Design and Optimization of Energy-Accuracy Tradeoff Networks for Mobile Platforms via Pretrained Deep Models

NITTHILAN KANAPPAN JAYAKODI, SYRINE BELAKARIA, ARYAN DESHWAL, and JANARDHAN RAO DOPPA, School of Electrical Engineering and Computer Science, Washington State University

Many real-world edge applications including object detection, robotics, and smart health are enabled by deploying deep neural networks (DNNs) on energy-constrained mobile platforms. In this article, we propose a novel approach to trade off energy and accuracy of inference at runtime using a design space called Learning Energy Accuracy Tradeoff Networks (LEANets). The key idea behind LEANets is to design classifiers of increasing complexity using pretrained DNNs to perform input-specific adaptive inference. The accuracy and energy consumption of the adaptive inference scheme depends on a set of thresholds, one for each classifier. To determine the set of threshold vectors to achieve different energy and accuracy tradeoffs, we propose a novel multiobjective optimization approach. We can select the appropriate threshold vector at runtime based on the desired tradeoff. We perform experiments on multiple pretrained DNNs including ConvNet, VGG-16, and MobileNet using diverse image classification datasets. Our results show that we get up to a 50% gain in energy for negligible loss in accuracy, and optimized LEANets achieve significantly better energy and accuracy tradeoff when compared to a state-of-the-art method referred to as Slimmable neural networks.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Computer systems organization** → *Embedded software*;

Additional Key Words and Phrases: Deep neural networks, inference, embedded systems, hardware and software codesign

ACM Reference format:
Nitthilan Kanappan Jayakodi, Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. 2020. Design and Optimization of Energy-Accuracy Tradeoff Networks for Mobile Platforms via Pretrained Deep Models. *ACM Trans. Embed. Comput. Syst.* 19, 1, Article 4 (February 2020), 24 pages.
https://doi.org/10.1145/3366636

1 INTRODUCTION

Deep neural networks (DNNs) are increasingly deployed in mobile platforms to enable diverse edge applications including image classification, activity recognition, robotics, and mobile health.

This work was supported in part by the National Science Foundation (NSF) under Grant OAC-1910213 and Grant IIS-1845922, and in part by the U.S. Army Research Office (ARO) under Grant W911NF-17-1-0485 and Grant W911NF-19-1-0162. The views expressed are those of the authors and do not reflect the official policy or position of NSF or ARO.
Authors' addresses: N. K. Jayakodi, S. Belakaria, A. Deshwal, and J. R. Doppa, School of Electrical Engineering and Computer Science, Washington State University, PO Box 642752, Pullman, WA, 99164-2752; emails: {n.kannappanjayakodi, syrine.belakaria, aryan.deshwal, jana.doppa}@wsu.edu.

1539-9087/2020/02-ART4 $15.00
https://doi.org/10.1145/3366636

The size of real-world DNNs is growing rapidly in order to achieve higher accuracies. For example, DenseNet has up to 264 layers [19] and ResNet has up to 152 layers [15]. However, such large and complex DNNs require a significant amount of resources (computational and power) to perform repeated inferences. Since mobile platforms are constrained by resources (power, computation, and memory), there is a great need for low-overhead solutions to perform energy-constrained inference. We performed all our experiments for deploying DNN models on an ODROID-XU4 board [31]. As we discussed in a related work section, prior work in this problem space has considered several methods based on hardware, software, and hardware/software codesign. However, there are limited approaches that *leverage pretrained DNNs* to automatically trade off energy and accuracy of inference at *runtime* on a target mobile device.

In this article, we propose a novel approach based on a formalism referred to as *LEANets* to trade off energy and accuracy of inference on mobile platforms. Our approach is complementary to most of the existing methods in the sense that it can reuse a pretrained DNN toward this goal. There are two key ideas behind our approach. First, we design the LEANet model as a sequence of classifiers of increasing complexity by using varying fractions of channels from the pretrained DNN. The main benefit of this design is the *reuse of computation* from classifiers at lower levels to perform incremental computation at higher levels *only if needed*. Second, some learned parameters (one threshold for each classifier in the sequence) of the LEANet model allow us to perform *adaptive inference* depending on the hardness of input examples. Figure 2 illustrates that easy images can be classified with simple classifiers to save energy and we only need complex classifiers for hard images. By varying the threshold values for different classifiers, we can achieve different energy and accuracy tradeoffs. We develop a novel methodology to optimize the design space of LEANet models conditioned on the given pretrained DNN. This includes selecting the number of levels, selecting the complexity of the classifier in each level, and finding the set of threshold vectors that achieve the best Pareto front for energy and accuracy tradeoff.

Contributions. The main contributions of this article include:

- A novel formalism referred to as LEANets to trade off energy and accuracy of inference with deep neural networks.
- Development of a principled methodology to design and optimize LEANets using a pretrained deep neural network to obtain the Pareto front of energy and accuracy tradeoff for a target mobile platform.
- Comprehensive experiments on pretrained deep networks including ConvNet, VGG16, and MobileNet to show the generality and effectiveness of our approach. Results show that we get up to a 50% gain in energy for a negligible loss in accuracy, and optimized LEANets achieve significantly better energy and accuracy tradeoff when compared to a state-of-the-art method referred to as Slimmable neural networks.

2 RELATED WORK

Hardware Approaches. Methods include the design of specialized hardware using data-flow knowledge [4], placement of memory closer to computation units [12], and on-chip integration of memory and computation [3]. Apart from the constraint of having an application-specific hardware, most of these solutions also have an overhead of analog-to-digital conversion [38].

Software Approaches. Most important software-level methods are as follows: reduced precision of weights and activations [28, 43]; binary weights [6] and variations like LightNN [9]; pruning and compression of large DNNs [14, 17], including "energy-aware" pruning that was shown to be more energy efficient [41]; and exploiting sparsity of DNN models [4]. Another class of methods

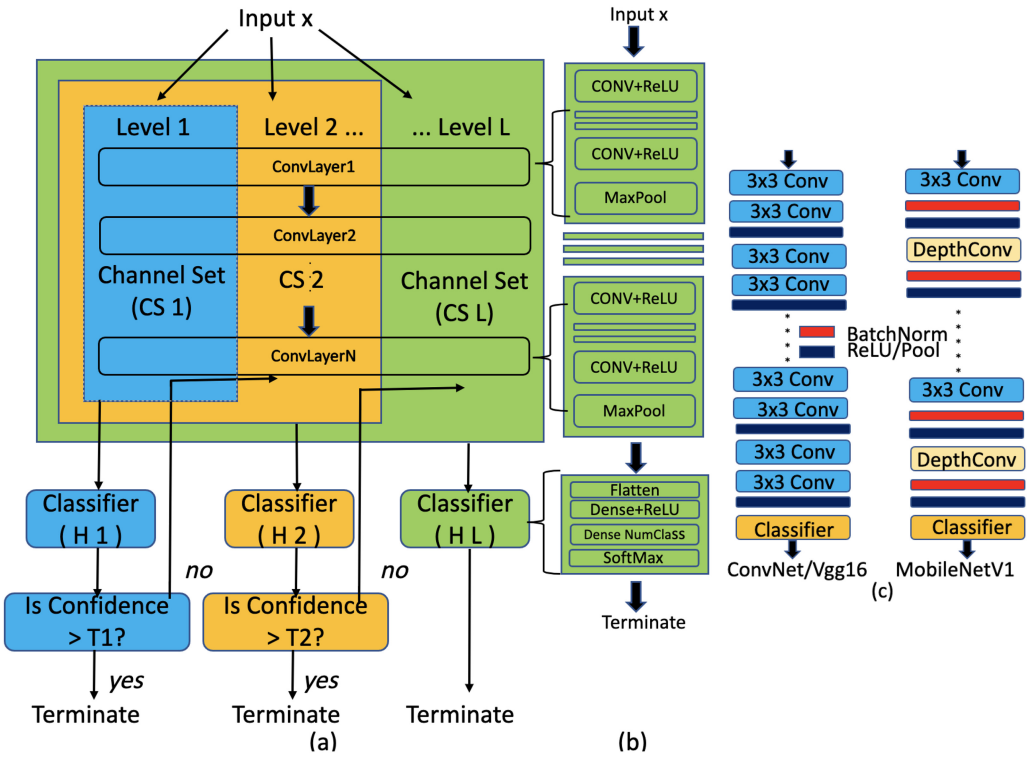


Fig. 1. Illustration of LEANet model. Given a pretrained network as shown in (b), we can split into multiple levels with reusable computation from lower levels as shown in (a). The nested boxes for different channel sets represent the reuse of computation from lower levels at higher levels. (c) Neural architectures of some popular DNNs including ConvNet, VGG16, and MobileNet.

includes the design of DNN architectures to reduce the computation and model size. The Inception module [39], Xception [5], and Mobilenets [18] employ depth-wise separable convolutions. The SqueezeNet [20] introduces a fire module that squeezes the filter dimension followed by expansion to reduce the number of filter coefficients, which helps in achieving the accuracy of AlexNet with 50× fewer parameters.

There are also approaches that employ a sequence of networks to perform conditional inference depending on the input example to save computation and energy. A cascaded CNN architecture for face detection [27] progressively prunes areas of the image that are not likely to contain a face. However, CNNs used in [27] do not reuse any features. The iterative CNN (ICNN) approach [30] makes multistage predictions for images, where a two-stage wavelet transform is applied to produce multiple small-scale images and train separate models to process them progressively to make predictions. Adaptive NN [37] employs two DNNs—a small one on a mobile and a large one on a remote server—and performs joint optimization. However, this approach is significantly suboptimal: it only allows two levels and does not reuse computation from a small DNN.

Recent work on the conditional deep learning (CDL) model [21, 32] also falls in this category. The CDL model [32] is related to our work, but it has a completely different architecture than LEANets. All these methods are characterized by a specific hand-designed sequence of networks that is tuned in an *ad hoc* manner without considering the mobile platform for deployment ([21] is one exception). In contrast, our proposed LEANets-based approach *automatically* defines a design

space of such a sequence of networks using *any* pretrained DNN model and selects the optimized LEANet design to trade off accuracy and energy of inference for the target mobile platform.

Hardware and Software Codesign Approaches. Many of these approaches involve design of hardware guided by software-level optimization techniques. [28] proposes a compiler specific to FPGA that analyzes CNNs and generates modules to improve the throughput of the system. In [13], the authors propose the design of the Energy Inference Engine (EIE) that deploys pruned DNN models. In [24], the knowledge of compressed sparse weights guides the hardware to read weights and perform MAC computations only for the nonzero activations, thereby reducing the energy cost by 45%. In [21], a *hand-designed* sequence of simple to complex networks was configured for a target mobile platform. Our LEANets-based approach automatically finds the optimized sequence of classifiers and can be seen as complementary to [21]. Slimmable neural networks (SlimNets) [42] are a very recent state-of-the-art approach that employs the adaptive computation graph technique. The approach behind SlimNets involves training a single neural network at different widths (number of channels). At runtime, we can adaptively execute SlimNet with one of the predefined widths as needed based on the resource constraints on a mobile platform.

Our proposed LEANet-based approach has several advantages over prior methods: **(1)** LEANet is complementary to methods including quantization, model pruning, hand-designed networks, and knowledge distillation. Specifically, we can consider LEANet as a wrapper approach that takes the output network from these methods to automatically trade off energy and accuracy. **(2)** LEANet employs the channels from the pretrained DNN given as input and only trains a small number of classifiers. The classifier training overhead is negligible when compared to SlimNets. **(3)** LEANet can achieve finer energy-accuracy tradeoffs even with a fixed number of levels by changing the confidence threshold vectors employed to perform input-specific adaptive inference. On the other hand, SlimNets with predefined width values can only achieve a coarse-grained tradeoff. **(4)** LEANet does not employ any extra network modules to perform dynamic inference.

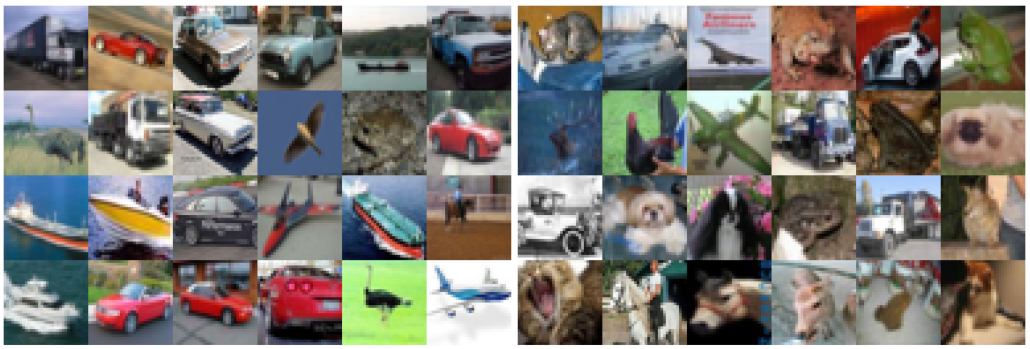
3 OVERVIEW OF LEANET FRAMEWORK

In this section, we provide a high-level overview of the Learning Energy Accuracy Tradeoff Networks (LEANet) framework. First, we explain the key insights behind LEANet. Second, we formally describe the LEANet model and the associated inference procedure.

3.1 Motivation

Deploying large DNN models on mobile platforms requires a lot of resources in terms of computation and energy. DNN architectures like MobileNet address the challenge of constrained resources on mobile platforms by compressing the network with some loss in accuracy. Even though these architectures address the resource requirements for inference, they still involve a huge cost for training them from scratch for a new real-world (edge) application. Toward the goal of overcoming their drawbacks and further improving the energy and computational efficiency of state-of-the-art DNNs like MobileNet, we propose the LEANet model.

LEANet can be interpreted as a wrapper approach that takes a pretrained DNN model as input and reuses the learned weights to automatically build a multilevel DNN, where the level number is proportional to the complexity of the corresponding classifier. The key idea is to slice the pretrained DNN vertically into sets of filter maps/channels and progressively go from simpler to more complex classifier *only if needed,* thereby reusing the computation of lower levels at higher levels to achieve up to a 50% reduction in energy consumption and execution time with little to no loss in accuracy.



(a) Images predicted at level 1 (b) Images predicted at level 5

Fig. 2. Images predicted at lowest and highest levels for LEANet(MobileNet) with CIFAR10.

Figures 2(a) and 2(b) show some sample images from our experiments that are predicted at level 1 and level 5, respectively, using an instantiation of LEANet with five levels based on MobileNet. Images classified using level 1 are simpler with a single object and clear background, while images classified using level 5 have overlapping, skewed, or hidden objects with confusing backgrounds. These results provide evidence for the key intuition behind our proposed LEANet approach.

3.2 LEANet Model and Inference

A LEANet model $\mathcal{M}$ with L levels is a sequence of classifiers of growing complexity resulting from an increased number of channels with growing levels as shown in Figure 1. There are three key components for each level in a LEANet model.

1) Channel sets: This consists of a fraction of learned filter maps/channels for each layer in the pretrained DNN. Each successive level employs an increasing number of channels. The parameters (denoted by CS_i) of channel sets are reused from the given pretrained DNN in the LEANet model. It takes the features computed in previous level L_{i-1} and input image (x) as input and produces more complex features L_i.

2) Classifier: This is a classification layer that takes the features computed by a given channel set and produces a probability distribution over all labels. In the LEANet model, we only learn the parameters (denoted by H_i) of this classification layer.

3) Confidence threshold. Given a predicted probability distribution over candidate labels, we can estimate the confidence of the classifier [40] using the *maximum probability*, where we take the probability of the highest scoring label. The confidence threshold T_i is employed to determine if the classifier is confident enough in its prediction to terminate and return the predicted label.

Inference in LEANet Model. Given a LEANet model $\mathcal{M}$ with all its parameters (CS_i, H_i, T_i for all levels $1 \leq i \leq L$) fully specified, inference computation for input example x is performed as follows. We sequentially go through the LEANet model starting from level 1. At each level i, the channel set CS_i takes the input x and reuses the features computed from the previous level CS_{i-1} to produce the features of CS_i. Then classifier layer H_i makes predictions using the features resulting from the given channel set CS_i. If the estimated confidence parameter $\hat{T}_i$ meets the threshold T_i or we reach the final level L, we terminate and return the predicted output $\hat{y}$.

4 DESIGN SPACE OF LEANET MODELS

In this section, we describe the design space of candidate LEANet models that can be constructed using a given pretrained DNN (e.g., MobileNet). For the sake of understanding, we first explain the case of the LEANet model with two levels and then generalize the space for more than two levels. We employ CNNs for image classification as a running example for illustrative purposes.

As shown in Figure 1(a), we divide the convolution layers of the given pretrained neural network (CNN) into multiple levels by using an increased number of channel sets with growing levels. Any general CNN architecture is composed of several convolution layers as shown in Figure 1(b). Let a convolution layer consist of N channels. For an L level LEANet, our method selects n_i channels of each convolution layer for each level i, where $\forall j > i, \sum_{k=1}^{j} n_k > \sum_{k=1}^{i} n_i$, and $\sum_{k=1}^{n} n_k = N$. Intuitively, each successive level builds a classifier of increasing complexity resulting from the increased number of channels n_i in each layer.

Two-Level LEANet. For illustration, we construct a two-level LEANet. Consider a convolution layer with a filter of width (C), height (C), and depth (M) convolving over an input layer (I) of M channels to produce an output layer (O) of N channels. We describe splitting at two levels using M_1 input channels and N_1 output channels for level L_1, and $M_1 + M_2 = M$ input channels and $N_1 + N_2 = N$ output channels for level L_2. In level L_1, the first N_1 output channels (O_n) are obtained from the first M_1 input channels (I_m) as given by

$$O_n^{L_1}[a,b] = \sum_{m=1}^{M_1} \sum_{i=-C/2}^{C/2} \sum_{j=-C/2}^{-C/2} I_m[i+a, j+b] \cdot w_{mn}[i,j], \quad \forall n \in 1, 2 \ldots N_1 \tag{1}$$

where w_n represents filter weights. The above equation can be rewritten as

$$O_n^{L_1}[a,b] = \sum_{m=1}^{M_1} C_m^*[a,b] \quad \forall n \in 1, 2 \ldots N_1$$
$$C_m^*[a,b] = \sum_{i=1}^{C} \sum_{j=1}^{C} I_m[i+a, j+b] \cdot w_n[i,j], \tag{2}$$

where C_m^* represents the convolution operation.

In level L_2, we consider all N output channels. The equation for the first N_1 output channels is as follows:

$$O_n^{L_2}[a,b] = \sum_{m=1}^{M_1+M_2} C_m^*[a,b] \quad \forall n \in 1, 2 \ldots N_1$$
$$O_n^{L_2}[a,b] = \sum_{m=1}^{M_1} C_m^*[a,b] + \sum_{p=M_1+1}^{M_2} C_p^*[a,b], \tag{3}$$

where the first part in Equation (3) can be reused from level L_1. This reuse of computation is illustrated in Figure 3. Furthermore, the remaining N_2 new output channels used in L_2 can be computed similar to Equation (1).

For a numerical illustration, consider a two-level LEANet setup for a convolution layer with $M = 48$ input channels and $N = 96$ output channels. Let $M_1 = 32$, $N_1 = 64$, $M_2 = 16$, and $N_2 = 32$. The computation in L_2 is proportional to MN of which we are reusing the M_1N_1 part from L_1.

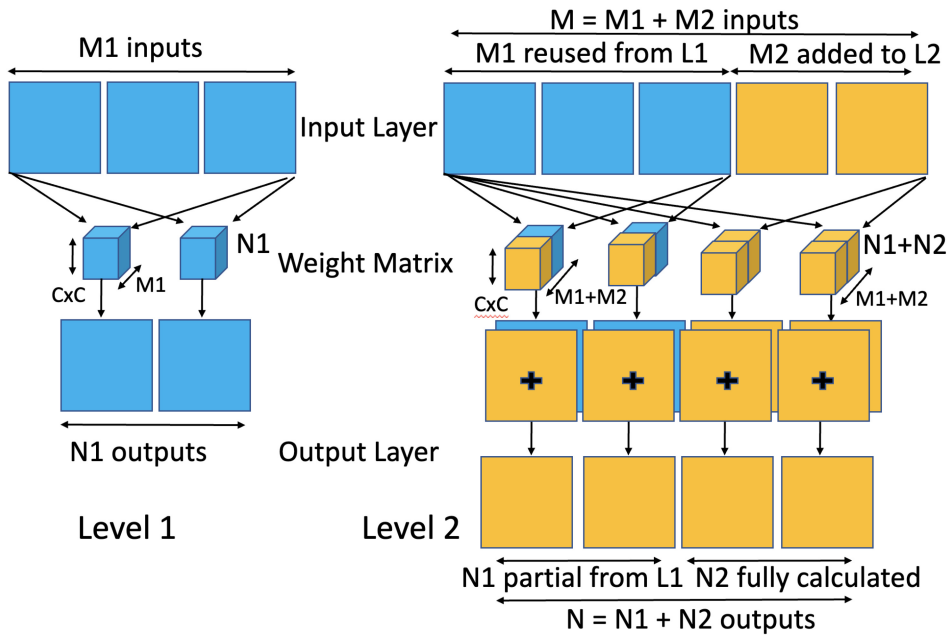


Fig. 3. Illustration of a two-level LEANet model with one convolution layer. The blue part from level 1 is reused in level 2.

Therefore, the reuse ratio is

$$\frac{M_1 \cdot N_1}{M \cdot N} = \frac{M_1 \cdot N_1}{(M_1 + M_2) \cdot (N_1 + N_2)} = \frac{32 \cdot 64}{48 \cdot 96} = 0.44.$$

The illustrative demonstration of the convolutional layer with two levels shows how much we are reusing the computation from the previous level to the next level. Equation (3) corresponds to this demonstration without pooling and nonactivation layers. Specifically, we are not reusing the solutions from the pooling and nonactivation layer operations. Instead, the partial output of the convolutional layer from previous level i-1 and the new convolutions at level i are *jointly* passed through the pooling and nonactivation layers (negligible computational overhead) to compute the input for the next convolutional layer for level i. We reuse the computation from the previous level in the next level to generate more features. Importantly, *these features are not necessarily equivalent to the features generated from scratch at each level in the base DNN.* The classifier used at the end of each level can compensate for the difference in the features generated (exact vs. approximate) by learning appropriate weights to be able to make correct predictions. Hence, learning classifiers tuned for approximate features allow us to trade off energy and accuracy of inference in our LEANet approach.

Multilevel LEANet. We can easily generalize the idea of LEANet with two levels to multiple levels ($L > 2$). As illustrated in Figure 3, while moving across successive levels, LEANet models reuse the computation from previous levels. To describe the general expression for this computational reuse factor, consider a multilevel LEANet model with $L > 2$ levels. Without loss of generality, consider a convolutional layer that is split into M_i input channels and N_i output channels corresponding to each level $1 \leq i \leq L$. Extending Equation (5), the reuse of computation going from level i to $i + 1$

is given by

$$\frac{\sum_{j=1}^{i} M_j \cdot \sum_{j=1}^{i} N_j}{\sum_{j=1}^{i+1} M_j \cdot \sum_{j=1}^{i+1} N_j}. \tag{4}$$

Though most networks use normal convolution, as shown in Figure 1(c), networks like MobileNet employ different types of layers. We now extend the LEANet splitting criterion to other layer types. *Pointwise 2D Convolution* is similar to a $C \times C$ convolution layer with $C = 1$. Thus, the reuse factor remains the same since M and N do not change. *Depthwise 2D Convolution* is another special case where the input channels $M = 1$. Therefore, we have single-parameter N output layers in LEANet. This makes the reuse factor $\frac{\sum_{j=1}^{i} N_j}{\sum_{j=1}^{i+1} N_j}$. *Batch Normalization* too has $M = 1$ like Depthwise 2D Convolution. This layer has four different parameters given by gamma weights, beta weights, moving mean (nontrainable), and moving variance (nontrainable). Thus, based on the above details, while constructing multiple levels, we carefully map the appropriate input channels to the appropriate output channels across levels to maximize the computational reuse, thereby minimizing the energy consumption of the overall LEANet model.

5 DESIGN OPTIMIZATION METHODOLOGY

In this section, we describe the details our optimization methodology to find the best LEANet model in terms of the Pareto front to trade off energy and accuracy of inference.

Design Space Definition. As explained in Section 4, each candidate LEANet model in our design space depends on three variables: number of levels (L), fraction of channels in channel set CS_i of each level given by $(F_1, F_2, \ldots, F_L)$, and confidence thresholds $(T_1, T_2, \ldots, T_{L-1})$ for each classifier H_i. Each F_i is equal to N_i/N, where N_i is the number of channels in level i and N is the total number of channels. This is a conditional search space in the sense that the size of the fraction-of-channels vector and confidence thresholds vector depend on the number of levels L.

Optimization Problem. Each candidate LEANet model (configuration of the three variables mentioned in the design space definition) corresponds to an accuracy and energy pair over a given evaluation set of classification examples. Given a pretrained DNN $\mathcal{N}$, training set TSR, and validation set VA of classification examples, our goal is to find the LEANet model configuration, which converges to the optimal energy and accuracy tradeoff Pareto front for a given target mobile platform $\mathcal{MP}$.

Optimization Methodology. As described in Algorithm 1, to obtain the optimal Pareto front, we iterate over the number of levels L starting from $L = 2$ until convergence. We determine the fraction of channels at each level $i = 1$ to L as follows. The final level L corresponds to the input pretrained DNN $\mathcal{N}$ that will achieve the highest accuracy. The fraction of channels at level 1 (F_1) is based on the *minimum energy budget* of target mobile platform $\mathcal{MP}$, which we want to use to be able to perform inference using an appropriate granularity of DNN. The highest level (level L) will employ all the channels ($F_L = 1.0\times$), consuming the maximum energy to provide the highest accuracy. The fraction of channels at levels ($i > 1$) is obtained by allocating the remaining channels equally for all the remaining L-2 levels. For example, if the minimum energy budget of mobile platform allows us to run at 0.5× of the base DNN, it would correspond to the lowest level. For three levels, the fraction of channels in channel sets would be [0.5×, 0.75×, 1.0×], and for four levels, the fraction of channels in channel sets would be [0.5×, 0.675×, 0.875×, 1.0×].

Given the number of levels L and fraction of channels in each level $(F_1, F_2, \ldots, F_L)$, we train the classifiers $H_1, H_2, \ldots, H_{L-1}$ with the training data TSR using stochastic gradient descent and back-propagation. H_L corresponds to the pretrained DNN. Subsequently, we determine a set of

confidence threshold vectors $\mathcal{T}_L$ based on the target mobile platform $\mathcal{MP}$ (to compute energy consumption) and the validation data VA (to compute accuracy) using a novel multiobjective Bayesian Optimization (BO) approach referred to as *UnPAc*. Each threshold vector $(T_1, T_2, \ldots, T_{L-1}) \in \mathcal{T}_L$ corresponds to a particular tradeoff between energy and accuracy. We observed diminishing returns as we increase the number of levels and Pareto fronts converge in at most five levels. Our overall optimization procedure including identifying the number of levels, training the classifiers at all levels, and finding the set of confidence threshold vectors corresponding to the energy and accuracy tradeoffs is executed *offline*. At *runtime*, we can select the appropriate confidence threshold vector for the required energy and accuracy tradeoff.

ALGORITHM 1: LEANet Design Optimization

Input: $\mathcal{N}$ = Pretrained deep model; TSR = Training set; VA = Validation set; and $\mathcal{MP}$ = Target mobile platform

1: **Initialization**: number of levels $L = 2$
2: **while** Convergence **do**
3: Set channel fractions $F_1, F_2, \ldots, F_L$ via domain knowledge
4: Train classifiers $H_1, H_2, \ldots, H_{L-1}$ using TSR
5: Threshold vectors $\mathcal{T}_L \leftarrow$ UnPAc($\mathcal{N}$, $\mathcal{MP}$, VA) // *Find the optimal Pareto front of energy and accuracy*
6: $L \leftarrow L + 1$ // *Increase the number of levels*
7: **end while**
8: **return** optimized LEANet model with L levels, classifiers at all levels, and confidence threshold vectors $\mathcal{T}_L$ to trade off energy and accuracy

5.1 UnPAc Algorithm to Estimate Thresholds

For a given number of levels L and the trained intermediate classifiers of a candidate LEANet model, our goal is to find a set of threshold vectors $\mathcal{T}_L$ that correspond to the Pareto front of energy and accuracy tradeoff. There are two key challenges in solving this problem: (1) The energy and accuracy objective functions are unknown and we need to perform experiments to evaluate each candidate threshold vector $(T_1, T_2, \ldots, T_{L-1}) \in \mathcal{T}$. In our specific problem, evaluation of both the objectives is expensive: evaluation of the energy objective on target mobile platform $\mathcal{MP}$ is much more expensive than the accuracy objective. (2) The objectives are conflicting in nature and they cannot be optimized simultaneously. Therefore, we need to find the *Pareto optimal* set of solutions. A solution is called Pareto optimal if one objective cannot be improved without compromising other objectives. The overall goal is to approximate the true Pareto set by minimizing the overall number of energy and accuracy evaluations.

Multiobjective Bayesian Optimization Formulation. We formulate the above problem in the framework of multiobjective Bayesian Optimization (BO): the input space $\mathcal{T}$ of all candidate threshold vectors is $[0, 1]^{L-1}$, and energy and accuracy over the validation set VA and target mobile platform $\mathcal{MP}$ as the two unknown objective functions. BO algorithms learn cheap surrogate models (e.g., Gaussian Process) from training data obtained from past evaluations of objective functions. They judiciously select the next candidate input $(T_1, T_2, \ldots, T_{L-1}) \in \mathcal{T}$ for evaluation by trading off exploration and exploitation to quickly direct the search toward approximating the true Pareto front. Acquisition functions are defined in terms of the statistical models to score the candidate inputs and guide this search process. PESMO [16] is the state-of-the-art approach to solve multiobjective BO problems based on entropy optimization. PESMO reduces the problem to single-objective optimization. It iteratively selects the input that maximizes the information gained about the true Pareto set. Unfortunately, this choice is suboptimal as it is hard to capture

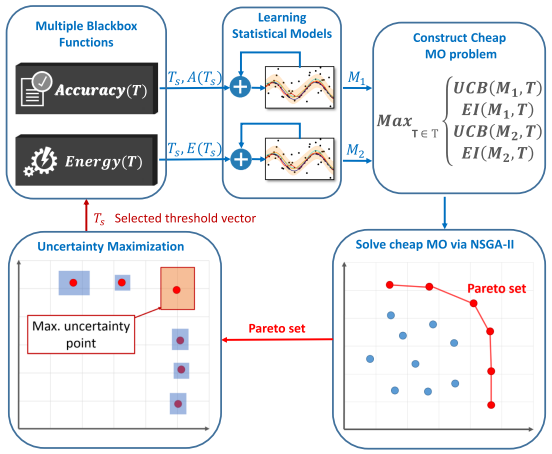


Fig. 4. Overview of the UnPAc algorithm.

the tradeoff between multiple objectives and can potentially lead to aggressive exploitation behavior. Indeed, our experiments demonstrate the inefficiency of PESMO. Motivated by the need for a more sample-efficient optimization approach, we propose a novel algorithm referred to as *Uncertainty Reduction via Portfolio Acquisition optimization (UnPAc).*

UnPAc Algorithm. As shown in Figure 4, UnPAc is an iterative algorithm that involves four key components as described below. The energy and accuracy objective functions are considered as the *two blackbox functions* to be optimized.

1) Learning statistical models. We build statistical models $\mathcal{M}_1$ and $\mathcal{M}_2$ for both unknown objective functions *accuracy* and *energy* from the training data in the form of past function evaluations. In each iteration of UnPAc, the learned statistical models $\mathcal{M}_1$ and $\mathcal{M}_2$ are employed to select the next candidate threshold vector $(T_1, T_2, \ldots, T_{L-1}) \in \mathcal{T}$ for evaluation.

2) Constructing cheap MO problem. We select a set of promising candidate inputs $\mathcal{T}_p \subset \mathcal{T}$ by solving a cheap multiobjective (MO) optimization problem defined using the statistical models $\mathcal{M}_1$ and $\mathcal{M}_2$. Specifically, we employ two acquisition functions for each statistical model: upper confidence bound (UCB) and expected improvement (EI) as defined below:

$$UCB(x) = \mu(x) + \beta^{1/2}\sigma(x) \tag{5}$$

$$LCB(x) = \mu(x) - \beta^{1/2}\sigma(x) \tag{6}$$

$$EI(x) = \sigma(x)(\alpha\Phi(\alpha) + \phi(\alpha)), \tag{7}$$

where $\alpha = \frac{\tau - \mu(x)}{\sigma(x)}$; $\mu(x)$ and $\sigma(x)$ correspond to the mean and standard deviation of the prediction from statistical model and represent exploitation and exploration scores, respectively; β is a parameter that balances exploration and exploitation, which is automatically specified based on the iteration number t based on theoretical analysis of convergence from Srinivas et al. [36]; τ is the best uncovered input; and Φ and ϕ are the CDF and PDF of normal distribution, respectively.

3) Solving the cheap MO problem. We use the popular NSGA-II algorithm [7] to solve the cheap MO problem to obtain the Pareto set $\mathcal{T}_p$ representing the most promising candidate inputs

for evaluation:

$$\mathcal{T}_p \leftarrow \max_{T \in \mathcal{T}} (\text{UCB}(\mathcal{M}_1, T), \text{EI}(\mathcal{M}_1, T), \text{UCB}(\mathcal{M}_2, T), \text{EI}(\mathcal{M}_2, T)).$$

The acquisition functions for each unknown function tell us the utility of evaluating a candidate threshold vector. For example, a threshold vector might have a high utility for accuracy but may have a lower utility for optimizing energy consumption. Therefore, the Pareto set found by solving the cheap MO problem captures the tradeoff between the utility for both unknown objective functions.

4) Uncertainty maximization. From the promising candidate set $\mathcal{T}_p$ obtained by solving the cheap MO problem, we need to select the best threshold vector such that it will guide the overall search toward the goal of quickly approximating the true Pareto set. We employ an uncertainty measure defined in terms of the statistical models $\mathcal{M}_1, \mathcal{M}_2$ to select the most promising candidate input for evaluation. We define the uncertainty measure as the volume of the uncertainty hyperrectangle:

$$U_{\beta_t}(T) = VOL(\{(LCB(\mathcal{M}_i, T), UCB(\mathcal{M}_i, T)\}_{i=1}^{2}), \tag{8}$$

where $\text{LCB}(\mathcal{M}_i, x)$ and $\text{UCB}(\mathcal{M}_i, T)$ represent the lower confidence bound and upper confidence bound, respectively, of the statistical model $\mathcal{M}_i$ for threshold vector T as defined in Equations (6) and (7). We measure the uncertainty volume measure for all threshold vectors $T \in \mathcal{T}_p$ and select the one with maximum uncertainty for evaluation: $T_s = \arg max_{T \in \mathcal{T}_p} U_{\beta_t}(T)$.

Finally, this selected threshold vector T_s is used for evaluation to get the corresponding energy $E(T_s)$ and accuracy $A(T_s)$. The next iteration starts after the statistical models $\mathcal{M}_1$ and $\mathcal{M}_2$ are updated using the new training example: input is T_s and output is $(E(T_s), A(T_s))$. Unlike prior methods including PESMO that reduce to single-objective optimization, UnPAc follows the above procedure to select better candidates for evaluation in each iteration. Recent work on multiobjective optimization [1] showed that output space entropy search is more effective than input space entropy search. It would be interesting future work to see how UnPAC compares to this approach.

6 EXPERIMENTS AND RESULTS

In this section, we first describe the experimental setup and then discuss results of LEANet along different dimensions.

6.1 Experimental Setup

Hardware Setup. All our experiments were performed by deploying DNN models on an `ODROID-XU4` board [31]. `ODROID-XU4` is an Octa-core heterogeneous multiprocessing system with ARM Big-Little architecture, which is very popular in current mobile devices. The `ODROID-XU4` board employs ARM Cortex-A15 Quad 2GHz CPU as the Big Cluster and Cortex-A7 Quad 1.4GHz CPU as the Little Cluster. The board boots up Ubuntu 16.04LTS with ODROID's Linux kernel version 3.10.y. We execute DNN models with Caffe-HRT [2], which employs the Caffe framework [22] with ARM Compute Library to speed up deep learning computations. We ran the software programs for our LEANet approach, full DNN, and SlimNets baseline on the mobile platforms to measure the energy consumption. Therefore, all the presented results are relative to the common software execution scheme. Studying optimized execution schemes for LEANet models to minimize memory overhead is part of our immediate future work. We employ SmartPower2 [34] to measure power. We compute the average power over the total execution time.

Energy Objective. Energy consumption of LEANet models is measured in terms of the normalized energy delay product (EDP). $\text{EDP} = \sum \text{P}\, \Delta\text{t} \cdot \text{T}$, where Δt is the time interval at which we record the power P and T is the total execution time. As power is measured at a regular interval, we simply

calculate EDP as EDP = $P_{avg} \cdot T^2$. This value is normalized with EDP of the pretrained DNN. We use EDP and energy interchangeably in the discussion of our results. However, all our presented results are for EDP unless specified otherwise.

Image Classification Task and Training Data. We demonstrate the general applicability of our LEANet-based approach using three different pretrained DNNs as shown in Figure 1(c): (1) Conv, a VGG-style neural network with 6 convolution layers; (2) VGG, a more complex network with 10 convolution layers; and (3) MobileNet. We ran extensive experiments using MNIST, CIFAR10, and SVHN image datasets to study the performance gains with classification tasks of varying difficulty. We used the standard 4:1:1 split ratio for the training, validation, and testing (10,000) set, respectively. Accuracy is measured as the fraction of input images whose labels are predicted correctly. We present all our results in terms of accuracy and normalized EDP for ease of exposition. To demonstrate the effectiveness of LEANets on large datasets, we ran experiments using the ImageNet dataset [33]. ImageNet has a training set of roughly 1.2 million images and around 50,000 validation and test images. Each image has 224 ×224 ×3 dimensionality and maps to one of the 1,000 candidate class labels. We present the results for ImageNet in terms of top 5 accuracy (commonly employed metric in the literature) and normalized EDP.

Classifier Training. We employ a multilayer Perceptron to learn classifiers at intermediate levels (H_i) as shown in Figure 1(b). In this multilayer Perceptron, input corresponds to the output of the final convolution layer and number of output class labels is the same as that of the classification task. The hidden layer is given a fixed value as a function of the size of the input feature vector. In practice, half the size of the input feature vector gives consistent results in all our experiments. To train the parameters of classifier (H_i) at different levels, we employ backpropagation using the RMS prop optimizer with learning rate of 0.0001 and a decay set to $1e^{-6}$. We use a batch size of 128 and run training for 200 epochs with sufficient data augmentation including horizontal flips, width, and height shifts. As shown in Figure 9, our design optimization algorithm converges at five levels in all cases. Therefore, for each pretrained DNN, we have to train four classifiers: 200 training epochs for only the classifier layer, which has a very negligible overhead (around 2% of the computational resources needed to train the full network).

Multiobjective BO for LEANet Models. For obtaining the confidence threshold vectors to trade off energy and accuracy, we employ different multiobjective BO algorithms. We compare our UnPAc algorithm with the state-of-the-art PESMO method [16]. We employ the PESMO code from the BO library Spearmint.[1] We use a GP-based statistical model with squared exponential (SE) kernel in all our experiments. The SE kernel is defined as $\kappa(x, x') = s \cdot \exp(\frac{-\|x-x'\|^2}{2\sigma^2})$, where s and σ correspond to scale and bandwidth parameters. These hyperparameters are estimated after every 10 function evaluations. We initialize the GP models for both objective functions by sampling initial points at random from a Sobol grid. This initialization procedure is the same as the one in-built in the Spearmint library. We run for a maximum of 200 iterations.

Pareto Hypervolume (PHV) Metric. PHV is a commonly employed metric to measure the quality of a given Pareto front [44]. PHV is defined as the volume between a reference point and the given Pareto front. After each BO iteration t (or number of function evaluations), we report the difference between the hypervolume of the ideal Pareto front ($\mathcal{T}^*$) and hypervolume of the estimated Pareto front ($\mathcal{T}_t$) by a given algorithm: $PHV_{diff} = PHV(\mathcal{T}^*) - PHV(\mathcal{T}_t)$. Since PHV_{diff} represents a regret function, when comparing algorithms, the smaller the better (i.e., the estimated Pareto front is closer to the ideal Pareto front).

[1] https://github.com/HIPS/Spearmint/tree/PESM.

Table 1. Number of Levels L and the Fraction of Channels (Width) at Different Levels When LEANet Optimization Approach Converges

Dataset (DNN)	Fraction of Channels (Width)
CIFAR10 (Conv/VGG/MobileNet)	0.55×, 0.625×, 0.75×, 0.875×, 1×
MNIST (Conv/VGG/MobileNet)	0.5×, 0.625×, 0.75×, 0.875×, 1×
SVHN (Conv/VGG/MobileNet)	0.5×, 0.625×, 0.75×, 0.875×, 1×

6.2 Results and Discussion

In this section, we first present the performance of a variant of the state-of-the-art method based on SlimNets followed by a discussion of improvements achieved by optimized LEANets. Subsequently, we explain the results of our design optimization methods including its convergence behavior and comparison of different multiobjective BO algorithms (UnPAc vs. PESMO). Finally, we present fine-grained analysis of optimized LEANet models.

Energy and Accuracy Tradeoffs with Varying Fraction of Channel. We employ each classifier at different levels of LEANet as shown in Table 1 to obtain the corresponding energy and accuracy values by performing *static* inference over all the testing examples. This gives us one energy and accuracy values pair for each level. It is easy to see that this setting is conceptually similar to Slimmable neural networks (SlimNet) with fixed-width values *without* the corresponding training to optimize its accuracy averaged for all widths [42]. For example, SlimNet(0.5×) corresponds to a neural network with a fraction of 50% channels. Figure 5 shows the energy and accuracy tradeoffs obtained using SlimNets with different widths. In this case, the tradeoff is obtained at five discrete points as per the widths shown in Table 1 for different datasets. We present the energy and accuracy tradeoff patterns for different datasets and pretrained DNN models. **SlimNet(Conv/SVHN):** From width 1.0× to 0.875×, we can see that accuracy drops by 2% and energy reduces by 35%. Similarly, from width 0.875× to 0.75×, accuracy drops by 8% and energy reduces by 65% w.r.t SlimNet(1.0×). **SlimNet(MobileNet/SVHN):** We see a 2.5% drop in accuracy and 35% gain in energy when we move from width 1.0× to 0.875×. **SlimNet(MobileNet/CIFAR10):** For a difficult dataset like CIFAR10, from width 1.0× to 0.875×, we can see that accuracy drops by 8.5% and energy reduces by 38%. Similarly, from width 0.875× to 0.75×, accuracy drops by 22% and energy reduces by 65% w.r.t SlimNet(1.0×). SlimNet(VGG) and dataset MNIST also show similar trends. In summary, we see significant gains in energy and relatively smaller losses in accuracy as we decrease the width in SlimNet irrespective of the difficulty of the dataset.

Optimized LEANet vs. SlimNets. Figure 6 shows the comparison of optimized LEANet and SlimNets as per the five level widths shown in Table 1. We compare them in terms of the Pareto front for energy and accuracy tradeoffs. We can see that optimized LEANet outperforms SlimNets in terms of the achievable tradeoffs for all cases. When compared to SlimNets, LEANets achieve a more fine-grained tradeoff between energy and accuracy by varying the threshold vectors employed to make classification decisions. We observe that LEANets reach the same accuracy values as SlimNet(1.0×) with a significant gain in energy. On the SVHN dataset, LEANet(MobileNet) and LEANet(VGG) achieve around **35%** and **45%** gains in energy, respectively, for achieving the maximum possible accuracy, i.e., SlimNet(1.0×). For CIFAR10, the corresponding energy gains are around **10%** and **40%,** respectively. We see a similar trend with the LEANet(ConvNet) and MNIST datasets. We observe that the energy gains we get with MNIST and SVHN are higher than those with CIFAR10. The significant energy gains using optimized LEANet are explained by the fact that many *easy* images are predicted at lower levels and only *hard* images are classified at higher levels. Remarkably, our LEANet-based approach is able to get energy savings even on MobileNet,

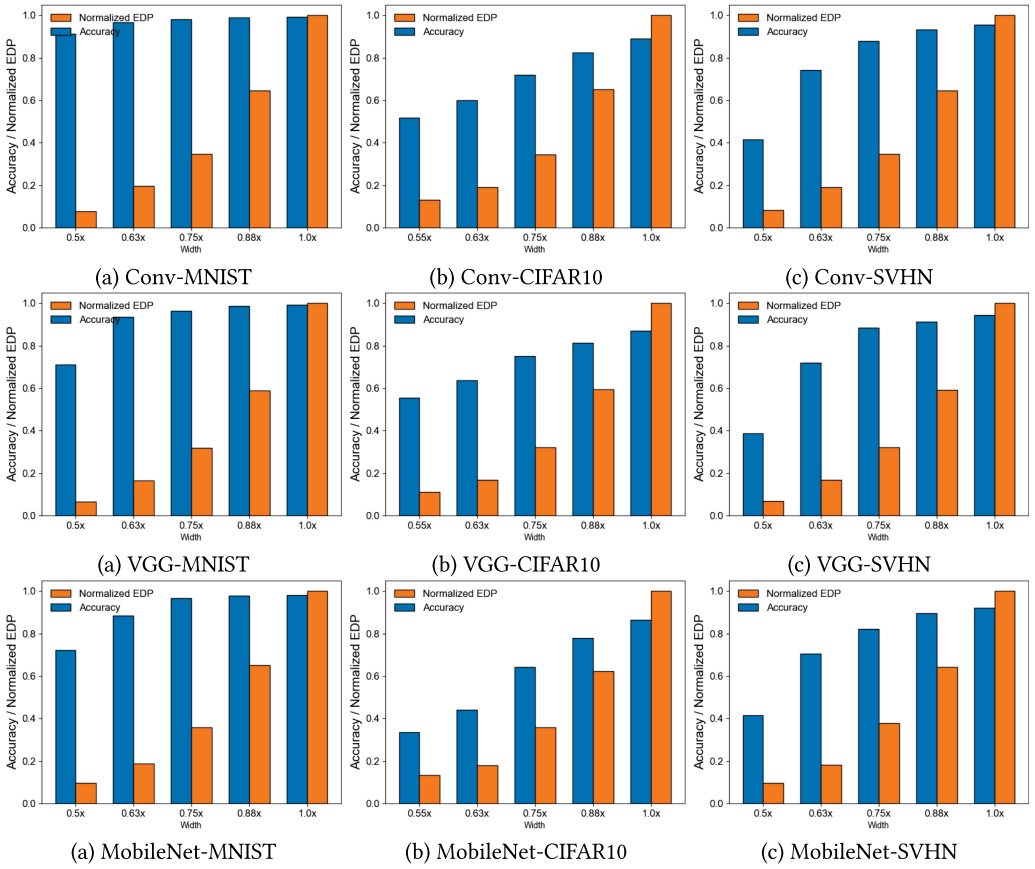
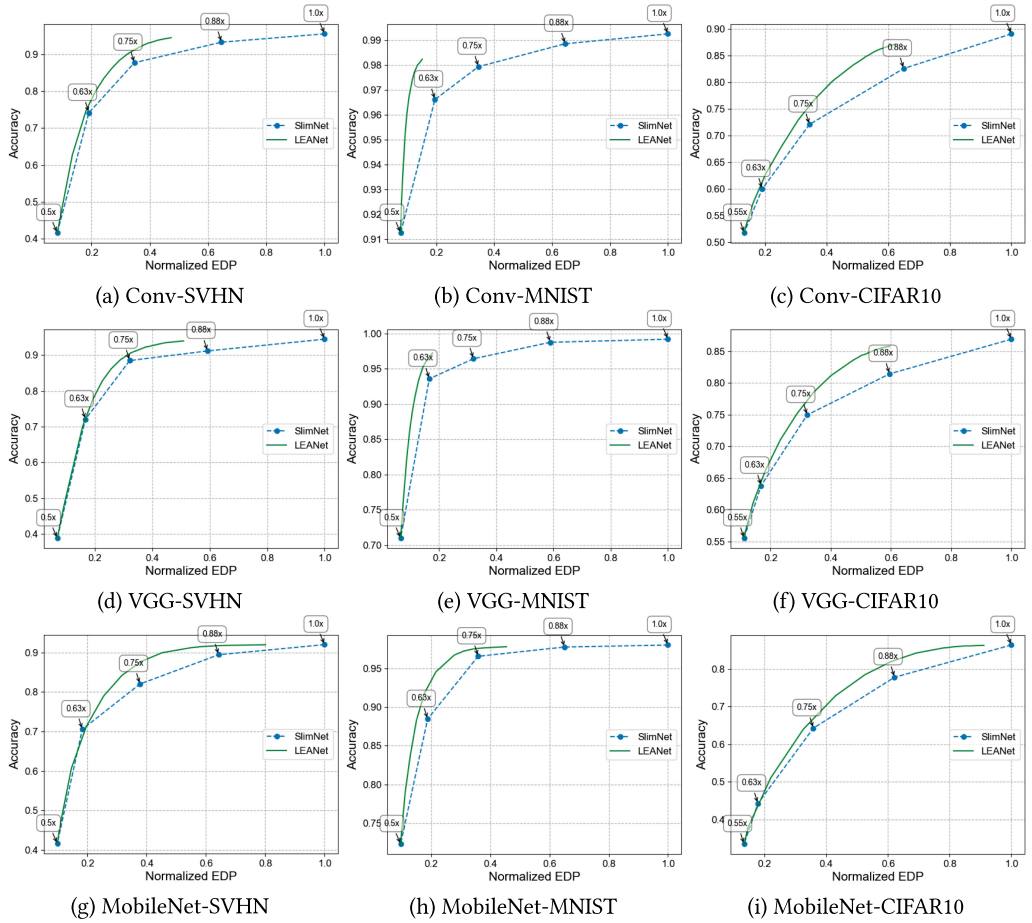


Fig. 5. Energy and accuracy of SlimNets with five different widths for each DNN and dataset. For CIFAR10, we use [0.55× 0.625× 0.75× 0.875× 1×], and for SVHN/MNIST, we use [0.5×, 0.625×, 0.75×, 0.875×, 1×].

which is hand-designed to save energy. In Figure 10, we show the gains in inference time obtained using optimized LEANets in comparison with the SlimNets baseline and observe a similar trend as energy.

Energy and Accuracy Tradeoff with Optimized LEANets. Optimized LEANets can achieve significant energy gains for a small loss in accuracy. Figure 7 shows the configurations that achieve 0.5%, 1.0%, 2.0%, and 5.0% accuracy loss for VGG and MobileNet to determine their energy gains for the SVHN and CIFAR10 datasets. For SVHN, a simple dataset, we see energy gains of 56% and 50% with an accuracy loss of 1% for LEANet(VGG) and LEANet(MobileNet), respectively. For CIFAR10, a more complex dataset, the corresponding energy gains are 40% and 22%, respectively. Similarly, for a 2% loss of accuracy, we see around 31% to 61% gains in accuracy for LEANets across different DNN and dataset combinations. The energy gains range from 39% to 70% for a 5% loss in accuracy. In summary, our approach using LEANet significantly reduces the energy consumption with a small loss in accuracy when compared to pretrained DNNs, i.e., SlimNets(1.0×).

Optimized LEANet Performance on ImageNet Dataset. To demonstrate the performance of LEANets on large datasets, we compare the MobileNet performance on ImageNets. In Figure 11(a), we present the energy and accuracy value pairs for each level for the fractions [0.5×, 0.625×,



(a) Conv-SVHN (b) Conv-MNIST (c) Conv-CIFAR10

(d) VGG-SVHN (e) VGG-MNIST (f) VGG-CIFAR10

(g) MobileNet-SVHN (h) MobileNet-MNIST (i) MobileNet-CIFAR10

Fig. 6. Energy and accuracy tradeoff results comparing optimized LEANet and SlimNets of different widths.

0.75×, 0.875×, 1×]. Similar to small datasets, from width 1.0× to 0.875×, the accuracy drops by 2%, resulting in an energy gain of 35%. From width 0.875× to 0.75×, the accuracy drops by another 2.5% for an energy gain of 30%, thus producing a significant gain in energy for a relatively small loss in accuracy. In Figure 11(c), we show the comparison of optimized LEANet and SlimNets as per the five level widths shown in Table 1 in terms of Pareto front of the energy and accuracy tradeoffs. Similar to other datasets, optimized LEANets not only offer a fine-grained tradeoff between energy and accuracy but also reach the same accuracy values as SlimNet(1.0×) with a significant gain in energy. In Figure 11(d), we show the gains in inference time obtained using optimized LEANets and observe a similar trend as energy. Finally, in Figure 11(b), we present the results for a significant energy gain for a small loss in accuracy.

Estimating Classifier Thresholds. We first compare the efficiency of multiobjective BO algorithms with a scalarization-based single-objective BO approach. We combine the energy and accuracy into a single objective using a scalarization parameter ($\alpha \in [0, 1]$) and run the single-objective BO algorithm with 10 different values of α in increments of 0.1 until convergence. To achieve the same Pareto front, our multiobjective BO algorithm (UnPAc) takes only 2% to 7% of the total evaluations of candidate threshold vectors taken by single-objective BO as shown in Table 2.

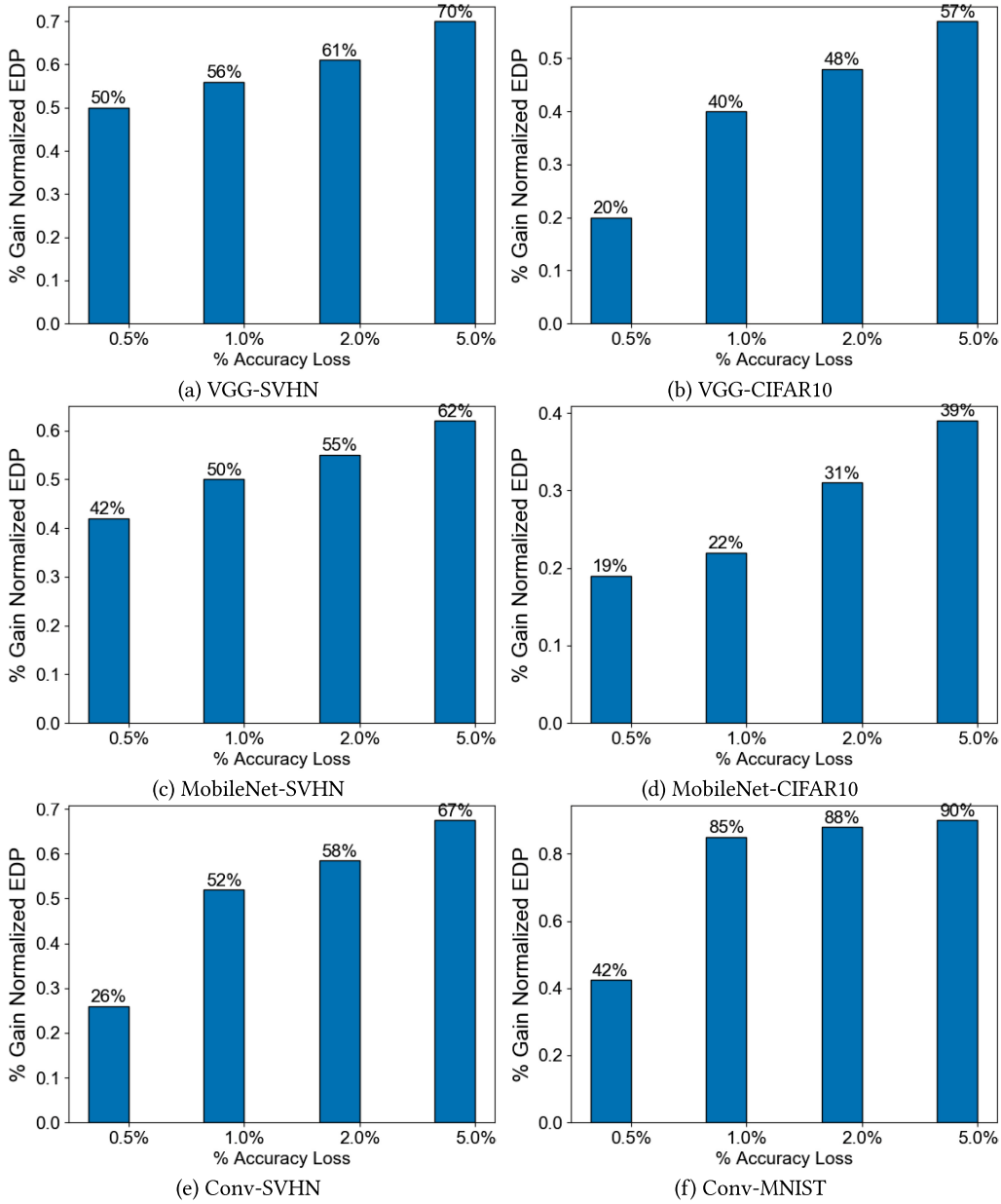


Fig. 7. Energy gain with different amounts of accuracy loss (0.5%, 1%, 2%, 5%) for optimized LEANet models.

We now compare UnPAc with PESMO, the state-of-the-art multiobjective BO method. Figure 8 shows the PHV difference as a function of the number of threshold vectors evaluated by the algorithms. We make the following observations: (1) UnPAc converges within 25 evaluations for all DNN and dataset combinations, except for MobileNet and MNIST, where it converges in 125 evaluations, and (2) UnPAc converges to a smaller regret value when compared to PESMO, resulting

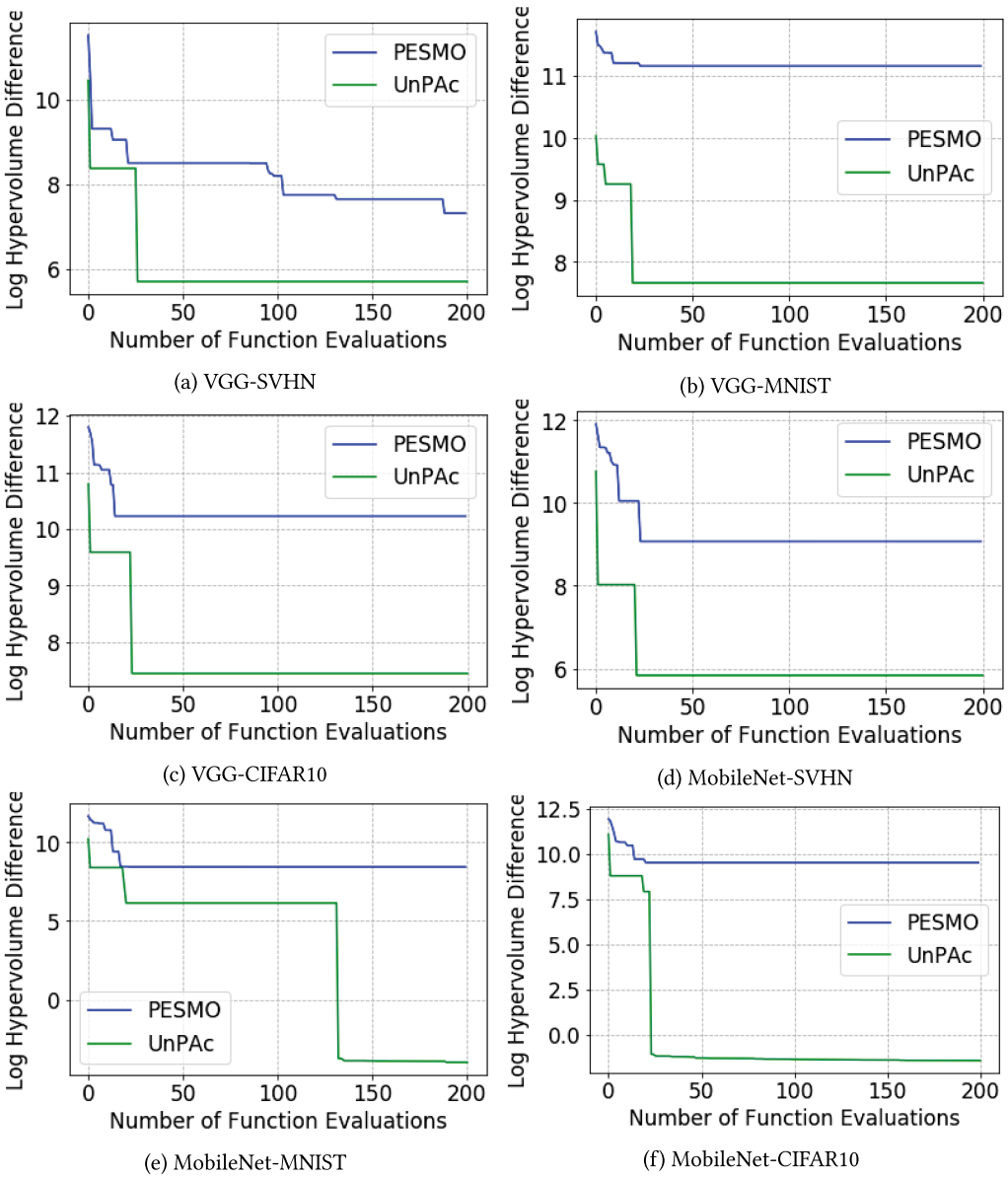


Fig. 8. Results comparing multiobjective BO algorithms UnPAc and PESMO to find the Pareto set of threshold vectors.

in a better Pareto front for energy and accuracy tradeoff. Since UnPAc is consistently shown to be better than PESMO, we will show all our results using the UnPAc algorithm.

Memory Requirement for Optimized LEANets. Since inference in LEANets occurs incrementally across levels until the confidence threshold to make the classification decision, we need to store intermediate computations to be reused across levels. In Table 3, we present the total memory required by the two optimized LEANets, namely, VGG19 and MobileNet, to perform inference for the ImageNet dataset on Odroid board. An Odroid board has a 2GB DDR memory. The presented

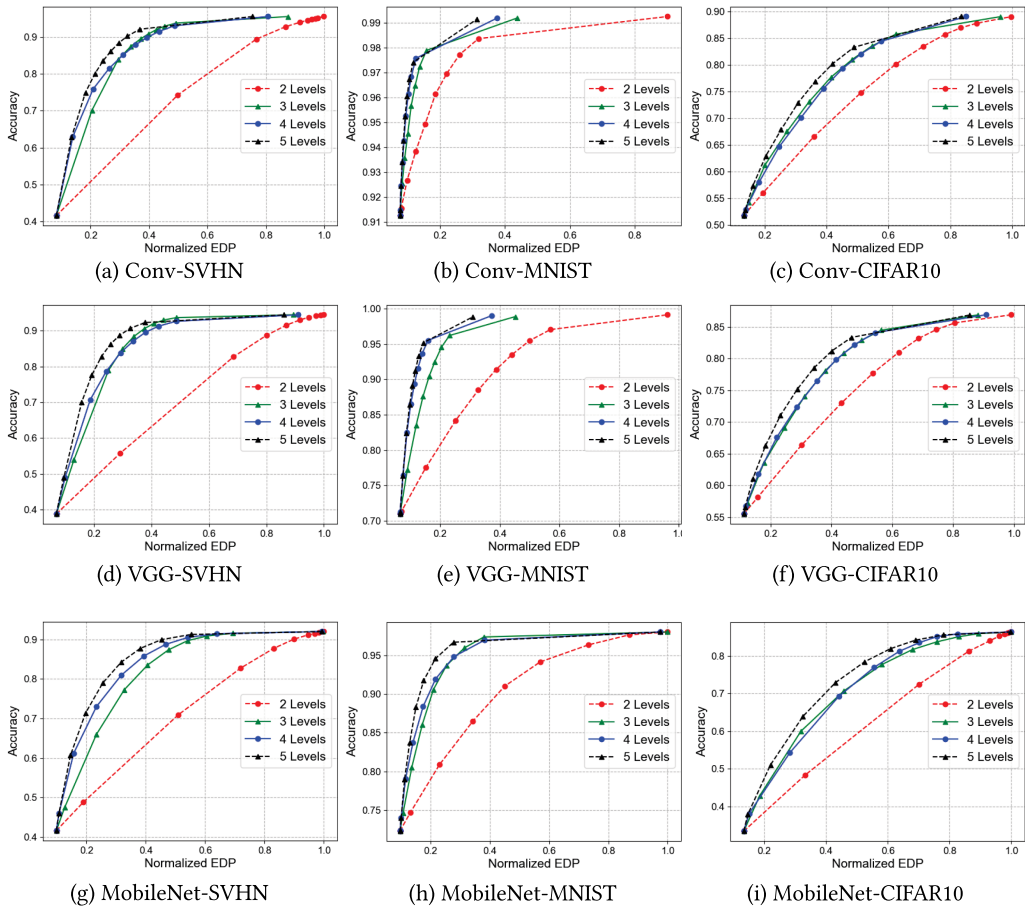
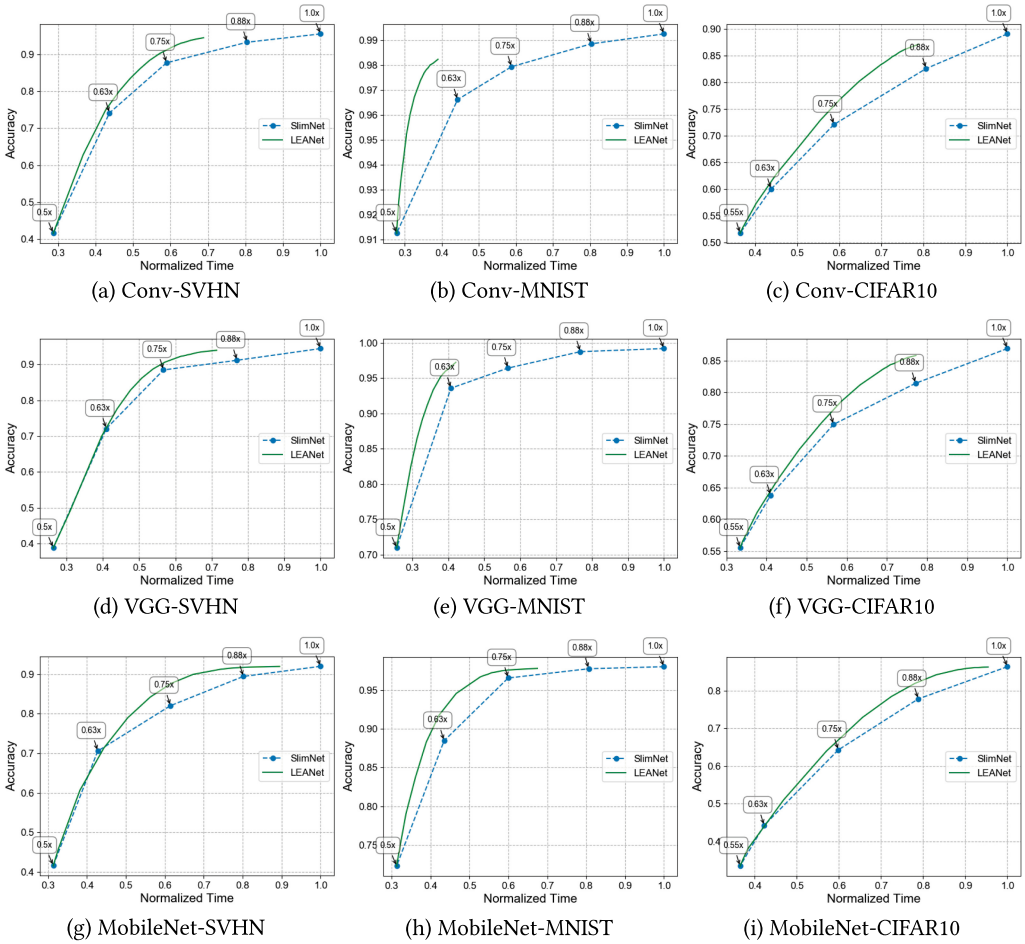


Fig. 9. Results of optimized LEANet with two levels [0.5×, 1.0×], three levels [0.5×, 0.75×, 1×], four levels [0.5×, 0.625×, 0.875×, 1×], and five levels [0.5×, 0.625×, 0.75×, 0.875×, 1×] for different DNN and dataset combinations.

Table 2. Results of UnPAc Algorithm in Terms of the Percentage of Candidate Threshold Vectors (w.r.t Total Evaluations by Scalarization-Based BO) Evaluated to Reach Convergence

DNN/Dataset	Percentage of Evaluations
VGG (SVHN/MNIST/CIFAR10)	1.25%
MobileNet (SVHN/CIFAR10)	1.25%
MobileNet (MNIST)	6.25%

memory results correspond to the setting when the prediction for a given input example is made at the last level of the optimized LEANet (*worst case*). From the results, we can see that the memory requirement is proportional to the number of layers in the network. A 19-layer VGG network consumes 21MB of memory and the 28-layer MobileNet consumes 41MB of memory, respectively. Recall that optimized LEANets make adaptive predictions depending on the hardness of the input examples: easy examples will be classified at lower levels (large fraction) and only a small number



(a) Conv-SVHN (b) Conv-MNIST (c) Conv-CIFAR10

(d) VGG-SVHN (e) VGG-MNIST (f) VGG-CIFAR10

(g) MobileNet-SVHN (h) MobileNet-MNIST (i) MobileNet-CIFAR10

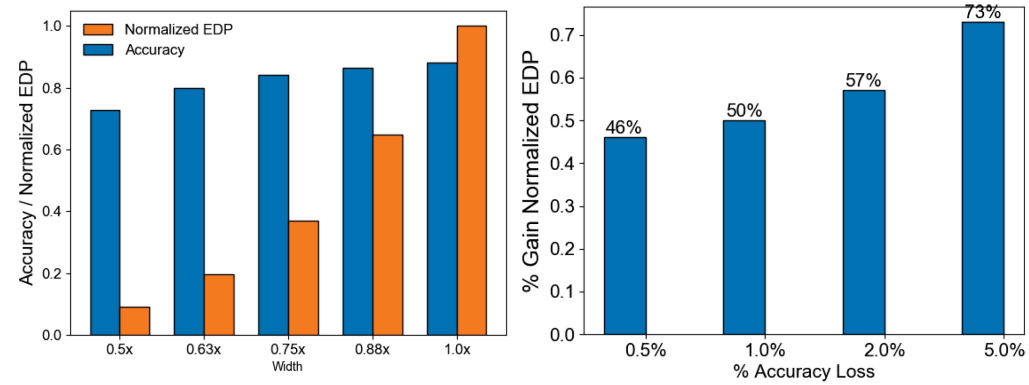
Fig. 10. Time and accuracy tradeoff results comparing optimized LEANet and SlimNets of different widths.

Table 3. Memory Requirement of Optimized LEANet Inference on Odroid Board for ImageNet Dataset

LEANets	Memory (MB)	Num Layers
VGG19	22	19
MobileNet	42	28

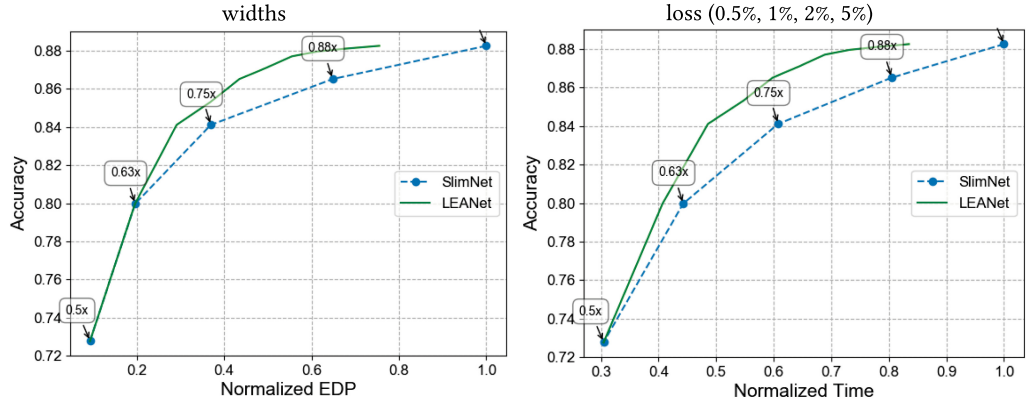
of hard examples will classified at higher levels (small fraction). Therefore, the memory overhead will be significantly smaller than the presented results for a large fraction of input examples. Overall, the memory overhead of LEANets to improve energy efficiency and inference time is negligible.

Convergence of LEANets Optimization. As explained in Section 5, our iterative optimization approach considers LEANet with increasing levels and stops at convergence. We employ the Pareto hypervolume indicator (PHI) metric to define the convergence criterion: the difference between PHI of Pareto fronts of two consecutive levels l and $l + 1$ is very small. Table 1 shows the number



(a) Energy and accuracy of SlimNets with 5 different widths

(b) Energy gain with different amounts of accuracy loss (0.5%, 1%, 2%, 5%)



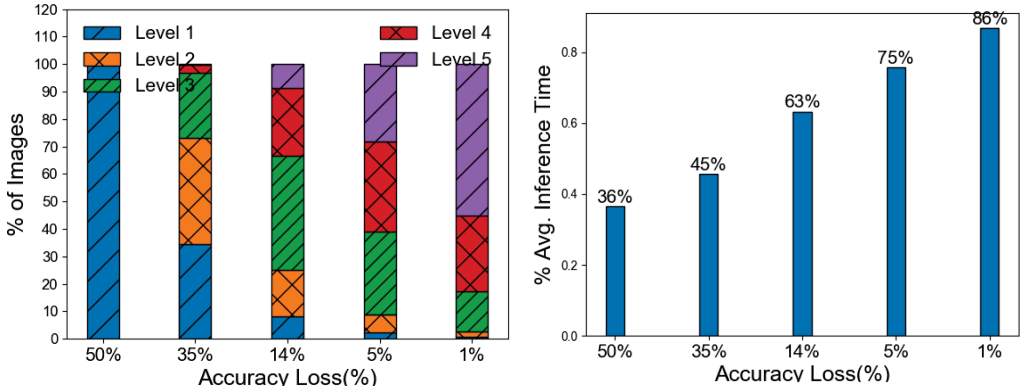
(c) Energy and accuracy trade-off optimized LEANet and SlimNets

(d) Time and accuracy trade-off optimized LEANet and SlimNets

Fig. 11. Optimized LEANets performance for large DNN MobileNet using ImageNet dataset.

of levels L and the fraction of channels at different levels when the LEANet optimization approach converges for each pretrained DNN and dataset pair. Figure 9 shows the convergence phenomenon of LEANets for all combinations of DNN and dataset pairs. The LEANet approach is parameterized by the number of levels. For a fixed number of levels (say L), a candidate threshold vector for each of the L-1 classifiers gives a concrete prediction engine. By executing this prediction engine on a set of input images, we can get the corresponding accuracy and energy consumed: a point on the normalized EDP vs. accuracy plot. For a fixed number of levels (say L), by varying the threshold vectors, we get different accuracy and energy values. The Pareto curve corresponds to the dominant solutions in the energy and accuracy tradeoff space obtained by the UnPAC algorithm. Therefore, we get one Pareto curve for a fixed number of levels L. We make the following observations: (1) going from two to three levels of LEANet, we see a significant improvement in the achievable energy and accuracy tradeoff, and (2) as we move beyond level 3, the improvements are reduced and converge in at most five levels. This is a practically beneficial result that shows that we can apply our design optimization approach to large-scale DNNs.

LEANet with Varying Accuracy. We perform fine-grained analysis to understand how LEANet achieved the above shown energy gains. We present this analysis for LEANet(MobileNet/CIFAR10), noting that analysis for other cases shows similar trends.



(a) % images predicted w/ levels (b) Prediction time

Fig. 12. Results for LEANet(MobileNet/CIFAR10) with varying accuracy loss.

The energy gains can be explained by understanding how many images from the testing set are classified at different levels. We make the following observations from Figure 12(a). With an accuracy loss of 50%, almost all the images are predicted at level 1 as the simplest classifier consumes the least energy. As the accuracy loss decreases, the number of images predicted by levels 2, 3, 4, and 5 slowly increase to achieve better accuracy. The key observation is that with accuracy loss of 1%, unlike SlimNet(1.0×), where all the images get predicted at level 5, we see that the 10,000 images are distributed as 5,512, 2,768, 1,453, 193, and 74 images across levels 5, 4, 3, 2, and 1, respectively. Since roughly 45% of images are predicted using a lower level, optimized LEANet models are able to achieve significant energy gains.

From Figure 12(b), we observe that the average prediction time gradually increases with a decrease in accuracy loss, which is explained by more and more images getting predicted at higher levels. With an accuracy loss of 35%, the prediction time of inference is 55% lesser when compared to the pretrained DNN, and with an accuracy loss of 5%, we save around 25% in prediction time. Therefore, in scenarios where we require real-time predictions, we can trade off the accuracy of LEANet for the target time-bound constraints.

Table 4 shows a sample Pareto front of LEANet(MobileNet) with five levels on the CIFAR10 dataset. It shows the candidate threshold vectors for variables T_1, T_2, T_3, T_4 obtained by the UnPAC algorithm.

7 CONCLUSIONS AND FUTURE WORK

We introduced a novel approach based on the LEANets formalism to trade off energy and accuracy of inference on mobile platforms at runtime. We also propose a principled algorithm to find optimized LEANet configurations by reusing pretrained neural networks. Our LEANet-based approach can be used as a complementary wrapper for other existing techniques to deploy deep neural networks on mobile devices. Results with optimized LEANets constructed from pretrained networks including VGG and MobileNet on image classification datasets including ImageNet show significant energy gains with minimal loss in accuracy. Future work includes applying dynamic resource management techniques [25, 29] and task mapping optimization based on machine learning [8, 23, 35] to further improve the power and thermal tradeoffs in heterogeneous mobile SoCs, and generalizing the LEANets formalism from simple classification tasks to structured output prediction tasks [10, 11, 26] including semantic segmentation and scene understanding that commonly arise in robotics and autonomous driving applications.

Table 4. Results of Pareto Front Obtained Using LEANet(MobileNet) with Five Levels on CIFAR10 Dataset

Accuracy	Norm. EDP	T4	T3	T2	T1
0.864	0.998	1.0	1.0	1.0	1.0
0.859	0.851	0.985	0.880	1.0	1.0
0.849	0.742	0.734	1.0	0.836	0.999
0.836	0.706	0.787	0.981	0.690	0.578
0.801	0.607	0.416	0.922	0.971	0.868
0.758	0.556	0.058	0.952	0.947	0.500
0.725	0.442	0.067	0.457	0.967	0.967
0.595	0.366	0.016	0.958	0.817	0.256
0.477	0.212	0.862	0.849	0.206	0.749
0.442	0.177	0.017	0.958	0.014	0.736
0.334	0.133	0.991	0.856	0.985	0.065

We show the candidate threshold vectors for variables T_1, T_2, T_3, and T_4 obtained by UnPAC to achieve different energy and accuracy trade-offs.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for the feedback that helped in improving the article.

REFERENCES

[1] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. 2019. Max-value entropy search for multi-objective Bayesian optimization. In *Advances in Neural Information Processing Systems (NeurIPS'19)*.

[2] Caffe-HRT. [n.d.]. Retrieved from https://github.com/OAID/Caffe-HRT.

[3] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, and Ling Li. 2014. DaDianNao: A machine-learning supercomputer. In *Proceedings of MICRO*. 609–622.

[4] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2017. Using dataflow to optimize energy efficiency of deep neural network accelerators. *IEEE Micro* 37, 3 (2017), 12–21. DOI: 10.1109/MM.2017.54

[5] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*. 1800–1807. DOI: 10.1109/CVPR.2017.195

[6] Matthieu Courbariaux and Yoshua Bengio. 2016. BinaryNet: Training deep neural networks with weights and activations constrained to +1 or −1. *CoRR* abs/1602.02830 (2016). arxiv:1602.02830 http://arxiv.org/abs/1602.02830.

[7] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, T. Meyarivan, and A. Fast. 2002. NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[8] Aryan Deshwal, Nitthilan Kannappan Jayakodi, Biresh Kumar Joardar, Janardhan Rao Doppa, and Partha Pratim Pande. 2019. MOOS: A multi-objective design space exploration and optimization framework for NoC enabled many-core systems. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 77:1–77:23. DOI: 10.1145/3358206

[9] Ruizhou Ding, Zeye Liu, R. D. Shawn Blanton, and Diana Marculescu. 2018. Quantized deep neural networks for energy efficient hardware-based inference. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference (ASP-DAC'18)*. DOI: 10.1109/ASPDAC.2018.8297274

[10] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. 2014. HC-Search: A learning framework for search-based structured prediction. *Journal of Artificial Intelligence Research* 50 (2014), 369–407.

[11] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. 2014. Structured prediction via output space search. *Journal of Machine Learning Research* 15, 1 (2014), 1317–1350.

[12] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and efficient neural network acceleration with 3D memory. *SIGARCH Computer Architecture News* 45, 1 (2017), 751–764. DOI: 10.1145/3093337.3037702

[13] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of ISCA*. 243–254.

[14] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural networks. *CoRR* abs/1506.02626 (2015). arxiv:1506.02626

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *CoRR* abs/1512.03385 (2015). arxiv:1512.03385 http://arxiv.org/abs/1512.03385.

[16] Daniel Hernandez-Lobato, Jose Miguel Hernandez-Lobato, Amar Shah, and Ryan P. Adams. 2016. Predictive entropy search for multi-objective Bayesian optimization. In *ICML*. 1492–1501.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2014. Distilling the knowledge in a neural network. In *NIPS Deep Learning Workshop*.

[18] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *CoRR* abs/1704.04861 (2017). arxiv:1704.04861 http://arxiv.org/abs/1704.04861.

[19] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. 2016. Densely connected convolutional networks. *CoRR* abs/1608.06993 (2016). arxiv:1608.06993 http://arxiv.org/abs/1608.06993.

[20] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50× fewer parameters and <1MB model size. *CoRR* abs/1602.07360 (2016).

[21] Nitthilan Kannappan Jayakodi, Anwesha Chatterjee, Wonje Choi, Janardhan Rao Doppa, and Partha Pratim Pande. 2018. Trading-off accuracy and energy of deep inference on embedded systems: A co-design approach. *IEEE TCAD* 37, 11 (2018), 2881–2893.

[22] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. *CoRR* abs/1408.5093 (2014). arxiv:1408.5093 http://arxiv.org/abs/1408.5093.

[23] Biresh Kumar Joardar, Ryan Gary Kim, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. 2018. Learning-based application-agnostic 3D NoC design for heterogeneous manycore systems. *IEEE Transactions on Computing* 68, 6 (2018), 852–866.

[24] Patrick Judd, Alberto Delmas, Sayeh Sharify, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *Proceedings of ISCA*. 1–13.

[25] Ryan Gary Kim, Wonje Choi, Zhuo Chen, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. 2017. Imitation learning for dynamic VFI control in large-scale manycore systems. *IEEE Transactions on VLSI Systems (TVLSI)* 25, 9 (2017), 2458–2471.

[26] Michael Lam, Janardhan Rao Doppa, Sinisa Todorovic, and Thomas G. Dietterich. 2015. HC-search for structured prediction in computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 4923–4932.

[27] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. 2015. A convolutional neural network cascade for face detection. In *Proceeding of CVPR*. 5325–5334.

[28] Yufei Ma, Naveen Suda, Yu Cao, Jae sun Seo, and Sarma Vrudhula. 2016. Scalable and modularized RTL compilation of convolutional neural networks onto FPGA. In *Proceedings of FPL*. 1–8.

[29] Sumit Mandal, Ganapati Bhatt, Chetan Arvid Patel, Janardhan Rao Doppa, Partha Pratim Pande, and Umit Ogras. 2019. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 12 (2019), 2842–2854.

[30] Katayoun Neshatpour, Farnaz Behnia, Houman Homayoun, and Avesta Sasan. 2018. ICNN: An iterative implementation of convolutional neural networks to enable energy and computational complexity aware dynamic approximation. In *Proceedings of DATE*.

[31] ODROIOD-XU4. 2017. Retrieved March 29, 2018, from https://wiki.odroid.com/odroid-xu4/hardware/hardware.

[32] Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. 2016. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *Proceedings of DATE*.

[33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. 2014. ImageNet large scale visual recognition challenge. *CoRR* abs/1409.0575 (2014). http://arxiv.org/abs/1409.0575

[34] SmartPower2. [n.d.]. Retrieved from https://wiki.odroid.com/accessory/power_supply%_battery/smartpower2.

[35] Das Sourav, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. 2017. Design-space exploration and optimization of an energy-efficient and reliable 3-D small-world network-on-chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 36, 5 (2017), 719–732.

[36] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. 2010. Gaussian process optimization in the bandit setting: No regret and experimental design. In *ICML*. 1015–1022.

[37] Dimitrios Stamoulis, Ting-Wu Chin, Anand Krishnan Prakash, Haocheng Fang, Sribhuvan Sajja, Mitchell Bognar, and Diana Marculescu. 2018. Designing adaptive neural networks for energy-constrained image classification. *CoRR* abs/1808.01550 (2018). arxiv:1808.01550 http://arxiv.org/abs/1808.01550.

[38] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE* 105, 12 (2017), 2295–2329.
[39] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 1–9. DOI: 10.1109/CVPR.2015.7298594
[40] Keze Wang, Dongyu Zhang, Ya Li, Ruimao Zhang, and Liang Lin. 2017. Cost-effective active learning for deep image classification. *CoRR* abs/1701.03551 (2017).
[41] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of CVPR*.
[42] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas S. Huang. 2019. Slimmable neural networks. In *ICLR*.
[43] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental network quantization: Towards lossless CNNs with low-precision weights. *Arxiv:1702.03044* (2017). arxiv:1702.03044
[44] Eckart Zitzler. 1999. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Vol. 63. Ithaca, NY: Shaker.

Received August 2019; revised September 2019; accepted September 2019