Fast and Scalable In-memory Deep Multitask Learning via Neural Weight Virtualization

Seulki Lee
University of North Carolina at Chapel Hill
seulki@cs.unc.edu

Shahriar Nirjon
University of North Carolina at Chapel Hill
nirjon@cs.unc.edu

ABSTRACT

This paper introduces the concept of *Neural Weight Virtualization* – which enables fast and scalable in-memory multitask deep learning on memory-constrained embedded systems. The goal of neural weight virtualization is two-fold: (1) packing multiple DNNs into a fixed-sized main memory whose combined memory requirement is larger than the main memory, and (2) enabling fast in-memory execution of the DNNs. To this end, we propose a two-phase approach: (1) *virtualization of weight parameters* for fine-grained parameter sharing at the level of weights that scales up to multiple heterogeneous DNNs of arbitrary network architectures, and (2) in-memory data structure and run-time execution framework for *in-memory execution and context-switching* of DNN tasks. We implement two multitask learning systems: (1) an embedded GPU-based mobile robot, and (2) a microcontroller-based IoT device. We thoroughly evaluate the proposed algorithms as well as the two systems that involve ten state-of-the-art DNNs. Our evaluation shows that weight virtualization improves memory efficiency, execution time, and energy efficiency of the multitask learning systems by 4.1x, 36.9x, and 4.2x, respectively.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Computer systems organization** → **Embedded software**.

KEYWORDS

Deep neural network, Virtualization, Multitask learning, In-memory

ACM Reference Format:
Seulki Lee and Shahriar Nirjon. 2020. Fast and Scalable In-memory Deep Multitask Learning via Neural Weight Virtualization. In *The 18th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '20), June 15–19, 2020, Toronto, ON, Canada.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3386901.3388947

1 INTRODUCTION

As deep learning algorithms for resource-constrained systems continue to become more efficient and more accurate [166–168], we see an increased number of intelligent embedded systems such as home IoT devices, social robots, and the plethora of portable, wearable, and mobile devices that are feature-packed with a wide variety of machine learning tasks running on the same device [9, 14, 71, 99, 112, 148]. Home hubs like Amazon Echo Show and Google Nest Hub nowadays are performing speech recognition [47, 72], speaker identification [50], gesture recognition [46], face recognition [48], facial expression and emotion recognition [45] in order to closely imitate human assistants. Similar classifiers are running on social, domestic, and personal robots [15, 40, 120, 133, 141], which also execute robotic application-specific learning tasks such as object recognition [101, 122], obstacle detection [163], scene understanding [91], self-localization [130], and navigation [16, 41]. While a naive approach to enable multiple classifiers on a device would be to train and execute each classifier independently, state-of-the-art multitask learning approaches suggest jointly training more than one correlated task in order to increase the accuracy of each learner by exploiting the commonalities and differences across different tasks [19, 125, 174, 175].

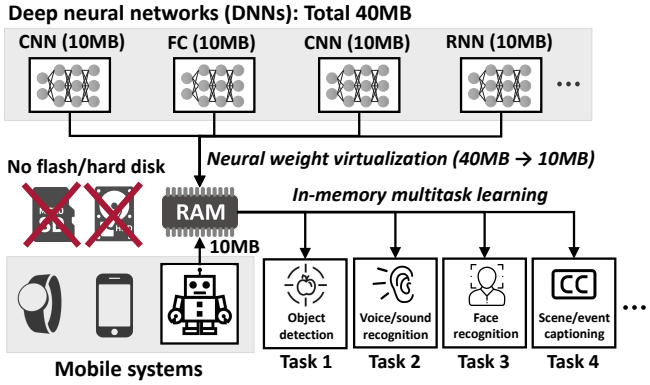


Figure 1: Neural weight virtualization packs multiple DNNs into the main memory of a system where the total size of the DNNs is larger than the capacity of the main memory (40MB vs. 10MB). It performs complete *in-memory* storage and execution of DNNs at run-time, which enables fast and real-time multitask learning on resource-constrained systems.

Unfortunately, multitask deep learning on mobile and embedded systems is not quite as effective as running the classifiers on a high-end machine due to their limited CPU and memory. Using powerful processors and/or larger (or external) memory is not feasible in these systems due to cost, space, heating, latency, and design constraints [62]. In general, lack of *scalability* and sluggish *response time* are the two major challenges to effective deep multitask learning on CPU and memory-constrained embedded systems:

• *Scalable Packing.* State-of-the-art DNNs require between hundreds of KB to hundreds of MB of main memory [57, 136, 146]. On the other hand, state-of-the-art embedded GPUs and low-power SoCs and MCUs typically contain 8KB–512MB of RAM [58, 64, 149]. Hence, the maximum number of DNNs that can reside in the main

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MobiSys '20, June 15–19, 2020, Toronto, ON, Canada
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7954-0/20/06…$15.00
https://doi.org/10.1145/3386901.3388947

memory is quite limited. Commonsense approaches such as compression and pruning [2, 54, 56, 59, 113, 153] DNNs do not quite solve the problem since these algorithms are applied on each DNN separately, they require significant fine-tuning, and more importantly, as opposed to multitask learning, these networks do not benefit from knowledge transfer as they are trained in isolation. Although by sharing network structure (typically, the first few layers), multitask learning achieves limited compression [60], its primary goal is to increase robustness and generalization of *correlated* and *similar-structured* learners. Thus, packing multiple *heterogeneous* learners into extremely scarce memory of an embedded system still remains an open problem.

• *Low-latency Context Switching.* A practical limitation of multitask learning *systems* in the wild – which is often overlooked by the main-stream deep multitask learning literature – is the overhead of switching DNN tasks at run-time. In memory-constrained multitask learning systems, where some of the DNN models must reside in the flash or the hard disk, context switching overhead is extremely high as memory operations are typically 10–100x faster than accessing flash or hard disks, and DNN models are large. Hence, the overhead of frequent swapping in and out of DNNs to and from the main memory causes severe latency, which in turn, degrades the responsiveness and usability of the system.

To address these challenges, we introduce the concept of *Neural Weight Virtualization* – which treats *consecutive memory locations containing weights of neural networks* as resources that can be virtualized, and thus, shared by more than one DNN. Weight virtualization enables scalable packing of *any*[1] number of DNNs into the main memory while achieving the fastest possible deep multitask learning on an embedded system that incurs near-zero context switching overhead due to complete in-memory storage and execution. An illustration of weight virtualization is shown in Figure 1, where four DNNs, requiring a total of 40MB memory, are packed into 10MB RAM of a mobile system.

Packing multiple DNNs into the main memory via weight virtualization is motivated by empirical observations that *only a small fraction of DNN weights have significant impacts on the inference result*, and *these high-significance weights are concentrated in a few blocks in the main memory.* This guides us in designing a scalable DNN packing algorithm that matches similar blocks of weights across multiple DNNs and combines them to construct a single new block of *virtual* weights that is shared by the DNNs. The matching process is followed by an optimization (retraining) process to gain back any loss of the inference accuracy of the DNNs. Efficient in-memory data structures and run-time framework for task management, execution, scheduling, and context-switching have been implemented to support scalable and fast in-memory deep multitask learning on embedded systems. The approach is architecture agnostic – it applies to any type of DNN, including fully-connected, convolutional, and recurrent layers.

We implement two deep multitask learning systems involving ten state-of-the-art DNNs: (1) a mobile robot that executes six DNNs (i.e., MobileNet [65, 129], FaceNet [132], Place205 [182], UrbanSound8K [128], GSC [158], and ShowAndTell [155]) on Jetson Nano embedded GPU platform [110]), and (2) an extremely resource-constrained IoT device having an MSP430 [149] microcontroller that executes five compressed DNNs (i.e., GTSRB [144], GSC [158], SVHN [108], MNIST [79], and CIFAR-10 [75]). Our experimental results demonstrate that with weight virtualization, these systems successfully packs all DNNs with a 4x compression ratio while achieving an improved latency of 36x and an improved energy efficiency of 4.34x due to in-memory operations and multitask learning. We have made our software open-source at a public repository[2].

[1] Although the proposed approach can pack an unlimited number of DNNs, ensuring a minimum accuracy for each DNN does impose a limit on this number–which is still larger than alternatives such as [8, 29, 60, 69, 85, 97, 98, 100, 103].

2 OVERVIEW

The general idea of virtualization technology is to create the illusion of having an extended resource given a limited physical resource – such as hardware, storage, and networks [31, 118, 139]. We formulate the *in-memory multitask learning problem* into a virtualization problem where multiple DNNs must reside in the same fixed-sized RAM. We assume that the main memory is at least as large as the largest DNN, but we do not impose any limit on the total memory required by all DNNs. We further assume that the DNNs are already trained, and they can be of any arbitrary network structures.

Our approach to in-memory deep multitask learning is to virtualize a portion of the main memory, which stores a carefully generated set of constant numbers that represent the weight parameters of one or more DNNs. We call this *weight virtualization* as opposed to virtualization of main memory since the memory locations, along with their content (offline-computed fixed numbers representing DNN weights) are virtualized. For example, a memory block, B_0, may simultaneously represent K consecutive weights of the L_i^{th} layer of the first DNN as well as K consecutive weights of the L_j^{th} layer of another DNN. Weight virtualization requires us to find a set of values to be stored in the main memory such that (1) each block of memory represents a block of weights of one or more DNNs, and (2) significant weights (if not all) of all DNNs are mapped to a weight page.

2.1 Feasibility of Weight Virtualization

Observations. Enabling weight virtualization to pack multiple DNNs in the main memory is motivated by two key observations on the *significance* of weight parameters towards the classification result of a DNN. While studying 13 popular DNNs used in state-of-the-art audio and visual learning tasks, we observe the following:

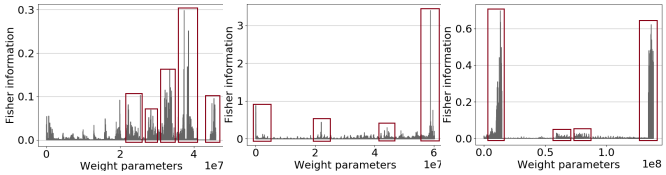
• *Disparity in weights' significance is globally sparse.* It is known that the significance (aka sensitivity or importance) of different weights of a neural network toward the classification result is different [54, 80]. When the significance is quantified (e.g., using Fisher information [84]), we observe a large disparity — only a fraction of the weight parameters' significance is markedly higher than the rest. This happens primarily due to the inherent redundancy in most state-of-the-art DNN models [27, 54]. This observation hints us that only a small fraction of high-significance weights per DNN must be stored unaltered in the main memory, while the rest can be altered and stored if there is room.

• *Disparity in weights' significance is locally dense.* Although only a fraction of the weight parameters is of high significance, they tend to be located in the vicinity of each other. In other words,

[2] https://github.com/learning1234embed/NeuralWeightVirtualization

when a DNN's weights are stored in the computer memory, a high-significance weight is more likely to be located next to another high-significance weight, and the same is true for low-significance weights. This is analogous to human brains, where only the neurons from a specific locality get activated by certain stimuli or tasks. This observation hints us that when sharing weights among multiple DNNs, we can consider an entire block of memory for possible sharing (which we call a *weight-page*), as opposed to bookkeeping each memory cell individually, and thus, expedite the memory sharing and management process.

Empirical Evidence. Of the 13 DNNs we studied, Figure 2 shows the significance score (Fisher information) of individual weights of three popular DNNs[3] on ImageNet classification [35], i.e.,— Inception-v4 [146], ResNet-152 [57], and VGG-16 [136]. The weights are listed in the order of layers, i.e., the weights of the first layer are followed by the second layer's, and so on.



(a) Inception-v4 (b) ResNet-152 (c) VGG-16

Figure 2: The importance of individual weight parameter to the inference accuracy measured by Fisher information [84]: Inception-v4 [146], ResNet-152 [57], and VGG-16 [136].

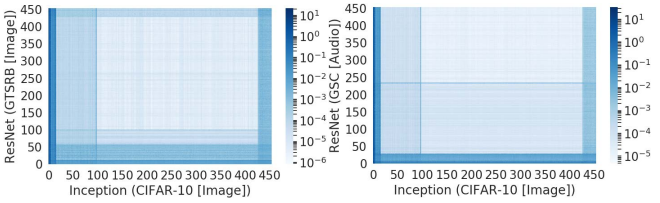
As shown in the figure, a few groups of neighboring weights (marked with red boxes) dominate in terms of their significance. For instance, the top 5% weights contribute to 90% Fisher score in Inception-v4, i.e., 95% of its memory space can be used by other DNNs if need be while retaining 90% accuracy of Inception-v4. Table 1 shows the percentages of weights of the 13 DNNs required to attain certain amounts of significance score.

DNN	Dataset	≥70%	≥80%	≥90%	≥95%	≥99%
Inception-v4 [146]	**ImageNet** [35]	1.0%	2.1%	5.0%	9.3%	23.1%
Inception-ResNet-v2 [146]	**ImageNet** [35]	0.06%	0.19%	0.9%	3.2%	14.9%
RestNet-152 [57]	**ImageNet** [35]	2.5%	5.0%	11.1%	18.9%	38.5%
VGG-16 [136]	**ImageNet** [35]	0.4%	0.7%	1.7%	2.8%	5.8%
PNASNet-5 [93]	**ImageNet** [35]	0.1%	0.3%	1.4%	4.7%	20.2%
MobileNet-v2 [129]	**ImageNet** [35]	0.3%	0.5%	0.8%	1.2%	2.0%
AlexNet [76]	**CIFAR-10** [75]	0.1%	0.4%	3.0%	8.9%	29.6%
GoogleNet [156]	**Place205** [182]	3.3%	5.8%	11.1%	17.6%	35.9%
FaceNet [132]	**VGGFace2** [18]	1.9%	4.7%	10.5%	16.6%	30.3%
ShowAndTell [155]	**MS COCO** [92]	0.1%	0.8%	4.4%	10.7%	28.5%
KWS [126]	**GSC** [126]	0.01%	0.16%	3.1%	12.3%	47.9%
LeNet-5 [79]	**MNIST** [79]	3.9%	6.0%	9.7%	13.3%	20.6%
Boosted-LeNet-4 [81]	**GTSRB** [144]	9.5%	14.8%	25.4%	36.9%	61.4%

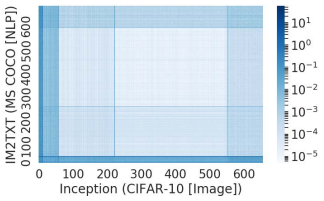
Table 1: The percentage of weight parameters required to attain different percentages of total significance (Fisher information [84]) in the 13 state-of-the-art DNNs; e.g., 5.0% of weight parameters are required to attain 90% of the total significance score in Inception-v4.

Weight Virtualization Landscape. Using three DNNs (Table 2 – Upper) and five datasets (Table 2 – Lower), we conduct an experiment where we compare the similarity of memory blocks of different pairs of DNNs. In Figure 3, we show three representative cases out of all $\binom{13}{2}$ pairs. In each figure, the darkness of the coordinate (x, y) represents the dissimilarity of memory block x of a DNN (X-axis) and the memory block y of another DNN (Y-axis).

[3]The remaining DNNs follow the same trend and are used in the evaluation section.



(a) Inception(CIFAR10)/ResNet(GTSRB) (b) Inception(CIFAR10)/ResNet(GSC)



(c) Inception(CIFAR10)/IM2TXT(MS)

DNN	Network Architecture
Inception	Convonlution+Residual [146]
ResNet	Convonlution+Residual [57]
IM2TXT	Convonlution+Recurrent [155]

Task	Dataset
Image	CIFAR-10 [75], GTSRB [144], MNIST [79]
Audio	GSC [126]
NLP	MS COCO [92] (Image caption)

Table 2: The DNN architecture and dataset.

Figure 3: The disparity in weights in the memory blocks of two DNNs having (a) similar architecture and similar tasks, (b) similar architecture but different tasks, and (c) different architecture and different tasks.

The dissimilarity score is computed using Equation 1 (explained later in Section 3), where a higher score or a darker dot represents larger differences between the memory blocks.

Figures 3a and 3b show that (1) DNNs of similar architectures, e.g., Inception and ResNet, tend to have similar memory blocks across the entire network except for the very beginning and the end, and (2) the number of similar blocks decreases when two DNNs perform different tasks. As shown in the figure, the area having paler color is smaller when the DNNs perform different tasks, i.e., image vs. audio classification (Figure 3b) than the case when they perform similar tasks, i.e., two image recognition tasks (Figure 3a).

Figure 3c shows that DNNs of different architectures, e.g., Inception and IM2TXT, performing different tasks (image vs. NLP) tend to have memory blocks containing similar weights at specific and limited areas due to their architectural differences, i.e., convolutional (and residual) vs. recurrent structure.

2.2 Weight Virtualization Framework

We propose a two-phase weight virtualization framework (Figure 4) having an offline and an online phase.

Weight Virtualization (Offline). Weight virtualization is divided into three steps. First, for each DNN, its weight parameters are split into fixed-sized weight-pages - which is the basic unit of the weight virtualization. In Figure 4, the page size is 100, and each DNN has up to four weight-pages as the main memory size is 400. Second, the weight-pages of all DNNs are matched, and similar weight-pages are grouped together. The matching process minimizes a cost function (defined later by Equation 1) that reduces the performance loss caused by weight sharing. Figure 4 shows four groups of weight-pages. Third, the matched weight-pages in each group are combined to create a single *virtual* weight-page by optimizing (re-training) all or some of the DNNs. The goal of this optimization is to retain the accuracy of each DNN that shares one or more virtual weight-pages with others. Their algorithmic details are described in Section 3.

In-Memory Execution (Online). The virtual weight-pages are loaded into the main memory of the system. For each DNN, a page

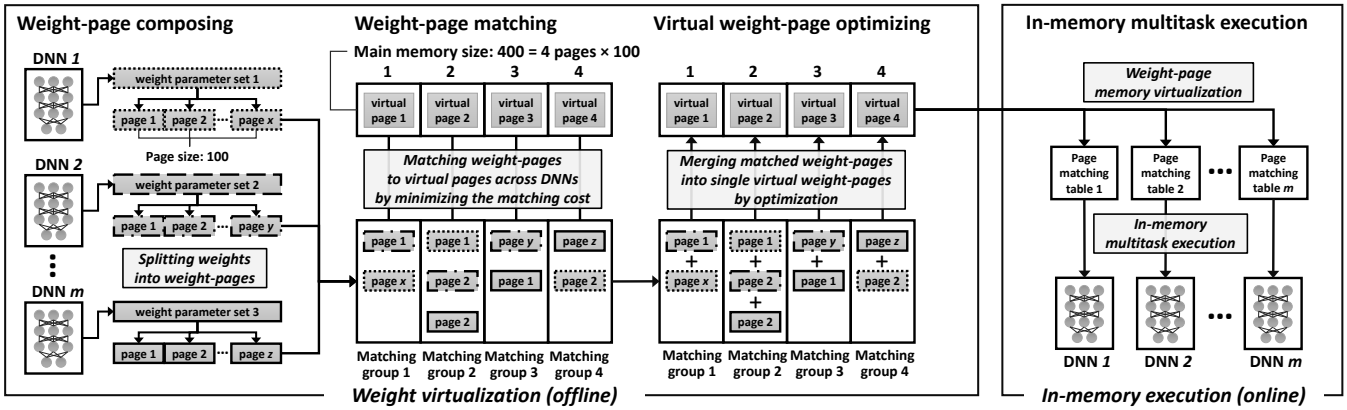


Figure 4: The proposed weight virtualization framework has two phases: (1) An offline weight virtualization phase, consisting of weight-page composing, matching, and optimizing steps, and (2) an online in-memory execution phase.

matching table that points a set of memory addresses of virtual weight-pages required by the DNN is generated. At run-time, a DNN is executed in the main memory with the necessary weight parameters obtained by dereferencing its page matching table. Further details of this phase are in Section 4.

2.3 Benefits of Weight Virtualization

Improved Multitask Learning. Unlike existing works on multitask learning [19, 96, 125, 165, 183] which are limited to similar network architectures and related tasks [19], neural weight virtualization allows multitask learning involving heterogeneous tasks that may have completely different network architectures. Because the structure sharing happens at the granularity of weights, weight virtualization can pack any types of networks, including fully-connected (FC), convolutional (CNN), and recurrent neural networks (RNN). By isolating the memory blocks used for weights from the network structure (graph), weight virtualization preserves the original network structure of the input DNNs – which is not supported by the state-of-the-art [125, 175]. This latter aspect is particularly important since nowadays there are many DNN models available for use, and neural weight virtualization is a practical way to integrate these DNNs to a system without requiring any structural modifications to them [30].

Efficient Parameter Representation. Virtual weight-pages not only reduce the amount of information required to represent the weight parameters of a DNN but also enables efficient unpacking and execution of DNNs at run-time. For example, to represent 1,000,000 weights using virtual weight-pages of size 100, only 10,000 page-pointers are required, which is only 1% of the original size of the weights (10,000 vs. 1,000,000). The larger the page-size, the less information is needed to represent the same amount of weight parameters. Hence, by suitably choosing a page-size, we can optimize the efficiency of parameter representation.

Efficient Task Management. In-memory multitask learning enables efficient management and scheduling of DNNs. Since all DNNs reside in the main memory, tasks can be executed and switched in real-time, enabling fast and responsive execution of multiple DNNs. By eliminating the need for repetitive resource allocation, e.g., loading DNNs from the flash or the hard disk, task management such as context switching and scheduling becomes simpler.

3 WEIGHT VIRTUALIZATION

Weight virtualization is a three-step process consisting of weight-page composing, matching, and optimizing, which collectively virtualize (combine) the weight parameters of DNNs to virtual weights that fit into the main memory. This is an offline process.

3.1 Weight-Page Composing

Weight-Page Composing. The first step of weight virtualization is to compose weight-pages for each DNN. Given a set of pre-trained DNN tasks, τ_i for $1 \leq i \leq m$, where m is the number of tasks, we define a *weight-page* as a sequence of s weights stored in consecutive memory locations of τ_i, where s denotes the page-size. Note that although the weights of a DNN are mathematically expressed as matrices when stored in the computer memory, they reside in a linear address space. Thus, weight-pages represent fixed-sized logical partitions of the portion of the main memory that contains the weight matrices. We define a *weight-page set*, P_i as all the weight-pages of DNN τ_i. Figure 5 illustrates the weight-page composition for a fully-connected layer. The process is similar for other architectures, such as convolutional and recurrent layers.

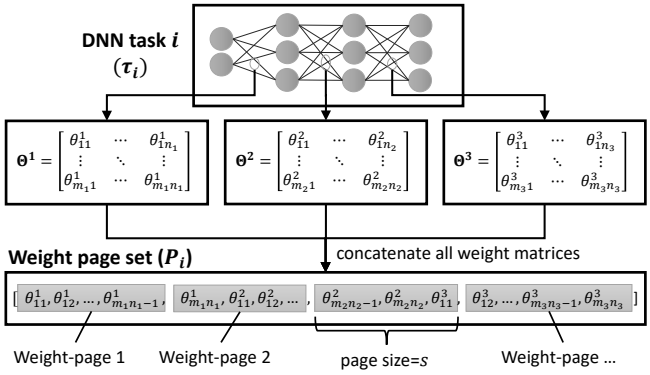


Figure 5: The weight-page set P_i for DNN τ_i is composed by segmenting s consecutive weights in memory. The matrix, Θ is an arbitrary weight or bias matrix, where θ_{ij} is an individual weight parameter.

Weight-Page-based Virtualization. We virtualize weights at the granularity of weight-pages for two reasons. First, it retains the functional and spatial locality of a DNN inside a weight-page, which increases the reusability of common locality patterns between DNNs.

Second, it enables efficient and flexible management of weight parameters when sharing and accessing them. A DNN task can easily reconstruct its weight parameters by locating weight-pages in the main memory, which is much more efficient in terms of memory and computation than finding individual weights one by one, considering the massive number of weight parameters in many state-of-the-art DNNs (e.g., ResNet-50 [57] and VGG-16 [136] have 26 and 138 million weights, respectively).

3.2 Weight-Page Matching

The next step of weight virtualization is to match each page within the weight-page sets of the DNNs to each page of the *virtual weight-page set*, P_0 where they are eventually merged. For multitask learning scenarios having more than two tasks, we keep a task, τ_0 as the final combined task and P_0 as the final virtual weight-page set, to which, $\tau_i(P_i)$'s are merged iteratively one by one, for $i > 0$. Hence, our goal is to find the best match between the virtual weight-page set P_0 of τ_0 and weight-page set P_i of the newly-added task τ_i – based on a one-to-one matching of each page.

Matching Objective. Given two sets of weight-pages, P_i and P_0, and a weight-page matching cost function, $C : P_i \times P_0 \rightarrow \mathbb{R}$, which is defined by Equation 1, the goal of the matching step is to find an injective mapping function $f_i : P_i \rightarrow P_0$ such that $\sum_{p \in P_i} C(p, f_i(p))$ is minimized, i.e., finding the closest set of $(p, f_i(p))$ pairs. Figure 6 illustrates a matching between task weight-page set P_i and the virtual weight-page set P_0.

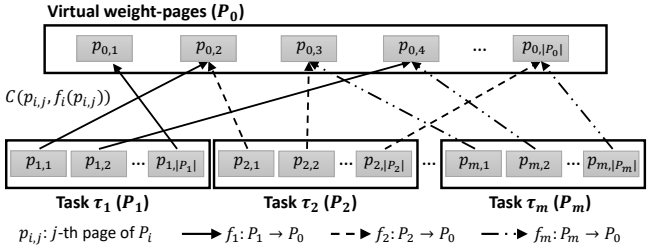


Figure 6: Each weight-page of task $\tau_i(P_i)$ is matched to a subset of the virtual weight-page set P_0. Each matching edge (arrows in the figure) incurs a matching cost of $C(p_{i,j}, f_i(p_{i,j}))$ between the matched weight-pages, whose summation is minimized.

Matching Cost Function. The *weight-page matching cost function* between two pages p and q is given by:

$$C(p,q) = \kappa \sum_{(\theta_p \in p, \theta_q \in q)} (\theta_p - \theta_q)^2 (\tilde{F}(\theta_p|\tau_p) + \tilde{F}(\theta_q|\tau_q)) \quad (1)$$

where κ is the matching regularizer, θ_p is a weight in p, θ_q is a weight in q, τ_p is the task of p, τ_q is the task of q, and $\tilde{F}(\theta_p|\tau_p)$ and $\tilde{F}(\theta_q|\tau_q)$ are Fisher information [84] of θ_p and θ_q, respectively (defined by Equation 2). The matching cost is minimized when two weights become similar, and the summation of Fisher information gets smaller. The matching cost function in Equation 1 primarily encourages virtualization of weights with similar values so that weight-pages of similar patterns can be combined together. However, at the same time, the summation of Fisher information works as a regularizer that discourages too much concentration of highly-important weight parameters on a few weight-pages in order to ensure the performance of virtualized DNNs by distributing the overall importance of weights over the entire memory.

Fisher Information. We use *Fisher Information* [84] in the matching cost function (Equation 1) since the difference of weights by itself does not carry the information on how impactful the change is to the final outputs of tasks [37, 73]. The Fisher information of weight, θ in $\tau_i(P_i)$ given data $X = \{ x_1, x_2, ..., x_n \}$ is defined by:

$$\tilde{F}(\theta|\tau_i) = \frac{1}{n} \sum_{j=1}^{n} \left(\frac{\partial}{\partial \theta} \log f_i(x_j|\theta) \right)^2 \quad (2)$$

where $f_i(x_j|\theta)$ is τ_i's probability density, i.e., the likelihood of data x_j conditioned on θ. Three key properties of Fisher score make it suitable for estimating the significance of a weight parameter [73, 117, 152]: first, it is equivalent to the second derivative of the loss near a minimum, second, it can be easily computed from the first-order derivatives alone and is thus easy to calculate for large models, and third, it is guaranteed to be positive semi-definite.

Weight-Page Matching Problem. Given m DNN tasks, the weight-page matching is formulated as a combinatorial optimization problem, called the assignment problem [131] aka maximum weighted bipartite matching problem [160], as follows:

$$\begin{aligned} \min \sum_{i=1}^{m} \sum_{(p,q) \in P_i \times P_0} C(p,q) \cdot x_{ipq} \\ \text{s.t.} \sum_{p \in P_i} x_{ipq} = 1 \quad 1 \le i \le m \text{ and } q \in P_0 \\ \sum_{q \in P_0} x_{ipq} = 1 \quad 1 \le i \le m \text{ and } p \in P_i \\ x_{ipq} \in \{0,1\} \quad 1 \le i \le m, p \in P_i \text{ and } q \in P_0 \end{aligned} \quad (3)$$

where, P_i is the weight-page set of task τ_i, P_0 is the weight-page set for the combined task τ_0, and $C(p,q)$ is the weight-page matching cost function between weight-page p and q in Equation 1. The variable x_{ipq} is one, if page p and q are matched, and zero, otherwise. Although $O(n^4)$ polynomial-time optimal algorithm [107] exists for two tasks, where n is the number of weight-pages, for a large number of weight-pages, the computational overhead of such matching is too high. For more than two tasks, the problem becomes a multiple-choice multiple-assignment problem [24, 140] of $O(n^{nm})$ complexity, which is computationally intractable.

Greedy Matching Algorithm. To solve the weight-page matching problem in Equation 3 which is computationally challenging, we propose an *iterative greedy matching algorithm* that starts with an unmatched weight-page, $p \in P_i$ having the largest Fisher information (i.e., weights that impact the final outputs more) and greedily finds an unmatched page, $q \in P_0$ that minimizes the matching cost $C(p,q)$ in Equation 1. It is based on the observation that Fisher information of most weights has near-zero values except only a few having significant magnitudes. Algorithm 1 describes the proposed greedy weight-page matching algorithm. For m tasks having n weight-pages each, it has $O(mn^2/p)$ of computational complexity, when computing weight-page matching cost in p-way parallel (line 10–15) using a GPU.

3.3 Weight-Page Optimization

The last step of weight virtualization is to combine the matched weight-pages into single pages and optimize (retrain) the tasks to retain any accuracy loss due to the changed weight parameters.

Algorithm 1: Greedy Weight-Page Matching

Input: Weight-page set P_i for $0 \le i \le m$
Fisher information set F_i for $0 < i \le m$
Output: $f_i : P_i \rightarrow P_0$ for $1 \le i \le m$

$(c_k)_{k=1}^{|P_0|} = (0, ...)$, $(u_k)_{k=1}^{|P_0|} := (P_0)$, $(s_k) = (v_k) = ()$, $n := 0$;
for $i \leftarrow 1$ **to** m **do**
  forall $(f, p) \in (F_i, P_i)$ **do**
    $n := n + 1$, $s_n := ||f||_1$, $v_n := p$;
  end
end
sort $(v_k)_{k=1}^{n}$ in descending order of $(s_k)_{k=1}^{n}$;
for $i \leftarrow 1$ **to** n **do**
  $(c'_i)_{k=1}^{|P_0|} := (c_i)_{k=1}^{|P_0|}$, $c_{\min} := \infty$, $\text{idx}_{\min} := 0$;
  for $j \leftarrow 1$ **to** $|P_0|$ **do in parallel**
    $c_j := c_j + C(v_i, u_j)$;
    if $c_j < c_{\min}$ **then**
      $c_{\min} := c_j$, $\text{idx}_{\min} := j$;
    end
  end
  $c_{\text{idx}_{\min}} := c_{\min}$;
  $t :=$ task number of v_i, $f_t(v_i) := u_{\text{idx}_{\min}}$;
end
return f_i *for* $1 \le i \le m$

Virtual Weight-Page. Once the task weight-pages are matched, they are combined to obtain single pages called *virtual weight-pages* by optimizing (retraining) all or some of the tasks. To retain the accuracy of tasks, we minimize the side-effects of combining on the set of to-be combined weight-pages $Q_i = \bigcup_{p \in P_i} Q_{i,p}$ when optimizing task τ_i, which is given by:

$$Q_{i,p} = \{ q \in \bigcup_{j=1, j \neq i}^{m} P_j \mid \exists f_j^{-1}(f_i(p)) \} \quad (4)$$

where, p is the page in the weight-page set P_i of τ_i, m is the number of tasks, P_j is the weight-page set of τ_j, f_j^{-1} is the inverse page mapping function of τ_j, and f_i is the page mapping function of τ_i obtained from the weight-page matching step in Section 3.2.

Optimization. Task τ_i is optimized by minimizing the summation of (1) the original loss function of τ_i, and (2) the total weight-page matching cost of Q_i in Equation 4, which is given by:

$$\mathcal{L}(\tau_i) = \mathcal{L}_o(\tau_i) + \kappa \sum_{p \in P_i} \frac{1}{|Q_{i,p}|} \sum_{q \in Q_{i,p}} \sum_{\theta \in q} (\theta - \theta^*)^2 \tilde{F}(\theta | \tau_q) \quad (5)$$

where $\mathcal{L}_o(\tau_i)$ is the original loss of τ_i, κ is the matching regularizer, Q_i is the weight-pages matched to the page $f_i(P_i)$ defined in Equation 4, θ is the weight in the page q, θ^* is the virtual weight in $f_i(p)$ being optimized and shared between tasks, τ_q is the task having page q, and $\tilde{F}(\theta|\tau_q)$ is the Fisher information of θ in τ_q. By jointly optimizing tasks with the additional loss on weight-page matching, which is expressed by the second term in Equation 5, it tries to find common local minima for the desired performance of all tasks. Even when the tasks are sequentially optimized one-by-one, the virtual weights tend to stay in a low-risk region, maintaining the performance of the previously optimized tasks [73].

Flexible Weight Sharing. Weight virtualization enables flexible weight sharing and achieves better performance than existing works such as continual learning [73, 114, 170] that tries to solve the catastrophic forgetting problem [39, 44]. While existing works can only share a fixed combination of weights in the same networks, weight virtualization allows weight sharing on arbitrary combinations of weights between tasks of different architectures. Furthermore, the accuracy of DNNs with weight virtualization is expected to be higher since the weight-pages are first matched to minimize the matching cost before optimization (retraining). Thus, similar weight-pages are more likely to be shared – which improves the performance of weight sharing between tasks.

4 IN-MEMORY EXECUTION

Weight virtualization enables fast, in-memory execution of multiple DNNs at run-time by efficiently accessing the virtualized weights supported by an efficient memory model and data structure.

4.1 Memory Model and Data Structure

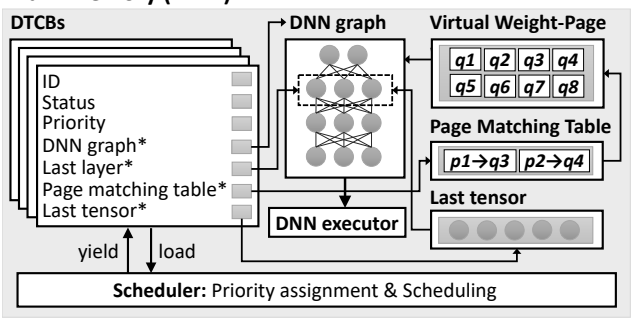


Figure 7: DNN Task memory map illustrating DTCB, DNN graph, DNN executor, virtual weight-pages, and page matching table. Virtual weight-pages and DNN executor are shared among tasks, while the other elements are exclusive to each task.

Memory Model. Figure 7 shows the memory model that supports neural weight virtualization, consisting of *DTCB* (DNN Task Control Block), *DNN graph, DNN executor, virtual weight-pages*, and *page matching table*. Unlike conventional multitasking models [154] that allocate private memory spaces to individual tasks, it allows memory overlapping between tasks for weight sharing via virtual weight-pages. Also, unlike traditional virtual memory [151] that expands to second-level memory, i.e., swapping DNNs with a flash or hard disk, it implements the entire memory hierarchy into the main memory since all DNN tasks fit into it.

DTCB. Similar to the Process Control Blocks (PCB) in modern operating systems [135], each DNN task is managed by in-memory *DNN Task Control Block (DTCB)* that is updated by the system at run-time. Each DTCB corresponds to a DNN task which contains the necessary task information, i.e., ID, status (e.g., running, suspended), priority (for scheduling), and pointers to DNN graph, page matching table, last layer, and last tensor as shown in the left side of Figure 7.

Virtual Weight-Pages. The *virtual weight-pages* generated by the weight virtualization in Section 3 are located in the main memory. They are shared by the DNN tasks. A task accesses to a subset of the virtual weight-pages by using its page matching table that performs the weight-page translation.

Page Matching Table. For each DNN task, a *page matching table* that translates the weight-pages of the DNN to the virtual weight-pages based on the matching result of Section 3.2 is provided. It consists of (1) a mapping list between weight-pages of the DNN and the virtual weight-pages, and (2) memory addresses of the matched virtual weight-pages. Figure 8 illustrates the access process of virtual weight-pages via a page matching table where a task locates the memory addresses of virtual weight-pages matched to its weight-pages.

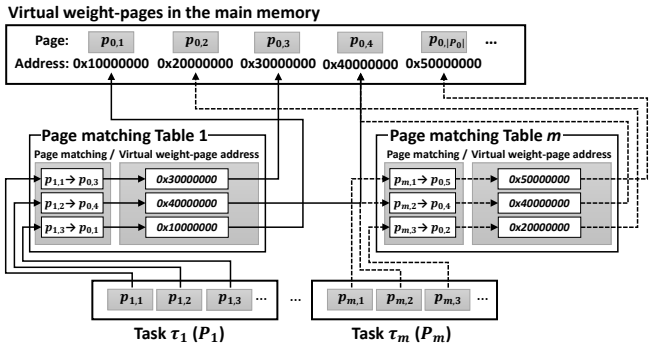


Figure 8: A page matching table translates weight-pages of a DNN task into virtual weight-pages in the memory.

DNN Graph and Executor. Each task has metadata called the *DNN graph* that defines its network architecture. The *DNN executor* is a common module, shared between all tasks, which executes a part or entire DNN graph given input tensors and weights.

4.2 Task Execution

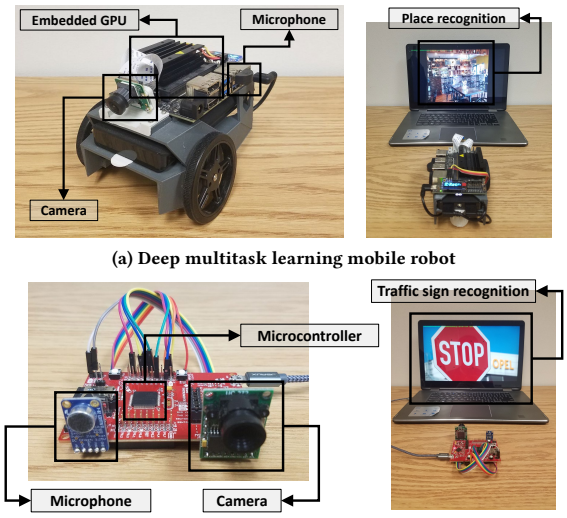
Task Execution. For the in-memory execution of a task, a task first unfolds its DNN graph and locates the last layer that was completed at the previous execution using its DTCB. Then, the last tensor values (also saved at the previous execution) and the necessary weight parameter values obtained via page matching table are passed to the DNN executor for execution. To improve memory efficiency, only the unfinished part of the graph from the last execution is unfolded and executed. It also allows fitting large DNNs into limited memory by unfolding and running a part of the graph in stages when the entire DNN does not fit into the memory.

Task Scheduling and Switching. DNN tasks are scheduled after every execution of a tensor. After executing a tensor of a task, the DNN executor gives control back to the scheduler that selects the next task based on its scheduling policy and task priorities. When a task is context-switched, the system saves the last layer of the current task completed by the DNN executor, including the corresponding tensor that is passed to the DNN executor when the task is scheduled to execute again. It allows DNN tasks to be executed by various schedulers, e.g., non-preemptive scheduling such as cooperative multitasking [10] or preemptive scheduling [134] that requires back-and-forth execution of tasks. In this paper, we use cooperative multitasking.

In-Memory Execution. Task execution and DNN context-switches – all happening in the main memory – improves the response time and the end-to-end execution time of tasks since (1) access time of the main memory is consistent and much faster (10X–100X) than bulk storage modules such as flash/hard disks, and (2) the main memory is far more flexible and efficient for random access. On the contrary, a system using secondary storage experiences not only significant but also unpredictable overhead, e.g., disk-writes in some storage modules such as flash and solid-state drives, have to erase an entire block before writing to it. By eliminating this overhead, in-memory execution enables fast and responsive back-to-back execution of multiple DNNs.

5 SYSTEM IMPLEMENTATION

We develop two real systems that implement neural weight virtualization: (1) an embedded GPU-based multitask learning mobile robot, and (2) a microcontroller-based low-power multitask learning IoT device. These systems are packed with 5-6 learning tasks. Figure 9 shows these systems performing an image recognition task (i.e., place and traffic sign recognition).



(a) Deep multitask learning mobile robot

(b) Deep multitask learning IoT device

Figure 9: Two application systems of neural weight virtualization: Multitask learning mobile robot and multitask learning IoT device.

5.1 Deep Multitask Learning Mobile Robot

System Overview. Intelligent robots are often tasked with multiple high-level perception tasks. For example, a social robot [15, 133] has to move around and assist a user – based on various human- or environment-generated information, such as voice and facial expression. As the first application system, we implement a multitask learning mobile robot (Figure 9a), representing an intelligent social agent, that assists humans in various situations by running six state-of-the-art DNNs, i.e., objection recognition (MobileNet-v2 [129]), face identification (FaceNet [132]), visual scene interpretation (ShowAndTell (IM2TXT) [155]), place classification (Place205 [182]), environmental sound classification (UrbanSound8K [145]), and voice recognition (GoogleSpeechCommands (GSC) [126]).

Hardware Platform. The robot is implemented using Jetson Nano [110], an embedded GPU platform having an NVIDIA Maxwell GPU and an ARM A57 CPU. The robot has a camera in the front, a microphone on the side, and two motors and wheels on both sides. We print the body of the robot with a 3D printer and install the components on it. For the offline phase of weight virtualization,

we use an NVIDIA RTX 2080 Ti GPU. For run-time in-memory multitask DNN execution, we use the embedded GPU of the robot.

Software Platform. We extend TensorFlow [1] to support weight virtualization algorithms and in-memory multitask execution.

Memory Budget. We allocate 146MB of GPU RAM to the virtual weight parameters and pack six DNN tasks in the GPU RAM whose total memory requirement is 604MB. The memory size of each DNN is listed in Figure 16 and Table 5.

5.2 Deep Multitask Learning IoT Device

System Overview. Recently, microcontroller-based embedded systems have started to support the execution of lightweight versions of state-of-the-art DNNs [42, 78, 83, 157, 167]. For example, Google recently released their TensorFlow Lite library targeting microcontrollers (MCUs) [49], and Amazon Alexa services are now supported on MCUs such as ARM Cortex M [32] that have less than 1MB memory [7]. However, the number of DNNs that runs on an embedded system [13, 20, 42] is limited by the capability of the MCU, memory-size, and battery-capacity. As the second application system, we implement a low-power multitask learning IoT device that performs multitask DNN learning on an MCU, having an extremely limited memory (256KB). The system is shown in Figure 9b. It packs five compressed DNNs, i.e., traffic sign recognition (German Traffic Sign Recognition Benchmark (GTSRB) [144]), voice command recognition (GoogleSpeechCommands (GSC) [126]), house/plate number classification (Street View House Numbers (SVHN) [108]), digits classification (MNIST [79]), and object recognition (CIFAR-10 [75]).

Hardware Platform. The system is implemented on an MSP430 MCU [149] that consumes little energy (≤1.8mA). For in-memory multitask DNN execution, we use 256KB FRAM [17] built in the MCU package. The FRAM is a non-volatile memory, performing read/write operations in nanoseconds [34]. For sensing, we connect a camera and a microphone to the MCU. For the offline phase of weight virtualization, we use an NVIDIA RTX 2080 Ti GPU.

Software Platform. We extend TensorFlow [1] for weight virtualization algorithms and implement the in-memory multitask execution framework using C language for efficient task execution.

Memory Budget. We allocate 129 KB of FRAM for weight virtualization and pack five DNNs whose total memory requirement is 523KB. The memory size is shown in Figure 21 and Table 7.

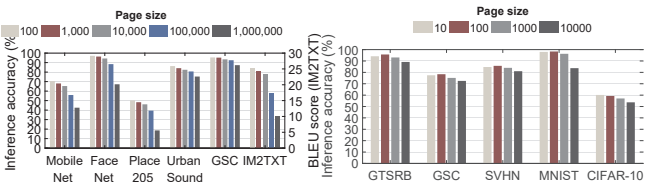
6 ALGORITHM EVALUATION

We first evaluate the weight virtualization phase described in Section 3 for the two systems we implement in Section 5. We use an NVIDIA RTX 2080 Ti GPU and an Intel Core i9-9900K CPU.

Training and Evaluation Datasets. For the training and evaluation of DNNs in the multitask mobile robot, we use ImageNet [35], VGGFace2 [18], LFW Face [66], Microsoft COCO [92], Place205 [182], UrbanSound8K [128], and GoogleSpeechCommands (GSC) [159] dataset. For the multitask IoT device, we use GTSRB [144], GSC [159], SVHN [108], MNIST [79], and CIFAR-10 [75] dataset.

6.1 Weight-Page Matching

Weight-Page Size. We evaluate the effect of weight-page size on weight virtualization. Figure 10 plots the inference accuracy of the two application systems over different weight-page sizes. In general, larger pages result in decreased accuracy for both systems, e.g., the accuracy of FaceNet for the mobile robot drops from 97% to 67% when the page size is increased from 100 to one million. Since smaller pages are more finely matched with each other, higher accuracy is achieved.



(a) Mobile Robot (b) IoT Device

Figure 10: The inference accuracy for various weight-page size for multitask learning mobile robot and the IoT device.

However, smaller page sizes increase the total number of virtual weight-pages and page matching time, which increases both the run-time and compile-time cost, as shown in Table 3 and 4. Thus, a suitable page size should be chosen based on the trade-off between the performance (e.g., inference accuracy) and computation/memory cost.

Weight-Page Size	Total Number of Virtual Weight-Pages	Total Number of Virtual Weights	Matching Time (s)
100	383,510	38,351,000	916.21
1,000	38,351	38,351,000	89.77
10,000	3,836	38,350,000	16.66
100,000	384	38,400,000	14.24
1,000,000	39	39,000,000	10.28
10,000,000	4	40,000,000	10.47

Table 3: The number of weight-pages, weights, and weight-page matching time over different page sizes for the multitask mobile robot.

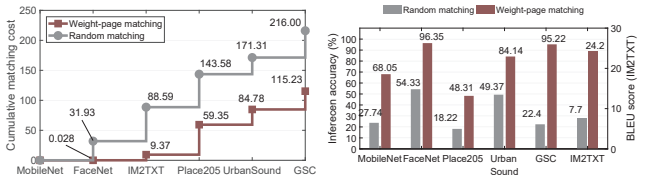
Weight-Page Size	Total Number of Virtual Weight-Pages	Total Number of Virtual Weights	Matching Time (s)
10	6,648	66,480	0.4822
100	665	66,500	0.1493
1,000	67	67,000	0.1076
10,000	7	70,000	0.1077

Table 4: The number of weight-pages, weights, and weight-page matching time over different page sizes for the multitask IoT device.

Weight-Page Matching vs. Random Matching. We evaluate the performance of the weight-page matching algorithm by comparing the weight-page matching cost defined in Equation 1 and the final inference accuracy of the two systems against random weight-page matching, as shown in Figures 11 and 12. In random matching, the weight-pages of DNNs are randomly matched for optimization without considering their similarity, as done in existing works [73, 170]. For both systems, the weight-page matching outperforms random matching, i.e., a maximum 1,140x less matching cost that results in a maximum of 72% inference accuracy improvement in the mobile robot (GSC in Figure 11b). It demonstrates that weight-page matching is an essential step for performance retention before performing weight-page optimization (combining), which significantly decreases the chance of combining weight-pages of big difference.

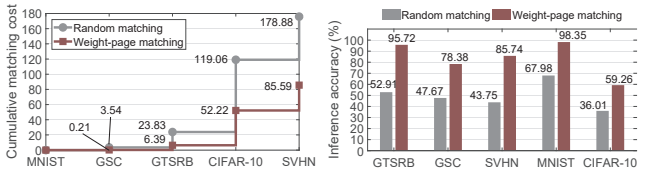
6.2 Weight-Page Optimization

Joint vs. Sequential Optimization. We evaluate two methods of weight-page optimization, i.e., joint vs. sequential optimization,



(a) Weight-page matching cost (b) Inference accuracy

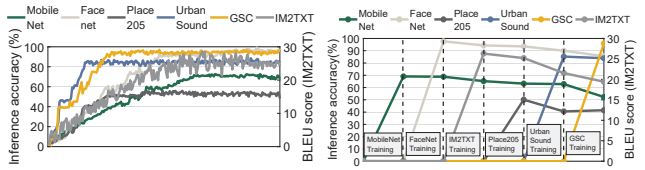
Figure 11: The weight-page matching cost and inference accuracy of the multitask learning mobile robot: The proposed weight-page matching vs. Random matching.



(a) Weight-page matching cost (b) Inference accuracy

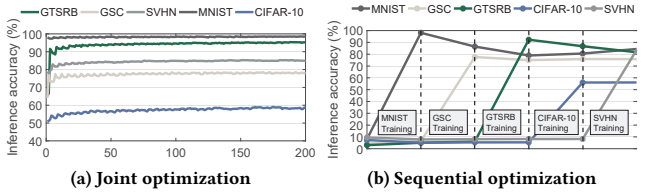
Figure 12: The weight-page matching cost and inference accuracy of the multitask learning IoT device: The proposed weight-page matching vs. Random matching.

by comparing their final inference accuracy. While all the DNN tasks are optimized together in the joint optimization, only a single task is optimized at a time after completing the optimization of the prior task in the sequential optimization. Figure 13 and 14 show the results for the two systems, respectively. Sequential optimization exhibits a pattern that a DNN recently optimized achieves its best accuracy while DNNs optimized prior to it experience accuracy degradation, e.g., from 69% to 52% for MobileNet for the mobile robot. On the other hand, joint optimization keeps the best accuracy of all DNNs, and sometimes achieves *higher* accuracy than the individual training of a DNN due to less overfitting achieved during joint training, e.g., from 69% to 78% for GSC in the multitask learning IoT device, by optimizing them together during the entire optimization iterations.



(a) Joint optimization (b) Sequential optimization

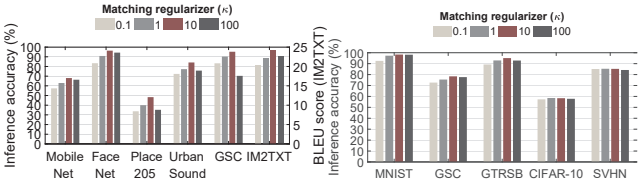
Figure 13: The inference accuracy of the multitask learning mobile robot: Joint vs. Sequential optimization.



(a) Joint optimization (b) Sequential optimization

Figure 14: The inference accuracy of the multitask learning IoT device: Joint vs. Sequential optimization.

Matching Regularizer. We evaluate the effect of matching regularizer, κ in Equation 1, and 5, which determines the extent of matching cost that works as a penalty in optimization, i.e., the larger κ, the more joint performance of all DNNs are considered during optimization. Figure 15 plots the inference accuracy with different matching regularizers, which shows that the inference accuracy of the DNNs tends to increase as the regularizer increases but starts to decrease after some point. That is because too large regularizer leads the optimizer to minimize the matching cost too much, preventing it from minimizing the original losses of DNNs.



(a) Multitask mobile robot (b) Multitask IoT device

Figure 15: The inference accuracy over the regularizer κ.

7 SYSTEM EVALUATION

In this section, we evaluate the run-time performance of in-memory multitask execution for the two multitask learning systems described in Section 5 (i.e., mobile robot and IoT device). The evaluation is performed on the same dataset used in Section 6.

7.1 Deep Multitask Learning Mobile Robot

Memory Packing Efficiency. We evaluate the memory efficiency of weight virtualization by measuring the memory usage of the DNNs, which shows the packing ratio of multiple DNNs that reside in the limited memory. Figure 16 and Table 5 present the number of weights and memory usage for the mobile robot, where a total of 144M weight parameters (604MB) are virtualized to 38M virtual weights (146MB). It achieves a 4.13x packing ratio when compared to baseline DNNs that are not virtualized.

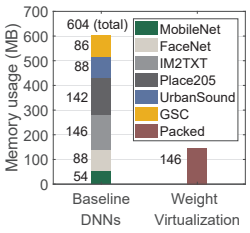


Figure 16: The memory usage of weight parameters: Baseline DNNs vs. Weight virtualization for the multitask learning mobile robot.

Baseline DNNs	Total Weights	Memory (MB)
MobileNet [35, 65, 129]	14M	54
FaceNet [18, 66, 132]	22M	88
IM2TXT [92, 155]	37M	142
Place205 [180–182]	38M	146
UrbanSound [128, 145, 179]	23M	88
GSC [22, 89, 126, 158]	22M	86
Total	**144M**	**604**

Weight Virtualization	Packed Weights	Memory (MB)
MobileNet+FaceNet +IM2TXT+Place205 +UrbanSound+GSC	**38M**	**146**
Packing Ratio	**4.13x**	**4.13x**

Table 5: Weight memory packing of the multitask learning mobile robot with weight virtualization.

Inference Accuracy. We evaluate the inference accuracy of the virtualized DNNs, as shown in Figure 17. Compared to the state-of-the-art baseline DNNs that are not virtualized, they achieve comparable performance with a maximum of 3% accuracy drop (MobileNet) or achieve better accuracy for some DNNs (0.4% up in UrbanSound and 0.72% up in GSC). The accuracy increase can be explained by the fact that weight parameter sharing via virtualization leads to (1) reduce the risk of overfitting [12], and (2) provide better generalization [125] with weight-page optimization.

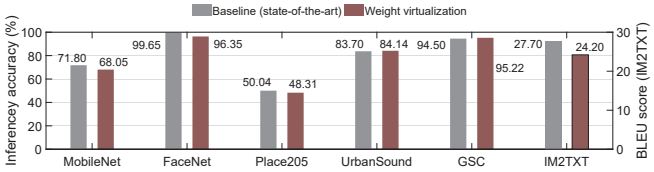


Figure 17: The inference accuracy of multitask mobile robot: Baseline (state-of-the-art) vs. Weight virtualization.

Execution Time. We evaluate the execution time of the virtualized DNNs against baseline DNNs that are not virtualized. Figure 18 shows the end-to-end execution time of consecutive DNNs for (1) baseline DNNs that use eMMC memory as a secondary storage module for loading DNNs to GPU RAM, and (2) virtualized DNNs that perform in-memory loading and execution entirely in GPU RAM. As shown in the figure, the execution time is significantly improved when executing DNNs consecutively, e.g., 36.9x faster when executing all six DNNs successively (39.12 vs. 1.06 seconds). Next, we breakdown the end-to-end execution time into two parts, i.e., (1) actual DNN execution and (2) DNN-switching (loading) time, and measure each of them separately. Figure 19 shows the execution time and switching (loading) time of each DNN. All the virtualized DNNs accelerate their switching (loading) by a maximum of 87x (9.16 vs. 0.105 seconds for IM2TXT) compared to the non-virtualized DNNs (the baselines), which demonstrates that in-memory execution yields much faster response time for multitasking DNNs.

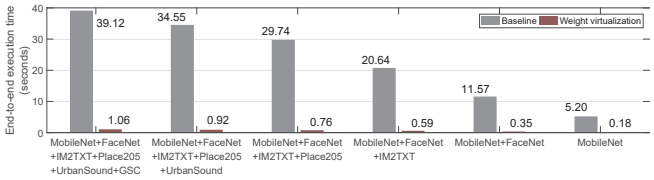
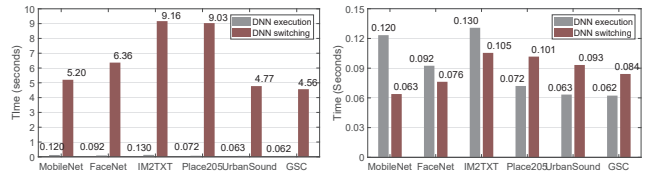


Figure 18: The end-to-end execution time of DNNs consecutively executed on the mobile robot: Baseline (no virtualization) vs. Weight virtualization.



(a) Baseline **(b) Weight virtualization**

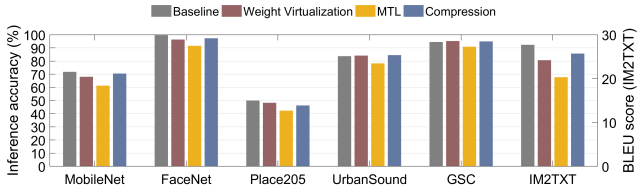
Figure 19: The DNN execution and switching time of the mobile robot: Baseline vs. Weight virtualization.

Comparison to Alternatives. We compare the inference accuracy and execution time (total time for inference plus switching) of the virtualized DNNs against two alternative methods: (1) multitask learning (MTL) and (2) individual model compression. Both use the same total amount of 146 MB memory as the virtualized DNNs. For MTL, the baseline DNNs are trained together using the state-of-the-art *K for-the-price-of 1* [106] approach, whose memory size is limited to 146 MB. For model compression, the baseline DNNs are individually compressed using pruning methods that are applicable to specific architectures [54, 55, 61, 161, 172]. Table 6 provides the memory usage of the baseline (uncompressed) and compressed DNNs. Note that the combined memory usage of the six compressed DNNs is 146 MB.

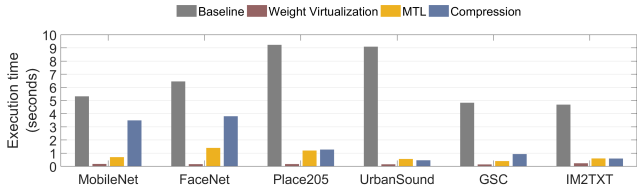
	MobileNet	FaceNet	IM2TXT	Place205	UrbanSound	GSC
Vanilla	54 MB	88 MB	142 MB	146 MB	88 MB	54 MB
Compressed	36 MB [61]	52 MB [161]	19 MB [172]	4 MB [54]	16 MB [54]	18 MB [55]

Table 6: The memory usage of the baseline DNNs (uncompressed vanilla models), and individually compressed DNNs for the multitask learning robot. The compression algorithms are cited inside the brackets.

Figure 20a shows their inference accuracy where the virtualized DNNs achieve comparable accuracy to the MTL and individually compression approach, i.e., they perform slightly worse than the compression approach (-0.3%) but better than the MTL (+2.4%). However, the virtualized DNNs run significantly faster than the other two, i.e., 10.1x on average, as shown in Figure 20b. This is because (1) MTL executes the entire network for all tasks, and (2) individually compressed DNNs incur a high model switching overhead and are barely expedited due to reduced network size [26]. This result demonstrates that weight virtualization achieves the best of both worlds, i.e., high accuracy and low execution latency.



(a) Inference accuracy with fixed memory size (146 MB)



(b) Execution time with fixed memory size (146 MB)

Figure 20: Comparison against multitask learning (MTL) and individually compressed DNNs.

7.2 Deep Multitask Learning IoT Device

Memory Packing Efficiency. Figure 21 and Table 7 show the number of weights and memory usage for the low-power multitask learning IoT device, where a total of 268,692 weights (523KB) are virtualized to 66,475 virtual weights (129KB). It achieves a 4.04x packing ratio when compared to the baseline DNNs not virtualized.

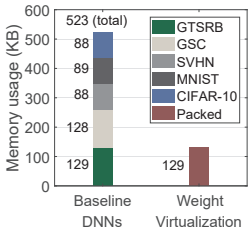


Figure 21: The memory usage of weight parameters: Baseline DNNs vs. Weight virtualization for the IoT device.

Baseline DNNs	Total Weights	Memory (KB)
GTSRB [83, 144]	66,475	129
GSC [22, 158]	65,531	128
SVHN [83, 108]	45,490	88
MNIST [43, 79]	45,706	89
CIFAR-10 [75, 168]	45,490	88
Total	**268,692**	**523**
Weight Virtualization	**Packed Weights**	**Memory (KB)**
GTSRB+GSC +SVHN+MNIST +CIFAR-10	**66,475**	**129**
Packing Ratio	**4.04x**	**4.04x**

Table 7: Weight memory packing with weight virtualization for the IoT device.

Inference Accuracy. Figure 22 shows the inference accuracy of the virtualized DNNs. Compared to the baseline DNNs that are not

virtualized, they all achieve higher accuracy, e.g., the accuracy of GSC increases from 69.86% to 76.38% since weight virtualization decreases overfitting in the individual DNN models.

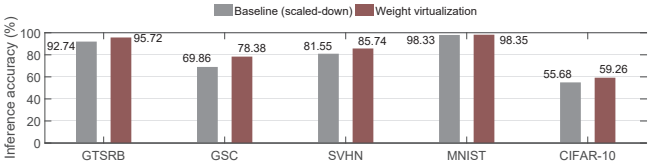


Figure 22: The inference accuracy of multitask IoT device: Baseline (scaled-down DNNs) vs. weight virtualization.

Execution Time. Figure 23 shows the end-to-end execution time of consecutive DNNs for (1) baseline DNNs that use an SD card as the secondary storage for loading DNNs to FRAM, and (2) virtualized DNNs that perform in-memory loading and execution entirely in FRAM. As shown in the figure, the execution time is improved when executing DNNs consecutively, e.g., 1.76x faster when executing all the five DNNs successively. Figure 24 shows the execution and switching (loading) time of each DNN. All the virtualized DNNs accelerate their switching (loading) by a maximum of 1,268x (2.41 vs. 0.0019 seconds in GTSRB) compared to the non-virtualized DNNs, which demonstrates that in-memory execution yields fast and responsive execution.

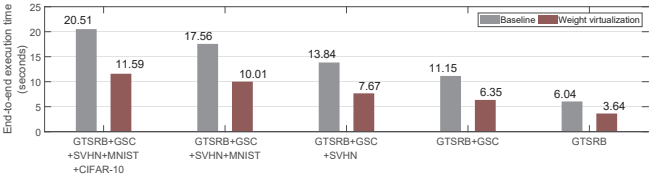
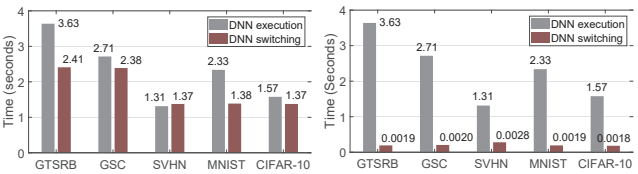


Figure 23: The end-to-end execution time of DNNs consecutively executed on the multitask learning IoT device: Baseline (no virtualization) vs. Weight virtualization.



(a) Baseline **(b) Weight virtualization**

Figure 24: The DNN execution and DNN-switching time of the IoT device: Baseline (no virtualization) vs. Weight virtualization. The bars of DNN-switching time in (b) are drawn lager than the real size.

Energy Consumption. We measure the energy consumption of DNN executions on the multitask learning IoT device against the baseline that does not use weight virtualization by using EnergyTrace [67] of MSP430 board. Figure 25 shows the total energy consumption of consecutive DNN executions for (1) the baseline DNNs that use an SD card, and (2) the virtualized DNNs that perform in-memory loading and execution in FRAM. In-memory execution reduces the total energy consumption by 4.24x when executing all five DNNs successively (432.95 vs. 102.09 mJ). Next, we breakdown the total energy consumption into two parts, i.e., (1) actual DNN execution and (2) DNN-switching (loading) energy, and measure each of them. Figure 26 shows the execution and switching energy for each DNN. All the virtualized DNNs improve their switching (loading) energy efficiency by a maximum of 7,413x (88.96 vs. 0.012 mJ in GTSRB) compared to the non-virtualized DNNs (the baselines).

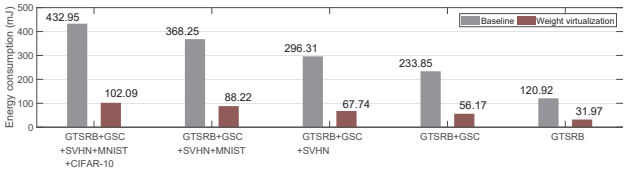
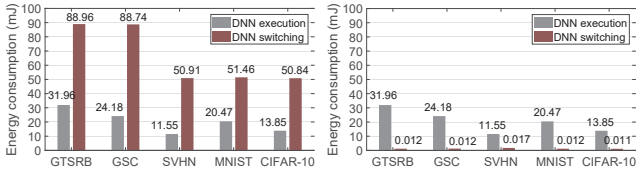


Figure 25: The total energy consumption of DNNs consecutively executed on the IoT device: Baseline (no virtualization) vs. Weight virtualization.



(a) Baseline **(b) Weight virtualization**

Figure 26: The energy consumption of DNN execution and DNN-switching for the IoT device: Baseline vs. Weight virtualization. The bars of DNN-switching energy in (b) are drawn larger than real.

Comparison to Alternatives. To evaluate the performance of virtualized DNNs against alternatives, we compare them with (1) a multitask learning (MTL) algorithm [38] and (2) individually compressed DNNs using [54] where the memory size of MTL and the combined memory size of all compressed DNNs is set to 129 KB for each. Table 8 shows the memory usages of compressed DNNs.

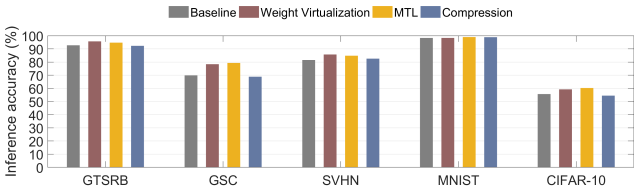
	GTSRB	GSC	SVHN	MNIST	CIFAR-10
Vanilla	129 KB	128 KB	88 KB	89 KB	88 KB
Compressed	27 KB [54]	34 KB [54]	25 KB [54]	10 KB [54]	30 KB [54]

Table 8: The memory usage of the baseline DNNs (uncompressed vanilla models), and individually compressed DNNs for the multitask learning IoT device. The compression algorithms are cited inside the brackets.

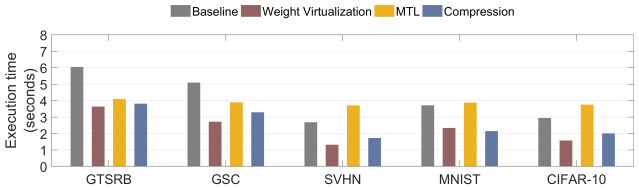
Figure 27a shows that the virtualized DNNs achieve comparable accuracy to the MTL that achieves the highest accuracy among all methods. The accuracy of MTL is average 3% higher than the baseline DNNs, and the virtualized DNNs provides 0.1% lower accuracy on average than the MTL. Also, the virtualized DNNs execute up to 2.8x faster than the two alternatives, as shown in Figure 27b. This result demonstrates that weight virtualization provides the benefit of multitask learning, i.e., accuracy improvement via joint training while enabling fast in-memory execution at the same time.

8 DISCUSSION

Fisher Information and Inference Accuracy. Fisher information [84] of weight parameters is used to optimize the loss function of a DNN, which is known as the natural gradient descent optimization [4–6, 111, 115, 117]. Since the weight parameters are proportionally updated to the Fisher information matrix in the natural gradient descent optimization in order to minimize the loss function that is closely related to the end-to-end accuracy, one can assume that reasonable accuracy is retained when the weight parameters of high Fisher information either remain the same or change very little. To understand their relationship, general proofs and theoretical assessment of Fisher information regarding the end-to-end inference accuracy and divergence from the optimal solution should be further studied. However, the exact mathematical relationship between the Fisher information and end-to-end accuracy is difficult to derive, which makes it challenging to provide



(a) Inference accuracy with fixed memory size (129 KB)



(b) Execution time with fixed memory size (129 KB)

Figure 27: Comparison against alternatives: multitask learning (MTL) and individually compressed DNNs.

a guaranteed inference accuracy when a large number of models are virtualized together.

Expansion to Secondary Storage. The proposed neural weight virtualization technique is primarily designed to operate purely in the main memory – without involving any secondary storage. An alternative design choice would be to extend the memory hierarchy of weight virtualization to include secondary storage or disks in the similar manner as in modern operating systems. It enables the system to retain the desired level of inference accuracy when the total model size is too large to fit into a small memory of the system in return for partial in-memory operation. By allowing disk-level weight virtualization, multitask learning of a more significant number of DNNs can be enabled with limited memory at scale without experiencing accuracy degradation.

9 RELATED WORK

Multitask Learning. Multitask learning jointly trains correlated tasks to increase the accuracy of each learner by exploiting the commonalities and differences across tasks [19, 125, 174, 175]. Typical approaches include common feature learning [88, 94, 95, 103, 105, 173, 178], low-rank parameter search [3, 23, 53, 102, 165, 171], task clustering [11, 52, 70, 77, 150, 183], and task relation learning [82, 96, 176, 177]. Although, by sharing network structure (typically, the first few layers) they achieve limited compression [60], their primary goal is to increase robustness and generalization of correlated and similar-structured learners. Thus, packing multiple heterogeneous learners into extremely scarce memory of an embedded system, as well as managing, context switching, and executing different DNN tasks efficiently at run-time, are challenging to them, both of which are achieved by weight virtualization.

DNN Combining (Packing). Stacked neural networks (SNN) [104, 142, 143] combine multiple DNNs by adding a layer on top of the features extracted from the DNNs. Although it may achieve better accuracy, the weights of each DNN needs to be maintained, which does not reduce the memory size, and the DNNs are not trained with multitask learning. PackNet [100] packs multiple DNNs to a single network with iterative pruning based on redundancies in DNNs to free up weights that can be employed for new tasks. However, the number of DNNs can be limited when free weights fall short as more DNNs are packed, and only a single network is maintained. Although [29] merges DNNs by integrating convolutional layers, it works with only two DNNs and requires to align layers to merge them. Learn-them-all [8, 69, 85] trains a single complex DNN to handle multiple tasks simultaneously. However, it is hard to choose a suitable architecture for learning all the tasks well in advance. Besides, learning from a large training data from different types or sources are demanding.

DNN Weight Sharing. For a single DNN, soft weight sharing approaches [109, 153] such as the Dirichlet process [124], k-means clustering [54, 138], or quantization [74] have been proposed. These techniques do not provide much benefit in terms of memory usage since the assignments of weights to connections must be stored additionally. There are several studies on weight sharing between multiple DNNs. MultiTask Zipping [60] merges DNNs for cross-model compression with a layer-wise neuron sharing. Sub-Network Routing [97] modularizes the shared layers into multiple layers of sub-networks. Cross-stitch Networks [103] apply weight sharing [36] after pooling and fully-connected layers of two DNNs. MMoE [98] learns to model task relationships from data by sharing the sub-models and weights across tasks. Tensor factorization [164] divides each set of parameters into shared and task-specific parts. However, their scope and methods of weight sharing are limited by network architecture and task type, unlike neural weight virtualization, where weights are shared without imposing a limitation on the network structure.

DNN Compression. The need to deploy DNNs on mobile systems motivated several techniques that reduce memory and computational costs, including knowledge distillation [21, 63, 123], low-rank factorization [68, 127, 147], pruning [51, 80, 87, 119, 169], quantization [54, 86, 162], compression with structured matrices [28, 137], network binarization [33, 90, 121], and hashing [25]. However, they do not provide cross-DNN compression that trains multiple DNNs together, unlike neural weight virtualization. Instead, each DNN is compressed individually with different compression methods, which is not scalable and does not achieve the benefits of multitask learning. Furthermore, a significantly compressed DNN does not run nearly as significantly faster since most parameters are pruned in fully-connected layers while convolutional layers consume most computation time, as shown in [51, 54, 116].

10 CONCLUSION

This paper introduces neural weight virtualization that enables scalable and fast multitask learning of DNNs on resource-constrained systems. By virtualizing network weights, it packs multiple DNNs into a limited size main memory. In-memory multitask learning enabled by weight virtualization improves the execution and response time, as well as the energy efficiency of the system. We implement two multitask learning systems: a mobile robot and an IoT device, and demonstrate that memory efficiency, execution time, and energy efficiency increases with weight virtualization.

ACKNOWLEDGMENTS

This paper was supported, in part, by NSF grants CNS-1816213 and CNS-1704469, and NIH grant 1R01LM013329-01.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Reza Abbasi-Asl and Bin Yu. 2017. Structural compression of convolutional neural networks based on greedy filter pruning. *arXiv preprint arXiv:1705.07356* (2017).

[3] Arvind Agarwal, Samuel Gerber, and Hal Daume. 2010. Learning multiple tasks using manifold regularization. In *Advances in neural information processing systems*. 46–54.

[4] Shun-ichi Amari. 1997. Neural learning in structured parameter spaces-natural Riemannian gradient. In *Advances in neural information processing systems*. 127–133.

[5] Shun-Ichi Amari. 1998. Natural gradient works efficiently in learning. *Neural computation* 10, 2 (1998), 251–276.

[6] Shun-ichi Amari, Koji Kurata, and Hiroshi Nagaoka. 1992. Information geometry of Boltzmann machines. *IEEE Transactions on neural networks* 3, 2 (1992), 260–271.

[7] Amazon. 2019. Introducing Alexa Voice Service Integration for AWS IoT Core, a new way to cost-effectively bring Alexa Voice to any type of connected device. https://aws.amazon.com/blogs/iot/introducing-alexa-voice-service-integration-for-aws-iot-core/.

[8] Yusuf Aytar, Carl Vondrick, and Antonio Torralba. 2017. See, hear, and read: Deep aligned representations. *arXiv preprint arXiv:1706.00932* (2017).

[9] Mallika Bariya, Hnin Yin Yin Nyein, and Ali Javey. 2018. Wearable sweat sensors. *Nature Electronics* 1, 3 (2018), 160–171.

[10] Joe Bartel. 2011. Non-preemptive multitasking. *Comput. J.* 30 (2011), 37–39.

[11] Aviad Barzilai and Koby Crammer. 2015. Convex multi-task learning by clustering. In *Artificial Intelligence and Statistics*. 65–73.

[12] Jonathan Baxter. 1997. A Bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine learning* 28, 1 (1997), 7–39.

[13] Sourav Bhattacharya and Nicholas D Lane. 2016. 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*. ACM, 176–189.

[14] Mark Billinghurst and Thad Starner. 1999. Wearable devices: new ways to manage information. *Computer* 32, 1 (1999), 57–64.

[15] Jonathan Bohren, Radu Bogdan Rusu, E Gil Jones, Eitan Marder-Eppstein, Caroline Pantofaru, Melonee Wise, Lorenz Mösenlechner, Wim Meeussen, and Stefan Holzer. 2011. Towards autonomous robotic butlers: Lessons learned with the PR2. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, 5568–5575.

[16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[17] Dudley Allen Buck. 1952. *Ferroelectrics for Digital Information Storage and Switching*. Technical Report. MASSACHUSETTS INST OF TECH CAMBRIDGE DIGITAL COMPUTER LAB.

[18] Qiong Cao, Li Shen, Weidi Xie, Omkar M Parkhi, and Andrew Zisserman. 2018. Vggface2: A dataset for recognising faces across pose and age. In *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*. IEEE, 67–74.

[19] Rich Caruana. 1997. Multitask learning. *Machine learning* 28, 1 (1997), 41–75.

[20] Jagmohan Chauhan, Suranga Seneviratne, Yining Hu, Archan Misra, Aruna Seneviratne, and Youngki Lee. 2018. Breathing-Based Authentication on Resource-Constrained IoT Devices using Recurrent Neural Networks. *Computer* 51, 5 (2018), 60–67.

[21] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. 2017. Learning efficient object detection models with knowledge distillation. In *Advances in Neural Information Processing Systems*. 742–751.

[22] Guoguo Chen, Carolina Parada, and Georg Heigold. 2014. Small-footprint keyword spotting using deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 4087–4091.

[23] Jianhui Chen, Lei Tang, Jun Liu, and Jieping Ye. 2009. A convex formulation for learning shared structures from multiple tasks. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 137–144.

[24] Liang-Hsuan Chen and Hai-Wen Lu. 2007. An extended assignment problem considering multiple inputs and outputs. *Applied Mathematical Modelling* 31, 10 (2007), 2239–2248.

[25] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*. 2285–2294.

[26] Xuhao Chen. 2018. Escoin: Efficient Sparse Convolutional Neural Network Inference on GPUs. *arXiv e-prints*, Article arXiv:1802.10280 (Feb 2018), arXiv:1802.10280 pages. arXiv:cs.DC/1802.10280

[27] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282* (2017).

[28] Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. 2015. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*. 2857–2865.

[29] Yi-Min Chou, Yi-Ming Chan, Jia-Hong Lee, Chih-Yi Chiu, and Chu-Song Chen. 2018. Merging Deep Neural Networks for Mobile Devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 1686–1694.

[30] Yi-Min Chou, Yi-Ming Chan, Jia-Hong Lee, Chih-Yi Chiu, and Chu-Song Chen. 2018. Unifying and merging well-trained deep neural networks for inference stage. *arXiv preprint arXiv:1805.04980* (2018).

[31] NM Mosharaf Kabir Chowdhury and Raouf Boutaba. 2010. A survey of network virtualization. *Computer Networks* 54, 5 (2010), 862–876.

[32] ARM Cortex. 2004. M3 Processor. *Cortex-M Series* (2004).

[33] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[34] Cypress. 2017. CY15B104Q. http://www.cypress.com/file/209146/download.

[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

[36] Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. 2015. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. 845–850.

[37] Eleazar Eskin, Alex J Smola, and SVN Vishwanathan. 2004. Laplace propagation. In *Advances in Neural Information Processing Systems*. 441–448.

[38] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 115–127.

[39] Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3, 4 (1999), 128–135.

[40] Bill Gates. 2007. A robot in every home. *Scientific American* 296, 1 (2007), 58–65.

[41] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. 2015. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters* 1, 2 (2015), 661–667.

[42] Graham Gobieski, Nathan Beckmann, and Brandon Lucia. 2018. Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems. *arXiv preprint arXiv:1810.07751* (2018).

[43] Graham Gobieski, Brandon Lucia, and Nathan Beckmann. 2019. Intelligence beyond the edge: Inference on intermittent embedded systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 199–213.

[44] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. 2013. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211* (2013).

[45] Google. 2019. Face Match on Google Nest Hub Max. https://support.google.com/googlenest/answer/9320885?hl=en.

[46] Google. 2019. FAQs on camera sensing: Google Nest Hub Max. https://support.google.com/googlenest/answer/9449279?hl=en.

[47] Google. 2019. Introducing Google Nest Hub. https://support.google.com/googlenest/answer/9136909?hl=en.

[48] Google. 2019. Learn about familiar face detection and how to manage your library. https://support.google.com/googlenest/answer/9268625?co=GENIE.Platform%3DAndroid&hl=en.

[49] Google. 2019. TensorFlow Lite for Microcontrollers. https://www.tensorflow.org/lite/microcontrollers.

[50] Google. 2019. Voice Match and media on Google Nest and Google Home speakers and displays. https://support.google.com/googlenest/answer/7342711?hl=en.

[51] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*. 1379–1387.

[52] Lei Han and Yu Zhang. 2015. Learning multi-level task groups in multi-task learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[53] Lei Han and Yu Zhang. 2016. Multi-stage multi-task learning with reduced rank. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[54] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[55] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.

[56] Babak Hassibi and David G Stork. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*. 164–171.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[58] Qihao He, Bruce Segee, and Vincent Weaver. 2016. Raspberry Pi 2 B+ GPU Power, Performance, and Energy Implications. In *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 163–167.

[59] Tianxing He, Yuchen Fan, Yanmin Qian, Tian Tan, and Kai Yu. 2014. Reshaping deep neural network for fast decoding by node-pruning. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 245–249.

[60] Xiaoxi He, Zimu Zhou, and Lothar Thiele. 2018. Multi-task zipping via layer-wise neuron sharing. In *Advances in Neural Information Processing Systems*. 6016–6026.

[61] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 784–800.

[62] Thomas A Henzinger and Joseph Sifakis. 2006. The embedded systems design challenge. In *International Symposium on Formal Methods*. Springer, 1–15.

[63] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[64] Jon Holton and Tim Fratangelo. 2012. Raspberry Pi Architecture. *Raspberry Pi Foundation, London, UK* (2012).

[65] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[66] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. 2008. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments.

[67] Texas Instrument. 2018. MSP EnergyTrace Technology. http://www.ti.com/tool/ENERGYTRACE

[68] Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. 2015. Training cnns with low-rank filters for efficient image classification. *arXiv preprint arXiv:1511.06744* (2015).

[69] Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. 2017. One model to learn them all. *arXiv preprint arXiv:1706.05137* (2017).

[70] Zhuoliang Kang, Kristen Grauman, and Fei Sha. 2011. Learning with Whom to Share in Multi-task Feature Learning.. In *ICML*, Vol. 2. 4.

[71] Fahim Kawsar, Chulhong Min, Akhil Mathur, Alessandro Montanari, Utku Günay Acer, and Marc Van den Broeck. 2018. eSense: Open Earable Platform for Human Sensing. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. ACM, 371–372.

[72] Changhyung Kim, Kyoung-Yoon Jeong, Sang-Geun Choi, Kyounghak Lee, and CBA Sohn. 2017. A Surveys based on Cloud Internet of Things Frameworks for Smart Home Service. In *2017 Manila International Conference on" Trends in Engineering and Technology"(MTET-17)*.

[73] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* 114, 13 (2017), 3521–3526.

[74] Fatih Köksal, Ethem Alpaydyn, and Günhan Dündar. 2001. Weight quantization for multi-layer perceptrons using soft weight sharing. In *International Conference on Artificial Neural Networks*. Springer, 211–216.

[75] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.

[76] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[77] Abhishek Kumar and Hal Daume III. 2012. Learning task grouping and overlap in multi-task learning. *arXiv preprint arXiv:1206.6417* (2012).

[78] Nicholas D Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. 2017. Squeezing deep learning into mobile and embedded devices. *IEEE Pervasive Computing* 3 (2017), 82–88.

[79] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[80] Yann LeCun, John S Denker, and Sara A Solla. 1990. Optimal brain damage. In *Advances in neural information processing systems*. 598–605.

[81] Yann LeCun, LD Jackel, Leon Bottou, A Brunot, Corinna Cortes, JS Denker, Harris Drucker, I Guyon, UA Muller, Eduard Sackinger, et al. 1995. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, Vol. 60. Perth, Australia, 53–60.

[82] Giwoong Lee, Eunho Yang, and Sung Hwang. 2016. Asymmetric multi-task learning based on task relatedness and loss. In *International Conference on Machine Learning*. 230–238.

[83] Seulki Lee and Shahriar Nirjon. 2019. Neuro. ZERO: a zero-energy neural network accelerator for embedded sensing and inference systems. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. ACM, 138–152.

[84] Erich L Lehmann and George Casella. 2006. *Theory of point estimation*. Springer Science & Business Media.

[85] Gil Levi and Tal Hassner. 2015. Age and gender classification using convolutional neural networks. In *Proceedings of the iEEE conference on computer vision and pattern recognition workshops*. 34–42.

[86] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. 2017. Training quantized nets: A deeper understanding. In *Advances in Neural Information Processing Systems*. 5811–5821.

[87] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).

[88] Sijin Li, Zhi-Qiang Liu, and Antoni B Chan. 2014. Heterogeneous multi-task learning for human pose estimation with deep convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 482–489.

[89] Xuejiao Li and Zixuan Zhou. [n.d.]. Speech Command Recognition with Convolutional Neural Network. ([n. d.]).

[90] Zhe Li, Xiaoyu Wang, Xutao Lv, and Tianbao Yang. 2017. Sep-nets: Small and effective pattern networks. *arXiv preprint arXiv:1706.03912* (2017).

[91] Yiyi Liao, Sarath Kodagoda, Yue Wang, Lei Shi, and Yong Liu. 2016. Understand scene categories by objects: A semantic regularized scene classifier using convolutional neural networks. In *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2318–2325.

[92] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 740–755.

[93] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 19–34.

[94] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2017. Adversarial multi-task learning for text classification. *arXiv preprint arXiv:1704.05742* (2017).

[95] Wu Liu, Tao Mei, Yongdong Zhang, Cherry Che, and Jiebo Luo. 2015. Multi-task deep visual-semantic embedding for video thumbnail selection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3707–3715.

[96] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and S Yu Philip. 2017. Learning multiple tasks with multilinear relationship networks. In *Advances in Neural Information Processing Systems*. 1594–1603.

[97] Jiaqi Ma, Zhe Zhao2 Jilin Chen2 Ang Li, and Lichan Hong. 2019. SNR: Sub-Network Routing for Flexible Parameter Sharing in Multi-task Learning. (2019).

[98] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. 2018. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1930–1939.

[99] Sumit Majumder, Tapas Mondal, and M Jamal Deen. 2017. Wearable sensors for remote health monitoring. *Sensors* 17, 1 (2017), 130.

[100] Arun Mallya and Svetlana Lazebnik. 2018. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7765–7773.

[101] Daniel Maturana and Sebastian Scherer. 2015. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 922–928.

[102] Andrew M McDonald, Massimiliano Pontil, and Dimitris Stamos. 2014. Spectral k-support norm regularization. In *Advances in neural information processing systems*. 3644–3652.

[103] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. 2016. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3994–4003.

[104] Milad Mohammadi and Subhasis Das. 2016. SNN: stacked neural networks. *arXiv preprint arXiv:1605.08512* (2016).

[105] Nikola Mrkšić, Diarmuid O Séaghdha, Blaise Thomson, Milica Gašić, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2015. Multi-domain dialog state tracking using recurrent neural networks. *arXiv preprint arXiv:1506.07190* (2015).

[106] Pramod Kaushik Mudrakarta, Mark Sandler, Andrey Zhmoginov, and Andrew Howard. 2018. K for the Price of 1: Parameter-efficient Multi-task and Transfer Learning. *arXiv preprint arXiv:1810.10703* (2018).

[107] James Munkres. 1957. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics* 5, 1 (1957), 32–38.

[108] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Reading digits in natural images with unsupervised feature learning. (2011).

[109] Steven J Nowlan and Geoffrey E Hinton. 1992. Simplifying neural networks by soft weight-sharing. *Neural computation* 4, 4 (1992), 473–493.

[110] NVIDIA. 2019. Jetson Nano. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[111] Yann Ollivier, Ludovic Arnold, Anne Auger, and Nikolaus Hansen. 2017. Information-geometric optimization algorithms: A unifying picture via invariance principles. *The Journal of Machine Learning Research* 18, 1 (2017), 564–628.

[112] Hiroki Ota, Minghan Chao, Yuji Gao, Eric Wu, Li-Chia Tai, Kevin Chen, Yasutomo Matsuoka, Kosuke Iwai, Hossain M Fahad, Wei Gao, et al. 2017. 3d printed "earable" smart devices for real-time detection of core body temperature. *ACS sensors* 2, 7 (2017), 990–997.

[113] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. 2017. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 27–40.

[114] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* (2019).

[115] Hyeyoung Park, S-I Amari, and Kenji Fukumizu. 2000. Adaptive natural gradient learning algorithms for various stochastic models. *Neural Networks* 13, 7 (2000), 755–764.

[116] Jongsoo Park, Sheng R Li, Wei Wen, Hai Li, Yiran Chen, and Pradeep Dubey. 2016. Holistic sparsecnn: Forging the trident of accuracy, speed, and size. *arXiv preprint arXiv:1608.01409* 1, 2 (2016).

[117] Razvan Pascanu and Yoshua Bengio. 2013. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584* (2013).

[118] Christian Plessl and Marco Platzner. 2004. Virtualization of Hardware-Introduction and Survey.. In *ERSA*. Citeseer, 63–69.

[119] Adam Polyak and Lior Wolf. 2015. Channel-level acceleration of deep face representations. *IEEE Access* 3 (2015), 2163–2175.

[120] Erwin Prassler and Kazuhiro Kosuge. 2008. Domestic robotics. *Springer handbook of robotics* (2008), 1253–1281.

[121] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, 525–542.

[122] Joseph Redmon and Anelia Angelova. 2015. Real-time grasp detection using convolutional neural networks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1316–1322.

[123] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* (2014).

[124] Wolfgang Roth and Franz Pernkopf. 2018. Bayesian Neural Networks with Weight Sharing Using Dirichlet Processes. *IEEE transactions on pattern analysis and machine intelligence* (2018).

[125] Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* (2017).

[126] Tara Sainath and Carolina Parada. 2015. Convolutional neural networks for small-footprint keyword spotting. (2015).

[127] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 6655–6659.

[128] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. 2014. A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 1041–1044.

[129] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.

[130] Duygu Sarikaya, Jason J Corso, and Khurshid A Guru. 2017. Detection and localization of robotic tools in robot-assisted surgery videos using deep neural networks for region proposal and detection. *IEEE transactions on medical imaging* 36, 7 (2017), 1542–1549.

[131] Alexander Schrijver. 2003. *Combinatorial optimization: polyhedra and efficiency*. Vol. 24. Springer Science & Business Media.

[132] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 815–823.

[133] Roland Siegwart, Kai O Arras, Samir Bouabdallah, Daniel Burnier, Gilles Froidevaux, Xavier Greppin, Björn Jensen, Antoine Lorotte, Laetitia Mayor, Mathieu Meisser, et al. 2003. Robox at Expo. 02: A large-scale installation of personal robots. *Robotics and Autonomous Systems* 42, 3-4 (2003), 203–222.

[134] Abraham Silberschatz, Greg Gagne, and Peter B Galvin. 2018. *Operating system concepts*. Wiley.

[135] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. 2012. Operating System Concepts. 9th.

[136] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[137] Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. 2015. Structured transforms for small-footprint deep learning. In *Advances in Neural Information Processing Systems*. 3088–3096.

[138] Sanghyun Son, Seungjun Nah, and Kyoung Mu Lee. 2018. Clustering convolutional kernels to compress deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 216–232.

[139] WU Song and JIN Hai. 2003. A survey of storage virtualization. *Mini-micro Systems* 4 (2003).

[140] Frits CR Spieksma. 2000. Multi index assignment problems: complexity, approximation, applications. In *Nonlinear assignment problems*. Springer, 1–12.

[141] Mouroutsos G Spyridon and Mitka Eleftheria. 2012. Classification of domestic robots. *Proceedings in ARSA-Advanced Research in Scientific Areas* 1, 7 (2012), 1693.

[142] Dasaratha V Sridhar, Eric B Bartlett, and Richard C Seagrave. 1999. An information theoretic approach for combining neural network process models. *Neural Networks* 12, 6 (1999), 915–926.

[143] Dasaratha V Sridhar, Richard C Seagrave, and Eric B Bartlett. 1996. Process modeling using stacked neural networks. *AIChE Journal* 42, 9 (1996), 2529–2539.

[144] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. 2011. The German Traffic Sign Recognition Benchmark: A multi-class classification competition.. In *IJCNN*, Vol. 6. 7.

[145] Yu Su, Ke Zhang, Jingyu Wang, and Kurosh Madani. 2019. Environment sound classification using a two-stream cnn based on decision-level fusion. *Sensors* 19, 7 (2019), 1733.

[146] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[147] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. 2015. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067* (2015).

[148] Kazuhiro Taniguchi, Hisashi Kondo, Mami Kurosawa, and Atsushi Nishikawa. 2018. Earable TEMPO: a novel, hands-free input device that uses the movement of the tongue measured with a wearable ear sensor. *Sensors* 18, 3 (2018), 733.

[149] TexasInstruments. 2018. MSP430FR5994. http://www.ti.com/product/MSP430FR5994.

[150] Sebastian Thrun and Joseph O'Sullivan. 1996. Discovering structure in multiple learning tasks: The TC algorithm. In *ICML*, Vol. 96. 489–497.

[151] Wing N Toy and Benjamin Zee. 1986. *Computer Hardware-Software Architecture*. Prentice Hall Professional Technical Reference.

[152] Ming Tu, Visar Berisha, Martin Woolf, Jae-sun Seo, and Yu Cao. 2016. Ranking the parameters of deep neural networks using the fisher information. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2647–2651.

[153] Karen Ullrich, Edward Meeds, and Max Welling. 2017. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008* (2017).

[154] Frank Uyeda. 2009. Lecture 7: Memory Management, CSE 120: Principles of Operating Systems. http://cseweb.ucsd.edu/classes/su09/cse120/lectures/Lecture7.pdf

[155] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3156–3164.

[156] Limin Wang, Sheng Guo, Weilin Huang, and Yu Qiao. 2015. Places205-vggnet models for scene recognition. *arXiv preprint arXiv:1508.01667* (2015).

[157] Yue Wang, Tan Nguyen, Yang Zhao, Zhangyang Wang, Yingyan Lin, and Richard Baraniuk. 2018. EnergyNet: Energy-Efficient Dynamic Inference. (2018).

[158] Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209* (2018).

[159] P. Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *ArXiv e-prints* (April 2018). arXiv:cs.CL/1804.03209 https://arxiv.org/abs/1804.03209

[160] Douglas Brent West et al. 1996. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, NJ.

[161] Chunpeng Wu, Wei Wen, Tariq Afzal, Yongmei Zhang, Yiran Chen, et al. 2017. A compact dnn: approaching googlenet-level accuracy of classification and domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5668–5677.

[162] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4820–4828.

[163] Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. 2017. Towards monocular vision based obstacle avoidance through deep reinforcement learning. *arXiv preprint arXiv:1706.09829* (2017).

[164] Yongxin Yang and Timothy Hospedales. 2016. Deep multi-task representation learning: A tensor factorisation approach. *arXiv preprint arXiv:1605.06391* (2016).

[165] Yongxin Yang and Timothy M Hospedales. 2016. Trace norm regularised deep multi-task learning. *arXiv preprint arXiv:1606.04038* (2016).

[166] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. 2017. 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, 351–360.

[167] Shuochao Yao, Yiran Zhao, Aston Zhang, Shaohan Hu, Huajie Shao, Chao Zhang, Lu Su, and Tarek Abdelzaher. 2018. Deep learning for the internet of things. *Computer* 51, 5 (2018), 32–41.

[168] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher. 2017. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 4.

[169] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. 2018. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9194–9203.

[170] Friedemann Zenke, Ben Poole, and Surya Ganguli. 2017. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 3987–3995.

[171] Jian Zhang, Zoubin Ghahramani, and Yiming Yang. 2008. Flexible latent variable models for multi-task learning. *Machine Learning* 73, 3 (2008), 221–242.

[172] Jie Zhang, Xiaolong Wang, Dawei Li, and Yalin Wang. 2018. Dynamically hierarchy revolution: dirnet for compressing recurrent neural network on mobile devices. *arXiv preprint arXiv:1806.01248* (2018).

[173] Wenlu Zhang, Rongjian Li, Tao Zeng, Qian Sun, Sudhir Kumar, Jieping Ye, and Shuiwang Ji. 2016. Deep model based transfer and multi-task learning for biological image analysis. *IEEE transactions on Big Data* (2016).

[174] Yu Zhang and Qiang Yang. 2017. An overview of multi-task learning. *National Science Review* 5, 1 (2017), 30–43.

[175] Yu Zhang and Qiang Yang. 2017. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114* (2017).

[176] Yu Zhang and Dit-Yan Yeung. 2013. Learning high-order task relationships in multi-task learning. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

[177] Yu Zhang and Dit-Yan Yeung. 2013. Multilabel relationship learning. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 7, 2 (2013), 7.

[178] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. 2014. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*. Springer, 94–108.

[179] Zhichao Zhang, Shugong Xu, Shan Cao, and Shunqing Zhang. 2018. Deep convolutional neural network with mixup for environmental sound classification. In *Chinese Conference on Pattern Recognition and Computer Vision (PRCV)*. Springer, 356–367.

[180] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. 2014. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856* (2014).

[181] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. 2016. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2921–2929.

[182] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. 2014. Learning deep features for scene recognition using places database. In *Advances in neural information processing systems*. 487–495.

[183] Qiang Zhou and Qi Zhao. 2015. Flexible clustered multi-task learning by learning representative tasks. *IEEE transactions on pattern analysis and machine intelligence* 38, 2 (2015), 266–278.