ECC: Platform-Independent Energy-Constrained Deep Neural Network Compression via a Bilinear Regression Model

Haichuan Yang[1], Yuhao Zhu[1], and Ji Liu[2,1]

[1]University of Rochester, Rochester, USA
[2]Kwai Seattle AI Lab, Seattle, USA

Abstract

Many DNN-enabled vision applications constantly operate under severe energy constraints such as unmanned aerial vehicles, Augmented Reality headsets, and smartphones. Designing DNNs that can meet a stringent energy budget is becoming increasingly important. This paper proposes ECC, a framework that compresses DNNs to meet a given energy constraint while minimizing accuracy loss. The key idea of ECC is to model the DNN energy consumption via a novel bilinear regression function. The energy estimate model allows us to formulate DNN compression as a constrained optimization that minimizes the DNN loss function over the energy constraint. The optimization problem, however, has nontrivial constraints. Therefore, existing deep learning solvers do not apply directly. We propose an optimization algorithm that combines the essence of the Alternating Direction Method of Multipliers (ADMM) framework with gradient-based learning algorithms. The algorithm decomposes the original constrained optimization into several subproblems that are solved iteratively and efficiently. ECC is also portable across different hardware platforms without requiring hardware knowledge. Experiments show that ECC achieves higher accuracy under the same or lower energy budget compared to state-of-the-art resource-constrained DNN compression techniques.

1. Introduction

Computer vision tasks are increasingly relying on deep neural networks (DNNs). DNNs have demonstrated superior results compared to classic methods that rely on handcrafted features. However, neural networks are often several orders of magnitude more computation intensive than conventional methods [34, 45]. As a result, DNN-based vision algorithms incur high latency and consume excessive energy, posing significant challenges to many latency-sensitive and energy-constrained scenarios in the real-world such as Augmented Reality (AR), autonomous drones, and mobile robots. For instance, running face detection continuously on a mobile AR device exhausts the battery in less than 45 minutes [20]. Reducing the latency and energy consumption of DNN-based vision algorithms not only improves the user satisfaction of today's vision applications, but also fuels the next-generation vision applications that require ever higher resolution and frame rate.

Both the computer vision and hardware architecture communities have been actively engaged in improving the compute-efficiency of DNNs, of which a prominent technique is compression (e.g., pruning). Network compression removes unimportant network weights, and thus reduces the amount of arithmetic operations. However, prior work [38, 39, 41] has shown that the number of non-zero weights in a network, or the network sparsity, does not directly correlate with execution latency and energy consumption. Thus, improving the network sparsity does not necessarily lead to latency and energy reduction.

Recognizing that sparsity is a poor, indirect metric for the actual metrics such as latency and energy consumption, lots of recent compression work has started directly optimizing for network latency [39, 11] and energy consumption [38], and achieve lower latency and/or energy consumption compare to the indirect approaches. Although different in algorithms and implementation details, these efforts share one common idea: they try to search the sparsity bound of each DNN layer in a way that the whole model satisfies the energy/latency constraint while minimizing the loss. In other words, they iteratively search the layer sparsity, layer by layer, until a given latency/energy goal is met. We refer to them as *search-based* approaches.

The effectiveness of the search-based approaches rests on how close to optimal they can find the per-layer sparsity combination. Different methods differ in how they search for the optimal sparsity combination. For instance,

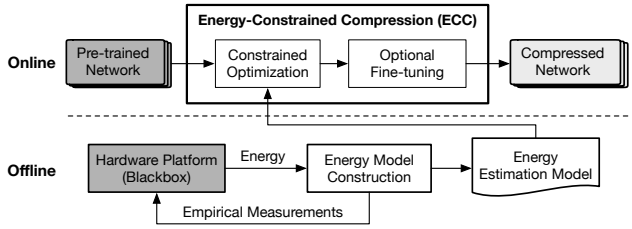


Figure 1: ECC framework overview.

NetAdapt [39] uses a heuristic-driven search algorithm whereas AMC [11][1] uses reinforcement learning. However, the search-based approaches are fundamentally limited by the search space, which could be huge for deep networks.

In this paper, we propose an alternative DNN compression algorithm that compresses all the DNN layers together rather than compressing it layer by layer. This strategy eliminates many of the heuristics and fine-tuning required in previous layer-wise approaches. As a result, it is able to find compression strategies that lead to better latency and energy reductions. Due to the lack of compression techniques that specifically target energy consumption, this paper focuses on energy consumption as a particular direct metric to demonstrate the effectiveness of our approach, but we expect our approach to be generally applicable to other direct metrics as well such latency and model size.

The key to our algorithm is to use a differentiable model that numerically estimates the energy consumption of a network. Leveraging this model, we formulate DNN compression as a constrained optimization problem (constrained by a given energy budget). We propose an efficient optimization algorithm that combines ideas from both classic constrained optimizations and gradient-based DNN training. Crucially, our approach is *platform-free* in that it treats the underlying hardware platform as a blackbox. Prior energy-constrained compressions all require deep understanding of the underlying hardware architecture [38, 37], and thus are necessarily tied to a particular hardware platform of choice. In contrast, our framework directly measures the energy consumption of the target hardware platform without requiring any hardware domain knowledges, and thus is portable across different platforms.

Leveraging the constrained optimization algorithm, we propose ECC, a DNN compression framework that automatically compresses a DNN to meet a given energy budget while maximizing its accuracy. ECC has two phases: an offline energy modeling phase and an online compression phase. Given a particular network to compress, the offline component profiles the network on a particular target platform and constructs an energy estimation model. The online component leverages the energy model to solve the constrained optimization problem followed by an optional fine-tuning phase before generating a compressed model. In summary, we make the following contributions:

- We propose a bilinear energy consumption model of DNN inference that models the DNN inference energy as a function of both its weights and sparsity settings. The energy model is constructed based on real hardware energy measurement and thus requires no domain-knowledge of the hardware architecture. Our model shows $< \pm 3\%$ error rate.
- We propose ECC, a DNN compression framework that maximizes the accuracy while meeting a given energy budget. ECC leverages the energy estimation model to formulate DNN compression as a constrained optimization problem. We present an efficient optimization algorithm that combines the classic ADMM framework with recent developments in gradient-based DNN algorithms. Although targeting energy in this paper, our framework can be extended to optimize other metrics such as latency and model size.
- We evaluate ECC using a wide range of computer vision tasks on both a mobile platform Jetson TX2 and a desktop platform with a GTX 1080 Ti GPU. We show that ECC achieves higher accuracy under the same energy budget compared to state-of-the-art resource-constrained compression methods including NetAdapt [39] and AMC [11].

Table 1: Comparison across different resource-constrained DNN compression techniques.

Properties/Methods	EAP [38]	AMC [11]	NetAdapt [39]	LcP [5]	**ECC**
Use direct metric?	✓	✓	✓	✓	✓
Target energy?	✓				✓
Optimization-based?					✓
Platform-free?		✓	✓	✓	✓

2. Related Work

Network Compression Network compression [18] is a key technique in reducing DNN model complexity. It leverages the observation that some network weights have less impact on the final results and thus could be removed (zeroed out). Compression techniques directly reduce a DNN model's size, which often also leads to latency and energy reductions. Early compression techniques focus exclusively on reducing the model size [9, 10, 36, 21, 44, 19] while latency and energy reductions are "byproducts." It is well-known now that model size, latency, and energy consumption are not directly correlated [38, 39, 41]. Therefore, compressing for one metric, such as model size, does not always translate to optimal compression results for other metrics such as latency and energy reduction, and vice versa.

Resource-Constrained Compression Researchers recently started investigating resource-constrained compres-

[1]The method proposed in AMC originally targets model size, FLOPs or latency, but can be extended to target energy consumption using our modeling method introduced in Section 3.2.

sion, which compresses DNNs under explicit resource constraints (e.g., energy, latency, the number of Multiply-accumulate operations) instead of using model size as a proxy, though the model size could also be used as a constraint itself. Table 1 compares four such state-of-the-art methods, EAP, AMC, and NetAdapt, and LcP. EAP [38] compresses a model to reduce its energy consumption while meeting a given accuracy threshold. AMC [11] compresses a model to meet a given resource (size, FLOPs or latency) constraint while maximizing the accuracy. NetAdapt [39] compresses a model to meet a given latency constraint while maximizing the accuracy. LcP [5] compresses a model to meet a given resource constraint (the number of parameters or the number of multiply-accumulate operations) while maximizing the accuracy.

Although the four techniques target different metrics and have different procedures, the core of them is to determine the optimal sparsity ratio of each layer in a way that the whole model meets the respective objectives. They differ in how they determine the layer-wise sparsity ratio. EAP, LcP, and NetAdapt all use heuristic-driven search algorithms. Specifically, EAP compresses layers in the order in which they contribute to the total energy, and prioritizes compressing the most energy-hungry layers; NetAdapt iteratively finds the per-layer sparsity by incrementally decreasing the latency budget; LcP assigns a score to each convolution filter, and prunes the filters based on the score until the resource constraint is satisfied. AMC uses reinforcement learning that determines the per-layer sparsity by "trial and error."

ECC is a different compression technique that, instead of compressing DNNs layer by layer, compresses all the network layers at the same time. It avoids heuristic searches and achieves better results.

Platform (In)dependence Previous energy-constrained compressions [38, 37] rely on energy modeling that is tied to a specific hardware architecture. ECC, in contrast, constructs the energy model directly from real hardware measurements without requiring platform knowledges, and is thus generally applicable to different hardware platforms. AMC and NetAdapt are also platform-free as they take empirical measurements from the hardware, but they target latency and model size. In particular, NetAdapt constructs a latency model using a look-up table whereas ECC's energy model is differentiable, which is key to formulating DNN compression as a constrained optimization problem that can be solved using gradient-based algorithms.

3. Method

This section introduces the proposed ECC framework for energy-constrained DNN compression. We first formulate DNN compression as a constrained optimization under the constraint of energy (Section 3.1). We then describe how the energy is estimated using a bilinear model (Section 3.2). Finally, we explain our novel gradient-based algorithm that solves the optimization problem (Section 3.3).

3.1. Problem Formulation

Our objective is to minimize the loss function ℓ under a predefined energy constraint:

$$\min_{\mathcal{W}} \quad \ell(\mathcal{W}) \tag{1a}$$

$$\mathcal{E}(\mathcal{W}) \leq E_{\text{budget}}, \tag{1b}$$

where $\mathcal{W} := \{\mathbf{w}^{(u)}\}_{u\in\mathcal{U}}$ ($\mathcal{U}$ is the set of all layers) stacks the weights tensors of all the layers, and $\mathcal{E}(\mathcal{W})$ denotes the real energy consumption of the network, which depends on the structure of DNN weights $\mathcal{W}$. Compression affects the DNN weights $\mathcal{W}$, and thus affects $\mathcal{E}(\mathcal{W})$. ℓ is the loss function specific to a given learning task. In deep learning, ℓ is a highly non-convex function.

There are two distinct classes of compression techniques. Unstructured, fine-grained compression prunes individual elements [9, 10]; whereas structured, coarse-grained compression prunes a regular structure of a DNN such as a filter channel. While our method is applicable to both methods, we particularly focus on the coarse-grained method that prunes channels in DNN layers [33, 44, 19, 22, 12, 23, 25, 7] because channel pruning is more effective on off-the-shelf DNN hardware platforms such as GPUs [41] whereas the fine-grained approaches require specialized hardware architectures to be effective [4, 8, 37].

With the channel pruning method, the optimization problem becomes finding the sparsity of each layer, i.e., the number of channels that are preserved in each layer, such that the total energy meets the given budget, that is,

$$\min_{\mathcal{W},\mathbf{s}} \quad \ell(\mathcal{W}) \tag{2a}$$

$$\text{s.t.} \quad \phi(\mathbf{w}^{(u)}) \leq s^{(u)}, \quad u \in \mathcal{U} \tag{2b}$$

$$\mathcal{E}(\mathbf{s}) \leq E_{\text{budget}}, \tag{2c}$$

where $s^{(u)}$ corresponds to the sparsity bound of layer $u \in \mathcal{U}$, and $\mathbf{s} := \{s^{(u)}\}_{u\in\mathcal{U}}$ stacks the (inverse) sparsities of all the layers. The energy consumption of a DNN $\mathcal{E}$ can now be expressed as a function of $\mathbf{s}$[2]. $\mathbf{w}^{(u)}$ denotes the weight tensor of layer u. The shape of $\mathbf{w}^{(u)}$ is $d^{(u)} \times c^{(u)} \times r_h^{(u)} \times r_w^{(u)}$ for the convolution layer u with $d^{(u)}$ output channels, $c^{(u)}$ input channels, and spatial kernel size $r_h^{(u)} \times r_w^{(u)}$. Without loss of generality, we treat the fully connected layer as a special convolution layer where $r_h^{(u)} = r_w^{(u)} = 1$. $\phi(\mathbf{w}^{(u)})$ calculates the layer-wise sparsity as $\sum_i \mathbf{I}(\|\mathbf{w}^{(u)}_{\cdot,i,\cdot,\cdot}\| \neq 0)$ where $\mathbf{I}(\cdot)$ is the indicator function which returns 1 if the inside condition is satisfied and 0 otherwise.

[2]For simplicity, we reuse the same notion $\mathcal{E}$ in both Equation (1b) and Equation (2c).

3.2. Bilinear Energy Consumption Model

The key step to solving Equation (2) is to identify the energy model $\mathcal{E}(\mathbf{s})$, i.e., to model the DNN energy as a function of the sparsity of each layer. This step is particularly important in that it provides an analytical form to characterize the energy consumption. Existing DNN energy models are specific to a particular hardware platform [38, 37], which requires deep understandings of the hardware and is not portable across different hardware architectures. In contrast, we construct an energy model directly from hardware measurements while treating the hardware platform as a blackbox. This is similar in spirit to NetAdapt [39], which constructs a latency model through hardware measurements. However, their model is a look-up table that is huge and not differentiable. Our goal, however, is to construct a differentiable model so that the optimization problem can be solved using conventional gradient-based algorithms.

Our key idea is that the energy model can be obtained via a data driven approach. Let $\hat{\mathcal{E}}$ be a differentiable function to approximate $\mathcal{E}$:

$$\hat{\mathcal{E}} = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{\mathbf{s}}[(f(\mathbf{s}) - \mathcal{E}(\mathbf{s}))^2], \tag{3}$$

where $\mathcal{F}$ is the space of all the potential energy models, and $\mathbb{E}_{\mathbf{s}}$ is the expectation with respect to $\mathbf{s} := [s_1, \cdots, s_{|\mathcal{U}|}, s_{|\mathcal{U}+1|}]$.

To find a differentiable energy model $\hat{\mathcal{E}}$, our intuition is that the energy consumption of a DNN layer is affected by the number of channels in its input and output feature maps, which in turn are equivalent to the sparsity of the current and the next layer, respectively. Therefore, the energy consumption of layer j can be captured by a function that models the interaction between s_j and s_{j+1}, where s_j denotes the (inverse) sparsity of layer j ($j \in [1, |\mathcal{U}|]$). Based on this intuition, we approximate the total network energy consumption using the following bilinear model: $\mathcal{F} := \{f(\mathbf{s}) = a_0 + \sum_{j=1}^{|\mathcal{U}|} a_j s_j s_{j+1} : \quad a_0, a_1, ..., a_{|\mathcal{U}|} \in \mathbb{R}_+\}$, where $s_{|\mathcal{U}|+1}$ is defined as the network output dimensionality, e.g., the number of classes in a classification task. Coefficients $a_0, a_1, ...a_{|\mathcal{U}|}$ are the variables defining this space.

The rationale behind using the bilinear structure is that the total number of arithmetic operations (multiplications and additions) during a DNN inference with layers defined by $\mathbf{s}$ would roughly be in a bilinear form. Although other more complex models are possible, the bilinear model is simple and tight, which is easy to train and also avoids overfitting effectively. We will show in Section 4 that our model achieves high prediction accuracy.

Sharp readers might ask what if the energy function fundamentally can not be modeled in the bilinear form (e.g., for some particular DNN architectures). In such a case, one could use a neural network to approximate the energy function since a three-layer neural network can theoretically approximate any function [13]. Note that our constrained optimization formulation requires only that the energy model is differentiable and thus is still applicable.

To obtain $\hat{\mathcal{E}}$, we sample $\mathbf{s}$ from a uniform distribution and measure the real energy consumption on the target hardware platform to get $\mathcal{E}(\mathbf{s})$. Please refer to Section 4.1 for a complete experimental setup. We then use the stochastic gradient descent (SGD) method to solve Equation (3) and obtain $\hat{\mathcal{E}}$. Note that this process is performed once for a given network and hardware combination. It is performed offline as shown in Figure 1 and thus has no runtime overhead.

3.3. Optimization Algorithm

The optimization problem posed by Equation (2) is a constrained optimization whereas conventional DNN training is unconstrained. Therefore, conventional gradient-based algorithms such as SGD and existing deep learning solvers do not directly apply here. Another idea is to extend the gradient-based algorithm to the projected version [32] – a (stochastic) gradient descent step followed by a projection step to the constraint [37]. When the projection is tractable [40] or the constraint is simple (e.g., linear) [29], Lagrangian methods can be used to solve constraints on the parameters or the outputs of DNNs. However, due to the complexity of our constraint, the projection step is extremely difficult.

In this paper, we apply the framework of ADMM [3], which is known to handle constrained optimizations effectively. ADMM is originally designed to solve optimization problems with linear equality constraints and convex objectives, both of which do not hold in our problem. Therefore, we propose a hybrid solver that is based on the ADMM framework while taking advantage of the recent advancements in gradient-based deep learning algorithms and solvers [15].

Algorithm Overview We first convert the original problem (2) to an *equivalent* minimax problem [3]:

$$\min_{\mathcal{W}, \mathbf{s}} \max_{z \geq 0, \mathbf{y} \geq \mathbf{0}} \mathcal{L}(\mathcal{W}, \mathbf{s}, \mathbf{y}, z) \tag{4}$$

where $\mathbf{y}$ is the dual variable introduced for the constraint (2b), and z is the dual variable for the constraint (2c). $\mathcal{L}$ is defined as the augmented Lagrangian $\mathcal{L}(\mathcal{W}, \mathbf{s}, \mathbf{y}, z) := \ell(\mathcal{W}) + \mathcal{L}_1(\mathcal{W}, \mathbf{s}, \mathbf{y}) + \mathcal{L}_2(\mathbf{s}, z)$, where $\mathcal{L}_1(\mathcal{W}, \mathbf{s}, \mathbf{y}) := \frac{\rho_1}{2} \sum_u [\phi(\mathbf{w}^{(u)}) - s^{(u)}]_+^2 + \sum_u y^{(u)}(\phi(\mathbf{w}^{(u)}) - s^{(u)})$, $\mathcal{L}_2(\mathbf{s}, z) := \frac{\rho_2}{2}[\hat{\mathcal{E}}(\mathbf{s}) - E_{\text{budget}}]_+^2 + z(\hat{\mathcal{E}}(\mathbf{s}) - E_{\text{budget}})$. $[\cdot]_+$ denotes the nonnegative clamp $\max(0, \cdot)$, and ρ_1, ρ_2 are two predefined nonnegative hyperparameters. Note that the choices of ρ_1 and ρ_2 affect only the efficiency of convergence, but not the convergent point (for convex optimization).

To compress a DNN under a given energy constraint, we start with a dense model denoted by $\mathcal{W}^{\text{dense}}$ (which could

Algorithm 1: Energy-Constrained DNN Compression.

Input: Energy budget E_{budget}, learning rates α, β, penalty parameters ρ_1, ρ_2.
Result: DNN weights $\mathcal{W}^*$.
Initialize $\mathcal{W} = \mathcal{W}^{\text{dense}}$, $\mathbf{s} = \{\phi(\mathbf{w}^{(u)})\}_{u\in\mathcal{U}}$, $\mathbf{y} = \mathbf{0}$, $z = 0$;
while $\hat{\mathcal{E}}(\mathbf{s}) > E_{budget}$ *or* $\exists u, \phi(\mathbf{w}^{(u)}) > s^{(u)}$ **do**
  Update $\mathcal{W}$ by proximal Adam update: $\mathcal{W}$ = (7);
  Update $\mathbf{s}$ by gradient descent: (10);
  Update $\mathbf{y}, z$ by projected gradient ascent: (11) and (12);
end
$\mathcal{W}^* = \mathcal{W}$.

be obtained by optimizing the unconstrained objective), and solve the problem in Equation (4) to obtain a compressed network. Inspired by the basic framework of ADMM, we solve Equation (4) iteratively, where each iteration updates the primal variables $\mathcal{W}, \mathbf{s}$ and dual variables $\mathbf{y}, z$. Algorithm 1 shows the pseudo-code of our algorithm.

Specifically, each iteration first updates the DNN weights $\mathcal{W}$ to minimize ℓ while preventing $\mathcal{W}$ to have layer-wise (inverse) sparsities larger than $\mathbf{s}$. $\mathbf{s}$ is then updated to reduce the energy estimation $\hat{\mathcal{E}}(\mathbf{s})$. Dual variables $\mathbf{y}$ and z can be seen as penalties that are dynamically changed based on how much $\mathcal{W}$ and $\mathbf{s}$ violate the constraints (2b) and (2c). We now elaborate on the three key updating steps.

3.3.1 Updating Primal Variable $\mathcal{W}$

We first fix the sparsity bounds $\mathbf{s}$ and the two dual variables $\mathbf{y}, z$ to update the primal variable weight tensor $\mathcal{W}$ by:

$$\arg\min_{\mathcal{W}} \ell(\mathcal{W}) + \mathcal{L}_1(\mathcal{W}, \mathbf{s}, \mathbf{y}). \tag{5}$$

The challenge here is that updating $\mathcal{W}$ is time-consuming, mainly due to the complexity of calculating $\arg\min_{\mathcal{W}} \ell(\mathcal{W})$, where ℓ is the non-convex loss function of the network. Stochastic ADMM [43] could simplify the complexity of the primal update by using stochastic gradient, but they consider only the convex problems with shallow models. Instead, we propose to improve the primal update's efficiency using a proxy of the loss $\ell(\mathcal{W})$ at $\mathcal{W}^t$:

$$\ell(\mathcal{W}^t) + \langle \hat{\nabla}\ell(\mathcal{W}^t), \mathcal{W} - \mathcal{W}^t \rangle + \frac{1}{2\alpha}\|\mathcal{W} - \mathcal{W}^t\|_B^2, \tag{6}$$

where B is a positive diagonal matrix, which is usually used in many adaptive optimizers such as ADADELTA [42] and Adam [15]; $\|\mathcal{W}\|_B$ is defined by the norm $\sqrt{\text{vec}(\mathcal{W})^\top B \text{vec}(\mathcal{W})}$ where $\text{vec}(\cdot)$ is the vectorization operation. Without loss of generality, we use the diagonal matrix B as in Adam. $\hat{\nabla}\ell(\mathcal{W}^t)$ is the stochastic gradient of ℓ at $\mathcal{W}^t$, and α is the learning rate for updating $\mathcal{W}$. Therefore, Equation (5) is simplified to a proximal [27] Adam update:

Algorithm 2: Proximal Operator $\mathbf{prox}_{\alpha\mathcal{L}_1}(\cdot)$.

Input: Input tensors $\bar{\mathcal{W}} = \{\bar{\mathbf{w}}^{(u)}\}_{u\in\mathcal{U}}$.
Result: Proximal operation result $\mathcal{W} = \{\mathbf{w}^{(u)}\}_{u\in\mathcal{U}}$.
Let $\mathbf{a}_i^{(u)} = \|\bar{\mathbf{w}}_{\cdot,i,\cdot,\cdot}^{(u)}\|^2_{B^{(u)}}, \forall u \in \mathcal{U}$;
Sort $\mathbf{a}^{(u)}$ in descending order, let $\mathbf{r}^{(u)}$ be the corresponding ranks of elements in $\mathbf{a}^{(u)}$;
foreach *Layer* $u \in \mathcal{U}$ **do**
  for $i \leftarrow 1$ **to** $c^{(u)}$ **do**
    if $\mathbf{a}_i^{(u)} > \rho_1\alpha([\mathbf{r}_i^{(u)} - s^{(u)}]_+^2 - [\mathbf{r}_i^{(u)} - 1 - s^{(u)}]_+^2) + 2\alpha y^{(u)}$ **then**
      $\mathbf{w}_{\cdot,i,\cdot,\cdot}^{(u)} = \bar{\mathbf{w}}_{\cdot,i,\cdot,\cdot}^{(u)}$;
    else
      $\mathbf{w}_{\cdot,i,\cdot,\cdot}^{(u)} = \mathbf{0}$;
    end
  end
end

$$\arg\min_{\mathcal{W}} \frac{1}{2\alpha}\|\mathcal{W} - (\mathcal{W}^t - \alpha B^{-1}\hat{\nabla}\ell(\mathcal{W}^t))\|_B^2 + \mathcal{L}_1(\mathcal{W}, \mathbf{s}, \mathbf{y}). \tag{7}$$

If we define $\mathbf{prox}_{\alpha\mathcal{L}_1}(\cdot)$ as the proximal operator of function $\alpha\mathcal{L}_1(\cdot, \mathbf{s}, y)$:

$$\mathbf{prox}_{\alpha\mathcal{L}_1}(\bar{\mathcal{W}}) := \arg\min_{\mathcal{W}} \frac{1}{2}\|\mathcal{W} - \bar{\mathcal{W}}\|_B^2 + \alpha\mathcal{L}_1(\mathcal{W}, \mathbf{s}, \mathbf{y}), \tag{8}$$

the optimal solution of problem (7) admits a closed form: $\mathbf{prox}_{\alpha\mathcal{L}_1}(\mathcal{W}^t - \alpha B^{-1}\hat{\nabla}\ell(\mathcal{W}^t))$. This update essentially performs pruning and fine-tuning simultaneously. The detailed algorithm for proximal operator $\mathbf{prox}_{\alpha\mathcal{L}_1}(\cdot)$ is shown in Algorithm 2.

3.3.2 Updating Primal Variable $\mathbf{s}$

In this step, we update the primal variable $\mathbf{s}$ by:

$$\arg\min_{\mathbf{s}} \mathcal{L}_1(\mathcal{W}, \mathbf{s}, \mathbf{y}) + \mathcal{L}_2(\mathbf{s}, z). \tag{9}$$

Similar as above, instead of searching for exactly solving this subproblem, we only apply a gradient descent step:

$$\mathbf{s}^{t+1} = \mathbf{s}^t - \beta(\nabla_{\mathbf{s}}\mathcal{L}_1(\mathcal{W}, \mathbf{s}^t, \mathbf{y}) + \nabla_{\mathbf{s}}\mathcal{L}_2(\mathbf{s}^t, z)), \tag{10}$$

where β is the learning rate for updating $\mathbf{s}$. To avoid removing a certain layer entirely, a lower bound is set for $s^{(u)}$. In our method, it is set as 1 if not explicitly mentioned.

3.3.3 Updating Dual Variables

The dual updates simply fix $\mathcal{W}, \mathbf{s}$ and update $\mathbf{y}, z$ by projected gradient ascent with learning rates ρ_1, ρ_2:

$$y^{(u)^{t+1}} = [y^{(u)^t} + \rho_1(\phi(\mathbf{w}^{(u)}) - s^{(u)})]_+, \tag{11}$$

$$z^{t+1} = [z^t + \rho_2(\hat{\mathcal{E}}(\mathbf{s}) - E_{\text{budget}})]_+. \tag{12}$$

To stabilize the training process, we perform some additional steps when updating the dual variables. The dual variable $\mathbf{y}$ controls the sparsity of each DNN layer, and larger $y^{(u)}$ prunes more channels in layer u. It is not necessary to penalize $\phi(\mathbf{w}^{(u)})$ when $\phi(\mathbf{w}^{(u)}) \leq s^{(u)}$, so $y^{(u)}$ is trimmed to meet $\phi(\mathbf{w}^{(u)}) \geq \lfloor s^{(u)} \rfloor$. The dual variable z is used to penalize the violation of energy cost $\hat{\mathcal{E}}(\mathbf{s})$, and larger z makes $\mathbf{s}$ prone to decrease $\hat{\mathcal{E}}(\mathbf{s})$. In the training process, we want $\hat{\mathcal{E}}(\mathbf{s})$ to be monotonically decreased. So we project the variable z to be large enough to meet $\max(\nabla_{\mathbf{s}}\mathcal{L}_1(\mathcal{W}, \mathbf{s}, \mathbf{y}) + \nabla_{\mathbf{s}}\mathcal{L}_2(\mathbf{s}, z)) \geq \epsilon$, where ϵ is a small positive quantity and we simply set 10^{-3}. The gradient of $\mathbf{s}$ is also clamped to be nonnegative.

4. Evaluation Results

We evaluate ECC on real vision tasks deployed on two different hardware platforms. We first introduce our experimental setup (Section 4.1), followed up by the accuracy of the energy prediction model (Section 4.2). Finally, we compare ECC with state-of-the-art methods (Section 4.3).

4.1. Experimental Setup

Vision tasks & Datasets We evaluate ECC on two important vision tasks: image classification and semantic segmentation. For image classification, we use the complete ImageNet dataset [31]. For semantic segmentation, we use the recently released large-scale segmentation benchmark Cityscapes [6] which contains pixel-level high resolution video sequences from 50 different cities.

DNN architectures For image classification, we use two representative DNN architectures AlexNet [17] and MobileNet [14]. The dense versions of the two models are from the official PyTorch model zoo and the official TensorFlow repository [1], respectively. For semantic segmentation, we choose the recently proposed ERFNet [30], which relies on residual connections and factorization structures, and is shown to be efficient on real-time segmentation. We use the pre-trained ERFNet released by the authors. The collection of the three networks allows us to evaluate ECC against different DNN layer characteristics including fully connected, convolutional, and transposed convolutional layers.

Hardware Platforms We experiment on two different GPU platforms. The first one is a GTX 1080 Ti GPU. We use the `nvidia-smi` utility [26] to obtain real-hardware energy measurements. The second one is the Nvidia Jetson TX2 embedded device, which is widely used in mobile vision systems and contains a mobile Pascal GPU. We retrieve the TX2's GPU power using the Texas Instruments INA 3221 voltage monitor IC through the I2C interface. The DNN architectures as well as our ECC framework are implemented using PyTorch [28].

Baseline We compare ECC with two most recent (as of submission) resource-constrained compression methods NetAdapt [39] and AMC [11]. We faithfully implement them according to what is disclosed in the papers. NetAdapt is originally designed to compress DNNs under latency constraints and AMC is designed to compress DNNs under constraint of model size, FLOPs or latency. We adapt them to obtain the energy-constrained versions for comparison. Both methods use channel pruning for compression, same as ECC.

Earlier channel pruning methods such as Network-Slimming [22], Bayesian Compression [23], and several others [7, 12, 25] are agnostic to resource constraints (e.g., energy) because they focus on sparsity itself. They require a sparsity bound (or regularization weight) for each layer to be *manually* set before compression. The compressed model is generated only based on the given sparsity bounds, regardless of the energy budget. We thus do not compare with them here.

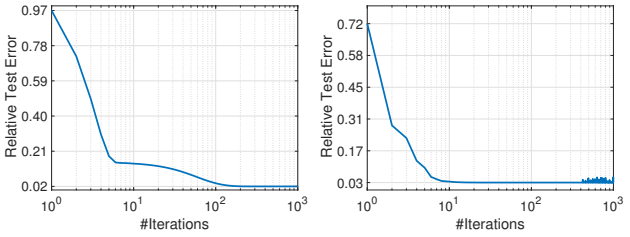
Hyper-parameters & implementation details The batch size is set to 128 for AlexNet and MobileNet and to 4 for ERFNet based on the GPU memory capacity. We use the Adam optimizer with its default Beta $(0.9, 0.999)$; its learning rate is set to 10^{-5}, and the weight decay is set as 10^{-4}.[3] All the compression methods are trained with the same number of data batches / iterations, which are about 300,000 for ImageNet and 30,000 for Cityscapes. These iterations correspond to the "short-term fine-tuning" [39] in NetAdapt and the "4-iteration pruning & fine-tuning" [11] in AMC. The reinforcement learning episodes in AMC is set to 400 as described in [11]. For the loss function ℓ, we add a knowledge distillation (KD) term [2] combined with the original loss (e.g., cross-entropy loss), since KD has been shown as effective in DNN compression tasks [24, 35, 37]. In our method, the learning rate β for the sparsity bounds $\mathbf{s}$ is set to reach the energy budgets with given iteration number, and the dual learning rates ρ_1, ρ_2 are set as 10 on ImageNet and 1 on Cityscapes.

After getting the compressed models with given energy budgets, we fine-tune each model for 100,000 iterations with aforementioned setup. For MobileNet, we additionally perform 300,000 iterations of fine-tuning with decayed learning rate (cosine decay from 3×10^{-5}) to minimize the cross-entropy loss. The fine-tuning procedures train the DNN with fixed non-zero positions in their weight tensors.

4.2. Energy Prediction Model

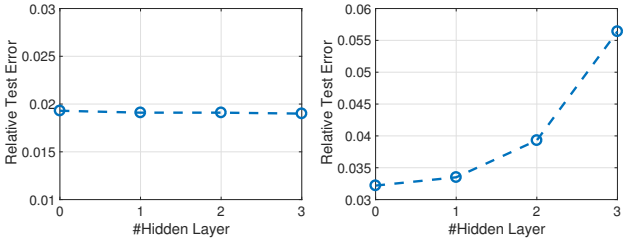
To train the energy prediction model, we obtain the real energy measurements under different layer-wise sparsity bounds $\mathbf{s}$. We first randomly sample $\mathbf{s}$ from the uniform distribution: $s^{(u)} \sim \text{unif}\{1, c^{(u)}\}$. For each sample, we

[3] They are chosen in favor of best pre-trained model accuracy rather than biasing toward any compression methods.



(a) MobileNet on GTX 1080 Ti. (b) MobileNet on TX2.

Figure 2: Relative test error of energy prediction using the proposed bilinear model.



(a) MobileNet on GTX 1080 Ti. (b) MobileNet on TX2.

Figure 3: Relative test error of energy prediction using an MLP model with different hidden layers.

then construct a corresponding DNN, and measure its energy $\mathcal{E}(\mathbf{s})$. We measure the energy by taking the average of multiple trials to minimize offset run-to-run variation. For each DNN architecture, we collect 10,000 $(\mathbf{s}, \mathcal{E}(\mathbf{s}))$ pairs to train the energy model $\hat{\mathcal{E}}$. We randomly choose 8,000 pairs as the training data and leave the rest as test data. To optimize problem (3), we use Adam optimizer with its default hyper-parameters, and the weight decay is set as 1.0. We set batch size as 8,000 (full training data) and train the energy model with 10,000 iterations. It should be noted that the bilinear energy model is linear in terms of the learnable parameters, which means a linear regression solver could be used. In this section, we also compare the bilinear model with a nonlinear model, for which Adam is more suitable.

In Figure 2, we show the relative test error defined as $\mathbb{E}_{\mathbf{s}\sim\text{testset}}[|\hat{\mathcal{E}}(\mathbf{s}) - \mathcal{E}(\mathbf{s})|/\mathcal{E}(\mathbf{s})]$ at each training iteration for MobileNet on both hardware platforms. We find that the relative test errors quickly converge to around 0.03. This indicates that our energy model is not only accurate, but is also efficient to construct. The same conclusions hold for other networks as well, but are omitted due to space limit.

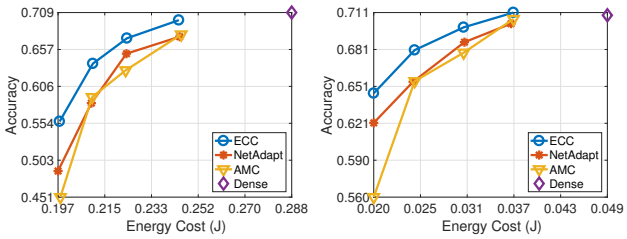
To assess whether the bilinear model is sufficient, we also experiment with a more complex prediction model by appending a multilayer perceptron (MLP) after the bilinear model. The widths of all the MLP hidden layers are set to 128, and we use the SELU [16] activation. We vary the number of hidden layers from 1 through 3. The prediction errors of this augmented energy model on MobileNet are shown in Figure 3, where the original bilinear model has zero hidden layer. We find that adding an MLP does not noticeable improve the prediction accuracy on GTX 1080 Ti, but significantly *reduce* the prediction accuracy on TX2. We thus use the plain bilinear model for the rest of the evaluation.

4.3. DNN Compression Results

We now show the evaluation results on two popular vision tasks: image classification and semantic segmentation.

4.3.1 ImageNet Classification

MobileNet In Figure 4, we show the validation accuracies of compressed MobileNet under different energy budgets, and the energy cost is shown by joule (J). We set four different energy budgets in descending order. The dense MobileNet model has a top-1 accuracy of 0.709. The energy cost of the dense model is 0.2877 J on GTX 1080 Ti and 0.0487 J on Jetson TX2.



(a) GTX 1080 Ti. (b) Jetson TX2.

Figure 4: Top-1 accuracy of image classification on MobileNet@ImageNet **after** fine-tuning.

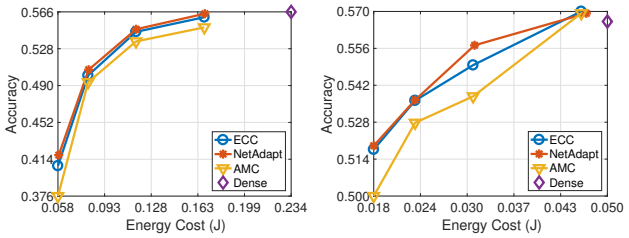
Figure 4 shows the top-1 accuracy v.s. energy comparisons (after fine-tuning) across the three methods. The results before fine-tuning is included in the supplementary material. ECC achieves higher accuracy than NetAdapt and AMC. For instance, on Jetson TX2 under the same 0.0247 J energy budget, ECC achieves 2.6% higher accuracy compared to NetAdapt. Compared to the dense model, ECC achieves 37% energy savings with $< 1\%$ accuracy loss on Jetson TX2. AMC has similar performance with NetAdapt when the energy budget is not too small.

The accuracy improvements of ECC over NetAdapt and AMC are more significant under lower energy budgets. This suggests that under tight energy budget, searching for the optimal per-layer sparsity combinations becomes difficult, whereas ECC, via its optimization process, is able to identify better layer sparsities than search-based approaches.

AlexNet We obtain similar conclusions on AlexNet. The dense model has a 0.566 top-1 accuracy. The energy cost of the dense model is 0.2339 J on GTX 1080 Ti and 0.0498 J on Jetson TX2. Figure 5 compares the three methods (after fine-tuning) on two platforms respectively. The

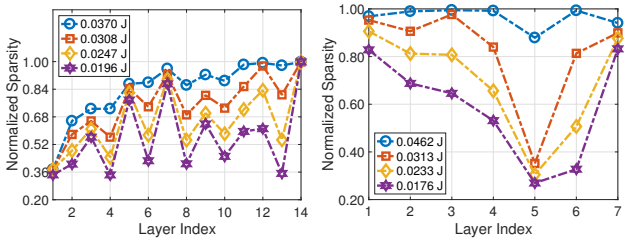
Table 2: Segmentation accuracy (averaged IoU) comparison on the Cityscapes dataset.

Methods / Energy Budget	GTX 1080 Ti					Jetson TX2				
	1.3843 J	1.4451 J	1.5238 J	1.6519 J	1.9708 J	0.8542 J	0.9051 J	0.9756 J	1.0724 J	1.3213 J
Dense	-	-	-	-	0.722	-	-	-	-	0.722
ECC	**0.6713**	**0.6830**	**0.7007**	**0.7163**	-	**0.6733**	**0.6914**	**0.7017**	**0.7183**	-
NetAdapt [39]	0.6361	0.6523	0.6865	0.7114	-	0.6567	0.6708	0.6916	0.7084	-
AMC [11]	0.6340	0.6374	0.6749	0.6992	-	0.6344	0.6491	0.6685	0.6976	-



(a) GTX 1080 Ti. (b) Jetson TX2.

Figure 5: Top-1 accuracy image classification on AlexNet@ImageNet **after** fine-tuning.



(a) MobileNet (inverse) sparsity. (b) AlexNet (inverse) sparsity.

Figure 6: Layer (inverse) sparsity after compressing on Jetson TX2.

results before fine-tuning are included in the supplementary material. Before fine-tuning, ECC outperforms NetAdapt, which however achieves similar or slightly better accuracy than ECC after fine-tuning. ECC consistently outperforms AMC before and after fine-tuning. Compared to dense models, ECC achieves 28% and 37% energy savings with $< 0.6\%$ and $< 1.7\%$ accuracy loss on GTX 1080 Ti and Jetson TX2, respectively.

Comparing the results on AlexNet (7 layers) and MobileNet (14 layers), we find that the advantage of ECC is more pronounced on deeper networks. This is because as the network becomes deeper the layer sparsity search space grows exponentially, which makes the search-based approaches such as NetAdapt and AMC less effective.

Sparsity Analysis Figure 6a and Figure 6b show the normalized (inverse) sparsity (i.e. #(nonzero channels)) of each layer in MobileNet and AlexNet respectively. Different colors represents different energy budgets used in Figure 4 and Figure 5. We find that the 5^{th} layer in AlexNet is pruned heaviest. In AlexNet, that is the first fully connected layer which has the most number of weights; pruning it saves lots of energy. We also observe many "spikes" in MobileNet. Our intuition is that every two consecutive layers can be seen as a low rank factorization of a larger layer (ignoring the nonlinear activation between layers). The spikes may suggest that low rank structure could be efficient in saving energy.

4.3.2 Cityscapes Segmentation

Now we apply ECC to ERFNet [30] for semantic segmentation. We use the well-established averaged Intersection-over-Union (IoU) metric, which is defined as TP/(FP+TP+FN) where TP, FP, and FN denote true positives, false positives, and false negatives, respectively. The training protocol is the same as the ImageNet experiments, except that the number of training iterations is 30,000 and the results are fine-tuned with 10,000 extra iterations. The dense model has an IoU of 0.722 and energy cost of 1.9708 J on GTX 1080 Ti and 1.3213 J on Jetson TX2.

Table 2 compares the IoUs of the three compression techniques under different energy budgets. ECC consistently achieves the highest accuracy under the same energy budget. Similar to MobileNet, ERFNet is also a deep network with 51 layers, which leads to large layer sparsity search space that makes search-based approaches ineffective. Compared to the dense model, ECC reduces energy by 16% and 19% with $< 0.6\%$ IoU loss on GTX 1080 Ti and TX2, respectively.

5. Conclusion

Future computer vision applications will be increasingly operating on energy-constrained platforms such as mobile robots, AR headsets, and ubiquitous sensor nodes. To accelerate the penetration of mobile computer vision, this paper proposes ECC, a framework that compresses DNN to meet a given energy budget while maximizing accuracy. We show that DNN compression can be formulated as a constrained optimization problem, which can be efficiently solved using gradient-based algorithms without many of the heuristics used in conventional DNN compressions. Although targeting energy as a case study in the paper, our framework is generally applicable to other resource constraints such as latency and model size. We hope that our work is a first step, not the final word, toward heuristics-free, optimization-based DNN improvements.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] J. Ba and R. Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[3] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[4] Y.-H. Chen, J. Emer, and V. Sze. Eyeriss: A Spatial Architecture for Energy-efficient Dataflow for Convolutional Neural Networks. In *Proc. of ISCA*, 2016.

[5] T.-W. Chin, C. Zhang, and D. Marculescu. Layer-compensated pruning for resource-constrained convolutional neural networks. *arXiv preprint arXiv:1810.00518*, 2018.

[6] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[7] B. Dai, C. Zhu, and D. Wipf. Compressing neural networks using the variational information bottleneck. *arXiv preprint arXiv:1802.10399*, 2018.

[8] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W. Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *Proc. of ISCA*, 2016.

[9] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[10] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[11] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision*, pages 784–800, 2018.

[12] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, 2017.

[13] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[15] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 971–980, 2017.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[18] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[19] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[20] R. LiKamWa, Z. Wang, A. Carroll, F. X. Lin, and L. Zhong. Draining our glass: An energy and heat characterization of google glass. In *Proceedings of 5th Asia-Pacific Workshop on Systems*, page 10. ACM, 2014.

[21] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015.

[22] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2755–2763. IEEE, 2017.

[23] C. Louizos, K. Ullrich, and M. Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3288–3298, 2017.

[24] A. Mishra and D. Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017.

[25] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*, pages 6775–6784, 2017.

[26] Nvidia. nvidia-smi. `https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf`.

[27] N. Parikh, S. Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[29] D. Pathak, P. Krahenbuhl, and T. Darrell. Constrained convolutional neural networks for weakly supervised segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1796–1804, 2015.

[30] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272, 2018.

[31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[32] O. Shamir and T. Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *International Conference on Machine Learning*, pages 71–79, 2013.

[33] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

[34] A. Suleiman, Y.-H. Chen, J. Emer, and V. Sze. Towards Closing the Energy Gap Between HOG and CNN Features for Embedded Vision. In *Proc. of ISCAS*, 2017.

[35] M. Tschannen, A. Khanna, and A. Anandkumar. Strassennets: Deep learning with a multiplication budget. *arXiv preprint arXiv:1712.03942*, 2017.

[36] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[37] H. Yang, Y. Zhu, and J. Liu. Energy-constrained compression for deep neural networks via weighted sparse projection and layer input masking. In *International Conference on Learning Representations*, 2019.

[38] T.-J. Yang, Y.-H. Chen, and V. Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5687–5695, 2017.

[39] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision*, pages 285–300, 2018.

[40] S. Ye, T. Zhang, K. Zhang, J. Li, J. Xie, Y. Liang, S. Liu, X. Lin, and Y. Wang. A unified framework of dnn weight pruning and weight clustering/quantization using admm. *arXiv preprint arXiv:1811.01907*, 2018.

[41] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. In *ACM SIGARCH Computer Architecture News*, volume 45, pages 548–560. ACM, 2017.

[42] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[43] S. Zheng and J. T. Kwok. Fast-and-light stochastic admm. In *IJCAI*, pages 2407–2613, 2016.

[44] H. Zhou, J. M. Alvarez, and F. Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, pages 662–677. Springer, 2016.

[45] Y. Zhu, A. Samajdar, M. Mattina, and P. Whatmough. Euphrates: Algorithm-soc co-design for low-power mobile continuous vision. *Proc. of ISCA*, 2018.