DeepIoT: Compressing Deep Neural Network Structures for Sensing Systems with a Compressor-Critic Framework

Shuochao Yao
University of Illinois Urbana
Champaign

Yiran Zhao
University of Illinois Urbana
Champaign

Aston Zhang
University of Illinois Urbana
Champaign

Lu Su
State University of New York at
Buffalo

Tarek Abdelzaher
University of Illinois Urbana
Champaign

ABSTRACT

Recent advances in deep learning motivate the use of deep neutral networks in sensing applications, but their excessive resource needs on constrained embedded devices remain an important impediment. A recently explored solution space lies in compressing (approximating or simplifying) deep neural networks in some manner before use on the device. We propose a new compression solution, called DeepIoT, that makes two key contributions in that space. First, unlike current solutions geared for compressing specific types of neural networks, DeepIoT presents a unified approach that compresses all commonly used deep learning structures for sensing applications, including fully-connected, convolutional, and recurrent neural networks, as well as their combinations. Second, unlike solutions that either sparsify weight matrices or assume linear structure within weight matrices, DeepIoT compresses neural network structures into smaller dense matrices by finding the minimum number of non-redundant hidden elements, such as filters and dimensions required by each layer, while keeping the performance of sensing applications the same. Importantly, it does so using an approach that obtains a global view of parameter redundancies, which is shown to produce superior compression. The compressed model generated by DeepIoT can directly use existing deep learning libraries that run on embedded and mobile systems without further modifications. We conduct experiments with five different sensing-related tasks on Intel Edison devices. DeepIoT outperforms all compared baseline algorithms with respect to execution time and energy consumption by a significant margin. It reduces the size of deep neural networks by 90% to 98.9%. It is thus able to shorten execution time by 71.4% to 94.5%, and decrease energy consumption by 72.2% to 95.7%. These improvements are achieved without loss of accuracy. The results underscore the potential of DeepIoT for advancing the exploitation of deep neural networks on resource-constrained embedded devices.

CCS CONCEPTS

•**Human-centered computing** →**Ubiquitous and mobile computing;** •**Computing methodologies** →**Machine learning;** •**Computer systems organization** →**Embedded and cyber-physical systems;**

KEYWORDS

Deep Learning, Model Compression, Structure Compression, Mobile Computing, Internet of Things

1 INTRODUCTION

This paper is motivated by the prospect of enabling a "smarter" and more user-friendly category of every-day physical objects capable of performing complex sensing and recognition tasks, such as those needed for understanding human context and enabling more natural interactions with users in emerging Internet of Things (IoT) applications.

Present-day sensing applications cover a broad range of areas including human interactions [27, 59], context sensing [6, 34, 39, 46, 54], crowd sensing [55, 58], object detection and tracking [7, 29, 42, 53]. The recent commercial interest in IoT technologies promises a proliferation of smart objects in human spaces at a much broader scale. Such objects will ideally have independent means of interacting with their surroundings to perform complex detection and recognition tasks, such as recognizing users, interpreting voice commands, and understanding human context. The paper explores the feasibility of implementing such functions using deep neural networks on computationally-constrained devices, such as Intel's suggested IoT platform: the Edison board[1].

The use of deep neural networks in sensing applications has recently gained popularity. Specific neural network models have been designed to fuse multiple sensory modalities and extract temporal relationships for sensing applications. These models have shown significant improvements on audio sensing [31], tracking and localization [8, 38, 52, 56], human activity recognition [37, 56], and user identification [31, 56].

[1]https://software.intel.com/en-us/iot/hardware/edison

Training the neural network can occur on a computationally capable node and, as such, is not of concern in this paper. The key impediment to deploying deep-learning-based sensing applications lies in the high memory consumption, execution time, and energy demand associated with storing and using the *trained network* on the *target device*. This leads to increased interest in compressing neural networks to enable exploitation of deep learning on low-end embedded devices.

We propose DeepIoT that compresses commonly used deep neural network structures for sensing applications through deciding the minimum number of elements in each layer. Previous illuminating studies on neural network compression sparsify large dense parameter matrices into large sparse matrices [4, 22, 24]. In contrast, DeepIoT minimizes the number of elements in each layer, which results in converting parameters into a set of small dense matrices. A small dense matrix does not require additional storage for element indices and is efficiently optimized for processing [19]. DeepIoT greatly reduces the effort of designing efficient neural structures for sensing applications by deciding the number of elements in each layer in a manner informed by the topology of the neural network.

DeepIoT borrows the idea of dropping hidden elements from a widely-used deep learning regularization method called dropout [44]. The dropout operation gives each hidden element a dropout probability. During the dropout process, hidden elements can be pruned according to their dropout probabilities. Then a "thinned" network structure can be generated. However, these dropout probabilities are usually set to a pre-defined value, such as 0.5. Such pre-defined values are not the optimal probabilities, thereby resulting in a less efficient exploration of the solution space. If we can obtain the optimal dropout probability for each hidden element, it becomes possible for us to generate the optimal slim network structure that preserves the accuracy of sensing applications while maximally reducing the resource consumption of sensing systems. An important purpose of DeepIoT is thus to find the optimal dropout probability for each hidden element in the neural network.

Notice that, dropout can be easily applied to all commonly used neural network structures. In fully-connected neural networks, neurons are dropped in each layer [44]; in convolutional neural networks, filters are dropped in each layer [14]; and in recurrent neural networks, dimensions are reduced in each layer [15]. This means that DeepIoT can be applied to all commonly-used neural network structures and their combinations.

To obtain the optimal dropout probabilities for the neural network, DeepIoT exploits the network parameters themselves. From the perspective of model compression, a hidden element that is connected to redundant model parameters should have a higher probability to be dropped. A contribution of DeepIoT lies in exploiting a novel *compressor* neural network to solve this problem. It takes model parameters of each layer as input, learns parameter redundancies, and generates the dropout probabilities accordingly. Since there are interconnections of parameters among different layers, we design the compressor neural network to be a recurrent neural network that can globally share the redundancy information and generate dropout probabilities layer by layer.

The compressor neural network is optimized jointly with the original neural network to be compressed through a compressor-critic framework that tries to minimize the loss function of the original

sensing applicaiton. The compressor-critic framework emulates the idea of the well-known actor-critic algorithm from reinforcement learning [28], optimizing two networks in an iterative manner.

We evaluate the DeepIoT framework on the Intel Edison computing platform [1], which Intel markets as an enabler platform for the computing elements of embedded "things" in IoT systems. We conduct two sets of experiments. The first set consists of three tasks that enable embedded systems to interact with humans with basic modalities, including handwritten text, vision, and speech, demonstrating superior accuracy of our produced neural networks, compared to others of similar size. The second set provides two examples of applying compressed neural networks to solving human-centric context sensing tasks; namely, human activity recognition and user identification, in a resource-constrained scenario.

We compare DeepIoT with other state-of-the-art magnitude-based [22] and sparse-coding-based [4] neural network compression methods. The resource consumption of resulting models on the Intel Edison module and the final performance of sensing applications are estimated for all compressed models. In all experiments, DeepIoT is shown to outperform the other algorithms by a large margin in terms of compression ratio, memory consumption, execution time, and energy consumption. In these experiments, when compared with the non-compressed neural networks, DeepIoT is able to reduce the model size by 90% to 98.9%, shorten the running time by 71.4% to 94.5%, and decrease the energy consumption by 72.2% to 95.7%. When compared with the state-out-the-art baseline algorithm, DeepIoT is able to reduce the model size by 11.6% to 83.2%, shorten the running time by 60.9% to 87.9%, and decrease the energy consumption by 64.1% to 88.7%. Importantly, these improvements are achieved without loss of accuracy. Experiments demonstrate the promise of DeepIoT in enabling resource-constrained embedded devices to benefit from advances in deep learning.

The rest of this paper is organized as follows. Section 2 introduces related work on optimizating sensing applications for resource-constrained devices. We describe the technical details of DeepIoT in Section 3. We describe system implementation in Section 4. The evaluation is presented in Section 5. Finally, we discuss the results in Section 6 and conclude in Section 7.

2 RELATED WORK

A key direction in embedded sensing literaure is to enable running progressively more interesting applications under the more pronounced resource constraints of embedded and mobile devices. Brouwers et al. reduced the energy consumption of Wi-Fi based localization with an incremental scanning strategy [5]. Hester et al. proposed an ultra-low-power hardware architecture and a companion software framework for energy-efficient sensing system [25]. Ferrari et al. and Schuss et al. focused on low-power wireless communication protocols [13, 41]. Wang et al. enabled energy efficient reliable broadcast by considering poorly correlated links [49]. Saifullah et al. designed a scalable and energy-efficient wireless sensor network (WSN) over white spaces [40]. Alsheikh et al. discussed about data compression in WSN with machine learning techniques [3].

Recent studies focused on compressing deep neural networks for embedded and mobile devices. Han et al. proposed a magnitude-based compression method with fine-tuning, which illustrated
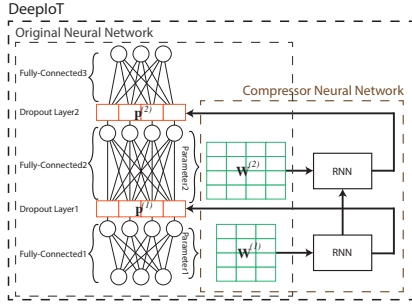
Figure 1: Overall DeepIoT system framework. Orange boxes represent dropout operations. Green boxes represent parameters of the original neural network.

promising compression results [24]. This method removes weight connections with low magnitude iteratively; however, it requires additional implementation of sparse matrix with more resource consumption. In addition, the aggressive pruning method increases the potential risk of irretrievable network damage. Guo et al. proposed a compression algorithm with connection splicing, which provided the chance of rehabilitation with a certain threshold [22]. However, the algorithm still focuses on weight level instead of structure level. Other than the magnitude-based method, another series of works focused on the factorization-based method that reduced the neural network complexity by exploiting low-rank structures in parameters. Denton et al. exploited various matrix factorization methods with fine-tunning to approximate the convolutional operations in order to reduce the neural network execution time [12]. Lane et al. applied sparse coding based and matrix factorization based method to reduce complexity of fully-connected layer and convolutional layer respectively [4]. However, factorization-based methods usually obtain lower compression ratio compared with magnitude-based methods, and the low-rank assumption may hurt the final network performance. Wang et al. applied the information of frequency domain for model compression [51]. However, additional implementation is required to speed-up the frequency-domain representations, and the method is not suitable for modern CNNs with small convolution filter sizes. Hinton et al. proposed a teacher-student framework that distilled the knowledge in an ensemble of models into a single model [26]. However, the framework focused more on compressing model ensemble into a single model instead of structure compression.

Our paper is partly inspired by deep reinforcement learning. With the aid of deep neural networks, reinforcement leaning has achieved great success on Atari games [33], Go chess [43], and multichannel access [50].

To the best of our knowledge, DeepIoT is the first framework for neural network structure compressing based on dropout operations and reducing parameter redundancies, where dropout operations provide DeepIoT the chance of rehabilitation with a certain probability. DeepIoT generates a more concise network structure for transplanting large-scale neural networks onto resource-constrained embedded devices.

3 SYSTEM FRAMEWORK

We introduce DeepIoT, a neural network structure compression framework for sensing applications. Without loss of generality,

before introducing the technical details, we first use an example of compressing a 3-layer fully-connected neural network structure to illustrate the overall pipeline of DeepIoT. The detailed illustration is shown in Figure 1. The basic steps of compressing neural network structures for sensing applications with DeepIoT can be summarized as follows.

(1) Insert operations that randomly zeroing out hidden elements with probabilities $\mathbf{p}^{(l)}$ called dropout (red boxes in Figure 1) into internal layers of the original neural network. The internal layers exclude input layers and output layers that have the fixed dimension for a sensing application. This step will be detailed in Section 3.1.

(2) Construct the compressor neural network. It takes the weight matrices $\mathbf{W}^{(l)}$ (green boxes in Figure 1) from the layers to be compressed in the original neural network as inputs, learns and shares the parameter redundancies among different layers, and generates optimal dropout probabilities $\mathbf{p}^{(l)}$, which is then fed back to the dropout operations in the original neural network. This step will be detailed in Section 3.2.

(3) Iteratively optimize the compressor neural network and the original neural network with the compressor-critic framework. The compressor neural network is optimized to produce better dropout probabilities that can generate a more efficient network structure for the original neural network. The original neural network is optimized to achieve a better performance with the more efficient structure for a sensing application. This step will be detailed in Section 3.3.

For the rest of this paper, all vectors are denoted by bold lower-case letters (*e.g.*, $\mathbf{x}$ and $\mathbf{y}$), and matrices and tensors are represented by bold upper-case letters (*e.g.*, $\mathbf{X}$ and $\mathbf{Y}$). For a column vector $\mathbf{x}$, the jth element is denoted by $\mathbf{x}_{[j]}$. For a tensor $\mathbf{X}$, the tth matrix along the third axis is denoted by $\mathbf{X}_{..t}$, and the other slicing denotations are defined similarly. The superscript l in $\mathbf{x}^{(l)}$ and $\mathbf{X}^{(l)}$ denote the vector and tensor for the lth layer of the neural network. We use calligraphic letters to denote sets (*e.g.*, $\mathcal{X}$ and $\mathcal{Y}$). For any set $\mathcal{X}$, $|\mathcal{X}|$ denotes the cardinality of $\mathcal{X}$.

3.1 Dropout Operations in the Original Neural Network

Dropout is commonly used as a regularization method that prevents feature co-adapting and model overfitting. The term "dropout" refers to dropping out units (hidden and visible) in a neural network. Since DeepIoT is a structure compression framework, we focus mainly on dropping out hidden units. The definitions of hidden units are distinct in different types of neural networks, and we will describe them in detail. The basic idea is that we regard neural networks with dropout operations as bayesian neural networks with Bernoulli variational distributions [14, 15, 44].

For the fully-connected neural networks, the fully-connected operation with dropout can be formulated as

$$
\begin{aligned}
\mathbf{z}_{[j]}^{(l)} &\sim \text{Bernoulli}\big(\mathbf{p}_{[j]}^{(l)}\big), \\
\tilde{\mathbf{W}}^{(l)} &= \mathbf{W}^{(l)}\text{diag}\big(\mathbf{z}^{(l)}\big), \\
\mathbf{Y}^{(l)} &= \mathbf{X}^{(l)}\tilde{\mathbf{W}}^{(l)} + \mathbf{b}^{(l)}, \\
\mathbf{X}^{(l+1)} &= f\big(\mathbf{Y}^{(l)}\big).
\end{aligned}
\tag{1}
$$

Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher

Refer to (1). The notation $l = 1, \cdots, L$ is the layer number in the fully-connected neural network. For any layer l, the weight matrix is denoted as $\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}$; the bias vector is denoted as $\mathbf{b}^{(l)} \in \mathbb{R}^{d^{(l)}}$; and the input is denoted as $\mathbf{X}^{(l)} \in \mathbb{R}^{1 \times d^{(l-1)}}$. In addition, $f(\cdot)$ is a nonlinear activation function.

As shown in (1), each hidden unit is controlled by a Bernoulli random variable. In the original dropout method, the success probabilities of $\mathbf{p}_{[j]}^{(l)}$ can be set to the same constant p for all hidden units [44], but DeepIoT uses the Bernoulli random variable with individual success probabilities for different hidden units in order to compress the neural network structure in a finer granularity.

For the convolutional neural networks, the basic fully-connected operation is replaced by the convolution operation [14]. However, the convolution can be reformulated as a linear operation as shown in (1). For any layer l, we denote $\mathcal{K}^{(l)} = \left\{ \mathbf{K}_k^{(l)} \right\}$ for $k = 1, \cdots, c^{(l)}$ as the set of convolutional neural network (CNN)'s kernels, where $\mathbf{K}_k^{(l)} \in \mathbb{R}^{h^{(l)} \times w^{(l)} \times c^{(l-1)}}$ is the kernel of CNN with height $h^{(l)}$, width $w^{(l)}$, and channel $c^{(l-1)}$. The input tensor of layer l is denoted as $\hat{\mathbf{X}}^{(l)} \in \mathbb{R}^{\hat{h}^{(l-1)} \times \hat{w}^{(l-1)} \times c^{(l-1)}}$ with height $\hat{h}^{(l-1)}$, width $\hat{w}^{(l-1)}$, and channel $c^{(l-1)}$.

Next, we convert convolving the kernels with the input into performing matrix product. We extract $h^{(l)} \times w^{(l)} \times c^{(l-1)}$ dimensional patches from the input $\hat{\mathbf{X}}^{(l)}$ with stride s and vectorize them. Collect these vectorized n patches to be the rows of our new input representation $\mathbf{X}^{(l)} \in \mathbb{R}^{n \times (h^{(l)} w^{(l)} c^{(l-1)})}$. The vectorized kernels form the columns of the weight matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{(h^{(l)} w^{(l)} c^{(l-1)}) \times c^{(l)}}$.

With this transformation, dropout operations can be applied to convolutional neural networks according to (1). The composition of pooling and activation functions can be regarded as the nonlinear function $f(\cdot)$ in (1). Instead of dropping out hidden elements in each layer, we drop out convolutional kernels in each layer. From the perspective of structure compression, DeepIoT tries to prune the number of kernels used in the convolutional neural networks.

For the recurrent neural network, we take a multi-layer Long Short Term Memory network (LSTM) as an example. The LSTM operation with dropout can be formulated as

$$
\begin{aligned}
\mathbf{z}_{[j]}^{(l)} &\sim \text{Bernoulli}(\mathbf{p}_{[j]}^{(l)}), \\
\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} &= \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} \mathbf{W}^{(l)} \begin{pmatrix} \mathbf{h}_t^{(l-1)} \odot \mathbf{z}^{(l-1)} \\ \mathbf{h}_{t-1}^{(l)} \odot \mathbf{z}^{(l)} \end{pmatrix}, \\
\mathbf{c}_t^{(l)} &= \mathbf{f} \odot \mathbf{c}_{t-1}^{(l)} + \mathbf{i} \odot \mathbf{g}, \\
\mathbf{h}_t^{(l)} &= \mathbf{o} \odot \tanh\left(\mathbf{c}_t^{(l)}\right).
\end{aligned}
\tag{2}
$$

The notation $l = 1, \cdots, L$ is the layer number and $t = 1, \cdots, T$ is the step number in the recurrent neural network. Element-wise multiplication is denoted by $\odot$. Operators $sigm$ and $tanh$ denote sigmoid function and hyperbolic tangent respectively. The vector $\mathbf{h}_t^{(l)} \in \mathbb{R}^{n^{(l)}}$ is the output of step t at layer l. The vector $\mathbf{h}_t^{(0)} = \mathbf{x}_t$ is the input for the whole neural network at step t. The matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{4n^{(l)} \times (n^{(l-1)} + n^{(l)})}$ is the weight matrix at layer l. We let $\mathbf{p}_{[j]}^{(0)} = 1$, since DeepIoT only drops hidden elements.

As shown in (2), DeepIoT uses the same vector of Bernoulli random variables $\mathbf{z}^{(l)}$ to control the dropping operations among different time steps in each layer, while individual Bernoulli random variables are used for different steps in the original LSTM dropout [57]. From the perspective of structure compression, DeepIoT tries to prune the number of hidden dimensions used in LSTM blocks. The dropout operation of other recurrent neural network architectures, such as Gated Recurrent Unit (GRU), can be designed similarly.

3.2 Compressor Neural Network

Now we introduce the architecture of the compressor neural network. As we described in Section 1, a hidden element in the original neural network that is connected to redundant model parameters should have a higher probability to be dropped. Therefore we design the compressor neural network to take the weights of an original neural network $\{\mathbf{W}^{(l)}\}$ as inputs, learn the redundancies among these weights, and generate dropout probabilities $\{\mathbf{p}^{(l)}\}$ for hidden elements that can be eventually used to compress the original neural network structure.

A straightforward solution is to train an individual fully-connected neural network for each layer in the original neural network. However, since there are interconnections among weight redundancies in different layers, DeepIoT uses a variant LSTM as the structure of compressor to share and use the parameter redundancy information among different layers.

According to the description in Section 3.1, the weight in layer l of fully-connected, convolutional, or recurrent neural network can all be represented as a single matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{d_f^{(l)} \times d_{\text{drop}}^{(l)}}$, where $d_{\text{drop}}^{(l)}$ denotes the dimension that dropout operation is applied and $d_f^{(l)}$ denotes the dimension of features within each dropout element. Here, we need to notice that the weight matrix of LSTM at layer l can be reshaped as $\mathbf{W}^{(l)} \in \mathbb{R}^{4 \cdot (n^{(l-1)} + n^{(l)}) \times n^{(l)}}$, where $d_{\text{drop}}^{(l)} = n^{(l)}$ and $d_f^{(l)} = 4 \cdot (n^{(l-1)} + n^{(l)})$. Hence, we take weights from the original network layer by layer, $\mathcal{W} = \left\{ \mathbf{W}^{(l)} \right\}$ with $l = 1, \cdots, L$, as the input of the compressor neural network. Instead of using a vanilla LSTM as the structure of compressor, we apply a variant l-step LSTM model shown as

$$
\begin{aligned}
\begin{pmatrix} \mathbf{v}_i^{\mathsf{T}} \\ \mathbf{v}_f^{\mathsf{T}} \\ \mathbf{v}_o^{\mathsf{T}} \\ \mathbf{v}_g^{\mathsf{T}} \end{pmatrix} &= \mathbf{W}_c^{(l)} \mathbf{W}^{(l)} \mathbf{W}_i^{(l)}, \qquad \begin{pmatrix} \mathbf{u}_i \\ \mathbf{u}_f \\ \mathbf{u}_o \\ \mathbf{u}_g \end{pmatrix} = \mathbf{W}_h \mathbf{h}_{l-1}, \\
\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} &= \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} \left(\begin{pmatrix} \mathbf{v}_i \\ \mathbf{v}_f \\ \mathbf{v}_o \\ \mathbf{v}_g \end{pmatrix} + \begin{pmatrix} \mathbf{u}_i \\ \mathbf{u}_f \\ \mathbf{u}_o \\ \mathbf{u}_g \end{pmatrix} \right), \\
\mathbf{c}_l &= \mathbf{f} \odot \mathbf{c}_{l-1} + \mathbf{i} \odot \mathbf{g}, \\
\mathbf{h}_l &= \mathbf{o} \odot \tanh\left(\mathbf{c}_l\right), \\
\mathbf{p}^{(l)} &= \mathbf{p}_t = \text{sigm}\left(\mathbf{W}_o^{(l)} \mathbf{h}_l\right), \\
\mathbf{z}_{[j]}^{(l)} &\sim \text{Bernoulli}(\mathbf{p}_{[j]}^{(l)}).
\end{aligned}
\tag{3}
$$

Refer to (3), we denote d_c as the dimension of the variant LSTM hidden state. Then $\mathbf{W}^{(l)} \in \mathbb{R}^{d_f^{(l)} \times d_{\text{drop}}^{(l)}}$, $\mathbf{W}_c^{(l)} \in \mathbb{R}^{4 \times d_f^{(l)}}$, $\mathbf{W}_i^{(l)} \in$

$\mathbb{R}^{d_{\text{drop}}^{(l)} \times d_c}$, $\mathbf{W}_h \in \mathbb{R}^{4d_c \times d_c}$, and $\mathbf{W}_o^{(l)} \in \mathbb{R}^{d_{\text{drop}}^{(l)} \times d_c}$. The set of training parameters of the compressor neural network is denoted as ϕ, where $\phi = \left\{ \mathbf{W}_c^{(l)}, \mathbf{W}_i^{(l)}, \mathbf{W}_h, \mathbf{W}_o^{(l)} \right\}$. The matrix $\mathbf{W}^{(l)}$ is the input matrix for step l in the compressor neural network, which is also the lth layer's parameters of the original neural network in (1) or (2).

Compared with the vanilla LSTM that requires vectorizing the original weight matrix as inputs, the variant LSTM model preserves the structure of original weight matrix and uses less learning parameters to extract the redundancy information among the dropout elements. In addition, $\mathbf{W}_c^{(l)}$ and $\mathbf{W}_i^{(l)}$ convert original weight matrix $\mathbf{W}^{(l)}$ with different sizes into fixed-size representations. The binary vector $\mathbf{z}^{(l)}$ is the dropout mask and probability $\mathbf{p}^{(l)}$ is the dropout probabilities for the lth layer in the original neural network used in (1) and (2), which is also the stochastic dropout policy learnt through observing the weight redundancies of the original neural network.

3.3 Compressor-Critic Framework

In Section 3.1 and Section 3.2, we have introduced customized dropout operations applied on the original neural networks that need to be compressed and the structure of compressor neural network used to learn dropout probabilities based on parameter redundancies. In this subsection, we will discuss the detail of compressor-critic compressing process. It optimizes the original neural network and the compressor neural network in an iterative manner and enables the compressor neural network to gradually compress the original neural network with soft deletion.

We denote the original neural network as $F_W(\mathbf{x}|\mathbf{z})$, and we call it critic. It takes $\mathbf{x}$ as inputs and generates predictions based on binary dropout masks $\mathbf{z}$ and model parameters $\mathcal{W}$ that refer to a set of weights $\mathcal{W} = \{\mathbf{W}^{(l)}\}$. We assume that $F_W(\mathbf{x}|\mathbf{z})$ is a pre-trained model. We denote the compressor neural network by $\mathbf{z} \sim \mu_\phi(\mathcal{W})$. It takes the weights of the critic as inputs and generates the probability distribution of the mask vector $\mathbf{z}$ based on its own parameters ϕ. In order to optimize the compressor to drop out hidden elements in the critic, DeepIoT follows the objective function

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_{\mathbf{z} \sim \mu_\phi}\left[L\left(\mathbf{y}, F_W(\mathbf{x}|\mathbf{z})\right)\right] \\ &= \sum_{\mathbf{z} \sim \{0,1\}^{|\mathbf{z}|}} \mu_\phi(\mathbf{W}) \cdot L\left(\mathbf{y}, F_W(\mathbf{x}|\mathbf{z})\right),\end{aligned} \tag{4}$$

where $L(\cdot, \cdot)$ is the objective function of the critic. The objective function can be interpreted as the expected loss of the original neural network over the dropout probabilities generated by the compressor.

DeepIoT optimizes the compressor and critic in an iterative manner. It reduces the expected loss as defined in (4) by applying the gradient descent method on compressor and critic iteratively. However, since there are discrete sampling operations, i.e., dropout operations, within the computational graph, backpropagation is not directly applicable. Therefore we apply an unbiased likelihood-ratio estimator to calculate the gradient over ϕ [17, 36]:

$$\begin{aligned}\nabla_\phi \mathcal{L} &= \sum_{\mathbf{z}} \nabla_\phi \mu_\phi(\mathcal{W}) \cdot L\left(\mathbf{y}, F_W(\mathbf{x}|\mathbf{z})\right) \\ &= \sum_{\mathbf{z}} \mu_\phi(\mathcal{W}) \nabla_\phi \log \mu_\phi(\mathcal{W}) \cdot L\left(\mathbf{y}, F_W(\mathbf{x}|\mathbf{z})\right) \\ &= \mathbb{E}_{\mathbf{z} \sim \mu_\phi}\left[\nabla_\phi \log \mu_\phi(\mathcal{W}) \cdot L\left(\mathbf{y}, F_W(\mathbf{x}|\mathbf{z})\right)\right].\end{aligned} \tag{5}$$

Therefore an unbiased estimator for (5) can be

$$\widehat{\nabla_\phi \mathcal{L}} = \nabla_\phi \log \mu_\phi(\mathcal{W}) \cdot L\left(\mathbf{y}, F_W(\mathbf{x}|\mathbf{z})\right) \quad \mathbf{z} \sim \mu_\phi. \tag{6}$$

The gradient over $\mathbf{W}^{(l)} \in \mathcal{W}$ is

$$\begin{aligned}\nabla_{\mathbf{W}^{(l)}} \mathcal{L} &= \sum_{\mathbf{z}} \mu_\phi(\mathcal{W}) \cdot \nabla_{\mathbf{W}^{(l)}} L\left(\mathbf{y}, F_W(\mathbf{x}|\mathbf{z})\right) \\ &= \mathbb{E}_{\mathbf{z} \sim \mu_\phi}\left[\nabla_{\mathbf{W}^{(l)}} L\left(\mathbf{y}, F_W(\mathbf{x}|\mathbf{z})\right)\right].\end{aligned} \tag{7}$$

Similarly, an unbiased estimator for (7) can be

$$\widehat{\nabla_{\mathbf{W}^{(l)}} \mathcal{L}} = \nabla_{\mathbf{W}^{(l)}} L\left(\mathbf{y}, F_W(\mathbf{x}|\mathbf{z})\right) \quad \mathbf{z} \sim \mu_\phi. \tag{8}$$

Now we provide more details of $\widehat{\nabla_\phi \mathcal{L}}$ in (6). Although the estimator (6) is an unbiased estimator, it tends to have a higher variance. A higher variance of estimator can make the convergence slower. Therefore, variance reduction techniques are typically required to make the optimization feasible in practice [21, 32].

One variance reduction technique is to subtract a constant c from learning signal $L\left(\mathbf{y}, F_W(\mathbf{x}|\mathbf{z})\right)$ in (5), which still keeps the expectation of the gradient unchanged [32]. Therefore, we keep track of the moving average of the learning signal $L\left(\mathbf{y}, F_W(\mathbf{x}|\mathbf{z})\right)$ denoted by c, and subtract c from the gradient estimator (6).

The other variance reduction technique is keeping track of the moving average of the signal variance v, and divides the learning signal by $\max(1, \sqrt{v})$ [21].

Combing the aforementioned two variance reduction techniques, the final estimator (6) for gradient over ϕ becomes

$$\widehat{\nabla_\phi \mathcal{L}} = \nabla_\phi \log \mu_\phi(\mathcal{W}) \cdot \frac{L\left(\mathbf{y}, F_W(\mathbf{x}|\mathbf{z})\right) - c}{\max(1, \sqrt{v})} \quad \mathbf{z} \sim \mu_\phi, \tag{9}$$

where c and v are the moving average of mean and the moving average of variance of learning signal $L\left(\mathbf{y}, F_W(\mathbf{x}|\mathbf{z})\right)$ respectively.

After introducing the basic optimization process in DeepIoT, now we are ready to deliver the details of the compressing process. Compared with previous compressing algorithms that gradually delete weights without rehabilitation [24], DeepIoT applies "soft" deletion by gradually suppressing the dropout probabilities of hidden elements with a decay factor $\gamma \in (0, 1)$. During the experiments in Section 5, we set γ as the default value 0.5. Since it is impossible to make the optimal compression decisions from the beginning, suppressing the dropout probabilities instead of deleting the hidden elements directly can provide the "deleted" hidden elements changes to recover. This less aggressive compression process reduces the potential risk of irretrievable network damage and learning inefficiency.

During the compressing process, DeepIoT gradually increases the threshold of dropout probability τ from 0 with step Δ. The hidden elements with dropout probability, $\mathbf{p}_{[j]}^{(l)}$ that is less than the threshold τ will be given decay on dropout probability, i.e., $\hat{\mathbf{p}}_{[j]}^{(l)} \leftarrow \gamma \cdot \mathbf{p}_{[j]}^{(l)}$. Therefore, the operation in compressor (3) can be updated as

$$\mathbf{z}_{[j]}^{(l)} \sim \text{Bernoulli}\left(\mathbf{p}_{[j]}^{(l)} \cdot \gamma^{\mathbb{1}^{\mathbf{p}_{[j]}^{(l)} \leq \tau}}\right), \tag{10}$$

where $\mathbb{1}$ is the indicator function; $\gamma \in (0, 1)$ is the decay factor; and $\tau \in [0, 1)$ is the threshold. Since the operation of suppressing dropout probability with the pre-defined decay factor γ is differentiable, we can still optimize the original and the compressor neural

---

Algorithm 1 Compressor-predictor compressing process

1: **Input:** pre-trained predictor $F_{\mathcal{W}}(\mathbf{x}|\mathbf{z})$
2: **Initialize:** compressor $\mu_\phi(\mathcal{W})$ with parameter ϕ, moving average c, moving average of variance v
3: **while** $\mu_\phi(\mathcal{W})$ is not convergent **do**
4:     $\mathbf{z} \sim \mu_\phi(\mathcal{W})$
5:     $c \leftarrow \text{movingAvg}\big(L\big(\mathbf{y}, F_{\mathcal{W}}(\mathbf{x}|\mathbf{z})\big)\big)$
6:     $v \leftarrow \text{movingVar}\big(L\big(\mathbf{y}, F_{\mathcal{W}}(\mathbf{x}|\mathbf{z})\big)\big)$
7:     $\phi \leftarrow \phi - \beta \cdot \nabla_\phi \log \mu_\phi(\mathcal{W}) \cdot \big(L\big(\mathbf{y}, F_{\mathcal{W}}(\mathbf{x}|\mathbf{z})\big) - c\big)/\max(1, \sqrt{v})$
8: **end while**
9: $\tau = 0$
10: **while** the percentage of left number of parameters in $F_{\mathcal{W}}(\mathbf{x}|\mathbf{z})$ is larger than α **do**
11:     $\mathbf{z} \sim \mu_\phi(\mathcal{W})$
12:     $c \leftarrow \text{movingAvg}\big(L\big(\mathbf{y}, F_{\mathcal{W}}(\mathbf{x}|\mathbf{z})\big)\big)$
13:     $v \leftarrow \text{movingVar}\big(L\big(\mathbf{y}, F_{\mathcal{W}}(\mathbf{x}|\mathbf{z})\big)\big)$
14:     $\phi \leftarrow \phi - \beta \cdot \nabla_\phi \log \mu_\phi(\mathcal{W}) \cdot \big(L\big(\mathbf{y}, F_{\mathcal{W}}(\mathbf{x}|\mathbf{z})\big) - c\big)/\max(1, \sqrt{v})$
15:     $\mathcal{W} \leftarrow \mathcal{W} - \beta \cdot \nabla_{\mathcal{W}} L\big(\mathbf{y}, F_{\mathcal{W}}(\mathbf{x}|\mathbf{z})\big)$
16:     update threshold τ: $\tau \leftarrow \tau + \Delta$ for every T rounds
17: **end while**
18: $\hat{\mathbf{z}}^{(l)}_{[j]} = \mathbb{1}\,\mathbf{p}^{(l)}_{[j]} > \tau$
19: **while** $F_{\mathcal{W}}(\mathbf{x}|\hat{\mathbf{z}})$ is not convergent **do**
20:     $\mathcal{W} \leftarrow \mathcal{W} - \beta \cdot \nabla_{\mathcal{W}} L\big(\mathbf{y}, F_{\mathcal{W}}(\mathbf{x}|\hat{\mathbf{z}})\big)$
21: **end while**

---

network through (8) and (9). The compression process will stop when the percentage of left number of parameters in $F_{\mathcal{W}}(\mathbf{x}|\mathbf{z})$ is smaller than a user-defined value $\alpha \in (0, 1)$.

After the compression, DeepIoT fine-tunes the compressed model $F_{\mathcal{W}}(\mathbf{x}|\hat{\mathbf{z}})$, with a fixed mask $\hat{\mathbf{z}}$, which is decided by the previous threshold τ. Therefore the mask generation step in (10) will be updated as

$$\hat{\mathbf{z}}^{(l)}_{[j]} = \mathbb{1}\,\mathbf{p}^{(l)}_{[j]} > \tau. \tag{11}$$

We summarize the compressor-critic compressing process of DeepIoT in Algorithm 1.

The algorithm consists of three parts. In the first part (Line 3 to Line 8), DeepIoT freezes the critic $F_{\mathcal{W}}(\mathbf{x}|\mathbf{z})$ and initializes the compressor $\mu_\phi(\mathcal{W})$ according to (9). In the second part (Line 9 to Line 17), DeepIoT optimizes the critic and compressor jointly with the gradients calculated by (8) and (9). At the same time, DeepIoT gradually compresses the predictor by suppressing dropout probabilities according to (10). In the final part (Line 18 to Line 21), DeepIoT fine-tunes the critic with the gradient calculated by (8) and a deterministic dropout mask is generated according to (11). After these three phases, DeepIoT generates a binary dropout mask $\hat{\mathbf{z}}$ and the fine-tuning parameters of the critic $\mathcal{W}$. With these two results, we can easily obtain the compressed model of the original neural network.

4 IMPLEMENTATION

In this section, we briefly describe the hardware, software, architecture, and performance summary of DeepIoT.

4.1 Hardware

Our hardware is based on Intel Edison computing platform [1]. The Intel Edison computing platform is powered by the Intel Atom SoC dual-core CPU at 500 MHz and is equipped with 1GB memory and 4GB flash storage. For fairness, all neural network models are run solely on CPU during experiments.

4.2 Software

All the original neural networks for all sensing applications mentioned in Section 5 are trained on the workstation with NVIDIA GeForce GTX Titan X. For all baseline algorithms mentioned in Section 5, the compressing processes are also conducted on the workstation. The compressed models are exported and loaded into the flash storage on Intel Edison for experiments.

We installed the Ubilinux operation system on Intel Edison computing platform [2]. Far fairness, all compressed deep learning models are run through Theano [48] with only CPU device on Intel Edison. The matrix multiplication operations and sparse matrix multiplication operations are optimized by BLAS and Sparse BLAS respectively during the implementation. No additional run-time optimization is applied for any compressed model and in all experiments.

4.3 Architecture

Given the original neural network structure and parameters as well as the device resource information, DeepIoT can automatically generate a compressed neural network that is ready to be run on embedded devices with sensor inputs. The system first obtains the memory information from the embedded device and sets the final compressed size of the neural network to fit in a pre-configured fraction of available memory, from which the needed compression ratio is computed. In the experiments, we manually set the ratio to exploit the capability of DeepIoT. This ratio, together with the parameters of the original model are then used to automatically generate the corresponding compressor neural network to compress the original neural network. The resulting compressed neural network is transferred to the embedded device. This model can be then called locally with a data input to decide on the output. The semantics of input and output are not known to the model.

4.4 Performance Summary

We list the resource consumption numbers of all compressed models without loss of accuracy generated by DeepIoT and their corresponding original model in Table 1 with the form of (original/compressed/reduction percentage). These models are explained in more detail in the evaluation, Section 5.

Table 1: Resource consumptions of model implementations on Intel Edison

Model	Size (MB)	Time (ms)	Energy (mJ)
LeNet5	1.72/0.04/97.6%	50.2/14.2/71.4%	47.1/12.5/73.5%
VGGNet	118.8/2.9/97.6%	1.5K/82.2/94.5%	1.7K/74/95.6%
Bi-LSTM	76.0/7.59/90.0%	71K/9.6K/86.5%	62.9K/8.1K/87.1%
DeepSense1	1.89/0.12/93.7%	130/36.7/71.8%	99.6/27.7/72.2%
DeepSense2	1.89/0.02/98.9%	130/25.1/80.7%	105.1/18.1/82.8%

Although the models generated by DeepIoT do not use sparse matrix representations, other baseline algorithms, as will be introduced in Section 5, may use sparse matrices to represent models. When the proportion of non-zero elements in the sparse matrix is larger than 20%, sparse matrix multiplications can even run slower than their non-sparse counterpart. Therefore, there is a tradeoff between memory consumption and execution time for sparse matrices with a large proportion of non-zero elements. In addition, convolution

operations conducted on CPU are also formulated and optimized as matrix multiplications, as mentioned in Section 3.1. Therefore, the tradeoff still exists. For all baseline algorithms in Section 5, we implement both the sparse matrix version and the non-sparse matrix version. During all the experiments with baseline algorithms, we "cheat", in their favor, by choosing the version that performs better according to the current evaluation metrics.

5   EVALUATION

In this section, we evaluate DeepIoT through two sets of experiments. The first set is motivated by the prospect of enabling future smarter embedded "things" (physical objects) to interact with humans using user-friendly modalities such as visual cues, handwritten text, and speech commands, while the second evaluates human-centric context sensing, such as human activity recognition and user identification. In the following subsections, we first describe the comparison baselines that are current state of the art deep neural network compression techniques. We then present the first set of experiments that demonstrate accuracy and resource demands observed if IoT-style smart objects interacted with users via natural human-centric modalities thanks to deep neural networks compressed, for the resource-constrained hardware, with the help of our DeepIoT framework. Finally, we present the second set of experiments that demonstrate accurancy and resource demands when applying DeepIoT to compress deep neural networks trained for human-centric context sensing applications. In both cases, we show significant advantages in the accuracy/resource trade-off over the compared state-of-the-art compression baselines.

5.1   Baseline Algorithms

We compare DeepIoT with other three baseline algorithms:

(1) **DyNS:** This is a magnitude-based network pruning algorithm [22]. The algorithm prunes weights in convolutional kernels and fully-connected layer based on the magnitude. It retrains the network connections after each pruning step and has the ability to recover the pruned weights. For convolutional and fully-connected layers, DyNS searches the optimal thresholds separately.

(2) **SparseSep:** This is a sparse-coding and factorization based algorithm [4]. The algorithm simplifies the fully-connected layer by finding the optimal code-book and code based on a sparse coding technique. For the convolutional layer, the algorithm compresses the model with matrix factorization methods. We greedily search for the optimal code-book and factorizaiton number from the bottom to the top layer.

(3) **DyNS-Ext:** The previous two algorithms mainly focus on compressing convolutional and fully-connected layers. Therefore we further enhance and extend the magnitude-based method used in DyNS to recurrent layers and call this algorithm DyNS-Ext. Just like DeepIoT, DyNS-Ext can be applied to all commonly used deep network modules, including fully-connected layers, convolutional layers, and recurrent layers. If the network structure does not contain recurrent layers, we apply DyNS instead of DyNS-Ext.

For magnitude-based pruning algorithms, DyNS and DyNS-Ext, hidden elements with zero input connections or zero output connections will be pruned to further compress the network structure. In addition, all models use 32-bit floats without any quantization.

5.2   Supporting Human-Centric Interaction Modalities

Three basic interaction modalities among humans are text, vision, and speech. In this section, we describe three different experiments that test implementations of these basic interaction modalities on low-end devices using trained and compressed neural networks. We train state-of-art neural networks on traditional benchmark datasets as original models. Then, we compress the original models using DeepIoT and the three baseline algorithms described above. Finally, we test the accuracy and resource consumption that result from using these compressed models on the embedded device.

5.2.1   Handwritten digits recognition with LeNet5. The first human interaction modality is recognizing handwritten text. In this experiment, we consider a meaningful subset of that; namely recognizing handwritten digits from visual inputs. An example application that uses this capability might be a smart wallet equipped with a camera and a tip calculator. We use MNIST[2] as our training and testing dataset. The MNIST is a dataset of handwritten digits that is commonly used for training various image processing systems. It has a training set of 60000 examples, and a test set of 10000 examples.

We test our algorithms and baselines on the LeNet-5 neural network model. The corresponding network structure is shown in Table 2. Notice that we omit all the polling layers in Table 2 for simplicity, because they do not contain training parameters.

The first column of Table 2 represents the network structure of LeNet-5, where "convX" represents the convolutional layer and "fcY" represents the fully-connected layer. The second column represents the number of hidden units or convolutional kernels we used in each layer. The third column represents the number of parameters used in each layer and in total. The original LeNet-5 is trained and achieves an error rate of 0.85% in the test dataset.

We then apply DeepIoT and two other baseline algorithms, DyNS and SparseSep, to compress LeNet-5. Note that, we do not use DyNS-Ext because the network does not contain a recurrent layer. The network statistics of the compressed model are shown in Table 2. DeepIoT is designed to prune the number of hidden units for a more efficient network structure. Therefore, we illustrate both the remaining number of hidden units and the proportion of the remaining number of parameters in Table 2. Both DeepIoT and DyNS can significantly compress the network without hurting the final performance. SparseSep shows an acceptable drop of performance. This is because SparseSep is designed without fine-tuning. It has the benefit of not fine-tuning the model, but it suffers the loss in the final performance at the same time.

The detailed tradeoff between testing accuracy and memory consumption by the model is illustrated in Fig 2a. We compress the original neural network with different compression ratios and recode the final testing accuracy. In the zoom-in illustration, DeepIoT achieves at least ×2 better tradeoff compared with the two baseline methods. This is mainly due to two reasons. One is that the compressor neural network in DeepIoT obtains a global view of parameter redundancies and is therefore better capable of eliminating them. The other is that DeepIoT prunes the hidden units directly, which

---

[2]http://yann.lecun.com/exdb/mnist/

Table 2: LeNet5 on MNIST dataset

Layer	Hidden Units	Params	DeepIoT (Hidden Units/ Params)		DyNS	SparseSep
conv1 (5×5)	20	0.5K	10	50.0%	24.2%	84%
conv2 (5×5)	50	25K	20	20.0%	20.7%	91%
fc1	500	400K	10	0.8%	1.0%	78.75%
fc2	10	5K	10	2.0%	16.34%	70.28%
total		431K		1.98%	2.35%	72.39%
Test Error	0.85%		0.85%		0.85%	1.05%
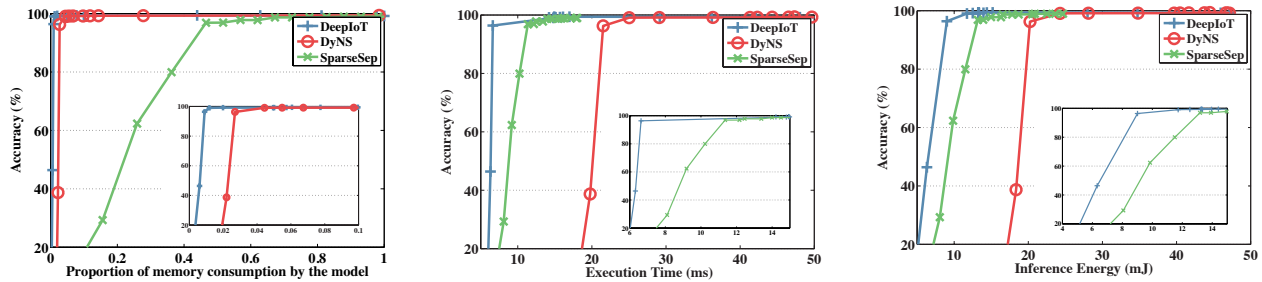
(a) The tradeoff between testing accuracy and memory consumption by models.

(b) The tradeoff between testing accuracy and execution time.

(c) The tradeoff between testing accuracy and energy consumption.

Figure 2: System performance tradeoff for LeNet5 on MNIST dataset

enables us to represent the compressed model parameters with a small dense matrix instead of a large sparse matrix. The sparse matrix consumes more memory for the indices of matrix elements. Algorithms such as DyNS generate models represented by sparse matrices that cause larger memory consumption.

The evaluation results on execution time of compressed models on Intel Edison, are illustrated in Fig. 2b. We run each compressed model on Intel Edison for 5000 times and use the mean value for generating the tradeoff curves.

DeepIoT still achieves the best tradeoff compared with other two baselines by a significant margin. DeepIoT takes 14.2ms to make a single inference, which reduces execution time by 71.4% compared with the original network without loss of accuracy. However SparseSep takes less execution time compared with DyNS at the cost of acceptable performance degradation (around 0.2% degradation on test error). The main reason for this observation is that, even though fully-connected layers occupy the most model parameters, most execution time is used by the convolution operations. SparseSep uses a matrix factorization method to covert the 2d convolutional kernel into two 1d convolutional kernels on two different dimensions [47]. Although this method makes low-rank assumption on convolutional kernel, it can speed up convolution operations if the size of convolutional kernel is large (5×5 in this experiment). It can sometimes speed up the operation even when two 1d kernels have more parameters in total compared with the original 2d kernel. However DyNS applies a magnitude-based method that prunes most of the parameters in fully-connected layers. For convolutional layers, DyNS does not reduce the number of convolutional operations effectively, and sparse matrix multiplication is less efficient compared with regular matrix with the same number of elements. DeepIoT directly reduces the number of convolutional kernels in each layer, which reduces the number of operations in convolutional

layers without making the low-rank assumption that can hurt the network performance.

The evaluation of energy consumption on Intel Edison is illustrated in Fig. 2c. For each compressed model, we run it for 5000 times and measure the total energy consumption by a power meter. Then, we calculate the expected energy consumption for one-time execution and use the one-time energy consumption to generate the tradeoff curves in Fig. 2c.

Not surprisingly, DeepIoT still achieves the best tradeoff in the evaluation on energy consumption by a significant margin. It reduces energy consumption by 73.7% compared with the original network without loss of accuracy. Being similar as the evaluation on execution time, energy consumption focuses more on the number of operations than the model size. Therefore, SparseSep can take less energy consumption compared with DyNS at the cost of acceptable loss on performance.

5.2.2 Image recognition with VGGNet. The second human interaction modality is through vision. During this experiment, we use CIFAR10[3] as our training and testing dataset. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. It is a standard testing benchmark dataset for the image recognition tasks. While not necessarily representative of seeing objects in the wild, it offers a more controlled environment for an apples-to-apples comparison.

During this evaluation, we use the VGGNet structure as our original network structure. It is a huge network with millions of parameters. VGGNet is chosen to show that DeepIoT is able to compress relative deep and large network structure. The detailed structure is shown Table 3.

---

[3]https://www.kaggle.com/c/cifar-10

Table 3: VGGNet on CIFAR-10 dataset

Layer	Hidden Units	Params	DeepIoT (Hidden Units/ Params)		DyNS	SparseSep
conv1 (3×3)	64	1.7K	27	42.2%	53.9%	93.1%
conv2 (3×3)	64	36.9K	47	31.0%	40.1%	57.3%
conv3 (3×3)	128	73.7K	53	30.4%	52.3%	85.1%
conv4 (3×3)	128	147.5K	68	22.0%	67.0%	56.8%
conv5 (3×3)	256	294.9K	104	21.6%	71.2%	85.1%
conv6 (3×3)	256	589.8K	97	15.4%	65.0%	56.8%
conv7 (3×3)	256	589.8K	89	13.2%	61.2%	56.8%
conv8 (3×3)	512	1.179M	122	8.3%	36.5%	85.2%
conv9 (3×3)	512	2.359M	95	4.4%	10.6%	56.8%
conv10 (3×3)	512	2.359M	64	2.3%	3.9%	56.8%
conv11 (2×2)	512	1.049M	128	3.1%	3.0%	85.2%
conv12 (2×2)	512	1.049M	112	5.5%	1.7%	85.2%
conv13 (2×2)	512	1.049M	149	6.4%	2.4%	85.2%
fc1	4096	2.097M	27	0.19%	2.2%	95.8%
fc2	4096	16.777M	371	0.06%	0.39%	135%
fc3	10	41K	10	9.1%	18.5%	90.2%
total		29.7M		2.44%	7.05%	112%
Test Accuracy	90.6%		90.6%		90.6%	87.1%
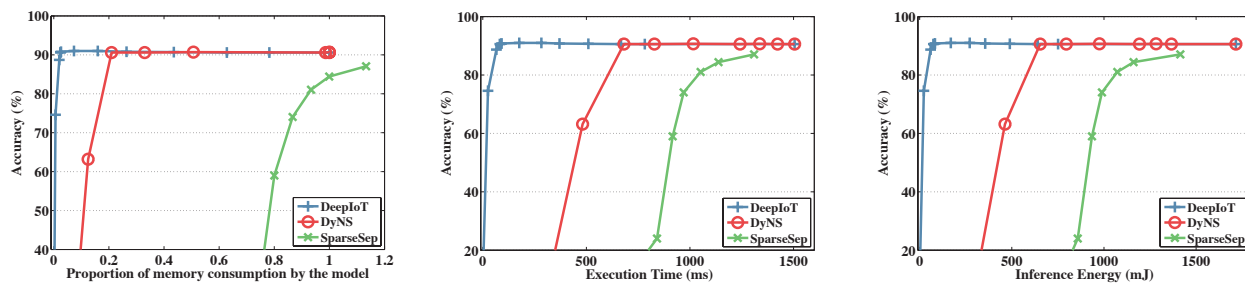
(a) The tradeoff between testing accuracy and memory consumption by models.

(b) The tradeoff between testing accuracy and execution time.

(c) The tradeoff between testing accuracy and energy consumption.

Figure 3: System performance tradeoff for VGGNet on CIFAR-10 dataset

In Table 3, we illustrate the detailed statistics of best compressed model that keeps the original testing accuracy for three algorithms. We clearly see that DeepIoT beats the other two baseline algorithms by a significant margin. This shows that the compressor in DeepIoT can handle networks with relatively deep structure. The compressor uses a variant of the LSTM architecture to share the redundancy information among different layers. Compared with other baselines considering only local information within each layer, sharing the global information among layers helps us learn about the parameter redundancy and compress the network structure. In addition, we observe performance loss in the compressed network generated by SparseSep. It is mainly due to the fact that SparseSep avoids the fine-tuning step. This experiment shows that fine-tuning (Line 18 to Line 21 in Algorithm 1) is important for model compression.

Fig. 3a shows the tradeoff between testing accuracy and memory consumption for different models. DeepIoT achieves a better performance by even a larger margin, because the model generated by DeepIoT can still be represented by a standard matrix, while other methods that use a sparse matrix representation require more memory consumption.

Fig. 3b shows the tradeoff between testing accuracy and execution time for different models. DeepIoT still achieves the best tradeoff. DeepIoT takes 82.2ms for a prediction, which reduces 94.5% execution time without the loss of accuracy. Different from the experiment with LeNet-5 on MNIST, DyNS uses less execution

time compared with SparseSep in this experiment. There are two reasons for this. One is that VGGNet use smaller convolutional kernel compared with LeNet-5. Therefore factorizing 2d kernel into two 1d kernel helps less on reducing computation time. The other point is that SparseSep fails to compress the original network into a small size while keeping the original performance. As we mentioned before, it is because SparseSep avoids the fine-tuning.

Fig. 3c shows the tradeoff between testing accuracy and energy consumption for different models. DeepIoT reduces energy consumption by 95.7% compared with the original VGGNet without loss of accuracy. It greatly helps us to develop a long-standing application with deep neural network in energy-constrained embedded devices.

5.2.3 Speech recognition with deep Bidirectional LSTM. The third human interaction modality is speech. The sensing system can take the voices of users from the microphone and automatically convert what users said into text. The previous two experiments mainly focus on the network structure with convolutional layers and fully-connected layers. We see how DeepIoT and the baseline algorithms work on the recurrent neural network in this section.

In this experiment, we use LibriSpeech ASR corpus [35] as our training and testing dataset. The LibriSpeech ASR corpus is a large-scale corpus of read English speech. It consists of 460-hour training data and 2-hour testing data.

Table 4: Deep bidirectional LSTM on LibriSpeech ASR corpus

Layer		Hidden Unit		Params		DeepIoT (Hidden Units/ Params)				DyNS-Ext	
LSTMf1	LSTMb1	512	512	1.090M	1.090M	55	20	10.74%	3.91%	34.9%	18.2%
LSTMf2	LSTMb2	512	512	2.097M	2.097M	192	71	4.03%	0.54%	37.2%	23.1%
LSTMf3	LSTMb3	512	512	2.097M	2.097M	240	76	17.58%	2.06%	43.1%	27.9%
LSTMf4	LSTMb4	512	512	2.097M	2.097M	258	81	23.62%	2.35%	52.3%	40.2%
LSTMf5	LSTMb5	512	512	2.097M	2.097M	294	90	28.93%	2.78%	72.6%	61.8%
fc1		29		59.3K		29		37.5%		69.0%	
total				19.016M				9.98%		37.1%	
Word error rate (WER)		9.31				9.20				9.62	
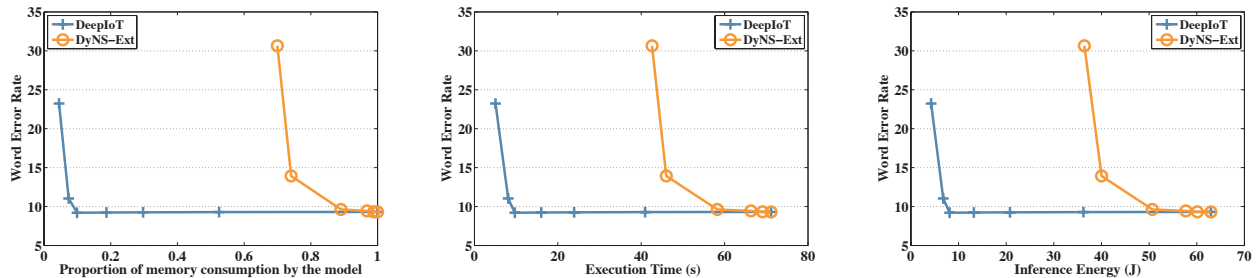
(a) The tradeoff between word error rate and memory consumption by models.

(b) The tradeoff between word error rate and execution time.

(c) The tradeoff between word error rate and energy consumption.

Figure 4: System performance tradeoff for deep bidirectional LSTM on LibriSpeech ASR corpus

We choose deep bidirectional LSTM as the original model [20] in this experiment. It takes mel frequency cepstral coefficient (MFCC) features of voices as inputs, and uses two 5-layer long short-term memory (LSTM) in both forward and backward direction. The output of two LSTM are jointly used to predict the spoken text. The detailed network structure is shown in the first column of Table 4, where "LSTMf" denotes the LSTM in forward direction and "LSTMb" denotes the LSTM in backward direction.

Two baseline algorithms are not applicable to the recurrent neural network, so we compared DeepIoT only with SyNS-Ext in this experiment. The word error rate (WER), defined as the edit distance between the true word sequence and the most probable word sequence predicted by the neural network, is used as the evaluation metric for this experiment.

We show the detailed statistics of best compressed model that keeps the original WER in Table 4. DeepIoT achieves a significantly better compression rate compared with DyNS-Ext, and the model generated by DeepIoT even has a little improvement on WER. However, compared with the previous two examples on convolutional neural network, DeepIoT fails to compress the model to less than 5% of the original parameters in the recurrent neural network case (still a 20-fold reduction though). The main reason is that compressing recurrent networks needs to prune both the output dimension and the hidden dimension. It has been shown that dropping hidden dimension can harm the network performance [57]. However DeepIoT is still successful in compressing network to less than 10% of parameters.

Fig. 4a shows the tradeoff between word error rate and memory consumption by compressed models. DeepIoT achieves around ×7 better tradeoff compared with magnitude-based method, DyNS-Ext. This means compressing recurrent neural networks requires more information about parameter redundancies within and among each layer. Compression using only local information, such as magnitude information, will cause degradation in the final performance.

Fig. 4b shows the tradeoff between word error rate and execution time. DeepIoT reduces execution time by 86.4% without degradation on WER compared with the original network. With the evaluation on Intel Edision, the original network requires 71.15 seconds in average to recognize one human speak voice example with the average length of 7.43 seconds. The compressed structure generated by DeepIoT reduces the average execution time to 9.68 seconds without performance loss, which improves responsiveness of human voice recognition.

Fig. 4c shows the tradeoff between word error rate and energy consumption. DeepIoT reduces energy by 87% compared with the original network. It performs better than DyNS-Ext by a large margin.

5.3 Supporting Human-Centric Context Sensing

In addition to experiments about supporting basic human-centric interaction modalities, we evaluate DeepIoT on several human-centric context sensing applications. We compress the state-of-the-art deep learning model, DeepSense, [56] for these problems and evaluate the accuracy and other system performance for the compressed networks. DeepSense contains all commonly used modules, including convolutional, recurrent, and fully-connected layers, which is also a good example to test the performance of compression algorithms on the combination of different types of neural network modules.

Two human-centric context sensing tasks we consider are heterogeneous human activity recognition (HHAR) and user identification with biometric motion analysis (UserID). The HHAR task recognizes human activities with motion sensors, accelerometer and gyroscope. "Heterogeneous" means that the task is focus on the generalization ability with human who has not appeared in the training set. The UserID task identifies users during a person's daily activities such as walking, running, siting, and standing.

Table 5: Heterogeneous human activity recognition

Layer		Hidden Unit		Params		DeepIoT (Hidden Units/ Params)				DyNS-Ext		DyNS		SparseSep	
conv1a	conv1b (2 × 9)	64	64	1.1K	1.1K	20	19	31.25%	29.69%	92%	95.7%	50.3%	60.0%	100%	100%
conv2a	conv2b (1 × 3)	64	64	12.3K	12.3K	20	14	9.76%	6.49%	70.1%	77.7%	25.3%	40.5%	114%	114%
conv3a	conv3b (1 × 3)	64	64	12.3K	12.3K	23	23	11.23%	7.86%	69.9%	66.2%	32.1%	35.4%	114%	114%
conv4 (2 × 8)		64		65.5K		10		5.61%		40.3%		20.4%		53.7%	
conv5 (1 × 6)		64		24.6K		12		2.93%		27.2%		18.3%		100%	
conv6 (1 × 4)		64		16.4K		17		4.98%		24.6%		12.0%		100%	
gru1		120		227.5K		27		5.8%		1.2%		100%		100%	
gru2		120		86.4K		31		6.24%		3.6%		100%		100%	
fc1		6		0.7K		6		25.83%		98.6%		99%		70%	
total				472.5K				6.16%		17.1%		74.5%		95.3%	
Test Accuracy		94.6%				94.7%				94.6%		94.6%		93.7%	

Table 6: User identification with biometric motion analysis

Layer		Hidden Unit		Params		DeepIoT (Hidden Units/ Params)				DyNS-Ext		DyNS		SparseSep	
conv1a	conv1b (2 × 9)	64	64	1.1K	1.1K	7	1	10.93%	1.56%	64.4%	75.5%	66.8%	65.6%	100%	100%
conv2a	conv2b (1 × 3)	64	64	12.3K	12.3K	7	4	1.2%	0.1%	32.5%	34.7%	36.6%	48.0%	114%	114%
conv3a	conv3b (1 × 3)	64	64	12.3K	12.3K	9	9	1.54%	0.88%	31.6%	28.6%	38.4%	43.5%	114%	114%
conv4 (2 × 8)		64		65.5K		7		1.54%		12.1%		29.2%		53.7%	
conv5 (1 × 6)		64		24.6K		5		0.85%		21.0%		23.3%		100%	
conv6 (1 × 4)		64		16.4K		7		0.85%		18.9%		16.0%		100%	
gru1		120		227.5K		13		1.18%		0.42%		100%		100%	
gru2		120		86.4K		9		0.69%		1.61%		100%		100%	
fc1		9		1.1K		9		7.5%		89.6%		98%		88%	
total				472.9K				1.13%		7.76%		77.0%		95.4%	
Test Accuracy		99.6%				99.6%				99.6%		99.6%		98.8%	
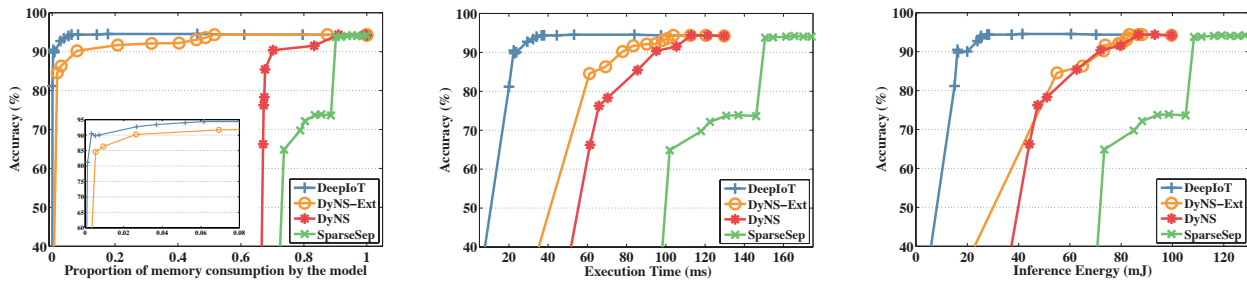
(a) The tradeoff between testing accuracy and memory consumption by models.
(b) The tradeoff between testing accuracy and execution time.
(c) The tradeoff between testing accuracy and energy consumption.

Figure 5: System performance tradeoff for heterogeneous human activity recognition
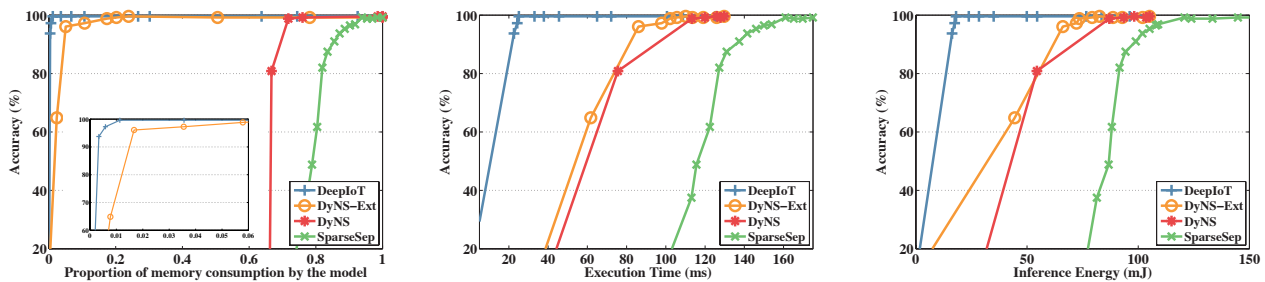
(a) The tradeoff between testing accuracy and memory consumption by models.
(b) The tradeoff between testing accuracy and execution time.
(c) The tradeoff between testing accuracy and energy consumption.

Figure 6: System performance tradeoff for user identification with biometric motion analysis

In this evaluation section, we use the dataset collected by Allan et al. [45]. This dataset contains readings from two motion sensors (accelerometer and gyroscope). Readings were recorded when users execute activities scripted in no specific order, while carrying smartwatches and smartphones. The dataset contains 9 users, 6 activities (biking, sitting, standing, walking, climbStairup, and climbStair-down), and 6 types of mobile devices. For both tasks, accelerometer and gyroscope measurements are model inputs. However, for HHAR tasks, activities are used as labels, and for UserID tasks, users' unique IDs are used as labels.

The original network structure of DeepSense is shown in the first two columns of Table 5 and 6. Both tasks use a unified neural network structure as introduced in [56]. The structure contains both convolutional and recurrent layers. Since SparseSep and DyNS are not directly applicable to recurrent layers, we keep the recurrent layers unchanged while using them. In addition, we also compare DeepIoT with DyNS-Ext in this experiment.

Table 5 and 6 illustrate the statistics of final pruned network generated by four algorithms that have no or acceptable degradation on testing accuracy. DeepIoT is the best-performing algorithm considering the remaining number of network parameters. This is mainly due to the design of compressor network and compressor-critic framework that jointly reduce the redundancies among parameters while maintaining a global view across different layers. DyNS and SparseSep are two algorithms that can be only applied to the fully-connected and convolutional layers in the original structure. Therefore there exists a lower bound of the left proportion of parameters, *i.e.*, the number of parameters in recurrent layers. This lower bound is around 66%.

The detailed tradeoffs between testing accuracy and memory consumption by the models are illustrated in Fig. 5a and 6a. DeepIoT still achieves the best tradeoff for sensing applications. Other than the compressor neural network providing global parameter redundancies, directly pruning hidden elements in each layer also enables DeepIoT to obtain more concise representations in matrix form, which results in less memory consumption.

The tradeoffs between system execution time and testing accuracy are shown in Fig. 5b and 6b. DeepIoT uses the least execution time when achieving the same testing accuracy compared with three baselines. It takes 36.7ms and 25.1ms for a single prediction, which reduces execution time by around 80.8% and 71.4% in UserID and HHAR, respectively, without loss of accuracy. DyNS and DyNS-Ext achieve better performance on time compared with SparseSep, which is different from the previous evaluations on LeNet-5. It is the structure of the original neural network that causes this difference. As shown in Table 5 and 6, the original network uses 1-d filters in its structure. The matrix factorization based kernel compressing method used in SparseSep cannot help to reduce or even increase the parameter redundancies and the number of operations involved. Therefore, there are constraints on the network structure when applying matrix factorization based compression algorithm. In addition, SparseSep cannot be applied to the recurrent layers in the network, which consumes a large proportion of operations during running the neural network.

The tradeoffs between energy consumption and testing accuracy are shown in Fig. 5c and 6c. DeepIoT is the best-performing algorithm for energy consumption. It reduces energy by around 83.3% and 72.2% in UserID and HHAR without loss of accuracy. Due to the aforementioned problem of SparseSep on 1-d filter, redundant factorization causes more execution time and energy consumption in the experiment.

6 DISCUSSION

This paper tries to apply state-of-the-art neural network models on resource-constrained embedded and mobile devices by simplifying network structure without hurting accuracy. Our solution, DeepIoT,

generates a simplified network structure by deciding which elements to drop in each layer. This whole process requires fine-tuning (Line 18 to Line 21 in Algorithm 1). However, we argue that the fine-tuning step should not be the obstacle in applying DeepIoT. First, all the compressing and fine-tuning steps are conducted on the workstation instead of embedded and mobile devices. We can easily apply the DeepIoT algorithm to compress and fine-tune the neural network ahead of time and then deploy the compressed model into embedded and mobile devices without any run-time processing. Second, the original training data must be easily accessible. Developers who want to apply neural networks to solve their own sensing problems will typically have access to their own datasets to fine-tune the model. For others, lots of large-scale datasets are available online, such as vision [11] and audio [16] data. Hence, for many categories of applications, fine-tuning is feasible.

DeepIoT mainly focuses on structure pruning or weight pruning, which is independent from other model compression methods such as weight quantization [9, 10, 18, 23]. Although weight quantization can compress the network complexity by using limited numerical precision or clustering parameters, the compression ratio of quantization is usually less than the structure pruning methods. Weight pruning and quantization are two non-conflicting methods. We can apply weight quantization after the structure pruning step or after any other compression algorithm. This is out of the scope of this paper. Also we do not exploit heterogeneous local device processors, such as DSPs [30], to speed up the running time during all the experiments, because this paper focuses on structure compression methods for deep neural networks instead of hardware speed-up.

7 CONCLUSION

In this paper, we described DeepIoT, a compression algorithm that learns a more succinct network structure for sensing applications. DeepIoT integrates the original network with dropout learning and generates stochastic hidden elements in each layer. We also described a novel compressor neural network to learn the parameter redundancies and generate dropout probabilities for original network layers. The compressor neural network is optimized jointly with the original neural network through the compressor-critic framework. DeepIoT outperforms other baseline compression algorithms by a significant margin in all experiments. The compressed structure greatly reduces the resource consumption on sensing system without hurting the performance, and makes a lot of the state-of-the-art deep neural networks deployable on resource-constrained embedded and mobile devices.

8 ACKNOWLEDGEMENTS

REFERENCES

[1] Intel edison compute module. http://www.intel.com/content/dam/support/us/en/documents/edison/sb/edison-module_HG_331189.pdf.

[2] Loading debian (ubilinux) on the edison. https://learn.sparkfun.com/tutorials/loading-debian-ubilinux-on-the-edison.

[3] M. A. Alsheikh, S. Lin, D. Niyato, and H.-P. Tan. Machine learning in wireless sensor networks: Algorithms, strategies, and applications. *IEEE Communications Surveys & Tutorials*, 16(4):1996–2018, 2014.

[4] S. Bhattacharya and N. D. Lane. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, pages 176–189. ACM, 2016.

[5] N. Brouwers, M. Zuniga, and K. Langendoen. Incremental wi-fi scanning for energy-efficient localization. In *Pervasive Computing and Communications (PerCom), 2014 IEEE International Conference on*, pages 156–162. IEEE, 2014.

[6] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on software engineering*, 29(10):929–945, 2003.

[7] E. Cho, K. Wong, O. Gnawali, M. Wicke, and L. Guibas. Inferring mobile trajectories using a network of binary proximity sensors. In *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2011 8th Annual IEEE Communications Society Conference on*, pages 188–196. IEEE, 2011.

[8] R. Clark, S. Wang, H. Wen, A. Markham, and N. Trigoni. Vinet: Visual inertial odometry as a sequence to sequence learning problem. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

[9] M. Courbariaux, Y. Bengio, and J.-P. David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

[10] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[12] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[13] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 1–14. ACM, 2012.

[14] Y. Gal and Z. Ghahramani. Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*, 2015.

[15] Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. 2016.

[16] J. F. Gemmeke, D. P. W. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.

[17] P. W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

[18] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[19] G. Goumas, K. Kourtis, N. Anastopoulos, V. Karakasis, and N. Koziris. Understanding the performance of sparse matrix-vector multiplication. In *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on*, pages 283–292. IEEE, 2008.

[20] A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *ICML*, volume 14, pages 1764–1772, 2014.

[21] S. Gu, S. Levine, I. Sutskever, and A. Mnih. Muprop: Unbiased backpropagation for stochastic neural networks. *arXiv preprint arXiv:1511.05176*, 2015.

[22] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

[23] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *ICML*, pages 1737–1746, 2015.

[24] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR, abs/1510.00149*, 2, 2015.

[25] J. Hester, T. Peters, T. Yun, R. Peterson, J. Skinner, B. Golla, K. Storer, S. Hearndon, K. Freeman, S. Lord, et al. Amulet: An energy-efficient, multi-application wearable platform. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, pages 216–229. ACM, 2016.

[26] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[27] E. Hoque, R. F. Dickerson, and J. A. Stankovic. Vocal-diary: A voice command based ground truth collection system for activity recognition. In *Proceedings of the Wireless Health 2014 on National Institutes of Health*, pages 1–6. ACM, 2014.

[28] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *NIPS*, volume 13, pages 1008–1014, 1999.

[29] B. Kusy, A. Ledeczi, and X. Koutsoukos. Tracking mobile nodes using rf doppler shifts. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 29–42. ACM, 2007.

[30] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International Conference on*, pages 1–12. IEEE, 2016.

[31] N. D. Lane, P. Georgiev, and L. Qendro. Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 283–294. ACM, 2015.

[32] A. Mnih and K. Gregor. Neural variational inference and learning in belief networks. 2014.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[34] S. Nirjon, R. F. Dickerson, Q. Li, P. Asare, J. A. Stankovic, D. Hong, B. Zhang, X. Jiang, G. Shen, and F. Zhao. Musicalheart: A hearty way of listening to music. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 43–56. ACM, 2012.

[35] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5206–5210. IEEE, 2015.

[36] J. Peters and S. Schaal. Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2219–2225. IEEE, 2006.

[37] V. Radu, N. D. Lane, S. Bhattacharya, C. Mascolo, M. K. Marina, and F. Kawsar. Towards multimodal deep learning for activity recognition on mobile devices. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 185–188. ACM, 2016.

[38] S. Rosa, X. Lu, H. Wen, and N. Trigoni. Leveraging user activities and mobile robots for semantic mapping and user localization. In *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 267–268. ACM, 2017.

[39] A. Rowe, M. Berges, and R. Rajkumar. Contactless sensing of appliance state transitions through variations in electromagnetic fields. In *Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building*, pages 19–24. ACM, 2010.

[40] A. Saifullah, M. Rahman, D. Ismail, C. Lu, R. Chandra, and J. Liu. Snow: Sensor network over white spaces. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, 2016.

[41] M. Schuss, C. A. Boano, M. Weber, and K. Roemer. A competition to push the dependability of low-power wireless protocols to the edge. In *Proceedings of the 14th International Conference on Embedded Wireless Systems and Networks (EWSN). Uppsala, Sweden*, 2017.

[42] Y. Shen, M. Hu, M. Yang, B. Wei, S. Lucey, and C. T. Chou. Face recognition on smartphones via optimised sparse representation classification. In *Information Processing in Sensor Networks, IPSN-14 Proceedings of the 13th International Symposium on*, pages 237–248. IEEE, 2014.

[43] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[44] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[45] A. Stisen, H. Blunck, S. Bhattacharya, T. S. Prentow, M. B. Kjærgaard, A. Dey, T. Sonne, and M. M. Jensen. Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 127–140. ACM, 2015.

[46] Z. Sun, A. Purohit, K. Chen, S. Pan, T. Pering, and P. Zhang. Pandaa: physical arrangement detection of networked devices through ambient-sound awareness. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 425–434. ACM, 2011.

[47] C. Tai, T. Xiao, Y. Zhang, X. Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

[48] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[49] S. Wang, S. M. Kim, Y. Liu, G. Tan, and T. He. Corlayer: A transparent link correlation layer for energy efficient broadcast. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 51–62. ACM, 2013.

[50] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari. Deep reinforcement learning for dynamic multichannel access. 2017.

[51] Y. Wang, C. Xu, S. You, D. Tao, and C. Xu. Cnnpack: packing convolutional neural networks in the frequency domain. In *Advances in Neural Information Processing Systems*, pages 253–261, 2016.

[52] H. Wen, S. Wang, R. Clark, and N. Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. *International Conference on Robotics and Automation*, 2017.

[53] J. Wilson and N. Patwari. See-through walls: Motion tracking using variance-based radio tomography networks. *IEEE Transactions on Mobile Computing*, 10(5):612–621, 2011.

[54] J. Yang, S. Sidhom, G. Chandrasekaran, T. Vu, H. Liu, N. Cecan, Y. Chen, M. Gruteser, and R. P. Martin. Detecting driver phone use leveraging car speakers. In *Proceedings of the 17th annual international conference on Mobile computing and networking*, pages 97–108. ACM, 2011.

[55] S. Yao, M. T. Amin, L. Su, S. Hu, S. Li, S. Wang, Y. Zhao, T. Abdelzaher, L. Kaplan, C. Aggarwal, et al. Recursive ground truth estimator for social data streams. In *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International*

Conference on, pages 1–12. IEEE, 2016.

[56] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher. Deepsense: a unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017.

[57] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[58] C. Zhang, K. Zhang, Q. Yuan, H. Peng, Y. Zheng, T. Hanratty, S. Wang, and J. Han. Regions, periods, activities: Uncovering urban dynamics via cross-modal representation learning. In *Proceedings of the 26th International Conference on World Wide Web*, pages 361–370. International World Wide Web Conferences Steering Committee, 2017.

[59] M. Zhang, A. Joshi, R. Kadmawala, K. Dantu, S. Poduri, and G. S. Sukhatme. Ocrdroid: A framework to digitize text using mobile phones. In *International Conference on Mobile Computing, Applications, and Services*, pages 273–292. Springer, 2009.