Neurocomputing 442 (2021) 116–124

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Auto-VirtualNet: Cost-adaptive dynamic architecture search for multi-task learning

Eunwoo Kim [a], Chanho Ahn [b], Songhwai Oh [b,*]

[a] School of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea
[b] Department of Electrical and Computer Engineering, ASRI, Seoul National University, Seoul, South Korea

ARTICLE INFO

Article history:
Received 11 September 2020
Revised 6 December 2020
Accepted 15 February 2021
Available online 4 March 2021
Communicated by Zidong Wang

Keywords:
Dynamic architecture search
Cost-adaptive learning
Multi-task learning
Task interference

ABSTRACT

Multi-task learning (MTL) improves learning efficiency by solving multiple tasks simultaneously compared to multiple instances of individual learning. However, despite its benefits, there still remain several major challenges: first, negative interference can reduce the learning efficiency when the number of tasks is high or the tasks are of limited relevance. Second, exploring an optimal model structure manually is quite restricted. Last but not least, offering cost-adaptive solutions has not been addressed in the MTL regime. In spite of its notable merits, the combined problem has not been well discussed. In this work, we propose a novel MTL approach to address the combinatorial problem while minimizing memory consumption. The proposed method discovers multiple network models dynamically from a pool of candidate models, and produces a set of widely distributed solutions with respect to different computational costs for each task. For the diversity of candidate models, we modularize the given backbone architecture that generates basic building blocks and then construct a hierarchical structure based on the building blocks. The proposed method is trained to optimize both task performance and computational costs of selected models. The proposed method dynamically generates optimal networks for each task and offers significant performance improvements over existing MTL approaches in a range of experiments.

© 2021 Elsevier B.V. All rights reserved.

1. Introduction

Deep learning has become a popular strategy in many fields, such as machine learning [1], computer vision [2–4] and natural language processing [5], due to its significantly enhanced performance over the traditional machine learning counterparts. To this end, many deeper and wider neural networks have been devised to achieve state-of-the-art performance for a designated task, such as image classification [2,4] and object detection [6,7]. However, as a result of pursuing this design philosophy, many deep neural networks have been well known to be over-parameterized and highly redundant as they produce a number of parameters making unimportant contributions [8,9]. Furthermore, due to a large number of parameters (or high memory footprint) of deep networks, it is hard to deploy them on devices of low memory capacity, such as mobile phones and robots.

To improve model efficiency and fully utilize network parameters, a number of learning strategies have been proposed under the name of multi-task learning (MTL) [10–18]. MTL is an efficient learning strategy that simultaneously learns multiple tasks for an efficient and versatile deep learning approach. Recent studies on MTL have shown that the joint learning regime successfully improves the performance of tasks in several applications, such as image classification [19], object detection [20], and semantic segmentation [14]. We are particularly interested in developing a multi-task learning approach in a single architecture (by sharing parameters) for resource efficiency in this work. Even if meaningful progress has been made towards efficient deep learning, most existing studies rely on human-designed architectures which demands dedicated efforts to find a promising model. Moreover, the hand-crafted structures are held fixed before training, thus it may cause a decrease in learning efficiency due to task interference when target tasks are of little relevance to one another [16,17]. It is inevitable to encounter negative interference when we perform different tasks.

Such expert efforts for architecture engineering have been recently relieved by automating the design process [21–23]. This line of research, which searches for a promising network model within well-developed hand-designed architectures, is called dynamic architecture search [24,25,17] or adaptive inference

* Corresponding author.
E-mail addresses: eunwoo@cau.ac.kr (E. Kim), mychahn@snu.ac.kr (C. Ahn), songhwai@snu.ac.kr (S. Oh).

https://doi.org/10.1016/j.neucom.2021.02.050
0925-2312/© 2021 Elsevier B.V. All rights reserved.

[26,27]. The hand-designed backbone architectures, such as residual networks [2,28], consist of predefined internal modules (e.g., residual modules, layers, and/or channel groups), which generate a diverse set of candidate models from their assembly. One can search for an internal model from a pool of candidate models in a backbone by dynamically constructing using the modules [24,25,17]. Automating architecture engineering and producing dynamic model structures can be a promising solution to avoid task interference in multi-task learning due to its dynamic search nature.

We also would like to highlight that a learning framework for multiple tasks or architecture search generally produces an architecture for a single desirable resource budget (e.g., the number of parameters and FLOPs) at a time. In other words, if a different computational budget is required, we may need to search for and train the corresponding model from scratch, entailing efforts with trial-and-error. It would be highly efficient to produce multiple architectures of different computational budgets in a single learning framework, even for a multi-task learning framework for efficiency and versatility. By learning such a framework, we can produce a wide range of solutions (i.e., cost-adaptive solutions) with respect to different tasks and computational budgets.

In this work, we aim to address the aforementioned important problems in a single learning framework simultaneously; hand-crafted model design, cost-adaptive solutions, and task interference arising when learning multiple tasks. To achieve this, we introduce a new learning framework that consists of a search agent and a backbone architecture, producing a pool of candidate models. The proposed approach explores multiple architectures at a time in the backbone, with respect to different resource budgets. This gives a trade-off between model performance and computational cost for each task in MTL scenarios to choose a proper model for a given resource requirement.

Interestingly, the proposed framework resembles a virtual machine (VM) framework [29] consisting of a hypervisor (a search agent) and VMs (network models) sharing resources of a physical machine (a backbone). Though a recent study [19] presents virtual networks within a single learned framework, they lack a crucial component, i.e., a search agent, to assign an optimal model to a task automatically. In [19], configurations of network models are determined using a rule-based method. Thus, the framework produces a predetermined fixed configuration of a network with respect to tasks and computation costs, not providing dynamic and optimal solutions. In contrast, the proposed framework contains both a backbone generating multiple networks and a search agent, which enables exploring an optimal model in the backbone.

The proposed approach performs a dynamic multi-model search for models with different resource budgets in the multi-task learning setup, performed within a single training procedure. The proposed framework consists of two major components: a backbone architecture and a search agent. To allow dynamic search, we modularize the backbone architecture by splitting the convolution channels in each layer into multiple groups which constitute basic building blocks. Based on the building blocks, we construct a hierarchical structure in such a way that a set of building blocks in a lower-level of hierarchy is a subset of a higher-level of hierarchy, creating a nested structure. This can produce cost-adaptive solutions because of the different numbers of building blocks (and thus different amounts of resources required) in different levels of hierarchy. The search agent identifies an optimal hierarchical structure using building blocks for each task. The structure for each task corresponds to a network sharing the resources of the backbone architecture, called a virtual network (VirtualNet). The entire procedure of the proposed method is illustrated in Fig. 1. The memory increment taken up by the search agent is negligible compared to that of the backbone architecture. The proposed approach is trained by considering the performance of each task, as well as the computational cost using stochastic gradient descent (SGD) in an alternating manner. Since our approach realizes a dynamic multi-model search in multi-task learning under different resource budgets in a single learning framework, it is highly efficient.

We evaluate the proposed approach for a range of multi-task learning scenarios using popular image classification benchmark datasets. The scenarios are designed to perform from a small number of similar tasks to many different tasks. Experimental results show that our approach achieves higher classification accuracy on the target tasks compared to other existing MTL approaches and provides adaptive solutions for different computational budgets. We further conduct comprehensive studies and confirm the effectiveness of the proposed approach.

The main contributions of this work are threefold:

- We propose a novel MTL method that addresses task interference, manual design efforts, and cost-adaptive learning simultaneously in a single learning framework.
- We present a framework that produces multiple network models sharing the resources of a backbone whose respective structures are explored by a search agent.
- Experimental results show the excellence of the proposed method compared to existing MTL approaches for diverse MTL scenarios.

2. Related work

2.1. Automatic architecture search

In recent years, there has been increasing interest in automating the design of neural network architectures [21,31,15,22,16,32]. In this paradigm of automatic model search, the search method automatically discovers an optimal configuration of a network from a backbone architecture [15,31] or a set of hyperparameters from which to choose [21,33]. Compared to model pruning or compression [34–36], which only controls the number of parameters or the model scale, a model search strategy is able to discover high-performing models from a large search space. While avoiding the effort of designing models manually, this approach can improve task performance over conventional hand-crafted network models under the same model capacity [2,3]. Recent studies have devised automated model searches for multi-task learning [15,16]. However, our study is different from those works in that the proposed framework is capable of cost-adaptive learning, i.e., inference with respect to different computational costs (see Table 1). To tailor a solution to a different computational cost, it is generally required to design multiple architectures and painstakingly train each one separately. Our method incorporates both automated architecture search and cost-adaptive learning, producing a large number of outputs for associated tasks and costs.

2.2. Cost-adaptive learning

One of the main goals of this work is cost-adaptive learning. This provides multiple solutions at a time for each task, covering a range of different computational costs rather than providing a single solution of a single cost that might not meet requirements. Several works construct k-in-1-type network structures within an architecture to provide adaptive solutions for multiple tasks [37–39] or different computational requirements [40,41]. By executing different internal models, these architectures can address multiple tasks (or budgets) without the need for multiple models (and their corresponding memory footprints). Recently, a similar method has been proposed [19] to perform inference with respect

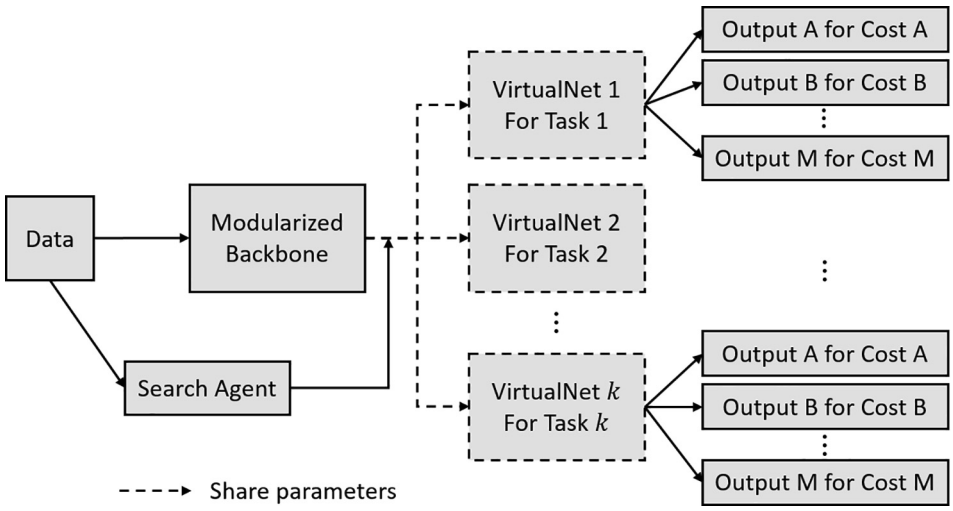

Fig. 1. Overall framework of the proposed method. Given a modularized backbone architecture, the search agent determines the configurations of virtual networks (termed VirtualNets) and their hierarchical structures for tasks. The hierarchical structure of a VirtualNet contains multiple subnetworks of different levels of hierarchy (or cost), which provides cost-adaptive solutions.

Table 1
Categorization of deep learning approaches with respect to functionality.

Method	Dynamic search	Multi-task learning	Cost-adaptive learning
Cross-Stitch Net [30]	X	✓	X
NASNet [22]	✓	X	X
RoutingNet [15]	✓	✓	X
VirtualNet [19]	X	✓	✓
DEN [17]	✓	✓	X
Proposed	✓	✓	✓

to different computational budgets in multi-task learning, allowing a larger number of solutions from a learned model. Although the method has made meaningful progress in realizing efficiency in deep neural networks, the approach is based on human-crafted models. This produces and keeps a fixed structure before training, which is vulnerable to task interference. Our method extends that of the previous study [19] to overcome its manual design limitation by performing dynamic model search on the fly (see the difference in functionality between [19] and the proposed method in Table 1). The difference is demonstrated in the experiments section.

2.3. Multi-task learning

Multi-task learning (MTL) [10] is a learning approach that learns the commonality and differences between associated tasks simultaneously to improve the performance of individual tasks. MTL is generally performed in a single shared network (i.e., hard parameter sharing [42,38]) or in multiple different networks according to the number of tasks with soft parameter sharing [43,30]. Most of the MTL approaches have focused on determining the relationships between tasks [44,30]. However, MTL approaches generally work on the assumption that tasks are related to one another. If tasks are less relevant, there will be a phenomenon, called destructive interference, among tasks [15–18]. Hence, learning all tasks through a single shared network may cause performance degradation. These issues can be mitigated by the proposed method in that the search agent dynamically selects a proper model for each task, making available a broad range of inference paths, not a single fixed path. Besides, unlike current MTL methods relying on dynamic search [15,17], the proposed method can provide a larger number of solutions with respect to different computational costs by incorporating the cost-adaptive learning.

3. The proposed method

3.1. Cost-adaptive learning

We first discuss the problem of cost-adaptive learning for a *single task* performing multiple inference operations with respect to different numbers of parameters (costs). To achieve this, we construct a network-in-network hierarchical (or nested) structure, following the strategy in [41]. Assume that a network consists of k disjoint subsets by splitting network parameters $\mathscr{W}$ into k subsets, i.e., $\mathscr{W} = [W_1, W_2, \ldots, W_k]$. Based on these subsets, we construct a hierarchical structure in a way that the i-th level of hierarchy ($i \geqslant 2$) contains the subsets in the $(i-1)$-th level and one additional subset. The hierarchical structure is illustrated in Fig. 2. In this work, the number of levels of hierarchy, denoted as n_h, is the same as the number of subsets, i.e., $n_h = k$. The structure allows n_h inference paths for n_h different computational requirements.

Given a dataset $\mathscr{D}$ consisting of image-label pairs and the number of levels of hierarchy n_h, the set of parameters $\mathscr{W}$ can be optimized by the sum of n_h loss functions

$$\min_{\mathscr{W}} \sum_{i=1}^{n_h} \mathscr{L}\left(h^i(\mathscr{W}); \mathscr{D}\right), \tag{1}$$

where $h^i(\mathscr{W})$ is a function that collects one or more subsets of $\mathscr{W}$ corresponding to the i-th level of hierarchy. The function has a constraint such that a higher level set includes a lower level set, i.e.,

$$h^l(\mathscr{W}) \subseteq h^m(\mathscr{W}), \quad l \leqslant m, \quad \forall l, \quad m \in [1, \ldots, n_h], \tag{2}$$

in a structure that shares parameters [41]. $\mathscr{L}(\cdot)$ is a standard loss function (e.g. cross-entropy) of a network associated with $\mathscr{D}$. The parameters $\mathscr{W}$ are learned by n_h losses determined by $h^i(\cdot)$. By solving (1), a learned network that has n_h inference paths (corresponding to n_h computational costs) is collected.

Note that the problem addressing different computational costs in (1) can be applied to a single task (i.e., a dataset $\mathscr{D}$). If we

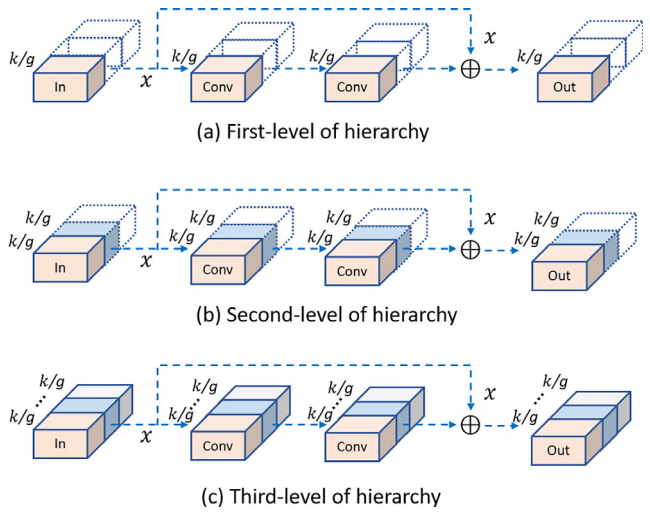

Fig. 2. A graphical illustration of a hierarchical (nested) structure in a residual module. Suppose that each layer has three blocks by splitting the filters into three blocks (subsets), i.e., $g = 3$, which is the same as the number of levels of hierarchy. Based on the blocks, we construct a three-level hierarchical structure in a way that the i-th level of hierarchy ($i \geqslant 2$) contains the subsets in the $(i-1)$-th level and one additional subset: (a) first-level of hierarchy containing a single (yellow) block, (b) second-level of hierarchy containing the (yellow) block of low-level and an additional (blue) block, and (c) third-level of hierarchy containing the (yellow and blue) blocks of middle-level and an additional (white) block, which is equivalent to the set containing all the blocks.

perform multiple tasks (or evaluate multiple datasets) while simultaneously providing cost-adaptive solutions for each task, it is inevitable to introduce task-specific individual networks. This will naturally entail memory increment due to the increasing number of networks with training-time trials-and-errors.

3.2. Model reshuffling to avoid destructive interference

Addressing both multiple tasks and different computational requirements within a single learned architecture would make the method highly efficient. Since the method tries not to increase the number of additional parameters, it is most likely to deploy a fixed number of parameters in an architecture to achieve the goal. A phenomenon when dealing with multiple tasks called destructive interference [16], which arises when tasks are of limited relevance to one another, destroying the learning efficiency in the multi-task learning regime, is also considered.

One possible strategy for dealing with this problem is to reshuffle the model structure (i.e., filters and the parameters) [19]. For example, the order of filters can be shuffled [45] while the entire structure of an architecture is preserved, rather than using the fixed order of the filters. This can not only distribute the importance of the parameters and increase the network capacity but also tailor the structure to each task (by assigning a reshuffled structure to a task). Thus, it will reduce the potential negative interference arising when multiple tasks are addressed simultaneously, due to the different network configurations and thus different inference flows. Note that unlike random shuffling during training in ShuffleNet [45], we reshuffle the structures guided by the search agent before learning the architecture and fix them during training and testing.

In order to do this, we first define a basic building block that is constructed by gathering a small number of convolution filters from each residual module in a backbone architecture. If we select a building block in every module across all the residual modules, this constitutes a small-scale network whose width is smaller than that of the backbone (see Fig. 2(a)). From this, we produce a redesigned backbone that contains a number of building blocks. By the use of building blocks, we obtain outputs varying in accuracy and computational cost. Then, we assemble the building blocks to construct network models of different configurations in a way that each model has a different order of building blocks. If we have g groups in every residual module, each network has a different order of the parameter groups among g^m possible configurations, where m is the number of residual modules.

Each model of a unique configuration corresponds to a virtual network developed using building blocks whose parameters are ordered differently from those of other models. Fig. 3 gives an illustration of virtual networks from a backbone architecture. Hence, each virtual network holds a unique model configuration and is specialized for a task. By reshuffling, we can produce multiple virtual models of different inference paths in a single fixed architecture, and each building block has different importance for different tasks. This is achieved by optimizing the following loss function:

$$\min_{\mathscr{W}} \sum_{j=1}^{k} \sum_{i=1}^{n_h} \mathscr{L}\left(h^{i,j}(\mathscr{W}); \mathscr{D}^j\right), \tag{3}$$

where k is the number of tasks and $\mathscr{D}^j$ denotes the dataset of the j-th task. $h^{i,j}(\cdot)$ is a function that determines the model configuration (or order of building blocks) for the i-th task and the j-th computation cost, and it has a constraint similar to (2) such that

$$h^{l,j}(\mathscr{W}) \subseteq h^{m,j}(\mathscr{W}), \quad l \leqslant m, \quad \forall l, \quad m \in [1, \ldots, n_h] \text{ and } j \in [1, \ldots, k]. \tag{4}$$

After training the framework by (3) and (4), we have $k \times n_h$ network models (and outputs) corresponding to tasks and computational costs having only consumed the memory of a single architecture.

3.3. Auto-VirtualNet: incorporating dynamic search

Now the question arises how to learn the function $h^{i,j}$ in (3), i.e., how to determine an optimal model structure of a virtual network for each task. The possible number of model configurations (or candidate models) is $(b+1)^m$, where b is the number of building blocks for each residual module and m is the number of residual modules. Recent work applies a rule-based method to produce virtually generated models [19], which has the search space of $k!(= k \times (k-1) \times \ldots \times 1)$. This determines configurations manually without knowledge of tasks, probably leading to a sub-optimal solution. More importantly, it does not reflect the relationship between tasks. Thus, it is required to explore an optimal model structure without relying on heuristics or expert knowledge and reflect task relationships.

To this end, we propose a new learning framework to discover optimal models dynamically for different tasks. We introduce a search agent $\mathscr{S}(\cdot, \mathscr{W}^s)$ (which is modeled as a neural network) to explore a pool of candidate models in the backbone architecture (see Fig. 1), where $\mathscr{W}^s$ denotes the set of parameters of the search agent. Since we have the search space of $(b+1)^m$ (which is larger than that of [19]), the chance of discovering a high-performing model will increase. The presented dynamic architecture search reduces to a problem of selecting building blocks in the backbone. Selecting building blocks is equivalent to producing a set of binary outputs from the search agent. By the proposed search mechanism, the proposed method gives a bespoke solution for each task, potentially avoid interference when learning multiple tasks.

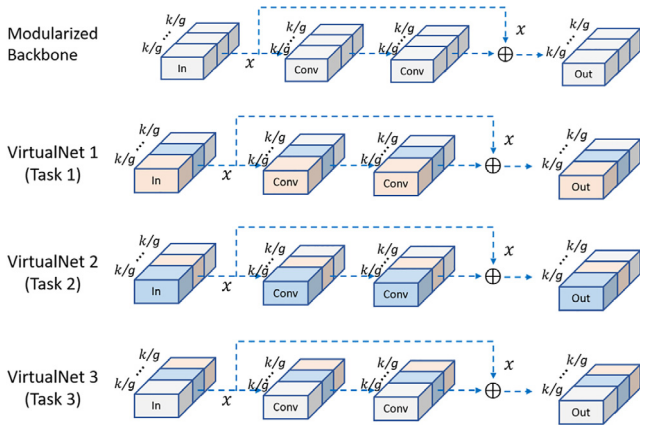

Fig. 3. An example of virtual networks (VirtualNets) produced by a modularized backbone architecture. Each VirtualNet has its unique configuration based on the order of the building blocks (the residual modules in each VirtualNet contain the same building blocks, represented by the same colors, in different orders) and is specialized for a task. A VirtualNet also has a hierarchical structure for different computational costs as shown in Fig. 2.

To represent a model in the backbone, we design an output consisting of real values generated from the search agent (network) such that

$$\mathscr{S}(D;\mathscr{W}^s) \in [0,1]^{(b+1)\times m}, \tag{5}$$

where D is an input. Here, $(b+1)$ denotes the number of building blocks and an additional case when nothing is selected (equivalent to a skip connection [2]). We have a constraint that the sum of the probability values in each module is set to one. From the output $\mathscr{S}(D;\mathscr{W}^s)$, we obtain the model configuration $\mathscr{M}$

$$\mathscr{M} = [m_{ij}] \in \{0,1\}^{(b+1)\times m} \sim \mathscr{S}(D;\mathscr{W}^s), \tag{6}$$

where $m_{ij} = 0$ or 1 indicates whether the building block is used to build a model or not.

To provide cost-adaptive solutions, the search agent simultaneously selects multiple model structures of different computation budgets. This can be achieved by applying different sparsity regularizations in the objective function. Suppose that a set of datasets $\mathscr{D}$ consists of multiple datasets for k tasks, i.e., $\mathscr{D} = \{D_1, D_2, \ldots, D_k\}$, where D_i is the i-th (task) dataset. The proposed method optimizes two sets of parameters, $\mathscr{W}^b$ and $\mathscr{W}^s$, for the backbone (producing a collection of models) and the search agent, respectively, from the following objective function:

$$\min_{\mathscr{W}^b,\mathscr{W}^s} \mathbb{E}_{D_t\in\mathscr{D},\mathscr{M}^i\sim\mathscr{S}(D_t;\mathscr{W}^s)} \sum_{i=1}^{n_h} \mathscr{L}(g(D_t;\mathscr{W}^b,\mathscr{M}^i)) + \mathscr{R}_i(\mathscr{M}^i), \tag{7}$$

where $\mathscr{L}$ denotes a loss function and g represents the model determined by the structure $\mathscr{M}^i$. $\mathscr{R}_i$ is the i-th sparsity regularizer that gives the average number of parameters of a residual module as

$$\mathscr{R}_i(\mathscr{M}^i) = \rho_i \cdot \left(\frac{1}{m}\sum_{j=1}^{m}\frac{p_j(\mathscr{M}^i)}{t_j}\right)^2, \tag{8}$$

where $p_j(\cdot)$ gives the number of parameters of the building blocks selected by $\mathscr{M}^i$ in the j-th residual module and t_j is the total number of parameters in the j-th residual module. ρ_i is a weighting factor. By learning the objective function with different sparsity regularizers $\mathscr{R}_i$, we can obtain multiple models of different computational costs for each task.

We apply the policy gradient method [46], which approximates computing the exact expected value, to learn the sets of parameters. Note that the learning step is similar to a single-step Markov decision process (MDP) [24] in that it discovers all model structures (corresponding to actions) at a single step based on the loss function (corresponding to reward). We update one set of parameters until convergence while the other set is held fixed and then vice versa, in an alternating optimization.

4. Experiments

4.1. Setup

The proposed method, which we name *Auto-VirtualNet*, was applied to several multi-task learning problems with different numbers of tasks. In this work, we focused on classification tasks using popular benchmark datasets. Our method was performed on five datasets: CIFAR-100 [47], Tiny-ImageNet,[1] STL-10 [48], Mini-ImageNet [49], and CIFAR-MTL [15].

We applied three multi-task learning scenarios to demonstrate the performance of the proposed method, from few-task to many-task learning. The first scenario is a three-task learning problem containing three different datasets, CIFAR-100, Tiny-ImageNet, and STL-10, where each dataset is assigned to a single task. The scenario addresses different image scales (from 32^2 to 96^2) and numbers of classes (from 10 to 200). The second scenario is based on CIFAR-MTL, the multi-task version of CIFAR-100, which contains 20 super-classes, each of which has five sub-classes. We treated each super-class as a separate task, so the second scenario consists of 20 tasks. The last scenario is a 10-task learning problem using the Mini-ImageNet dataset. For this dataset, we followed the procedure in [15] of randomly choosing 50 classes and then distributing them evenly between 10 tasks. Due to the randomness of results for this scenario, we generated 10 different sets of 50 classes to obtain the average performance per task for this dataset. Note that the scenarios mentioned above contain different tasks (and classes) where some are not compatible when learning them simultaneously due to different knowledge of classes. This will cause harmful interference and degrade the learning efficiency.

The proposed method was compared with recent multi-task learning approaches: Cross-Stitch network [30], Routing network (RoutingNet) [15], PackNet [38], NestedNet [41], deep virtual network (VirtualNet) [19], and deep elastic network (DEN) [17]. Due to the nature of cost-adaptive learning, the proposed method and VirtualNet produce n_h times larger solutions than the other methods compared. Experimental results of PackNet, NestedNet, and VirtualNet for the first scenario were taken from [19], and results of Cross-Stitch network and RoutingNet for the latter two scenarios were taken from the report in [15]. The result of DEN was taken from the paper. We evaluated the proposed method for all experiments using the same architecture to the compared methods under similar experimental conditions for a fair comparison.

4.2. Implementation details

We take two convolutional neural networks of small and large scales, respectively. For the first scenario, the proposed method is based on a residual network [2], and we follow the common practice to design the architecture used for the Tiny-ImageNet dataset. For the latter two scenarios, we adopted a simple convolutional network introduced in [15] to compare with other multi-task learning approaches under the same architecture. When constructing the architectures, we do not have groups for fully-connected layers throughout all the experiments.

The search agent is a small convolutional neural network with three conventional residual blocks and one fully-connected layer

[1] https://tiny-imagenet.herokuapp.com/.

and was applied for all experiments. The memory increment taken up by the search agent is smaller than those of the backbone networks. The proposed method first pretrains the modularized backbone network, similar to a strategy in multi-task learning where multiple tasks are learned in a single shared architecture [10]. Then, we alternatively optimize the parameters of selected network models on a per-sample basis, and the search agent, until convergence. When we pretrain the backbone network, we randomly drop some building blocks to increase the learning capacity and prevent performance degradation of building block selection by the search agent. This follows the technique detailed in [50], and we find that this strategy improves the performance of the discovered models.

All the parameters of the compared methods were initialized with Xavier initialization [51]. The backbone architecture, including selected models, was trained by the SGD optimizer with a Nesterov momentum of 0.9, and the search agent was trained by the Adam optimizer [52]. Standard weight decay was applied while training the selected models. The batch size of 64 was applied for the first scenario, where the initial learning rate was 0.1 and decreased by 10 when converging. For the rest of the scenarios, we set the batch size to 256, and the initial learning rate was 0.01 and was reduced by 10. To allow dynamic exploration of candidate models from the policy gradient method when learning the proposed method, we applied the epsilon-greedy algorithm [53].

4.3. Results

4.3.1. Scenario 1

We applied the methods to three datasets with different input scales from 32×32 to 96×96 and different numbers of classes from 10 to 200. We compared Auto-VirtualNet with other recently proposed multi-task learning approaches, including single-task learning (learning an individual architecture for each task) and multi-task learning (learning a single shared architecture for all tasks) baselines. One major competitor, VirtualNet, performs cost-adaptive learning while learning multiple tasks. According to the strategy, the number of budgets is the same as the number of tasks, and the proposed method learns with the same number of budgets. Unlike VirtualNet, however, the proposed has the flexibility to increase the number of models regardless of the number of tasks. Moreover, it can address task interference more effectively due to its automatic search nature. We produced three different models for each task selected by the search agent. Note here that even if Auto-VirtualNet adjusts the parameter budget from the search agent, it does not fix the structure manually but produces models from the search agent. Thus, depending on the task's difficulty, the search agent automatically explores an optimal model between the performance and the computational cost.

Table 2 shows the experimental results for the first scenario on classification accuracy and the corresponding number of parameters. The single-task learning baseline achieves decent accuracy, but it requires three times more parameters than other multi-task learning approaches to perform the same tasks. Whereas, the multi-task learning baseline performs less accurately than the single task learning baseline and the other methods, possibly due to the single fixed model structure. PackNet and NestedNet perform much better than the multi-task learning baseline while consuming fewer parameters for the first two tasks. They even perform better than the single task learning counterpart on average. Auto-VirtualNet and VirtualNet each give three solutions per task according to the number of computation budgets. Due to the automatic proposal of optimal architectures, Auto-VirtualNet performs better than VirtualNet on average under similar budgets (especially for the Tiny-ImageNet and STL-10 tasks). From the results, we can see that incorporating the automatic architecture search is beneficial in the multi-task learning regime, which can also alleviate potential interference risks between tasks.

4.3.2. Scenario 2

For learning a larger number of tasks, we applied a 20-task learning problem using the CIFAR-MTL dataset. In this scenario, we compared Auto-VirtualNet with other multi-task learning approaches: Cross-Stitch network and RoutingNet, both of which can address a large number of tasks. Note that the approaches compared in the first scenarios, PackNet, NestedNet, and VirtualNet, are not configured to learn many tasks due to their limited capacities (i.e., the number of parameters for each task decreases significantly as the number of tasks increases, resulting in performance degradation). In order to compare with a multi-task learning approach that can choose a proper model automatically from a pool of candidate models, RoutingNet was chosen for comparison. Note that Cross-Stitch network and RoutingNet require a larger number of parameters than the baseline and Auto-VirtualNet, whereas the latter does not increase the model size and does not introduce multiple parallel models. Auto-VirtualNet produces three models of different numbers of parameters from a single training process, whereas the compared approaches have a single model for each task.

The experimental results of the scenario are summarized in Table 3. Since we have 20 tasks in the scenario, the results are obtained by taking the average results of the tasks. The multi-task baseline performs poorly compared to other frameworks due to its fixed structure. It does not appear to address many tasks sufficiently. RoutingNet performs better than the baseline and Cross-Stitch network owing to its routing mechanism. The proposed method performs much better than the others by a large margin while consuming fewer parameters. Auto-VirtualNet has 26.9% higher accuracy while all its model sizes are much smaller than that of RoutingNet. The performance gap may be from the difference in the search process; while RoutingNet locally applies the router to select a sequence of function blocks iteratively, the search agent in Auto-VirtualNet globally produces an entire model structure at a time. Thus, the proposed global search will reduce the time to search for a sequence of blocks and avoid getting stuck in a local solution. Note also that Auto-VirtualNet produces a restructured model for each task (discovered by the search agent) while sharing some parameters for tasks, demonstrating that it can efficiently deal with commonality and differences among tasks.

4.3.3. Scenario 3

We further compared Auto-VirtualNet with other multi-task learning approaches for another scenario with a variety of classes using Mini-ImageNet. Mini-ImageNet is a subset of the ImageNet dataset with a smaller image scale, containing 100 classes. We used the three methods for comparison for this scenario, including a recently proposed model search approach, DEN. Here, Auto-VirtualNet produces two models of different parameter budgets. Similar to the previous tasks, Cross-Stitch network and RoutingNet involve a much larger number of parameters than the proposed.

Table 4 shows the average performance results. The compared methods have similar trends to the previous scenario, and the routing mechanism-based method outperforms the baseline and Cross-Stitch network. However, the proposed method performs better than all the other methods compared, while using fewer parameters (the smaller parameter model from Auto-VirtualNet performs with 5.9% better accuracy than RoutingNet while consuming less than half of the number of parameters). If we choose a larger model, the performance gap is more significant. The proposed method performs better than DEN with a 2.3% margin while requiring less than half of the number of parameters. We also note that the proposal's performance variation according to the

Table 2
Results of the first scenario containing three datasets (3 tasks). Baseline (Single) denotes an individual learning approach for different tasks and Baseline (Multi) is a multi-task learning method of the tasks using a single shared architecture with multiple task-specific outputs. No. parameters is the number of parameters used to perform each task. Since VirtualNet and Auto-VirtualNet provide a wide range of solutions regarding different numbers of parameters (costs), we show the results for three parameter budgets.

	Task 1		Task 2		Task 3		
	Accuracy	Params	Accuracy	Params	Accuracy	Params	Total Params
Baseline (Single)	72.7%	29.8 M	55.8%	29.8 M	71.5%	29.8 M	89.4 M
Baseline (Multi)	58.6%	29.8 M	45.5%	29.8 M	70.7%	29.8 M	29.8 M
PackNet [38]	69.6%	7.5 M	54.1%	16.7 M	73.9%	29.8 M	29.8 M
NestedNet [41]	71.9%	7.5 M	55.5%	16.7 M	74.3%	29.8 M	29.8 M
VirtualNet [19]	73.2%	7.5 M	55.5%	7.5 M	76.6%	7.5 M	29.8 M
	74.0%	16.7 M	57.9%	16.7 M	77.4%	16.7 M	
	74.5%	29.8 M	58.8%	29.8 M	77.9%	29.8 M	
Auto-VirtualNet	73.6%	15.2 M	58.6%	19.1 M	81.6%	10.7 M	29.8 M
	74.0%	18.2 M	58.9%	21.5 M	82.0%	12.5 M	
	74.2%	21.5 M	59.3%	24.7 M	82.5%	19.1 M	

Table 3
Average results on CIFAR-MTL (20 tasks). The proposed method shows results for three different numbers of parameters.

	Accuracy	No. parameters
Baseline (Multi-Task)	42.0%	74 K
Cross-Stitch network [30]	54.0%	>1.5 M
Routing network [15]	61.0%	>74 K
Auto-VirtualNet (Proposed)	87.9%	25 K
	88.1%	35 K
	88.2%	43 K

Table 4
Average results on Mini-ImageNet (10 tasks). The proposed method shows results for two different numbers of parameters.

	Accuracy	No. parameters
Baseline (Multi-Task)	51.0%	140 K
Cross-Stitch network [30]	56.0%	>1.4 M
Routing network [15]	59.0%	>140 K
DEN [17]	62.6%	140 K
Auto-VirtualNet (Proposed)	64.9%	59 K
	65.2%	82 K

variation of the number of parameters is not significant, making this method highly efficient.

4.3.4. Analysis

Since we propose a neural architecture search method containing a search agent, it is crucial to see how the search agent discovers a network architecture for each instance. Again, the proposed method explores the search space and selects a proper model on a per-sample basis. Given a query image and its corresponding network architecture, we collected five images whose model structures are similar to that of the query image. Note that similar images can share the same class, similar shapes, or similar textures, and the search agent will capture their similarity. This reveals that the search agent can select a similar model structure for a similar image. Fig. 4 shows the results of a query (left standalone image) and five similar images in every row. From the results, we can see that the search agent is able to propose a structure for similar images without any prior knowledge of the images. Most similar images are from the same classes as the query images (first to fourth rows), but the search agent also associates similar images of a similar shape if no class is provided (last row). The search agent automatically learns the relation of samples and searches for optimal models accordingly.

We further analyzed the model structures discovered by the proposed method for different parameter budgets. We show nine

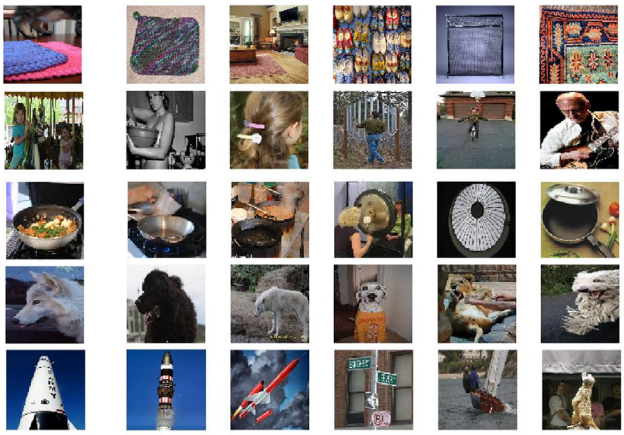

Fig. 4. Given a query image (left) from Tiny-ImageNet and its corresponding model structure produced by the search agent, we show five images whose model structures are similar to that of the query image for each row.

different model structures provided by the search agent for the first scenario (three datasets × three budgets) in Fig. 5. Each subfigure corresponds to an output of the search agent where it can search an optimal number of building blocks (convolution groups) for each residual module. If it chooses nothing, it means the module is skipped. As shown in the figure, for a higher (lower) parameter budget, the search agent selects a larger (smaller) number of building blocks. The model of a higher budget usually contains most of the building blocks of the models of lower budgets, resulting in smooth performance variation, as shown in the previous results. Using the simple strategy to construct different model sizes using different regularizers in (8), we can efficiently train the search agent. Note that it also explores different model structures (shown in different rows) for different tasks. This will avoid any potential risks of task interference due to the bespoke solution for each task.

5. Conclusion

In this study, we have proposed *Auto-VirtualNet*, a dynamic multi-model search approach to explore multiple models with respect to different tasks and computational budgets. To achieve the goal with a restricted backbone architecture size, the search agent discovers reshuffled structures sharing the same parameters. This can significantly reduce the required number of parameters to search for an optimal model compared to existing approaches. A modularization strategy is applied to introduce basic building

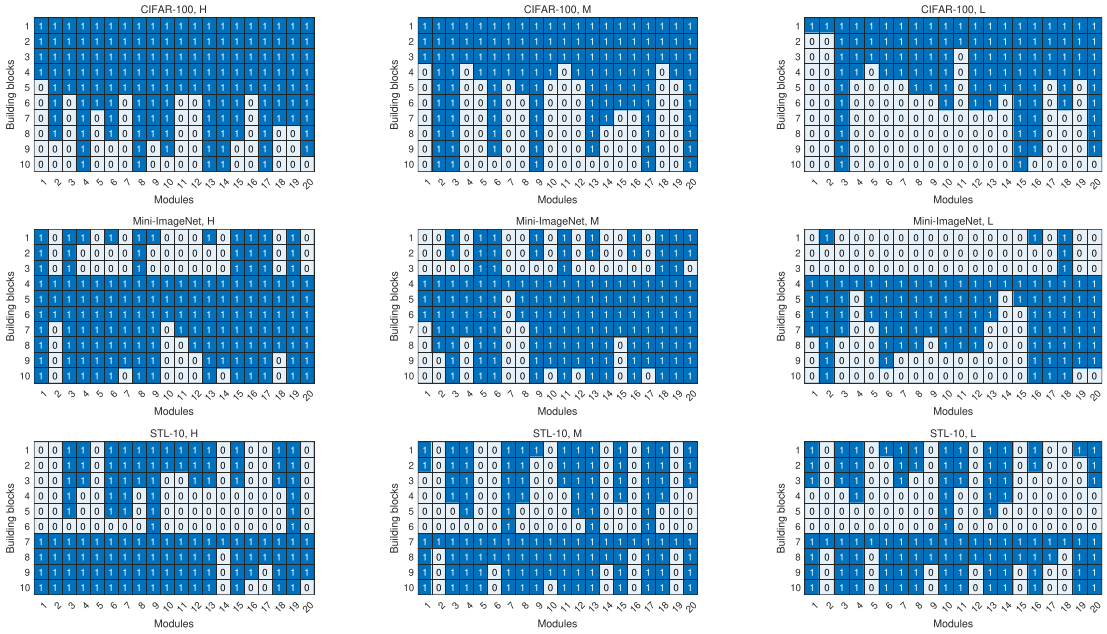

Fig. 5. Selected model structures with respect to different computational costs and tasks on the first scenario. For each subfigure, y-axis denotes building blocks and x-axis denotes the number of residual modules. We represent selected building blocks as 1, otherwise 0 (i.e., a larger number of 1's means the chosen model contains more building blocks). H, M, and L denote high-, medium-, and low-parameter budgets, respectively.

blocks and constructed a nested structure based on those building blocks for smooth performance variation. The proposed method also produces multiple models in a single learning framework with an off-the-shelf architecture, instead of having numerous learning frameworks. Experimental results show that the proposed method performs competitively compared to other multi-task learning methods while allowing more diverse inferences.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2020R1F1A1048332), and in part by the ICT R&D program of MSIT/IITP [2020-0-00857, Development of Cloud Robot Intelligence Augmentation, Sharing and Framework Technology to Integrate and Enhance the Intelligence of Multiple Robots].

References

[1] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[2] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, CVPR (2016).
[3] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, MobileNets: efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861..
[4] G. Huang, Z. Liu, K.Q. Weinberger, L. van der Maaten, Densely connected convolutional networks, CVPR (2017).
[5] T. Young, D. Hazarika, S. Poria, E. Cambria, Recent trends in deep learning based natural language processing, IEEE Comput. Intell. Mag. 13 (3) (2018) 55–75.
[6] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: towards real-time object detection with region proposal networks, NIPS (2015).
[7] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: unified, real-time object detection, CVPR (2016).
[8] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, NIPS (2015).
[9] P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning convolutional neural networks for resource efficient inference, ICLR (2017).
[10] R. Caruana, Multitask learning, Mach. Learn. 28 (1) (1997) 41–75.
[11] L. Nie, L. Zhang, Y. Yang, M. Wang, R. Hong, T.-S. Chua, Beyond doctors: future health prediction from multimedia and multimodal observations, in: Proceedings of the 23rd ACM International Conference on Multimedia, 2015.
[12] X. Song, L. Nie, L. Zhang, M. Liu, T.-S. Chua, Interest inference via structure-constrained multi-source multi-task learning, IJCAI (2015).
[13] S. Ruder, An overview of multi-task learning in deep neural networks, arXiv preprint arXiv:1706.05098..
[14] A. Kendall, Y. Gal, R. Cipolla, Multi-task learning using uncertainty to weigh losses for scene geometry and semantics, CVPR (2018).
[15] C. Rosenbaum, T. Klinger, M. Riemer, Routing networks: adaptive selection of non-linear functions for multi-task learning, ICLR (2017).
[16] X. Zhao, H. Li, X. Shen, X. Liang, Y. Wu, A modulation module for multi-task learning with applications in image retrieval, ECCV (2018).
[17] C. Ahn, E. Kim, S. Oh, Deep elastic networks with model selection for multi-task learning, in: ICCV, 2019..
[18] X. Sun, R. Panda, R. Feris, K. Saenko, Adashare: learning what to share for efficient deep multi-task learning, NeurIPS (2020).
[19] E. Kim, C. Ahn, P.H. Torr, S. Oh, Deep virtual networks for memory efficient inference of multiple tasks, CVPR (2019).
[20] R. Girshick, Fast R-CNN, in: ICCV, 2015..
[21] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, arXiv preprint arXiv:1611.01578..
[22] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition, CVPR (2018).
[23] T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: a survey, arXiv preprint arXiv:1808.05377..
[24] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L.S. Davis, K. Grauman, R. Feris, Blockdrop: dynamic inference paths in residual networks, CVPR (2018).
[25] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, J.E. Gonzalez, Skipnet: learning dynamic routing in convolutional networks, ECCV (2018).
[26] A. Veit, S. Belongie, Convolutional networks with adaptive inference graphs, ECCV (2018).
[27] J. Lin, Y. Rao, J. Lu, J. Zhou, Runtime neural pruning, NIPS (2017).
[28] S. Zagoruyko, N. Komodakis, Wide residual networks, BMVC (2016).
[29] J. Smith, R. Nair, Virtual Machines: Versatile Platforms for Systems and Processes, Elsevier, 2005.
[30] I. Misra, A. Shrivastava, A. Gupta, M. Hebert, Cross-stitch networks for multi-task learning, CVPR (2016).
[31] H. Liu, K. Simonyan, Y. Yang, Darts: differentiable architecture search, ICLR (2018).
[32] H. Liu, D. Li, J. Peng, Q. Zhao, L. Tian, Y. Shan, Mtnas: search multi-task networks for autonomous driving, ACCV (2020).
[33] H. Pham, M.Y. Guan, B. Zoph, Q.V. Le, J. Dean, Efficient neural architecture search via parameter sharing, in: ICML, 2018..
[34] H. Zhou, J.M. Alvarez, F. Porikli, Less is more: towards compact cnns, ECCV (2016).
[35] H. Li, A. Kadav, I. Durdanovic, H. Samet, H.P. Graf, Pruning filters for efficient convnets, ICLR (2017).

[36] X. Yu, T. Liu, X. Wang, D. Tao, On compressing deep models by low rank and sparse decomposition, CVPR (2017).
[37] A.R. Zamir, T.-L. Wu, L. Sun, W.B. Shen, B.E. Shi, J. Malik, S. Savarese, Feedback networks, CVPR (2017).
[38] A. Mallya, S. Lazebnik, PackNet: adding multiple tasks to a single network by iterative pruning, CVPR (2018).
[39] A. Mallya, D. Davis, S. Lazebnik, Piggyback: adapting a single network to multiple tasks by learning to mask weights, ECCV (2018).
[40] I. Kokkinos, Ubernet: training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory, CVPR (2017).
[41] E. Kim, C. Ahn, S. Oh, NestedNet: learning nested sparse structures in deep neural networks, CVPR (2018).
[42] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, T. Darrell, Decaf: a deep convolutional activation feature for generic visual recognition, in: ICML, 2014..
[43] Y. Yang, T.M. Hospedales, Trace norm regularised deep multi-task learning, arXiv preprint arXiv:1606.04038..
[44] B. Jou, S.-F. Chang, Deep cross residual learning for multitask visual recognition, in: Proceedings of the 24th ACM International Conference on Multimedia, ACM, 2016.
[45] X. Zhang, X. Zhou, M. Lin, J. Sun, Shufflenet: an extremely efficient convolutional neural network for mobile devices, CVPR (2018).
[46] R.S. Sutton, D.A. McAllester, S.P. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, NIPS (2000).
[47] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, Tech. rep., University of Toronto (2009).
[48] A. Coates, H. Lee, A.Y. Ng, An analysis of single layer networks in unsupervised feature learning, AISTATS (2011).
[49] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al., Matching networks for one shot learning, NIPS (2016).
[50] G. Huang, Y. Sun, Z. Liu, D. Sedra, K.Q. Weinberger, Deep networks with stochastic depth, ECCV (2016).
[51] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, AISTATS (2010).
[52] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, ICLR (2014).
[53] M. Tokic, Adaptive *varepsilon*-greedy exploration in reinforcement learning based on value differences, in: Annual Conference on Artificial Intelligence, 2010..

Eunwoo Kim received the B.S. degree in Electrical and Electronics Engineering from Chung-Ang University, Seoul, Korea, in 2011, and the M.S. and Ph.D. degrees in Electrical Engineering and Computer Science from Seoul National University, Seoul, Korea in 2013 and 2017, respectively. He is currently an assistant professor with the School of Computer Science and Engineering, Chung- Ang University, Seoul, Korea. From 2017 to 2018, he was a postdoctoral researcher with the Department of Electrical Engineering and Computer Science, Seoul National University, Seoul, Korea. From 2018 to 2019, he was a postdoctoral researcher with the Department of Engineering Science, University of Oxford, Oxford, UK. His research interests include machine learning, deep learning, subspace representation, and computer vision.

Chanho Ahn received the B.S. degree in electrical and computer engineering from Seoul National University, Seoul, Korea, in 2016. He is currently a Ph.D. student in the Department of Electrical and Computer Engineering at Seoul National University, Seoul, Korea. His research interests include machine learning, multi-task learning, continual learning, and computer vision.

Songhwai Oh received the B.S. (Hons.), M.S., and Ph.D. degrees in electrical engineering and computer sciences from the University of California, Berkeley, CA, USA, in 1995, 2003, and 2006, respectively. He is currently a Professor with the Department of Electrical and Computer Engineering, Seoul National University, Seoul, Korea. Before his Ph.D. studies, he was a Senior Software Engineer at Synopsys, Inc., Mountain View, CA, USA, and a Microprocessor Design Engineer at Intel Corporation, Santa Clara, CA, USA. In 2007, he was a Post-Doctoral Researcher with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, USA. From 2007 to 2009, he was an Assistant Professor of Electrical Engineering and Computer Science in the School of Engineering, University of California, Merced, CA, USA. His current research interests include robotics, computer vision, cyber-physical systems, and machine learning.