# ElasticDNN

## On-Device Neural Network Remodeling for Adapting Evolving Vision Domains at Edge

Zhang, Qinglong; Han, Rui; Liu, Chi Harold; Wang, Guoren; Chen, Lydia Y.

**DOI**
10.1109/TC.2024.3375608

**Publication date**
2024

**Document Version**
Final published version

**Published in**
IEEE Transactions on Computers

# ElasticDNN: On-Device Neural Network Remodeling for Adapting Evolving Vision Domains at Edge

Qinglong Zhang, Rui Han, Chi Harold Liu, *Senior Member, IEEE*, Guoren Wang, *Senior Member, IEEE*, and Lydia Y. Chen, *Senior Member, IEEE*

***Abstract*—Executing deep neural networks (DNN) based vision tasks on edge devices encounters challenging scenarios of significant and continually evolving data domains (e.g. background or subpopulation shift). With limited resources, the state-of-the-art domain adaptation (DA) methods either cause high training overheads on large DNN models, or incur significant accuracy losses when adapting small/compressed models in an online fashion. The inefficient resource scheduling among multiple applications further degrades their overall model accuracy. In this paper, we present ElasticDNN, a framework that enables online DNN remodeling for applications encountering evolving domain drifts at edge. Its first key component is the master-surrogate DNN models, which can dynamically generate a small surrogate DNN by retaining and training the large master DNN's most relevant regions pertinent to the new domain. The second novelty of ElasticDNN is the filter-grained resource scheduling, which allocates GPU resources based on online accuracy estimation and DNN remodeling of co-running applications. We fully implement ElasticDNN and demonstrate its effectiveness through extensive experiments. The results show that, compared to existing online DA methods using the same model sizes, ElasticDNN improves accuracy by 23.31% and reduces adaption time by 35.67x. In the more challenging multi-application scenario, ElasticDNN improves accuracy by an average of 25.91%.**

***Index Terms*—Edge vision, deep neural networks, domain adaptation.**

## I. Introduction

TREMENDOUS success of deep neural networks (DNN) has been demonstrated on vision applications executed on edge devices [1], e.g. image recognition [2], semantic segmentation [3], and object detection [4]. These DNNs are pre-trained using massive labeled data (called *source domain*) in cloud, and the *compressed* small DNNs are deployed on edge devices for various vision tasks. The input data stream on edge devices usually comes with a continually changing distribution, known as **evolving domain shifts** [5], which may cause a significant accuracy degradation of the deployed DNN. For instance, when an automatic vehicle moves to different locations such as residential area, city center and rural area, different new domains with diverse backgrounds, features, and subpopulations differing from the source domain may appear. Adapting and retraining DNNs in accordance to the evolution of *target domains* faces several challenges. First of all, such model retraining needs to be executed *on device*, due to communication and data privacy issues [6]. Second, input data on edge devices has unknown (or unpredictable) pattern and unfortunately unlabeled [6]. Last but not least, the most relevant and realistic execution scenario is that **multiple vision applications** are concurrently deployed on a single edge device whose computation resources, e.g. GPU cycles, can only be provided in a limited fashion. All in all, retraining multiple DNN applications on edge devices needs to consider the trade-off among: evolution of domain shifts, accuracy requirements, and available GPU resources across applications' DNNs.

**Motivation**. We use vision applications of image classification (ResNet56 [2]) and object detection (YOLOV3 [7]) to highlight the challenges of running concurrent domain adaptation (DA), shown in Fig. 1's example. We analyze existing model adaptation techniques and gain insights on their limitations on a resource-constrained edge device (NVIDIA Jetson AGX Orin).

**Limitation 1: Accuracy-overhead dilemma in a single application**. When facing evolving non-stationary vision domains, e.g. can be high up to 30 target domains in Fig. 1, current online DA methods can be divided into three types: (1) they train compressed/small models using feature alignment [8]; (2) they only tune batch normalization (BN) layers in original/large models [9]; and (3) they first retrain original models and then distill and retrain small models from them [6]. Fig. 1(a) and 1(b) illustrate these methods' model accuracies and retraining time. For the first two types of methods, only retraining small models or BN layers of large models results in low accuracies, because *the learning capacity of their backbone model tends to saturate easily when encountering drastically changing data domains*. Although the third type of methods achieves much higher accuracies, these methods incur much longer retraining time and become infeasible when having

Manuscript received 20 April 2023; revised 8 January 2024; accepted 21 February 2024. Date of publication 14 March 2024; date of current version 10 May 2024. This work was supported in part by the National Key R&D Program of China under Grant 2021YFB3301503, and in part by the National Natural Science Foundation of China under Grant 62272046, Grant 62132019, and Grant 61872337. Recommended for acceptance by D. Bertozzi. *(Corresponding author: Rui Han.)*

Qinglong Zhang, Rui Han, Chi Harold Liu, and Guoren Wang are with Beijing Institute of Technology, Beijing 100081, P.R. China (e-mail: hanrui@bit.edu.cn).

Lydia Y. Chen is with the TU Delft, 2628 Delft, The Netherlands.

Digital Object Identifier 10.1109/TC.2024.3375608

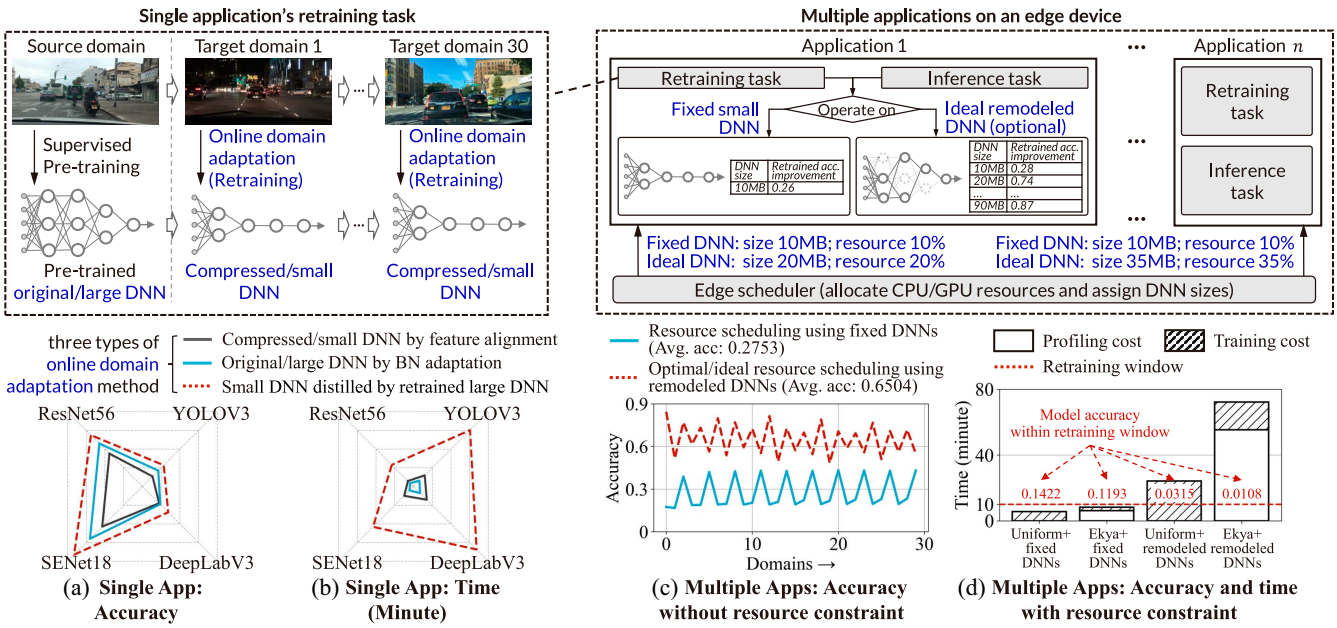


Fig. 1. Motivation examples in the single/multi-application scenarios to understand DA methods at edge.

limited retraining window at edge. Therefore, the **first challenge** lies in designing *an adaptation scheme that can dynamically adapt/retrain a large model into a small one while preserving the high accuracy for the incoming target domains on the fly*.

**Limitation 2: Inefficient resource scheduling among multiple co-running applications**. Current edge vision systems allocate an even amount of GPU resources to co-running vision applications, under an assumption that their DNN model sizes are *fixed* during the adaptation [4]. Shown by the prior art [1], [4], larger models have superior performance in DA than smaller ones, especially when handling difficult target domains, meanwhile requiring more computation resources. We demonstrate in Fig. 1(c) that compared to the optimal GPU cycle allocation, the accuracy degradation from even and fixed GPU cycle scheduling can be severe (up to 37.51%), because it does not factor in the difficulties of target domains across applications, i.e., changing applications' model sizes adapting to domain difficulties. However, existing remodeling techniques, i.e. adapting DNN models and retraining their weights, consume prohibitively high overheads and hence when applying them in edge devices, they adversely have the lowest accuracy (e.g. 0.0148 in Fig. 1(d)) due to the high profiling and retraining overheads. The **second challenge** thus is *how to support DNN remodeling with low overheads and leverage such remodeling to perform fine-grained resource scheduling across multiple co-running edge applications*.

In this work, we present ElasticDNN, a framework that proactively addresses the challenges of *evolving and drastic domain changes* and enables *online DNN remodeling* for DA on edge-based vision systems. The core of ElasticDNN is a master-surrogate framework which leverages the filters of a large pre-trained master DNN to adaptively generate and retrain small surrogate DNN on edge devices upon detecting domain shifts on the fly. When scheduling GPU cycles among multiple applications at runtime, ElasticDNN replaces *fixed* network adaptation with *domain-aware* network remodeling, and dynamically identifies and retrains a subset of filters that are most relevant to the current target domain's accuracy. The remodeling and retraining granularity of ElasticDNN is *the subset of filters* in the master DNN, which enables finer-grained resource allocation than the existing studies. As such, ElasticDNN is able to fully explore the advantages of high accuracy of the large master DNN and low overhead by using its small domain-specific surrogate DNN.

ElasticDNN presents the following novel techniques to fully unleash the optimal performance of co-running vision applications facing drastic domain changes:

- **Master-surrogate DNN adaptation framework** is proposed to jointly leverage the large learning capacity of the master DNN and the efficient model retraining and low inference latency in its compact surrogate DNNs. The framework enables dynamic architecture changes in the master DNN while preserving the trainability [10] when collaboratively training both master and surrogate DNNs.
- **Domain-specific DNN remodeling and retraining for single application** (the first challenge). Upon detecting a domain shift, it leverages the most accuracy-relevant filters of the master DNN to generate a domain-specific surrogate DNN online. We further design a novel collaborative training approach, which retrains only a small domain-sensitive region in the surrogate DNN and then updates the master DNN correspondingly.
- **Lightweight and filter-grained resource scheduling** (the second challenge). It is designed with the aim to further optimize the overall accuracy of model retraining among multiple applications and it is composed of two steps. First, ElasticDNN develops an online

estimator to quickly predict a DNN's accuracies before and after retraining for the target domain, and thus estimates the effectiveness of GPU allocations when retraining DNNs of different sizes. Second, ElasticDNN performs filter-grained scheduling such that the limited resources are first assigned to the filters contributing most to accuracy improvement.

**Summary of experimental results**. We fully implement ElasticDNN on top of TensorFlow Lite and PyTorch to support vision applications on both smart phones and embedded devices. We conduct extensive experiments against the state-of-the-art techniques to evaluate the performance of ElasticDNN, using image classification, semantic segmentation, and object detection scenarios, and six commodity edge devices (Huawei Mate20 X, Xiaomi 5S, and four types of Jetson devices). Each scenario has a mix of four to six prevalent vision benchmarks to represent the drastic and continuous domain changes in edge-based vision systems. The results demonstrate that: (1) in the single-application scenario, compared to baselines on the same model size, ElasticDNN improves the model accuracy by 23.31% while also achieving 35.67x reduction in adaptation time; compared to baselines with similar accuracy, ElasticDNN speeds up the adaptation by 144.70x, reduces inference latency and memory footprint by 2.55x and 4.14x, thus saving overall energy consumption by 7.63x on average, with only marginal accuracy drops of 0.63% to 2.70% (1.66% on average); (2) in the multi-application scenario, ElasticDNN improves the model accuracy by an average of 25.91% within the same retraining window; (3) ElasticDNN is applicable to different DNNs and DA algorithms, and its scheduling algorithm can be integrated into existing resource schedulers to improve model training performance.

## II. Background and Related Work

DNN-based edge vision system has been extensively studied in recent years for delivering low-latency applications, such as object detection [1] and video analytic [4]. Its basic paradigm is to pre-train large DNNs in the cloud, deploy *compressed* small DNNs [4] on resource-constrained edge devices for low inference latency, and continuously retrains the deployed models to mitigate accuracy degradation cased by two types of non-stationary input data distributions: domain shift and new taskes/classes. Specifically, incremental/continual learning techniques retrain a DNN to learn new tasks or classes using massive *labeled* data [11], and their major cost comes from labeling and storing samples of previous tasks. In contrast, DA techniques retrain a DNN to adapt to new target domains using *unlabeled* data. This work focuses on the later case and we discuss existing work from the perspective of single and multiple applications.

**DA techniques for a single application**. Traditional DA techniques are designed to adapt a model to *a single, stationary target domain*, using massive samples from the source and target domains [6]. They transfer well between two similar domains based on the assumption that target data is available during the source domain training. However, in edge computing scenarios, input data varies continuously and online DA methods are proposed to adapt to new domains using the fixed DNN networks without having their labeled data. Typical methods include: (i) *BN tuning techniques* fast tune the BN layers of a DNN model [9], [12]; (ii) *feature alignment techniques* align the feature (e.g. feature map or feature distribution) between source domain and target domain, using loss functions [8], [13], [14] and GAN [15]; (iii) *whole network training techniques* retrain all layers in a DNN, using synthesized image [3] and meta-learning [5]. As the DNN sizes are fixed for all three types of methods, they are limited in handling drastically evolving domain shifts, which are much more challenging.

**Resource scheduling techniques for multiple edge vision applications**. Allocating resources to co-running retraining and inference tasks on an edge device requires careful performance estimation. Ekya [4] performs online micro-profiling to estimate accuracy improvements under different hyper-parameters (e.g. training iterations) and resource allocations, thus finding the best solution to achieve the global optimal accuracy of all tasks. Similarly, RECL [16] monitors run-time training metrics and uses them to guide resource allocation. In retraining, RECL maintains a model zoo of previously trained models and selects the proper one for the current target domain. While those studies shed light the importance of allocating resources across co-running application, they are limited in dealing with learning applications with *fixed DNN*, which are shown insufficient to tackle drastic domain shifts.

We note that some recent techniques study the *neural architecture search (NAS)* for the underlying edge devices [17], [18]. They rely on labeled source domain data to pre-generate optimal DNN models for specific devices *offline*. In contrast, ElasticDNN needs no prior knowledge when generating a new network architecture at run-time. In addition, expandable networks are widely used in incremental/continual learning to solve such capacity limitation when learning new tasks/classes [11]. However, the training of expanded networks are based on massive labeled samples, which are not available in unsupervised DA scenarios.

## III. Design of ElasticDNN

### A. Overview

We design ElasticDNN to adapt DNN models to evolving domains on edge devices through the proposed feature of online DNN remodeling. As shown in Fig. 2, ElasticDNN is composed of five modules and they work together to address limitations/challenges 1 and 2 proposed in Section I.

**Online DNN remodeling and collaborative training for single application (Section III-B)**. The key enabling factor of ElasticDNN is the master-surrogate framework that enables both high accuracy and small model (challenge 1). This framework generates a new surrogate DNN upon detecting a domain shift. The surrogate model contains the master DNN's most accuracy-relevant filters to the new target domain detected. To achieve efficient DA on resource-constrained edge devices, ElasticDNN only updates a subset of filters that are

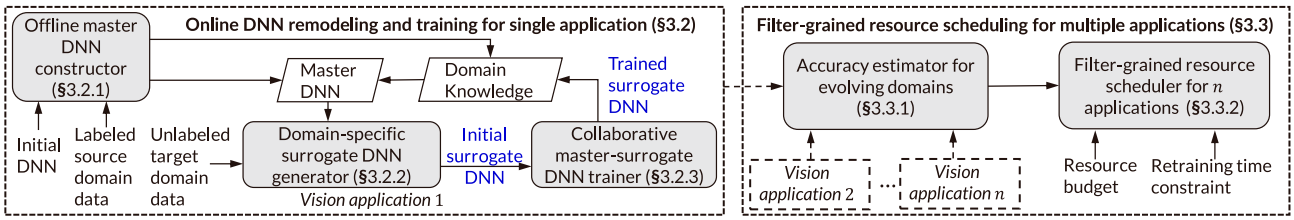


Fig. 2. ElasticDNN overview.

most sensitive to the target data, and accumulates seen domains' knowledge without expensive training of the master DNN.

- *Offline master DNN constructor* (Section III-B1). In order to support the master-surrogate DNN framework, the constructor uses the source domain data to pre-train the master DNN with twofold objectives. First, the **master DNN** supports dynamic architecture changes at run-time and it has good **trainability** [10], which decides the easiness to optimize a DNN. Second, the constructor generates a separate data structure, termed **domain knowledge**, to store each seen target domain's trained DNN as its *weight matrices*. This data structure is then used to update the master DNN while maintaining its trainability. The pre-training is performed once at the offline stage.
- *Domain-specific surrogate DNN generator* (Section III-B2). Once a new target domain is detected, the generator extracts a small domain-specific **surrogate DNN** from the master DNN, using one quick inference of the master DNN on the target domain's most representative sample. The surrogate DNN consists of only the most *accuracy-relevant* filters of the master DNN.
- *Collaborative master-surrogate DNN trainer* (Section III-B3). The trainer has two stages. Stage 1 quickly trains the generated surrogate DNN by only updating a small portion of its filters that are most relevant to the model convergence. Stage 2 first stores the trained surrogate DNN into the domain knowledge, and then updates the master DNN. Note that the master DNN is updated layer by layer to save memory footprint and its size keeps unchanged with new domain knowledge is stored.

**Filter-grained resource scheduling for multiple applications** (Section III-C). On edge devices, ElasticDNN supports low-overhead DNN remodelling and fine-grained resource scheduling among multiple applications with two modules (challenge 2). First, the *accuracy estimator for evolving domains* quickly estimates each DNN's accuracy improvement in DA under different model sizes. The estimation provides multiple optional model sizes for each application and thus enables the scheduler to adjust its DNN at the filter granularity. When considering multiple applications, the *filter-grained resource scheduler* optimizes resource allocation such that it assign more resources to a DNN bring more accuracy improvement in DA. That is, this DNN is expanded with more filters in remodeling and it is then retrained.

- *Accuracy estimator for evolving domains* (Section III-C1). The estimator has two steps. Step 1 predicts the surrogate DNN's accuracy improvement in DA given its model size and the current target domain's information. Step 2 then calibrates the prediction based on the *actual* predication results of previous domains. This calibration can improve the estimation precision because similar domains also have similar accuracy improvements in DA.
- *Filter-grained resource scheduler for $n$ applications* (Section III-C2). Given the estimated accuracy improvements and retraining time of all applications, the scheduler constructs an optimization problem that maximizes the overall accuracy be deciding each application's DNN size and assigned resource. The problem is solved by an evolutionary search approach.

### *B. Online DNN Remodeling and Collaborative Training for Single Application*

*1) Offline Master DNN Constructor:* The constructor pre-trains the master DNN and the domain knowledge for two purposes. First, the master DNN has sufficient learning ability and it is capable of remodeling into a small surrogate DNN when detecting a new target domain. Second, the domain knowledge accumulates the learned DNNs' knowledge of all seen domains.

In the **master DNN**, each convolutional layer is extended with a Feature Boosting and Suppression (FBS) module [19]. FBS allows the layer computing without some filters according to the layer's input, that is, it remodels a large layer to a specific small layer. The addition of FBS only slightly increases the model size (e.g. 1.87% in ResNet18 [2]).

Let $\theta_{i,j}$ be the weight of $i$-th $(1 \leq i \leq N)$ convolutional layer's $j$-th $(1 \leq j \leq N_i)$ filter, the **domain knowledge** $\Theta^*$ is defined as a list of weight matrix sets:

$$\begin{aligned} \Theta^* &= \{\Theta_1^*, \Theta_2^*, \cdots, \Theta_N^*\} \\ \Theta_i^* &= \{\theta_{i,1}^*, \theta_{i,2}^*, \cdots, \theta_{i,N_i}^*\} \end{aligned} \tag{1}$$

where $\Theta_i^*$ is a weight matrix set and $\theta_{i,j}^*$ $(1 \leq j \leq N_i)$ is a weight matrix. In the master DNN and the domain knowledge, each pair of $\theta_{i,j}^*$ and $\theta_{i,j}$ has the same format and hence these two matrix sets can interact with each other using addition or multiplication.

**Master DNN and domain knowledge pre-training**. The constructor uses two losses in the pre-training. The first loss (e.g. the cross-entropy in image classification task) initializes the master DNN's learning ability and remodeling ability. The second loss defines the interaction between the master DNN and the domain knowledge. This loss

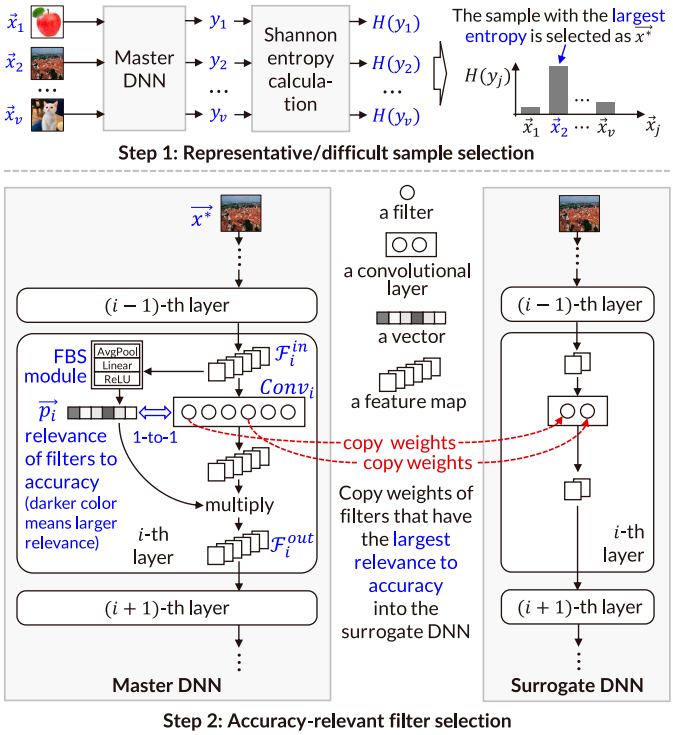


Fig. 3. Two steps in domain-specific surrogate DNN generator.

is defined as $\min \sum_{i=1}^{N} \sum_{j=1}^{N_i} \|\theta_{i,j} - \sum_{k=1}^{N_i} w_i(j,k) \cdot \theta_{i,k}^*\|_2^2$, where $w_i(j,k)$ represents the *relevance* between the filter $\theta_{i,j}$ in the master DNN and the weight matrix $\theta_{i,k}^*$ in the domain knowledge. After pre-training, the loss is minimized to 0 so that a filter in the $i$-th layer $\theta_{i,j}$ equals to a *linear combination* of the $i$-th weight set $\Theta_i^*$ in the domain knowledge. This combination relationship allows each master DNN's filter to receive an appropriate amount of knowledge from the domain knowledge, without affecting master DNN's remodeling ability.

*2) Domain-Specific Surrogate DNN Generator:* The domain-specific DNN generator remodels the master DNN into a small domain-specific surrogate DNN at run-time with two steps.

**Representative/difficult sample selection**. Given unlabeled samples in a target domain, step 1 selects the most difficult one to represent this domain. This sample requires the highest level of activations in model weights/filters to capture its semantic information [19], and such activation level is also sufficient to handle other simpler samples in the target domain. To this end, step 1 defines the difficulty of a sample $\vec{x}_i$ as the Shannon entropy $H(y_i) = -\sum_c p(y_i^c) \cdot log(p(y_i^c))$ of the master model's output $y_i$, where $p(y_i^c)$ denotes the probability that $\vec{x}_i$ is predicted as class $c$. A larger value of prediction entropy means the model is less confident about this prediction, namely $\vec{x}_i$ is more difficult [13]. As shown at the top of Fig. 3, given a batch of $v$ samples $\{\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_v\}$ in the target domain, step 1 first obtains the master DNN's outputs $\{y_1, y_2, \cdots, y_v\}$ and calculates their Shannon entropy $\{H(y_1), H(y_2), \cdots, H(y_v)\}$, and then selects the most difficult sample that has the largest entropy value $\vec{x}^* = \vec{x}_j$ $(j = \underset{j}{\text{argmax}}\, H(y_j))$.

**Accuracy-relevant filter selection**. Step 2 conducts one inference of the master model on sample $\vec{x}^*$ and selects the most accuracy-relevant filters from the master DNN. In inference, for the $i$-th convolution layer $Conv_i$, the *relevance* of its filters to accuracy is calculated by its FBS module. As shown in the bottom of Fig. 3, its FBS module receives the input feature map $\mathcal{F}_i^{in}$ and outputs a one-dimensional vector $\vec{p}_i$. In $\vec{p}_i$, the $j$-th element $p_{i,j}$ $(0 \leq p_{i,j} \leq 1)$ corresponds to the $j$-th filter's relevance to model accuracy. After calculating all filters' relevance values, the convolution layer outputs feature map $\mathcal{F}_i^{out}$:

$$\vec{p}_i = ReLU(Linear(AvgPool2d(\mathcal{F}_i^{in})))$$
$$\mathcal{F}_i^{out} = Conv_i(\mathcal{F}_i^{in}) * \vec{p}_i \quad (2)$$

where $ReLU$, $Linear$, and $AvgPool2d$ represent a ReLU layer, fully connected layer, and average pooling layer in the FBS module, respectively. The original output feature map is multiplied by filters' relevance values $\vec{p}_i$. After selecting a proportion (e.g. 30%) of filters with the largest relevance values, step 2 reassembles these filters into a new surrogate DNN model. This reassembling process is conducted layer by layer in the master DNN to save memory footprint.

*3) Collaborative Master-Surrogate DNN Trainer:* **Surrogate DNN retraining**. The trainer selects and trains a small proportion of filters in the surrogate DNN. This is based on the observation that in model training, most of the model weights have small gradients and contribute little to model accuracy, but the calculation of these gradients takes most of the model training time and memory footprint [20]. Our approach chooses to select model weights at the filter level because a filter is basic unit to extract feature information of input data [19] and hence the weights in the filter have similar gradients. The retraining has two steps.

*Step 1. Filter gradient calculation*. A straightforward approach to calculate the magnitude of a filter's gradient is to conduct one full forward and backward with target data and then check the filters' generated gradients. On resource-constrained edge devices, the proposed trainer employs a lightweight calculation method via the BN layer, because the BN layer calibrates the filter's output distribution and the magnitudes of their gradients are approximately proportional to each other. Formally, the magnitude of a filter's gradient is calculated as the magnitude of its corresponding BN channel's parameter changes (i.e. the magnitude of gradients): $s = \|\frac{1}{v}\sum_{i=1}^{v} \nabla_\beta \mathcal{L}(y_i)\|_1 + \|\frac{1}{v}\sum_{i=1}^{v} \nabla_\gamma \mathcal{L}(y_i)\|_1$, where $\beta$ and $\gamma$ represent the BN channel's weight and bias. This calculation is much cheaper than the calculation using all layers. For example, given a filter in ResNet56, the BN-based calculation reduces memory footprint by 192x and FLOPs by 144x.

*Step 2. Filter selection and model training*. The trainer ranks the filters in descending order according to their gradients and selects the top ranked filters whose accumulated gradient meets the threshold (e.g. 80%) of the summarized gradient. It then applies an unsupervised training algorithm (e.g. information maximization [13] or feature alignment [8]) to train the selected filters of the surrogate DNN. Note that compared with supervised algorithms, unsupervised algorithms use the model's output to calculate loss in gradient descent and

they also employ back propagation to update model parameters. For example, given only the model's outputs of $v$ samples $\{y_1, \cdots, y_v\}$, information maximization first calculates loss as $-\frac{1}{v}\sum_{i=1}^{v}\sum_c p(y_i^c)\cdot log(p(y_i^c)) + \sum_c p(\hat{y^c})\cdot log(p(\hat{y^c}))$, where $p(y_i^c)$ denotes the probability that the $i$-th sample is predicted as class $c$ and $p(\hat{y^c}) = \frac{1}{v}\sum_{i=1}^{v} p(y_i^c)$, and then back propagates the loss to update model parameters in the feature extractor.

**Master DNN updating**. Motivated by the problem that the master DNN needs to learn from unseen target domains without incurring high training overheads and sacrificing trainability, our trainer updates the master DNN's weights based on the *domain knowledge*. That is, the trainer first accumulates a seen target domain's full information in the domain knowledge and then transfers an approximation of this information to the master DNN. Formally, let $\Delta\theta_{i,j}$ be the change (i.e. learned knowledge) of a filter's weight $\theta_{i,j}$ in the surrogate DNN, the change is first stored into the corresponding weight set $\Theta_i^*$ of the domain knowledge:

$$\forall\ \theta_{i,k}^* \in \Theta_i^*,\ \theta_{i,k}^* \leftarrow \theta_{i,k}^* + w_i(j,k)\cdot\Delta\theta_{i,j} \tag{3}$$

where $\leftarrow$ means assignment operation. We can see that the domain knowledge stores the newly learned knowledge by changing the values of existing weights $\theta_{i,k}^*$, rather than explicitly storing a new set of surrogate DNN's weights. Subsequently, the trainer updates a filter $\theta_{i,j}$ in the master DNN using the weight set $\Theta_i^*$:

$$\theta_{i,j} \leftarrow \sum_{\forall\theta_{i,k}^*\in\Theta_i^*} w_i(j,k)\cdot\theta_{i,k}^* \tag{4}$$

Note that Equations 3 and 4 are executed multiple times using different values of $i$, $j$, and $k$, thus updating the master DNN layer by layer.

**Running Example**. Fig. 4 demonstrates how the three modules of ElasticDNN work together to complete an online remodeling and collaborative training on ResNet18. At the offline stage, the *offline master DNN constructor* trains the master DNN using 40k iterations. At run-time, suppose a domain shift is detected, the *domain-specific surrogate DNN generator* first estimates the accuracy relevance of all filters according to the most difficult sample, and then generates a surrogate DNN with 20% of the most accuracy-relevant filters (i.e. filters with darker colors). Subsequently, the trainer calculates the magnitude of surrogate DNN filters' gradients, and trains only 30% of them because these filters contribute to 80% of the summarized gradients. Finally, the *collaborative master-surrogate DNN trainer* updates the master DNN's model weights according to the accumulated domain knowledge.

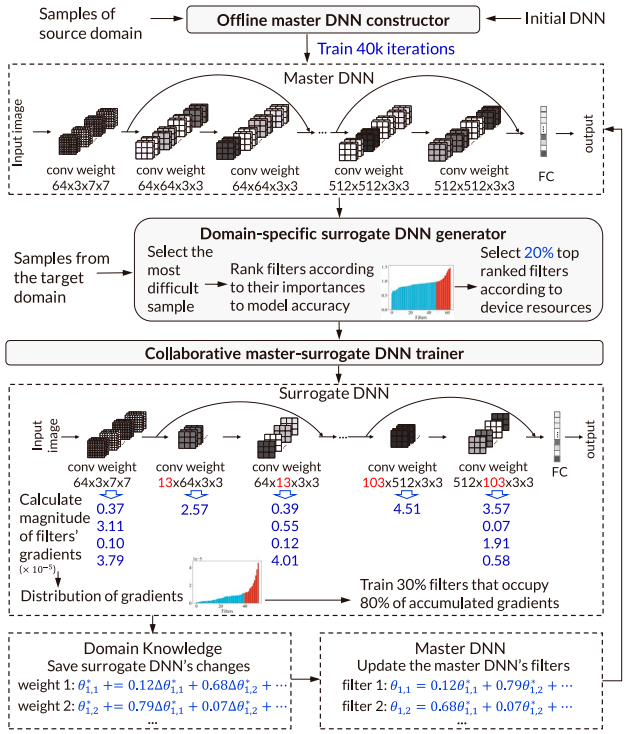


Fig. 4. An example of online remodeling and collaborative training.

Second, the scheduler uses the estimated accuracies to decide all applications' model sizes and assigned resources, with the goal of achieving the highest overall accuracy within the retraining window $t^{max}$.

*1) Accuracy Estimator for Evolving Domains:* The estimator first gives an initial prediction of model accuracy for target domain $k$ and then corrects the prediction based on previous accuracies of the past $k-1$ domains. This is based on the observation that the model accuracy of domain $k$ should be similar to the model accuracy of a seen domain if two domains are similar.

**Accuracy prediction**. For the $j$-th model size of the $i$-th application, the prediction takes three inputs: (i) remodeling ratio $s_{i,j}$ is the $j$-th model size divided by the original/uncompressed model size; (ii) domain distance $dis_{0,k}$ is the distance between source domain (0-th domain) and target domain $k$; (iii) domain representation denotes the mean feature vectors of source and target domain. By concatenating these inputs as a vector $\mathbf{I}$, the prediction employs a tiny fully connected network to output a vector $\mathbf{O} = (A_{i,j}^{(k)}, \tilde{A}_{i,j}^{(k)})$ that contains the estimated accuracies before and after retraining.

## C. Filter-Grained Resource Scheduling for Multiple Applications

Given $n$ applications, each application has its target domain $k$ and $C$ surrogate DNNs of different model sizes, the scheduling has two stages. First, the accuracy estimator predicts the $i$-th application's accuracy $A_{i,j}^{(k)}$ before retraining and accuracy $\tilde{A}_{i,j}^{(k)}$ after retraining when it uses the $j$-th model size ($1 \le j \le C$).

$$\sigma_1, \cdots, \sigma_{k-1} \leftarrow softmax\left(\frac{1}{dis_{k,1}}, \cdots, \frac{1}{dis_{k,k-1}}\right) \tag{5}$$

$$A_{i,j}^{(k)} \leftarrow \frac{1}{2}A_{i,j}^{(k)} + \frac{1}{2}\left(\sum_{a=1}^{k-1}\sigma_a \cdot rA_{i,j}^{(a)}\right) \tag{6}$$

$$\tilde{A}_{i,j}^{(k)} \leftarrow \frac{1}{2}\tilde{A}_{i,j}^{(k)} + \frac{1}{2}\left(\sum_{a=1}^{k-1}\sigma_a \cdot \widetilde{rA}_{i,j}^{(a)}\right) \tag{7}$$

**Accuracy calibration**. The calibration corrects $A_{i,j}^{(k)}$ and $\tilde{A}_{i,j}^{(k)}$ using three steps. Step 1 calculates the distance $dis_{k,a}$ between the current target domain $k$ and each past domain $a$ ($1 \leq a \leq k-1$), and normalizes these distances' inverses by softmax (Equation 5). Using the normalized distances as weight, step 2 calculates a weighted average of the past monitored accuracies before training $\{rA_{i,j}^{(1)}, \cdots, rA_{i,j}^{(k-1)}\}$, and averages it and $A_{i,j}^{(k)}$ as the calibrated $A_{i,j}^{(k)}$ (Equation 6). That is, the monitored accuracies in similar domains have larger influences in calibration. Finally, step 3 calculates a weighted average of the past monitored accuracies after training $\{\widetilde{rA}_{i,j}^{(1)}, \cdots, \widetilde{rA}_{i,j}^{(k-1)}\}$, and averages it and $\tilde{A}_{i,j}^{(k)}$ as the calibrated $\tilde{A}_{i,j}^{(k)}$ (Equation 7).

*2) Filter-Grained Resource Scheduler for $n$ Applications:* Based on the estimated accuracies, the scheduler finds the optimal model size and the amount of allocated resources for each application. The optimization objective is to maximize the overall accuracy among $n$ applications whose models need retraining, where each application's accuracy is the model accuracy averaged over the retraining window (aggregating accuracies before and after the retraining) [4]. Equation 8 defines the optimization objective in the target domain $k$.

$$\max \sum_{i=1}^{n} \sum_{j=1}^{C} \left[ A_{i,j}^{(k)} \cdot \left( t^{max} - \frac{T_{i,j}}{\lambda_i^{(k)}} \right) + \tilde{A}_{i,j}^{(k)} \cdot \frac{T_{i,j}}{\lambda_i^{(k)}} \right] \cdot O_{i,j}^{(k)} \tag{8}$$

$s.t.$

$$O_{i,j}^{(k)} \in \{0,1\}, \ \sum_{j=1}^{C} O_{i,j}^{(k)} = 1 \tag{9}$$

$$\lambda_i^{(k)} = \frac{\sum_{b=1}^{C} s_{i,b} \cdot O_{i,b}^{(k)}}{\sum_{a=1}^{n} \sum_{b=1}^{C} s_{a,b} \cdot O_{a,b}^{(k)}} \tag{10}$$

$$\forall \ 1 \leq i \leq n, \ \sum_{j=1}^{C} \frac{T_{i,j}}{\lambda_i^{(k)}} \cdot O_{i,j}^{(k)} \leq t^{max} \tag{11}$$

- The decision variable $O_{i,j}^{(k)}$ for selecting a model size is either 0 (not selected) or 1 (selected). For each application, only one model size is selected (Equation 9);
- $\lambda_i^{(k)}$ represents the percentage of allocated resources to the $i$-th application. $\lambda_i^{(k)}$ is proportional to the model size (i.e. the remodeling ratio $s_{i,j}$) of the $i$-th application (Equation 10);
- $T_{i,j}$ represents the retraining time of the $i$-th application under the $j$-th model size when it is allocated 100% of resources. When it is allocated only $\lambda_i^{(k)}$ of resources, its retraining time needs to be scaled by $\lambda_i^{(k)}$ (Equation 8 and 11);
- The time constraint is applied in each application such that the retraining time should not exceed the retraining window $t^{max}$ (Equation 11).

The scheduler utilizes an evolutionary search approach [18] to solve this optimization problem. The search process consists of multiple iterations. At each iteration, it either generates new random solutions or randomly mutates previously preserved solutions, evaluates them, and retains better solutions for the next iteration. When no better solution can be found, the search process is early-stopped and outputs the optimal solution. Due to the small dimension of the solution space (no more than 32 in our case, with $n = 4$ and $C = 8$), the search only takes ~5 seconds on edge devices. After solving the problem, suppose the $i$-th application generates a surrogate DNN under the $j$-th ($O_{i,j}^{(k)} = 1$) model size, this application is allocated $\lambda_i^{(k)}$ of resources in retraining. The resource allocation is developed based on lightweight Kubernetes/K3s on edge devices. All components of K3s consume less than 100MB of memory.

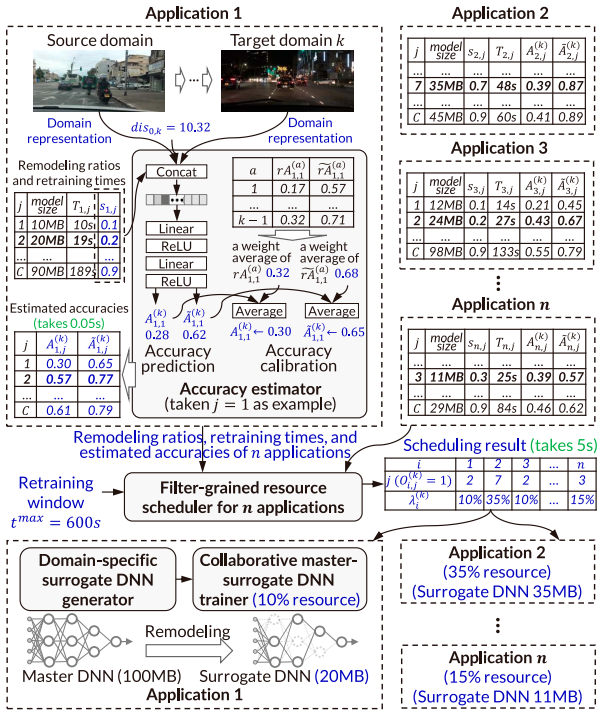


Fig. 5. An example of filter-grained resource scheduling.

**Running Example**. Fig. 5 demonstrates how ElasticDNN performs filter-grained resource scheduling among $n$ applications. For each application, the *accuracy estimator* first quickly predicts its surrogate DNN's accuracies $A_{i,j}^{(k)}, \tilde{A}_{i,j}^{(k)}$ before and after DA under different model sizes (e.g. within 0.5 seconds). Subsequently, the *resource scheduler* takes the estimated accuracies, the retraining time $T_{i,j}$, the remodeling ratios $s_{i,j}$ of $n$ applications, and the retraining window $t^{max}$ as input, and spends 5 seconds to search the optimal scheduling solution: each application's model size and assigned resource. We can see that application 2 is allocated the largest proportion (35%) of resources because its DNN brings the highest accuracy improvement in DA.

## IV. Evaluation

In this section, we evaluate the full implementation of ElasticDNN with an extensive set of evaluations.

TABLE I
SUMMARY OF FOUR WORKLOADS IN EVALUATION

| Workload | Source Domain | Target Domain | Model | Acc. Metric |
|---|---|---|---|---|
| Image Classification ($32 \times 32$) | CIFAR-10 [21] SVHN [22] | MNIST [23] STL-10 [24] USPS [23] | ResNet56 [2] | top-1 acc. |
| Image Classification ($224 \times 224$) | ImageNet [25] SYN SIGNS [26] | Caltech256 [27] GTSRB [28] DomainNet [29] | SENet18 [30] | top-1 acc. |
| Semantic Segmentation | GTA5 [31] Supervisely [32] | CityScapes [33] BaiduPerson [34] | DeepLabV3 [35] | mIoU |
| Object Detection | MSCOCO2017 [36] WI Face Mask [37] | VOC2012 [38] MakeML Mask [39] | YOLOV3 [7] | mAP@0.5 |

### A. Basic Settings

**Hardware setup**. We choose six edge devices of heterogeneous hardware architectures: (1) Huawei Mate20 X has eight 2.6GHz ARM-based cores (Kirin 980 GPU) and 8 GB memory; (2) Xiaomi 5S Plus has four 2.35GHz Quad cores (Qualcomm Snapdragon 821 CPU) and 4 GB memory; (3) NVIDIA Jetson TX2 has 256-core NVIDIA Pascal GPU and 8 GB memory; (4) NVIDIA Xavier NX has 384-core NVIDIA Volta GPU and 16GB memory; (5) NVIDIA AGX Xavier has 512-core NVIDIA Volta GPU and 32GB memory; and (6) NVIDIA AGX Orin has 1792-core NVIDIA Ampere GPU and 32GB memory. The two smart phones run Android OS 10.0 and support DNNs in TensorFlow Lite 2.8.0. The four NVIDIA platforms run Ubuntu 18.04.5 LTS and support DNNs in PyTorch 1.9.0 (Python 3.6.9).

**DNN models, evolving datasets, and applications**. We construct four workloads in three representative edge-based vision scenarios: image classification, semantic segmentation, and object detection. The configurations of workloads are detailed in Table I. These workloads provide evolving and drastic domain shifts by interleaving the arrival of target domains over time and 30 target domains are tested in each workload.

### B. Evaluation of Single-Application Scenario

**Compared baselines**. We implement and compare three categories of state-of-the art online DA methods: (1) *BN tuning methods* including ONline DA (ONDA) [12] and Tent [9]; (2) *feature alignment methods* including Source HypOthesis Transfer (SHOT) [13], ConDA [14], Bottom-Up Feature Restoration (BUFR) [40], Incremental Adversarial Domain Adaptation (IADA) [15], and CUA [8]; and (3) *whole network training methods* including Adapting to changing environments (ACE) [3]. We also report the model accuracies without DA as *source*.

**Offline model pre-training**. For each workload, all methods start from the same large DNN pre-trained using the source domain data. Subsequently, for baseline methods, four pre-trained models are compressed by removing 80% filters. For ElasticDNN, we use the pre-trained model to initialize master DNN and make sure that its surrogate DNNs have exactly the *same* model size as baseline methods' compressed models. All offline training tasks are executed on a GPU server with 48-GB Quadro RTX 8000 Graphics Card. For each workload, the training time of baseline methods ranges between 2.93 to 5.87 hours, and this time ranges between 6.40 to 13.78 hours for ElasticDNN. For each method, the offline pre-training is executed once.

**Online DA settings**. We implement and evaluate the online DA using a benchmark for evolving domains at edge [41]. The hyperparameters of all methods are automatically searched in this benchmark before evaluation. For each target domain, the number of available unlabeled samples is set to 100 to reflect realistic online DA scenario [5]. All workloads are evaluated on Huawei Mate20 X, Xiaomi 5S Plus, NVIDIA Xavier NX, and NVIDIA AGX Xavier, respectively.

**Metrics**. We measure three metrics: (1) *model accuracy*: it denotes the accuracy once the model is adapted to a new target domain; (2) *online DA time*: it is the model retraining time for baseline methods; and it includes both surrogate DNN retraining time and master DNN updating time for ElasticDNN; (3) *online DA overheads* including inference latency, memory footprint in the retraining, and energy consumption by `BatteryManager` from Android API and `in_power0_input` interface on NVIDIA devices.

*1) Accuracy Improvement Under the Same Model Size:* Fig. 6 demonstrates the accuracies of all methods in four workloads. We have three key observations from the results. First, ElasticDNN consistently delivers higher accuracies than all baselines for two reasons: (1) the surrogate DNN is customized for the target data hence contains its most accuracy-relevant filters; (2) the master DNN continuously learns from seen domains to generate more robust surrogate DNNs in the domains later. Second, when encountering more challenging scenarios with complex DNNs and vision tasks (Fig. 6(c) and 6(d)), ElasticDNN outperforms existing methods significantly because baseline methods' small models lack enough learning capacity for complex vision tasks. Finally, ElasticDNN is applicable to the DNN models of all workloads. In contrast, BUFR may cause logit explosion and lead to low accuracy (Fig. 6(b)); SHOT, ConDA, and BUFR are inapplicable to last two workloads because their pseudo-labeling or retraining algorithms are designed for specific image classification applications.

**Comparison of DA time**. As listed in Table II, two BN tuning methods (ONDA and Tent) have the lowest DA time, but they also produce the lowest accuracies among all methods. Except these two methods, ElasticDNN achieves the shortest DA time for two reasons: (1) before training, ElasticDNN's domain-specific surrogate DNN has a higher accuracy than most of baseline methods' models, so its training converges faster; (2) the surrogate DNN is trained using a small proportion of filters that have the largest gradients.

**Comparison of memory footprint.** In this evaluation, we choose the first domain as an example to test the memory footprint of each method (different domains have similar memory footprints). From the perspective of memory usage, baseline methods (e.g. CUA) have three stages in retraining: (1) *workload initialization* that loads model parameters and target data into memory; (2) *retraining* the loaded model; and (3) *retraining completion* that releases the target data and cache from memory. In contrast, ElasticDNN has two additional stages before and after the original retraining stage: (4) *surrogate DNN generation* that loads and uses the master DNN in one

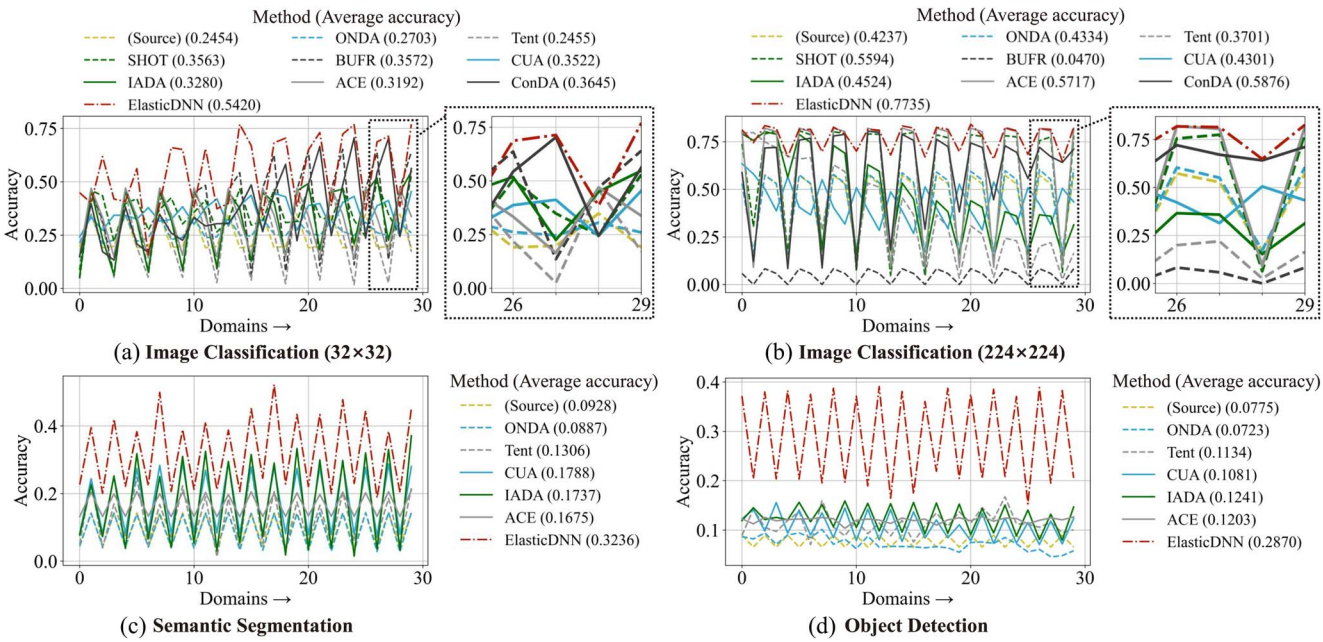


Fig. 6. Accuracies in continuous domain adaptation of eight baselines and ElasticDNN.

TABLE II
DA TIME (MINUTES) ON SMALL DNNS

| | Image Classification (32×32) | Image Classification (224×224) | Semantic Segmentation | Object Detection |
|---|---|---|---|---|
| ONDA | 0.09 | 0.22 | 0.88 | 1.01 |
| Tent | 0.42 | 9.42 | 0.91 | 2.52 |
| SHOT | 629.25 | 821.20 | N/A | N/A |
| BUFR | 149.53 | 251.43 | N/A | N/A |
| CUA | 1596.70 | 3378.14 | 309.48 | 57.19 |
| IADA | 241.39 | 207.90 | 153.97 | 70.73 |
| ACE | 2354.95 | 5219.28 | 649.91 | 344.54 |
| ConDA | 846.93 | 1723.43 | N/A | N/A |
| ElasticDNN | 26.48 | 58.20 | 9.97 | 4.42 |

inference operation and (5) *master DNN updating* that loads and updates the master DNN. Note that fastest BN layer tuning methods are not considered here because they have the lowest accuracies (26.60% lower than ElasticDNN). Fig. 7 illustrates the evaluation results and we have two observations.

First, ElasticDNN has the lowest average memory footprint than all baseline methods, because it proposes three strategies to reduce the memory footprint of the large master DNN: (i) ElasticDNN only requires one quick inference of the master DNN to generate the surrogate DNN. In inference, ElasticDNN interleaves the loading and forward operation of layers; (ii) after retraining, ElasticDNN updates the master DNN layer by layer; and (iii) ElasticDNN only retrains a small proportion of filters with the largest gradients.

Second, CUA, ACE, and IADA consume the largest memory footprint, because they retrain the entire model/feature extractor and maintain additional samples or models in the memory. Specifically, CUA randomly stores some previous samples at each retraining iteration, ACE uses an image generator to generate high-resolution images, and IADA introduces a generator and a discriminator to perform feature alignment.

*Discussion of NAS-based DA*. We note that the NAS-based DA methods also search for the optimal network architecture according to the given data. However, they take prohibitively long time on resource-constrained edge devices. Take the image classification (224×224) workload as an example, latest techniques such as OFA [18] and PIT [17] are expected to take about 211 and 3.8 days when running on NVIDIA AGX Orin (the most powerful edge device). This searching time is estimated by running the two techniques for 5% of search iterations, because they have the same searching time each iteration.

**Results**. *Compared to baseline techniques using the same model size, ElasticDNN increases accuracy by an average of 23.31%, accelerates the DA process by an average of 35.67x, and saves the memory footprint by an average of 43.10%.*

*2) Energy Consumption Reduction With Similar Accuracy:* Following the settings of the previous section, we compare baseline methods on the original large DNNs, which achieve similar accuracies as ElasticDNN. BN layer tuning methods are not considered because using large DNNs, they still have similar accuracies with the *source* method without adaptation and much lower accuracies than other methods. The evaluation results in Fig. 9 show that ElasticDNN significantly reduces the inference latency, DA time, and memory footprint while delivering 4.36% higher accuracy. In the following experiments, we focus on the energy consumption metric.

**Comparison of energy consumptions on heterogeneous edge devices**. In this evaluation, we test two image classification workloads on six edge devices and the last two workloads on three NVIDIA GPU devices (these workloads cause out-of-memory (OOM) error on smartphones). For measuring inference energy consumption of each workload, we emulate its real

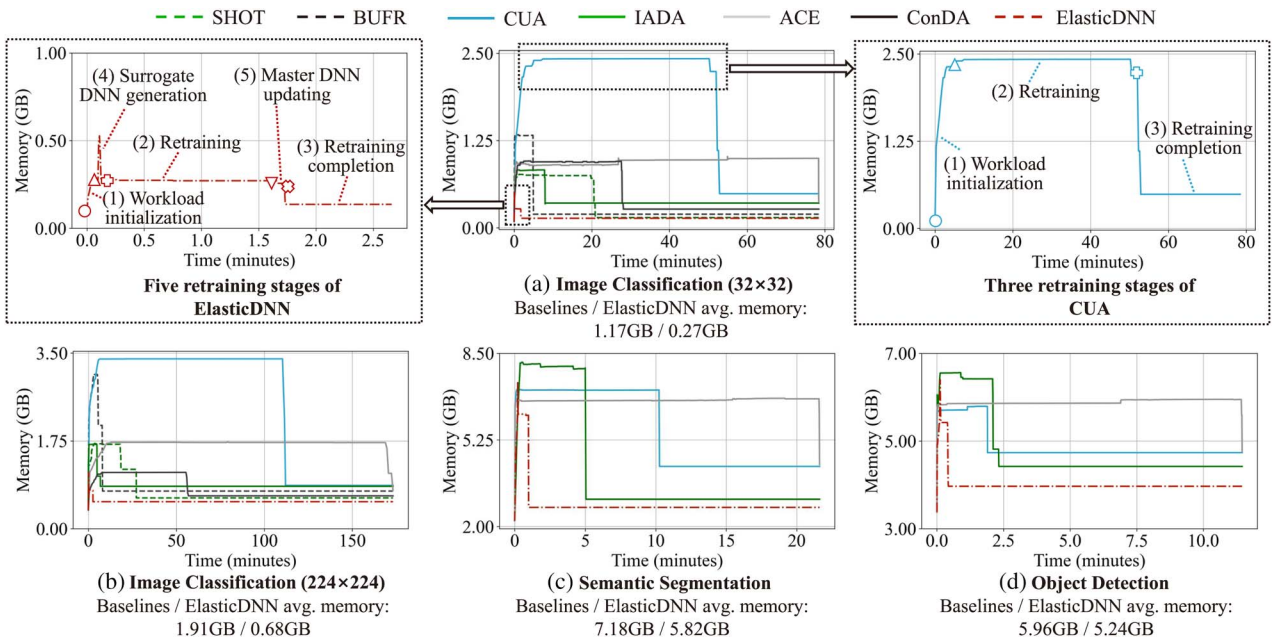


Fig. 7. Memory footprint of baselines methods and ElasticDNN in the first domain.

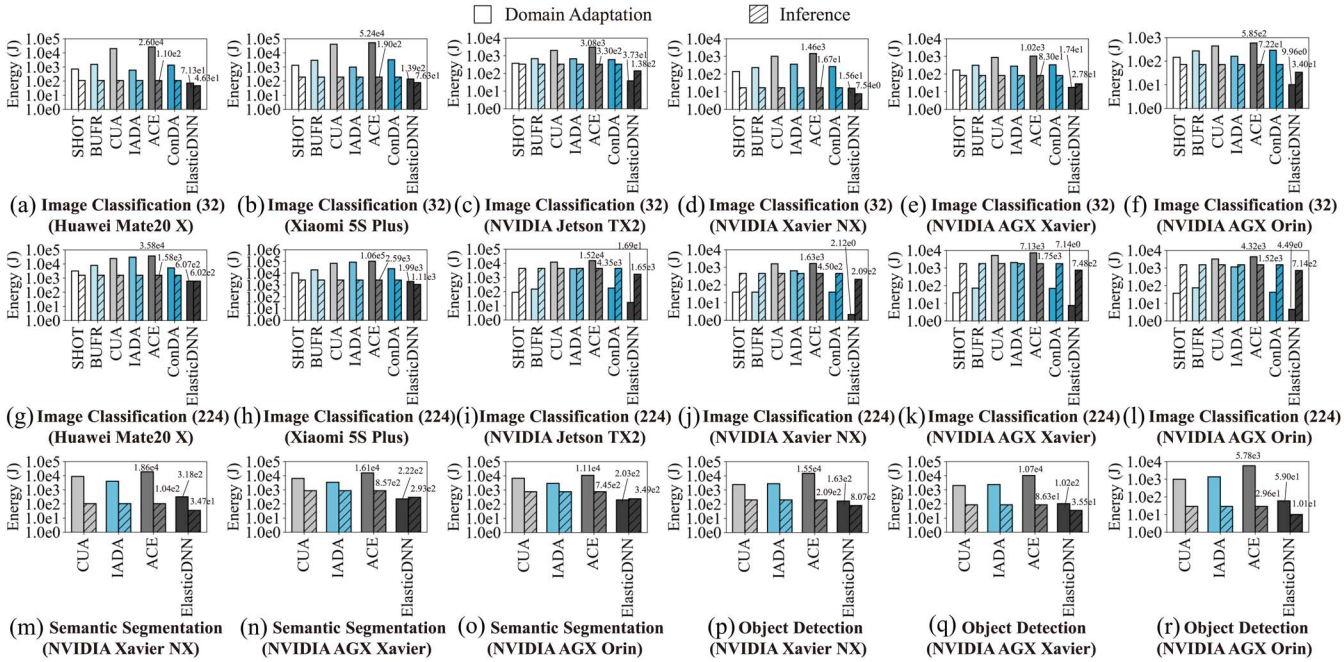


Fig. 8. Average energy consumption in each domain on heterogeneous devices.

edge-based vision scenario by conducting 1000 inferences in each target domain. Each evaluation is repeated for three times and the average energy consumption is reported.

Fig. 8 shows evaluation results (y-axis is in log scale). We can see that in DA, ElasticDNN achieves remarkable energy saving in all workloads (59.74x on average). In most cases, ACE consumes the largest energy because it not only caches old data samples for future training but also introduces another huge network to generate high-resolution images. In inference, the small surrogate DNN in ElasticDNN saves much energy consumptions compared to baselines methods (2.55x on average). Table III summarizes ElasticDNN's energy saving ratios compared to baseline methods, where "-" notation means the method suffers from OOM error. Overall, ElasticDNN saves energy consumptions by an average of 7.63x.

**Discussion of power modes**. Table IV reports the average DA energy consumption on edge devices' different power modes: (a-c) represent normal/performance/ultra power

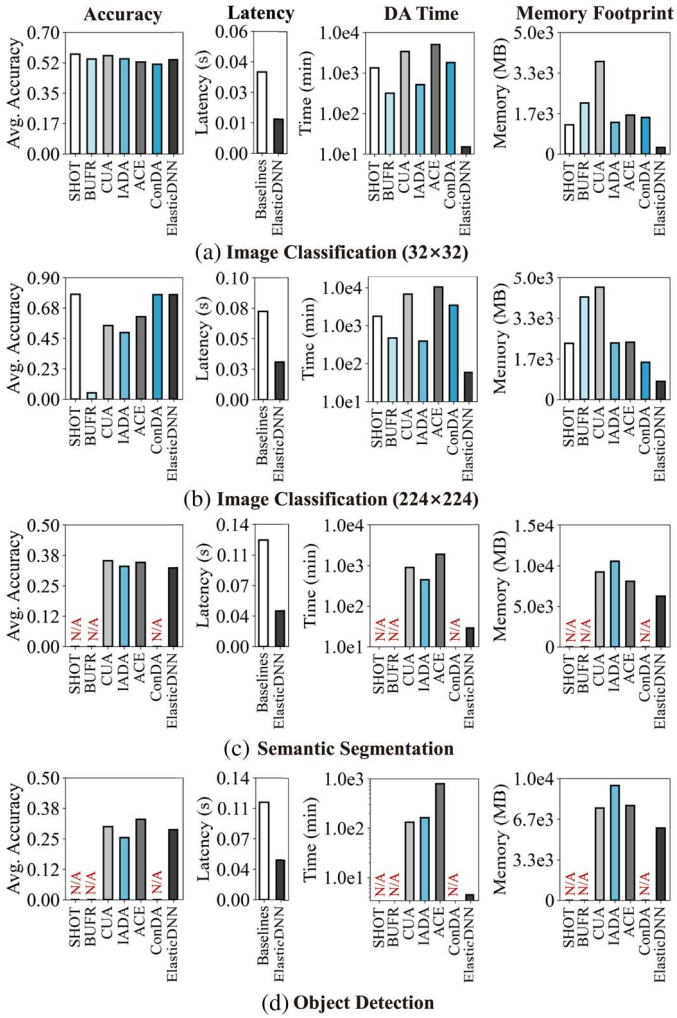


Fig. 9. Accuracy and overheads when baselines have similar accuracy to ElasticDNN.

saving mode respectively on Huawei Mate20 X and Xiaomi 5S Plus, and (d-f) represent 10W Desktop/15W 6Core/10W 2Core on NVIDIA Xavier NX respectively. Overall, ElasticDNN saves energy consumption by an average of 71.61x, 79.80x, 81.55x, 40.04x, 46.50x, 41.12x in the six power modes ((a) to (f)), respectively.

**Results**. *Compared to all baseline methods using the original large DNNs, ElasticDNN achieves 4.36% higher accuracy and has only up to 2.70% marginal accuracy drop when just comparing to the method that achieve the highest accuracy. Based on the similar accuracy, ElasticDNN reduces DA time, inference latency, memory footprint, and energy consumptions by an average of 144.70x, 2.55x, 4.14x, and 7.63x, respectively.*

### C. Evaluation of Multi-Application Scenario

In this section, we extend the above evaluation by testing the multi-application scenarios in Ekya [4]. On an edge device (NVIDIA Xavier NX or NVIDIA AGX Orin), we test 2 to 4 concurrent vision applications (image classification, semantic segmentation, or object detection). Within each retraining window (10 minutes), each application has both retraining/adaptation and inference tasks, and the tested scheduler aims to maximize their overall accuracy of all applications' tasks.

TABLE III
ENERGY SAVING RATIO IN ELASTICDNN

| | | Huawei Mate20 X | Xiaomi 5S Plus | NVIDIA Jetson TX2 | NVIDIA Xavier NX | NVIDIA AGX Xavier | NVIDIA AGX Orin |
|---|---|---|---|---|---|---|---|
| Image Classification (32×32) | DA | 117.25x | 121.41x | 33.19x | 36.93x | 28.56x | 31.56x |
| | Inference | 2.37x | 2.49x | 2.39x | 2.21x | 2.99x | 2.12x |
| Image Classification (224×224) | DA | 28.98x | 26.29x | 313.75x | 311.89x | 325.18x | 327.84x |
| | Inference | 2.63x | 2.34x | 2.63x | 2.15x | 2.34x | 2.13x |
| Semantic Segmentation | DA | - | - | - | 32.85x | 37.97x | 34.83x |
| | Inference | - | - | - | 3.07x | 2.91x | 3.14x |
| Object Detection | DA | - | - | - | 40.54x | 47.22x | 46.42x |
| | Inference | - | - | - | 2.46x | 2.45x | 2.87x |

TABLE IV
AVERAGE ENERGY CONSUMPTION (J) IN DA UNDER DIFFERENT POWER MODES

| | | SHOT | BUFR | CUA | IADA | ACE | ConDA | ElasticDNN |
|---|---|---|---|---|---|---|---|---|
| Image Classification (32×32) | (a) | 7.15e2 | 1.51e3 | 2.00e4 | 5.80e2 | 2.60e4 | 1.33e3 | 7.13e1 |
| | (b) | 9.32e2 | 1.92e3 | 2.16e4 | 1.01e3 | 3.56e4 | 2.10e3 | 7.83e1 |
| | (c) | 9.63e2 | 2.01e3 | 1.90e4 | 1.34e3 | 4.05e4 | 2.39e3 | 8.01e1 |
| Image Classification (224×224) | (a) | 1.05e4 | 1.91e4 | 6.93e4 | 8.51e4 | 1.06e5 | 2.40e4 | 1.99e3 |
| | (b) | 1.35e4 | 2.52e4 | 9.39e4 | 1.10e5 | 1.43e5 | 2.85e4 | 2.70e3 |
| | (c) | 1.40e4 | 2.87e4 | 9.21e4 | 1.11e5 | 1.38e5 | 3.20e4 | 2.76e3 |
| Object Detection | (d) | N/A | N/A | 2.42e3 | 2.47e3 | 1.49e4 | N/A | 1.65e2 |
| | (e) | N/A | N/A | 3.26e3 | 4.18e3 | 1.67e4 | N/A | 1.73e2 |
| | (f) | N/A | N/A | 2.20e3 | 2.82e3 | 1.09e4 | N/A | 1.29e2 |

**Compared baselines**. We implement and compare with two edge schedulers: Ekya [4] and its uniform version, denoted as Uniform, which uses the fixed retraining configuration and averaged resource allocation for each application. We also discuss RECL [16], which reuses past retrained models and schedules resources according to the real-time monitored training metrics. In addition, we integrate our approach with three resource schedulers that optimize retraining tasks' makespan in GPU servers (AFS [42], Synergy [43], and Muri [44]), and explore how ElasticDNN can improve these schedulers' performance.

*1) Estimation Accuracy and Overhead:* The effectiveness of resource scheduler is considerably impacted by its ability to estimate models' accuracy improvement under diverse target domains. We test the target domains in Section IV-B's four workloads and report the estimation errors in ElasticDNN and Ekya. Fig. 11 illustrates the estimation errors' distributions, average time usages, and average memory consumptions. We can see that ElasticDNN's estimation error is much lower, because of its prediction is based on sufficient offline training and online calibration using monitored accuracies. Moreover, ElasticDNN considerably reduces the estimation overheads because Ekya needs to execute tens of retraining trials for profiling one DNN. This low estimation overhead as well as the small memory usage in retraining (Fig. 7) indicate ElasticDNN can also achieve the lowest memory footprint in multi-application scenarios.

**Result**. *In estimation, ElasticDNN reduces time usage and memory consumption by an average of 8.91x and 1.93x, and achieves 7.74% lower estimation error.*

*2) Overall Performance:* Fig. 10 demonstrates the overall model accuracies of ElasticDNN as well as two baseline methods. Each sub-figure represents an evaluation on a specific mix

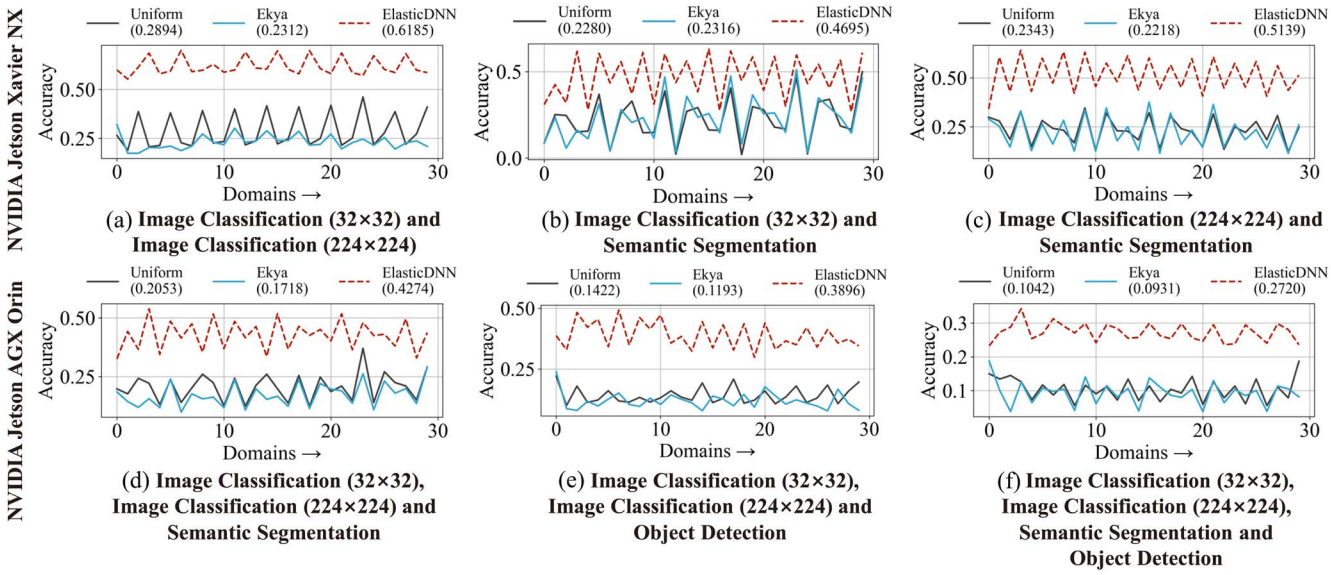


Fig. 10. Overall accuracies in the multi-application scenario of two baselines and ElasticDNN.

of applications. The results show that ElasticDNN consistently outperforms two baselines, because it employs lightweight accuracy estimation that leaves more time for retraining, and conducts filter-grained resource scheduling that better utilizes limited resources. In contrast, Ekya's heavy profiling takes up a large proportion of time that would be spent in retraining, making it performs even worse than Uniform in most cases (except Fig. 10(b)).

*Discussion of re-using trained models*. A complementary approach of Ekya is RECL [16], which reuses past trained models for faster convergence in model retraining. By testing it using the mix of applications in Fig. 10(f), the results shows that RECL still performs much worse than our approach by 13.94%, because all its stored models have a fixed network architecture and thus have limited learning capacity.

**Result**. *ElasticDNN achieves 17.34% to 35.82% accuracy improvement compared to the baselines, 25.91% on average.*

*3) Integration With Resource Scheduler in GPU Servers:* In this evaluation, we test three typical server-based schedulers using the the mix of applications in Fig. 10(f). These schedulers consider two conditions in their DNNs: fixed models or domain-adaptive models created by ElasticDNN. The results show that ElasticDNN are capable of being integrated into these resource schedulers and improves their accuracies by 19.04%, 17.64%, and 18.25%, respectively. This is because ElasticDNN's domain-adaptive surrogate DNN have larger learning capacity, while allowing these schedulers perform resource allocations at the filter granularity.

**Result**. *When integrating with the server-based schedulers, ElasticDNN improves accuracies by an average of 18.31%.*

### D. Ablation Study

We further conduct a breakdown analysis of the benefit brought by each feature of ElasticDNN. The experiments are performed on the image classification (32×32) and object detection workloads.

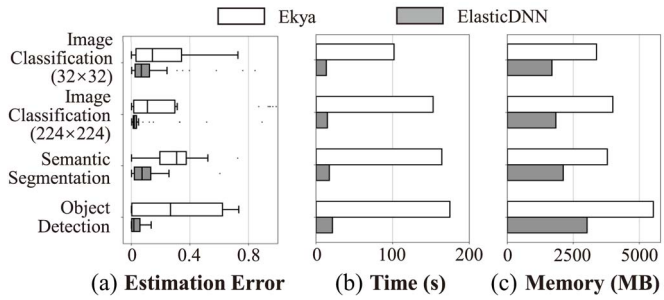


Fig. 11. A comparsion of estimation error and overhead of ElasticDNN and Ekya.

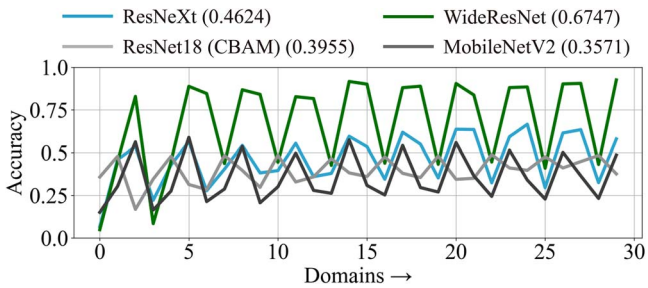


Fig. 12. Model accuracies under different model types.

**Generation strategy of surrogate DNN**. Compared to randomly choosing some filters to generate surrogate DNN, choosing the most accuracy-relevant filters in ElasticDNN achieves 4.42% and 2.61% higher accuracies in two workloads respectively. This is because these filters, which are chosen by the most difficult sample, can extract most of the features in the target domain.

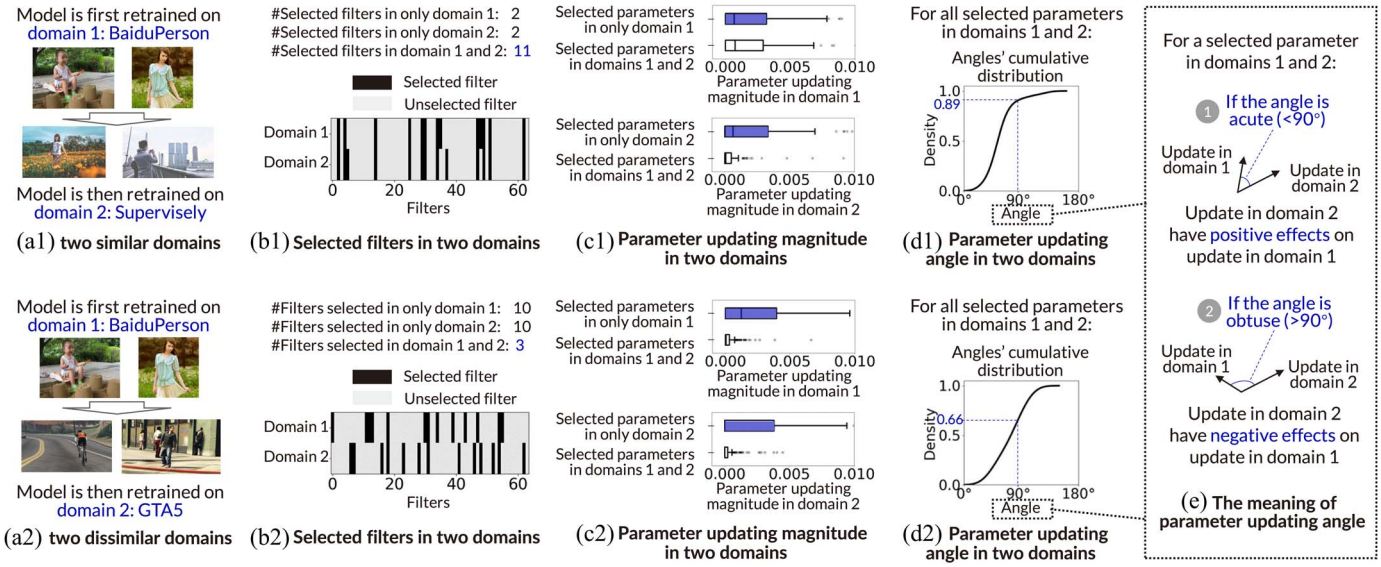


Fig. 13. Discussion of model parameter updating conflicts in two consecutive domains.

**Filter selection strategy**. We compare the accuracy when three strategies are used to select filters to be trained: (1) randomly choosing 20% filters; (2) choosing 20% filters that have the minimum cumulative gradients; (3) choosing 20% filters that have the maximum cumulative gradients, i.e. the default strategy applied by ElasticDNN. The evaluation result shows that the third strategy brings 10.47% and 6.53% accuracy improvements in two workloads respectively, because the filters that have the maximum magnitude of gradients contribute most to model accuracy.

**Updating master DNN**. We compare the accuracies of ElasticDNN when the master DNN is updated using the accumulated domain knowledge or not. The result shows that without updating, the surrogate model has 11.33% and 6.62% lower accuracies in two workloads, and it also has larger fluctuations in accuracies. This means ElasticDNN's collaborative master-surrogate DNN training not only improves the accuracy of both DNNs, but also increases the stability of these models when domain shift happens.

### *E. Discussion*

**Applicability of ElasticDNN to unsupervised DA techniques**. ElasticDNN represents the first framework that supports online DNN remodeling for edge vision systems. ElasticDNN can be generalized to various unsupervised DA techniques because its online remodeling is independent of the underlying retraining algorithm. For instance, ElasticDNN can be applied to two baselines Tent and CUA, improving their accuracies by an average of 15.43% and 16.21% in image classification (32×32) and object detection workloads, respectively.

**Applicability of ElasticDNN to various DNNs**. The above evaluation tests two types of DNNs: ResNet56 as a deep and multi-path model and SENet18 with feature map exploitation. We note that ElasticDNN can be generalized to support most of prevalent DNNs. To support this claim, we apply ElasticDNN in four DNNs [45] belonging to other prevalent types using the image classification (32×32) workload: (1) width: ResNeXt29 (2×64d) and WideResNet; (2) attention: ResNet18 with CBAM; and (3) lightweight DNN: MobileNetV2. Fig. 12 shows that ElasticDNN works well for these DNNs.

**Discussion of master DNN's capacity**. In ElasticDNN, the master DNN acts as the knowledge base of source and seen target domains, so its capacity decides the upper bound of accuracy in its surrogate DNN. An evaluation shows that the surrogate DNN generated by a larger master DNN (with 2x model size) achieves 14.83% and 15.10% higher accuracies than the one generated by a smaller master DNN (with 0.5x model size) in image classification (32×32) and object detection workloads, respectively.

**Discussion of model parameter updating conflict in two consecutive domains.** In ElasticDNN, one surrogate DNN is generated for a target domain and all surrogate DNNs feedback their learned knowledge into the same master DNN. This may cause *conflicts* between two consecutive domains because their surrogate DNNs can update the same model parameters in the master DNN. For example, if these two domains are similar (e.g. BaiduPerson and Supervisely both have real photos, as shown in Fig. 13(a1)), their surrogate DNNs thus contains many overlapped filters from the master DNN. If two domains are dissimilar (e.g. BaiduPerson and GTA5 in Fig. 13(a2)), their surrogate DNNs may have different update directions for the same model parameters in the master DNN. Here we take DeepLabV3 as an example and design three experiments to discuss this conflict. Note that the discussion focuses on the master DNN's last convolutional layer, because this layer contains more domain-specific features than other layers. Our evaluation reports three results.

*The percentage of jointly selected filters in two domains.* As shown in Fig. 13(b1) and 13(b2), 84.6% of filters are the same in two similar domains, while this percentage is only 23.1% in

two dissimilar domains. In master DNN updating, conflicts can only occur in these jointly selected filters and hence our two following experiments focus on model parameters in them.

*Parameter updating magnitude in two domains.* This magnitude refers to the absolute difference between a model parameter before and after retraining. A small magnitude indicates small conflicts in model parameter updating. As shown in Fig. 13(c1), in two similar domains, after the parameters are sufficiently updated in the first domain, their magnitudes become much smaller in the second domain and thus have negligible conflicts. In contrast, Fig. 13(c2) shows that in two dissimilar domains, the jointly selected parameters' magnitudes are much smaller than those of other model parameters. This result means the updating of the former parameters have small impacts on model accuracy (namely they cause small conflicts).

*Parameter updating angle between two domains.* As illustrated in Fig. 13(e), given a model parameter, two updates in two domains have two different directions that form an angle. An acute angle means the updating in the second domain has a positive effect (that is, improving model accuracy) on the updating of the first domain, and vice versa. Fig. 13(d1) and 13(d2) show that 89% and 66% of angles are acute between two similar and dissimilar domains, respectively. This result means the model parameter updating of the jointly selected parameters have no conflicts in most of the cases.

In conclusion, ElasticDNN's master-surrogate model parameter updating mechanism cause small conflicts for two reasons. First, two dissimilar domains only have a small percentage of jointly selected model parameters that may have conflicts. Second, in two similar domains, a majority of jointly selected model parameters have acute updating directions and their updating magnitudes are much smaller than those of other parameters.

## V. Conclusion

This paper presents the design, implementation and evaluation of ElasticDNN, a framework based on on-device DNN remodeling for adapting evolving vision domains at edge. Our approach can provide the ideal balance between accuracy and retraining cost in the single-application scenario, and enable the efficient filter-grained resource scheduling in the multi-application scenario. Extensive evaluation results prove ElasticDNN's superior accuracy, efficiency, and applicability compared to existing techniques.

## References


[1] R. Han, Q. Zhang, C. H. Liu, G. Wang, J. Tang, and L. Y. Chen, "LegoDNN: Block-grained scaling of deep neural networks for mobile vision," in *Proc. 27th Annu. Int. Conf. Mobile Comput. Netw. (ACM MobiCom '21)*, New Orleans, LA, USA, 2021, pp. 406–419.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770–778.

[3] Z. Wu, X. Wang, J. E. Gonzalez, T. Goldstein, and L. S. Davis, "ACE: Adapting to changing environments for semantic segmentation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Seoul, Korea (South), 2019, pp. 2121–2130.

[4] R. Bhardwaj et al., "Ekya: Continuous learning of video analytics models on edge compute servers," in *Proc. 19th USENIX Symp. Netw. Syst. Des. Implementation (NSDI)*, Renton, WA, USA, 2022, pp. 119–135.

[5] H. Liu, M. Long, J. Wang, and Y. Wang, "Learning to adapt to evolving domains," in *Proc. Adv. Neural Inf. Process. Syst. 33: Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 22338–22348.

[6] J. Yang, H. Zou, S. Cao, Z. Chen, and L. Xie, "MobileDA: Toward edge-domain adaptation," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6909–6918, 2020.

[7] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2015, *arXiv:1804.02767*.

[8] A. Bobu, E. Tzeng, J. Hoffman, and T. Darrell, "Adapting to continuously shifting domains," in *Proc. 6th Int. Conf. Learn. Representations (ICLR)*, Vancouver, BC, Canada, in Workshop Track Proceedings, 2018, pp. 1–4.

[9] D. Wang, E. Shelhamer, S. Liu, B. A. Olshausen, and T. Darrell, "Tent: Fully test-time adaptation by entropy minimization," in *Proc. 9th Int. Conf. Learn. Representations (ICLR)*, Virtual Event, Austria, 2021.

[10] H. Wang and Y. Fu, "Trainability preserving neural pruning," in *Proc. 11th Int. Conf. Learn. Representations (ICLR)*, Kigali, Rwanda, 2023.

[11] M. Kanakis, D. Bruggemann, S. Saha, S. Georgoulis, A. Obukhov, and L. V. Gool, "Reparameterizing convolutions for incremental multi-task learning without task interference," in *Proc. 16th Eur. Conf. Comput. Vis. (ECCV)*, Glasgow, U.K., vol. 12365. New York, NY, USA: Springer-Verlag, 2020, pp. 689–707.

[12] M. Mancini, H. Karaoguz, E. Ricci, P. Jensfelt, and B. Caputo, "Kitting in the wild through online domain adaptation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Madrid, Spain. Piscataway, NJ, USA: IEEE Press, 2018, pp. 1103–1109.

[13] J. Liang, D. Hu, and J. Feng, "Do we really need to access the source data? Source hypothesis transfer for unsupervised domain adaptation," in *Proc. 37th Int. Conf. Mach. Learn. (ICML)*, Virtual Event, vol. 119. PMLR, 2020, pp. 6028–6039.

[14] A. M. N. Taufique, C. S. Jahan, and A. Savakis, "ConDA: Continual unsupervised domain adaptation," 2021, *arXiv:2103.11056*.

[15] M. Wulfmeier, A. Bewley, and I. Posner, "Incremental adversarial domain adaptation for continually changing environments," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, Brisbane, Australia. Piscataway, NJ, USA: IEEE Press, 2018, pp. 4489–4495.

[16] M. Khani et al., "RECL: Responsive resource-efficient continuous learning for video analytics," in *Proc. 20th USENIX Symp. Netw. Syst. Des. Implementation (NSDI)*, Boston, MA, USA, 2023, pp. 917–932.

[17] M. Risso et al., "Lightweight neural architecture search for temporal convolutional networks at the edge," *IEEE Trans. Comput.*, vol. 72, no. 3, pp. 744–758, Mar. 2023.

[18] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *Proc. 8th Int. Conf. Learn. Representations (ICLR)*, Addis Ababa, Ethiopia, 2020, pp. 1–15.

[19] X. Gao, Y. Zhao, Ł. Dudziak, R. Mullins, and C.-z. Xu, "Dynamic channel pruning: Feature boosting and suppression," in *Proc. 7th Int. Conf. Learn. Representations (ICLR)*, New Orleans, LA, USA, 2019, pp. 1–14.

[20] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "FreezeOut: Accelerate training by progressively freezing layers," 2017, *arXiv:1706.04983*.

[21] "CIFAR-10 database." lecun.com. Accessed: Mar. 2023. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[22] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, vol. 2011, no. 5, p. 7.

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, pp. 2278–2324, Nov. 1998.

[24] A. Coates, A. Y. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proc. 14th Int. Conf. Artif. Intell. Statist. (AISTATS)*, Fort Lauderdale, USA, vol. 15, 2011, pp. 215–223.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. 25: 26th Annu. Conf. Neural Inf. Process. Syst.*, 2012, Lake Tahoe, NV, USA, 2012, pp. 1106–1114.

[26] B. Moiseev, A. Konev, A. Chigorin, and A. Konushin, "Evaluation of traffic sign recognition methods trained on synthetically generated data," in *Proc. 15th Int. Conf. Adv. Concepts Intell. Vis. Syst. (ACIVS)*, Poznań, Poland, vol. 8192, 2013, pp. 576–583.

[27] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," Pasadena, CA, USA: California Institute of Technology, 2007.

[28] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The German traffic sign recognition benchmark: A multi-class classification competition,"

in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, San Jose, CA, USA, 2011, pp. 1453–1460.
[29] X. Peng, Q. Bai, X. Xia, Z. Huang, K. Saenko, and B. Wang, "Moment matching for multi-source domain adaptation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Seoul, Korea (South), 2019, pp. 1406–1415.
[30] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 8, pp. 2011–2023, Aug. 2020.
[31] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *Proc. 14th Eur. Conf. Comput. Vis. (ECCV), Part II*, Amsterdam, The Netherlands, vol. 9906, 2016, pp. 102–118.
[32] "Supervisely person dataset." Supervisely. Accessed: Mar. 2023. [Online]. Available: https://supervise.ly/
[33] M. Cordts et al., "The cityscapes dataset for semantic urban scene understanding," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, J2016, pp. 3213–3223.
[34] Z. Wu, Y. Huang, Y. Yu, L. Wang, and T. Tan, "Early hierarchical contexts learned by convolutional networks for image segmentation," in *Proc. 22nd Int. Conf. Pattern Recognit. (ICPR)*, Stockholm, Sweden, 2014, pp. 1538–1543.
[35] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017, *arXiv:1706.05587*.
[36] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in *Proc. 13th Eur. Conf. Comput. Vis. (ECCV), Part V*, Zurich, Switzerland, vol. 8693, 2014, pp. 740–755.
[37] "Face mask detection dataset." Kaggle, San Francisco, CA, USA. Accessed: Mar. 2023. [Online]. Available: https://www.kaggle.com/datasets/wobotintelligence/face-mask-detection-dataset
[38] M. Everingham, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, "The PASCAL Visual Object Classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2010.
[39] "Mask dataset." Make ML. Accessed: Mar. 2023. [Online]. Available: https://makeml.app/datasets/mask
[40] C. Eastwood, I. Mason, C. Williams, and B. Schölkopf, "Source-free adaptation to measurement shift via bottom-up feature restoration," in *Proc. 10th Int. Conf. Learn. Representations (ICLR)*, Virtual Event, 2022, pp. 1–14.
[41] Q. Zhang, R. Han, C. H. Liu, G. Wang, and L. Y. Chen, "EdgeVision-Bench: A benchmark of evolving input domains for vision applications at edge," in *Proc. 39th IEEE Int. Conf. Data Eng. (ICDE)*, Anaheim, CA, USA, 2023, pp. 3643–3646.
[42] C. Hwang, T. Kim, S. Kim, J. Shin, and K. Park, "Elastic resource sharing for distributed deep learning," in *Proc. 18th USENIX Symp. Netw. Syst. Des. Implementation (NSDI)*, 2021, pp. 721–739.
[43] J. Mohan, A. Phanishayee, J. Kulkarni, and V. Chidambaram, "Looking beyond {GPUs} for {DNN} scheduling on {Multi-Tenant} clusters," in *Proc. 16th USENIX Symp. Operating Syst. Des. Implementation (OSDI)*, Carlsbad, CA, USA, 2022, pp. 579–596.
[44] Y. Zhao, Y. Liu, Y. Peng, Y. Zhu, X. Liu, and X. Jin, "Multi-resource interleaving for deep learning training," in *Proc. ACM SIGCOMM Conf. (SIGCOMM '22)*, Amsterdam, The Netherlands, 2022, pp. 428–440.
[45] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5455–5516, 2020.

**Qinglong Zhang** is a Master Student with the School of Computer Science and Technology, Beijing Institute of Technology. His research interests include edge intelligence and deep learning applications.

**Rui Han** received M.Sc. degree with honor from Tsinghua University, China, in 2010, and the Ph.D. degree from Imperial College London, U.K., in 2014. He is an Associate Professor with the School of Computer Science and Technology, Beijing Institute of Technology, China. His research interests include system optimization for cloud data center workloads (in particular highly parallel services and deep learning applications). He has over 40 publications in these areas, including papers at MobiCOM, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, INFOCOM, and ICDCS.

**Chi Harold Liu** (Senior Member, IEEE) received the B.Eng. degree from Tsinghua University, Beijing, China, and the Ph.D. degree from the Imperial College London, London, U.K. He is currently a Full Professor and the Vice Dean with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing. Before that, he worked for IBM Research - China and Deutsche Telekom Laboratories, Berlin, Germany, and IBM T. J. Watson Research Center, USA. He is now an Associate Editor for IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING. His research interests include the big data analytics, mobile computing, and deep learning. He is a fellow of IET, and a fellow of Royal Society of the Arts.

**Guoren Wang** (Senior Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees from the Department of Computer Science, Northeastern University, Shenyang, China, in 1988, 1991, and 1996, respectively. He is now a Full Professor and a Director with the Institute of Data Science and Knowledge Engineering, Department of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. He was an Assistant President with Northeastern University, China. His research interests include XML data management, query processing and optimization, bioinformatics, high dimensional indexing, parallel database systems, and cloud data management. He has published more than 150 research papers in top conferences and journals like IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, ICDE, SIGMOD, *VLDB*, etc.

**Lydia Y. Chen** (Senior Member, IEEE) received the B.A. degree from the National Taiwan University and the Ph.D. degree from Pennsylvania State University. She is an Associate Professor with the Department of Computer Science, Technology University Delft. Prior to joining TU Delft, she was a Research Staff Member with the IBM Zurich Research Lab from 2007 to 2018. Her research interests include dependability management, resource allocation, and privacy enhancement for large scale data processing systems and services. She has published more than 80 papers in journals, e.g., IEEE TRANSACTIONS ON DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON SERVICE COMPUTING, and conference proceedings, e.g., INFOCOM, Sigmetrics, DSN, and Eurosys. She was a co-recipient of the Best Paper Awards at CCgrid'15 and eEnergy'15.