CaptorX: A Class-Adaptive Convolutional Neural Network Reconfiguration Framework

Zhuwei Qin[1*], Fuxun Yu[1*], Zirui Xu[1], Chenchen Liu[2],and Xiang Chen[1]
[1]School of Electrical and Computer Engineering, George Mason University, Fairfax, VA 22030 USA
[2] University of Maryland, Baltimore County, Baltimore, MD 21250 USA

***Abstract* – Nowadays, the evolution of deep learning and cloud service significantly promotes neural network based mobile applications. Although intelligent and prolific, those applications still lack certain flexibility: For classification tasks, neural networks are generally trained with vast classification targets to cover various utilization contexts. However, only partial classes are practically inferred due to individual mobile user preference and application specificity, which causes unnecessary computation consumption. Thus, we proposed *CaptorX* – a class-adaptive convolutional neural network (CNN) reconfiguration framework to adaptively prune convolutional filters associated with unneeded classes. *CaptorX* can reconfigure a pre-trained full-class CNN model into class-specific lightweight models based on the visualization analysis of convolutional filters' exclusive functionality for a single class. These lightweight models can be directly deployed to mobile devices without the retraining cost of traditional pruning-based reconfiguration. Furthermore, we can apply the *CaptorX* framework into a distributed collaboration setting. With dedicated local training regulation and collaborative aggregation schemes, the class-adaptive models on individual mobile devices can further contribute back to the central full-class model. Experiments on representative CNNs and image classification datasets show that, *CaptorX* can reduce the CNN computation workload up to 50.22% and save 46.58% energy consumption for varied local devices, meanwhile improving accuracy for their targeted classes with better task focus. With our distributed collaboration paradigm, *CaptorX* also provides further potential to enhance the central model accuracy, while reducing up to 37.58% communication cost compared to traditional distributed learning methods.**

***Index Terms*—CNN Reconfiguration, CNN Visualization, Mobile Computing, Distributed Learning**

∗The first two authors contributed equally to this work.

I. Introduction

Promoted by the evolution of artificial intelligence and cloud services, more and more intelligent applications have emerged on mobile devices. As one of the most representative deep learning technologies, Convolutional Neural Network (CNN) has been widely adopted by those applications for classification tasks [1]–[4]. For common CNN-based mobile applications, most of current CNN models are pre-trained with vast classification targets to cover as much application tasks as possible (*e.g.*, ImageNet with 1.2 million training images from 1,000 different classes). However in most real-world cases, only partial classes are practically inferred depending on individual users' preference and application specificity. For example, we analyzed the "Top 100 Intelligent Android Applications" and found that over 22% of the applications only focus on a specific kind of classification task (*e.g.*, animals, plants, books recognition, *etc.*) [5]. Therefore, a large amount of classes are actually unneeded regarding the well-trained full-class model. As a result, the class-level functionality redundancy introduces considerable computation costs, which restricts the deployment of CNN models on resource-constrained mobile devices.

To adapt CNN models for different mobile applications, conventional approaches either manually train a specialized neural network from scratch or fine-tune the existing pre-trained CNN model on specific data set for each application scenario [6]. However, considering vast mobile applications types and diverse user preferences, the computationally expensive and time-consuming training efforts still hinder the scalability of such class-adaptive CNN reconfiguration, bringing insufficient feasibility issues [**?**], [7].

To meet the flexibility requirements of class-oriented reconfiguration, one potential way is to conduct class-level CNN reconfiguration, *i.e.*, to adaptively reduce unneeded classification targets of a pre-trained full-class model without affecting the remaining classes' functionalities [8]. For example, Guo *et al.* have proposed a class-adaptive CNN pruning method based on filter sensitivity analysis [9]. By applying a certain amount of perturbation to each filter, they measured the corresponding network output score of each class target. The filter causing a smaller output score to a certain class target can be pruned when this class is unneeded. SaiRam *et al.* utilized a similar filter analysis method and trained CNN models into class-specific tree structures, which can be reconfigured to a different subset of class targets [10].

Although these methods addressed the class-level CNN computation optimization to some extent, they still have certain shortcomings: First, these methods lack sufficient theoretic support and the reconfigured models usually suffer from significant accuracy drop. To achieve satisfying classification accuracy without retraining, the computation reduction of the above methods is limited. Second, it's a challenge to find

Manuscript received xxxx x, 201x; revised xxxx x, 201x. Corresponding author: X. Chen (email: xchen26@gmu.edu).

an optimal setting for the pruning thresholds for choosing how many filters to prune. Setting the same threshold for all layers may not be appropriate because the different layers have varying sensitivities to the classification targets, depending on their position in the network (shallow versus deep layers).

To achieve the goal of class-level CNN reconfiguration, the first challenge is the lack of understanding of CNN component functionality, or their relationships with various classification targets. Although some CNN visualization works have been proposed with certain interpretation, the functionality identification of the CNN components is still hard to achieve for different application tasks [11]. Besides the difficulty in functionality interpretation, the second challenge is the optimization design for removing functionality redundancy. Although there are many model compression methods, such as weight sparsity and filter pruning, these methods only focus on weight/neuron removal and cannot well adapt to the class-level optimization context. Meanwhile, such methods' inevitable time-consuming retraining process also cannot meet the flexible deployment requirement of mobile applications.

To tackle the above challenges, we proposed *CaptorX* – a class-adaptive CNN reconfiguration framework for efficient mobile application deployment. *CaptorX* takes advantage of CNN visualization analysis, where convolutional filters' exclusive activation preference (*i.e.*, filter functionality) to specific classification targets is identified. The filters with the same functionality are then grouped into independent clusters and identified as the functional meta-component for this class. After the functionality components identification, *CaptorX* can adaptively prune the filter clusters associated with unneeded classes and reconfigure a general pre-trained full-class CNN to class-specific lightweight models for dedicated mobile applications. Benefited from the precise filter functionality identification, the reconfigured lightweight models can still function well on the reserved classification targets without harming the accuracy. Thus, such class-specific models can be directly deployed onto mobile devices without any expensive retraining, bringing great flexibility advantages than previous retraining-based reconfiguration methods.

Furthermore, our *CaptorX* supports the model collaborative learning paradigm in a distributed manner to utilize the mobile devices' available training capability and further improve central model's accuracy. Specifically, with a dedicated local training algorithm, mobile devices can conduct training to improve the performance of the local class-specific models. Then through a novel collaboration policy, these distributed class-specific models can contribute back to the central full-class model, while achieving optimal accuracy for better inference and further deployment.

The contribution of this work can be summarized as follows:

- We analyzed convolutional filters' class activation preference through CNN visualization analysis, and identified filters' exclusive functionality. Then individual class-specific meta-component is identified by filter clustering to precisely group filters with the similar functionality;
- We then proposed a class-level CNN reconfiguration framework. By pruning filter clusters of the unneeded classes, the pre-trained full-class CNN model can be reconfigured into class-specific models without losing accuracy and thus no retraining effort is caused;
- We applied our class-level reconfiguration framework in a distributed manner. With the novel local training regulation and collaborative paradigm, we show that we could utilize vast mobile device's computing capability to further enhance the central model's performance.

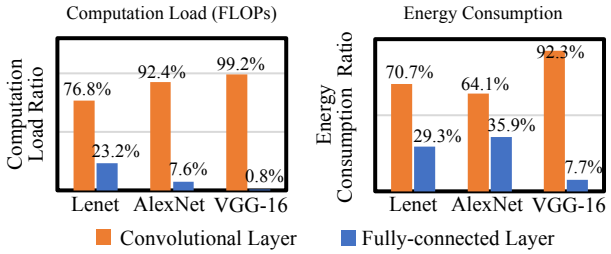


Fig. 1: CNNs Computation Load and Energy Consumption.

Experiments show that, *CaptorX* can effectively reconfigure a CNN model into class-specific models with any small subset of classification targets from different datasets, such as CIFAR10, CIFAR100, and ImageNet. Notably, *CaptorX* can reduce the CNN computation load up to 50.22% and save 46.58% energy consumption, while achieving even higher accuracy than the pre-trained models due to fewer class targets and better classification focus. In the distributed setting, *CaptorX* can reduce up to 37.58% communication cost without defecting accuracy.

II. Background

A. Mobile CNN Computation Consumption

The sophisticated CNN structure consists of multiple convolutional layers (CLs), pooling layers (PLs), and fully-connected layers (FLs). Such a high volume structure enables outstanding performance for cognitive tasks, but at the cost of intensive computation load and energy consumption [12], [13]. Among these CNN components, the multiplication and addition operations in CLs and FLs account for over 99% of total CNN computation load, making those two types of layers the major optimization targets [14].

Fig. 1 analyzes the computation load and energy consumption of CLs and FLs with three representative CNNs on a Google Nexus 5X, *i.e.*, *LeNet* [15], *AlexNet* [16], and *VGG-16* [17]. Fig. 1 demonstrates that: Although the three network have different structures, CLs contribute the most computation load (*i.e.*, 76.8%∼99.2%) as well as energy consumption (*i.e.*, 64.1%∼92.3%). As the model becomes deeper from *LeNet* to *VGG-16*, the cost of CLs also significantly increases.

While mobile storage space issue is relative ignorable nowadays [18], the computation load and energy consumption remain the major concerns for mobile applications. Therefore, we take CLs as our major optimization target in this work.

B. CNN Reconfiguration for Efficient Computing

To resolve the conflict between the massive computation load required for CNN inference and the limited computation resources on mobile devices, many CNN reconfiguration

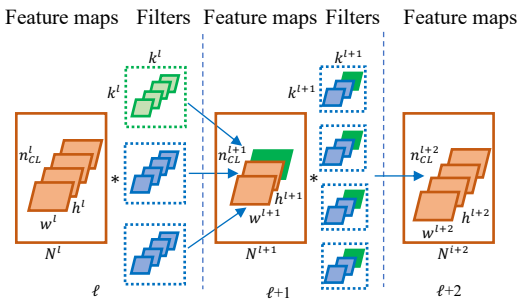


Fig. 2: Filter Pruning for CNN Reconfiguration.

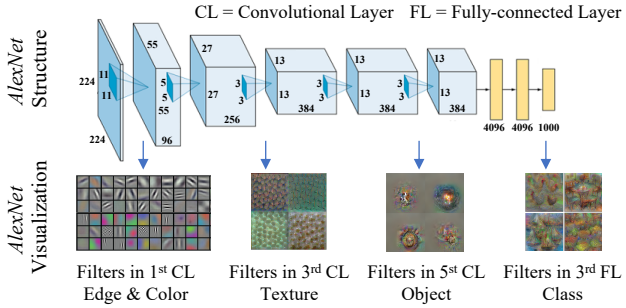


Fig. 3: Filter Visualization in AlexNet.

works have been proposed, such as weight sparsity [19], [20], filter pruning [21]–[25], low-rank decomposition [26], [27], quantization [28], [29], *etc*. As shown in later of this work, we consider the convolutional filter as the fundamental interpretable component, and mainly leverage the filter pruning based methods as the primary reconfiguration approach.

Filter pruning methods identify the convolutional filters with the least significance based on different metrics. By removing these filters and repeatedly retraining the model, CNN model volume can be significantly reduced and therefore decrease the computation load. For example, Li *et al.* ranked convolutional filters with their absolute values to identify and prune the least significant filters [21]. Inspired by the weight pruning method [30], [31], Molchanov *et al.* pruned filters based on the Taylor expansion that evaluates the significance of the filter by their second-order derivatives [32].

Fig. 2 demonstrates the mechanism of the filter pruning through three consecutive CLs. Given a l_{th} CL consisting of a set of filters $\mathcal{F}_i^l \in R^{k^l \times k^l \times n_{CL}^l}$ (represented by blue and green blocks). Each filter also contains multiple convolutional kernels of $K \in R^{k^l \times k^l}$. The convolutional operation generates and passes feature maps between layers (*i.e.*, $N^l \in R^{h^l \times w^l \times n^l}$ and $N^{l+1} \in R^{h^{l+1} \times w^{l+1} \times n^{l+1}}$ as the input and output of l_{th} layer). The convolutional filter pruning scheme aims to prune certain "insignificant" filters in the layer of l_{th} (represented by green blocks). As a filter in the l_{th} layer is pruned, its connected output feature maps and the corresponding convolutional kernels in $(l+1)_{th}$ layer's filters are also pruned. Meanwhile, the filter number in $(l+1)_{th}$ remains the same as well as the output feature map size of $N^{l+2} \in R^{h^{l+2} \times w^{l+2} \times n^{l+2}}$ in $(l+2)_{th}$. Therefore the pruned filters and feature maps could effectively reduce certain computation load.

Assume the total computation load for i_{th} CL as:

$$W_{CL_{l_{th}}} = n_{CL}^{l+1} \times (n_{CL}^{l} \times (k^l)^2) \times (h^{l+1} \times w^{l+1}), \quad (1)$$

where n^l is the total amount of filters and output feature maps. By pruning each filter in l_{th} CL, the computation load reduction can be formulated as:

$$\begin{aligned} W_{CL_{l_{th}}}^{-} =& (n_{CL}^{l} \times (k^l)^2) \times (h^{l+1} \times w^{l+1}) + \\ & n_{CL}^{l+1} \times (k^{l+1})^2 \times (h^{l+2} \times w^{l+2}). \end{aligned} \quad (2)$$

As shown above, such a significance-ranking based filter pruning method only considers the per-filter significance and has no connection with the actual interpretable functionality. Therefore, they cannot be directly used for our class-adaptive reconfiguration. Instead, we prefer to prune filters associated with unneeded classes as "insignificant" ones to achieve the class-level CNN reconfiguration. In the next section, we will introduce the CNN visualization to identify filters' functionality preference for dedicate classification target.

C. CNN Filter Functionality Interpretation

In CNN models, the convolutional filters are designed to capture certain input features. As a result, the semantics of the captured features can indicate the functionality of each filter. However, as a 3D weight matrix, the semantics of a filter is also usually hard to be directly interpreted.

Recently, more and more research works started to interpret the functionality of convolutional filters [33]. Zhou *et al.* demonstrated that convolutional filters have dedicated activation preferences for particular features in the input [34]. Such a feature preference can be defined as a filter's functionality. Yosinski *et al.* illustrated the divergence of the activation preference across convolutional layers [35]. Bau *et al.* further verified that, after a certain degree of functionality divergence, the convolutional filter in deep layers only extract class-specific features eventually [11].

To illustrate convolutional filters' functionality for qualitative analysis, most aforementioned methods utilized the CNN visualization technique. The CNN visualization technique was firstly introduced by Erhan *et al.* as the Activation Maximization (AM) method, which synthesizes the most preferred input pattern of a particular convolutional filter (with the largest activation value) [36]. Later, Simonyan *et al.* extended the AM method beyond the input patterns and identify the preferred class target for each convolutional filter [37].

Fig. 3 shows the visualized analysis of AlexNet – one of the most representative CNNs for image classification, which is composed of 5 CLs and 3 FLs. From the visualization analysis, we can see that the CNN visual feature abstraction starts with local fine details in the very first CL, then progresses into general textures and partial objects with deep layers, and ends in class determination with FLs.

In this work, we will utilize the AM method for convolutional filter functionality analysis. Specifically, we will qualitatively identify certain convolutional filters' exclusive activation preference to the individual classification target. Then, we will leverage the filter pruning with the interpretable filter functionality to reconfigure the CNN model.

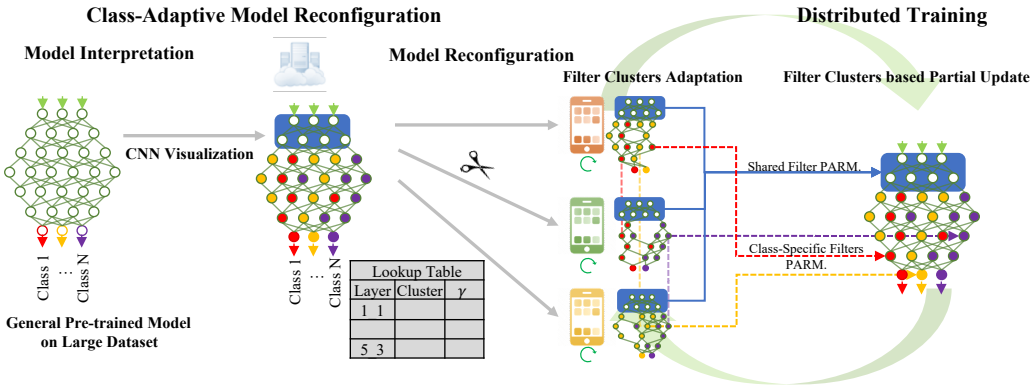


Fig. 4: Framework Overview of *CaptorX*. Given a pre-trained CNN model with *N*-class targets, its convolutional filters are pre-analyzed by CNN visualization, and the connections between filters and class targets are identified. The pre-trained CNN model can be reconfigured into class-specific models with small class-specific meta-components. The class-specific models can be either directly used for local inference or further training. After local training, the improved class-specific models can be collected back to the full-class model for better inference performance and further reconfiguration.

D. *Distributed CNN Collaborative Learning*

Distributed CNN collaborative learning has draw great attention recently with the advantages of harvesting vast edge computation resources, protecting local data privacy, *etc.* Many distributed learning frameworks have been proposed, such as Parameter Server [38] and Federated Learning [39]–[41]. By training locally on the distributed devices, these distributed models could contribute back by gradient or weight aggregation, and thus improve the performance of the central model.

In this work, we can also apply our class-adaptive reconfiguration framework in a distributed learning manner. In our framework, as each mobile device is deployed with a lightweight class-specific model, we conduct collaborative learning with these class-specific models in a distributed manner. With the dedicated local training and collaboration policy, *CaptorX* is able to aggregate these lightweight models to the central full-class model and thus improve the global full-class CNN model's performance.

III. Framework Overview of *CaptorX*

This section presents an overview of our proposed *CaptorX* framework. *CaptorX* aims to quickly reconfigure a pre-trained CNN model into lightweight class-specific models for mobile applications with dedicated classification tasks. Moreover, these lightweight models can be trained on devices, and contribute back to the full-class model. As shown in Fig. 4, a pre-trained CNN model can be reconfigured by the *CaptorX* and achieve further collaborative learning through following three stages: CNN Model Interpretation, Class-adaptive Model Reconfiguration, and Distributed Collaboration.

(1) CNN Model Functionality Interpretation. This stage identifies the functionalities of different convolutional filters. Given a generally pre-trained CNN model that covers a large number of the classification targets, we interpret the convolutional filters' functionality by CNN visualization. Specifically, Activation Maximization (AM) is used to identify the convolutional filters' exclusive activation preferences to specific class targets. The filters with same functionality will be grouped into filter clusters for later reconfiguration (This part will be detailedly shown in Sec. IV).

(2) Class-adaptive Model Reconfiguration. This stage reconfigures a pre-trained model into class-specific models. Based on the CNN functionality analysis, we have identified the filter clusters' dedicated functionality for specific classification target. Then we reconfigure the pre-trained model by preserving the filter clusters associated with the required classification targets while pruning out the rest. As the functionality of each filter cluster is relatively independent, all the filter clusters can be removed in a parallel manner with minor accuracy defection. Then, our reconfigured class-specific models can be directly deployed to mobile devices without retraining, achieving both optimal computation efficiency and reconfiguration feasibility. (This part will be detailedly shown in Sec. V).

(3) Distributed Model Collaborative Learning. In this stage, we applied our class-adaptive CNN reconfiguration framework in a distributed manner, in which the class-specific distributed models can contribute back to full-class model. As each model is deployed to distributed mobile devices, *CaptorX* conducts local training for all class-specific models and then collects the corresponding filter clusters from different class-specific models to a full class model. The aggregated model could well recognize all the class targets with improved model accuracy, and can be used for further model reconfiguration. (This part will be detailedly shown in Sec. VI).

IV. Visualization Analysis of Filter Functionality

To identify the functionalities of different filters towards specific class targets, we first analyze convolutional filters' exclusive activation preferences by using CNN visualization. Regarding the massive volume of convolutional filters, we introduce a clustering scheme that groups the filters based on their similarity and then identify their shared functionality.

A. *Filter Functionality Visualization*

As the fundamental feature extract components, convolutional filters have dedicated activation preference for different

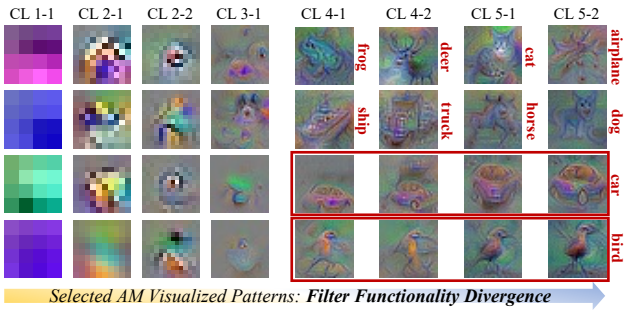


Fig. 5: Synthesized Patterns of Convolutional Filters with Certain Functionality Similarities, *e.g.*, Car, Bird, *etc.*

input features. We define such a characteristic of extracting specified features as filters' functionality. To analyze the feature extraction preference (*i.e.*, functionality), CNN visualization has been proposed in many previous works [34], [35]. In our work, we adopt the Activation Maximization (AM) visualization technique to qualitatively analyze the functionality of convolutional filters [35].

Formally, the feature extraction preference of the i_{th} filter $\mathcal{F}_i^l$ in the l_{th} layer is represented by a synthesized input pattern $V(\mathcal{F}_i^l)$ that can cause the maximum activation of $\mathcal{F}_i^l$ (*i.e.*, the convolutional feature map value). The synthesis gradient ascent process of such an input pattern can be formulated as:

$$V(\mathcal{F}_i^l) = \underset{X}{\operatorname{argmax}} A_i^l(X), \qquad X \leftarrow X + \eta \cdot \frac{\partial A_i^l(X)}{\partial X}, \tag{3}$$

where $A_i^l(X)$ is the activation of filter $\mathcal{F}_i^l$ from an input image X, η is the gradient ascent step size. With X initialized as an input image of random noises, each pixel is iteratively changed along $\partial A_i^l(X)/\partial X$ increment direction to achieve the maximum activation. Eventually, X demonstrates a specific visualized pattern $V(\mathcal{F}_i^l)$, which contains the filter's most sensitive input features with certain semantics, and represents the filter's functional preference.

To interpret all the filters' functionality, we apply AM visualization to filters in different layers and qualitatively analyze the convolutional filter functionality divergence. Note that the major reconfiguration overhead of *CaptorX* comes from the AM pattern generation, which can be ignored by only conducting once by a powerful server.

Algorithm 1 k-means based Filter Clustering Scheme

Input: CNN, Layers number L, and filter number in each layer I_l
Output: L Clusters

1: Graphic pattern synthesizing:
  for each *Layer* L_l **do** $List_l$=[];
    for each *Filter* F_i^l **do** $V(F_i^l)$=X_i^l;
      $List_l$.append($V(F_i^l)$); # Store the graphic patterns
2: Graphic pattern clustering:
  for each *Layer* L_l **do**
    c = $I_l/2$, C_l=kmeans.cluster($List_l$, c);
    for C_l^c in C_l **do** C_{locked}^l = [];
      if Len(C_c^l)== 1 **then** # Merge single filter clusters
      C_{locked}^l.append(C_c^l);
      c=c-1;
  c=Max(c)
  return C_c^l

Fig. 5 illustrates some AM visualized input patterns, which come from 8 selected convolutional layers (CLs) trained with *VGG-16* model and *CIFAR-10* dataset [42]. From Fig. 5, we can see that: (1) The functionality visualization of filters in shallow layers (*e.g.*, CL1_1 and CL2_1) tend to be colors or simple shapes, which are mostly basic features and have no clear class activation preference. (2) As the layer goes deeper, the complexity and variation of visualized patterns increase. The visualized patterns can contain clear class object patterns (*e.g.*, "car" and "bird"), indicating that these filters begin to form activation preference for dedicated classification targets. (3) For each class, there are certain groups of convolutional filters with similar dedicated class activation preference (indicated by the red rectangles), which can compose of a determinant meta-component for this class activation across deep layers. (4) After the CNN is well trained on specific datasets, these phenomena are universal for all classes (*i.e.*, 10 classes in the *CIFAR-10*).

Based on this observation, we found that many filters may share dedicated class activation preferences *w.r.t.* filter functionality. To further prepare those filters for class-adaptive reconfiguration, we also cluster them based on the AM visualized patterns.

B. Filter Functionality-oriented Clustering

The visualized input patterns demonstrate certain filters' class activation preference and similarity. To briefly interpret the connections between filters and specific class targets, we therefore conduct clustering upon the visualized input patterns.

As illustrated in Algorithm 1, we first utilize the AM to obtain the visualized input patterns of each convolutional filter. Then, we adopt k-means algorithm with pixel-level *Euclidean-distance* between the AM visualized input patterns of filter $\mathcal{F}_k^l$ and $\mathcal{F}_i^l$:

$$S_E[V(\mathcal{F}_i^l), V(\mathcal{F}_k^l)] = \|V(\mathcal{F}_i^l) - V(\mathcal{F}_k^l)\|^2, \tag{4}$$

which we use to indicate the functionality similarity of any two convolutional filters.

To determine the proper cluster number (*i.e.* c) in each layer, we first choose the cluster number as half of the total filter number (*i.e.*, $I_l/2$) in each layer. Then, during running the k-means clustering algorithm, many clusters may only contain one filter with extremely minimal similarity with others. These non-clustered filters are merged in a special cluster C_{locked}^l, which is considered to have unique features and won't participate the later class adaptive filter pruning. Finally, the final maximal c after merging the one filter clusters is selected as final cluster number for each layer.

To demonstrate the effectiveness of our proposed clustering method, we compare with the functional similarity evaluation of filters based on cosine similarity and Euclidean-distance between the filter weights. As shown in Fig. 6 the visualized filter pattern in the first column is visually similar to the second column one. However, their cosine similarity score did not truly reflects this visual similarity. For instance, the filter's visualized pattern in the first column of Fig. 6 has large invariant compared to the fourth pattern while filter's cosine similarity is smaller than the other three patterns. This indicates the cosine similarity of filter's weight fails to

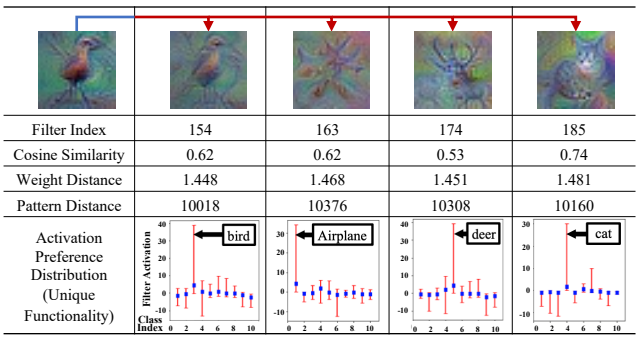


Fig. 6: The evaluation of filter similarity by different distance metrics (*i.e.* cosine similarity of filter weights and euclidean distance of filter weights and AM pattern) for filters in CL5_2 of *VGG-16*.

represent the function of convolutional filter from humans perspective. Although the Euclidean-distance between filter weights could also reflect the functioanlity similarity between filters to some extent, the distance difference between filter weights is much smaller. Hence, in our work, we utilized the Euclidean-distance between the AM pattern for clustering.

In addition, we evaluate the activation (feature map) of the selected filter by feeding in the testing images. As shown is the fifth row of Fig. 6, the x-axis is the class index and the y-axis is average filter activation on corresponding test images. We can see that each filter can be activated by their corresponding class images with distinctive high confidence. In other words, the filter can only be activated by one specific classes. This also proves the correctness and effectiveness of our AM pattern based filter functionality identification.

In Table. I, we show the filter cluster distribution based on *k*-means analysis. For a *VGG-16* model with 13 convolutional layers, the filters in each layer are grouped into 14 to 61 clusters. With the layer depth increment, the cluster number also becomes larger: in layer Conv5_2, the cluster number is as large as 61. This is because of the feature complexity increases with more divergent visualized activation preference patterns. Meanwhile, our proposed method can effectively cluster most filters. The minimum cluster ratio across all convolutional layers remains above 85%. In average, about 93% filters are well clustered through the whole model, indicating our method's sufficient filter functionality similarity analysis capability.

C. *Clusters' Class Activation Preference Identification*

Based on the visualized patterns, we have clustered the filters in each convolutional layer based on their feature preference similarity. And from the observation from Fig. 5, we can also find that the feature preference may demonstrate certain class-related patterns, *e.g.*, car, bird, *etc.* Therefore, in this part, we aim to quantitatively identify the class activation preference for filter clusters to associate each of them with its dedicated classification functionality.

TABLE I: Filter Cluster State of *VGG16* Model.

Layer	K	Filters	Ratio	Layer	K	Filters	Ratio
Conv1_1	15	62	96.8%	Conv4_1	27	447	87.3%
Conv1_2	17	64	100%	Conv4_2	28	437	85.4%
Conv2_1	20	128	100%	Conv4_3	38	439	85.7%
Conv2_2	31	121	94.5%	Conv5_1	46	500	97.7%
Conv3_1	26	251	98.0%	Conv5_2	61	503	98.2%
Conv3_2	24	251	98.0%	Conv5_3	40	506	98.8%
Conv3_3	14	239	93.4%	**Sum**	**387**	**3942**	**93.3%**

Specifically, we evaluate the class activation impact of each filter cluster by examining their backward-propagation gradients from different class logits, which demonstrate the impact of each convolutional filter towards a given class:

$$\gamma_{i,y_j} = \frac{1}{N}\sum_{n=1}^{N}\left\|\frac{\partial P(y_j)}{\partial A_i(x_n)}\right\|, \tag{5}$$

where $P(y_j)$ is the output probability of a sample image n corresponding to the class y_j, and $A_i(x_n)$ is the activation (output feature map) of filter F_i for each test image n. $\partial P(y_j)/\partial A_i(x_n)$ is the backward-propagation gradient of class y_i to filter F_i, which indicates the activation contribution of filter F_i to class y_j. Specifically, the gradient calculation is conducted on a large batch (N) of samples in the training dataset so as to get an unbiased estimation of the gradient. And the averaged value of γ_{i,y_j} on all tested samples will be considered the filter's activation preference to a certain class.

For each individual cluster, the class activation preference of all I filters are averaged, and the largest activation class target is considered as the cluster's class functionality preference:

$$\gamma_{c,y_j} = \frac{1}{I}\sum_{i=1}^{I}\gamma_{i,y_j}. \tag{6}$$

By doing so, we utilized the class activation preference to quantitatively associated each filter cluster with their dedicated classification target (*i.e.*, a class label, maximum γ_{c,y_j}), thus identifying their functionalities.

In the next section, we will introduce how to reconfigure a pre-trained model in class-level by removing filter clusters associated with unneeded classification targets.

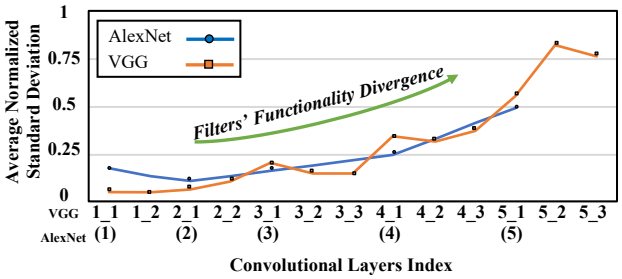
V. Class-adaptive model Reconfiguration

This section illustrates our class-adaptive CNN reconfiguration framework. Specifically, we design a set of class-adaptive CNN filter pruning methods for our reconfiguration. After that, we deploy the light-weight and class-adaptive models onto the mobile devices based on their specific application.

A. *Class-adaptive CNN Filter Pruning*

Based on our previous filter clustering and class activation preference identification, most convolutional filter clusters are associated with certain dedicated class targets. Based on the visualization analysis, different layers demonstrate various class activation preference divergence. Meanwhile, in each layer, filter clusters with same class label also have different activation preference strength. Therefore, we introduce CNN layer-level and cluster-level pruning schemes in order to maintain the optimal pruned model accuracy.

Layer-level: We first introduce the CNN layer-level pruning for different convolutional layers. Our design motivations come from the visualization results in Fig. 5. Through the feature preference visualization, we find the filters demonstrate stronger class activation preferences in deep CLs. Therefore,



*The upper horizontal axis indicates VGG, and the bottom one indicates AlexNet.

Fig. 7: The Evaluation of Class Activation Preferences of Different CLs in *VGG-16* and *AlexNet* model.

CaptorX should conduct filter pruning from the deep CLs to shallow CLs since filters in deep CLs are more class-specific.

Quantitatively, to evaluate the class activation preference differences, we define the class activation preferences level of l_{th} CL as:

$$L_l = \sqrt{\frac{1}{C}\sum_{c=1}^{C} S_{c,l}}, \quad S_{c,l} = \sqrt{\frac{1}{J}\sum_{j=1}^{J}(\gamma_{c,y_j}-\mu)^2}, \tag{7}$$

where C is the total cluster amount in l_{th} layer, and $S_{c,l}$ is the standard deviation of each filter's class activation towards different classes in one cluster. By such a definition, the standard deviation could indicate how much the filter clusters differ in their class activation preference: The larger the standard deviation, the more distinct class activation preference that different filter clusters would have. L_l is the average standard deviation of all filter cluster activation preference on different classes. Thus, the layer has larger L_l indicate stronger class functionality divergence, which should be pruned first.

Fig. 7 shows the normalized layer class activation preference level versus different CLs in *VGG-16* and *AlexNet* model. The layer number of *AlexNet* is represented with brackets in the x-axis. We can observe that: (1) As the layer number increases, the layer class activation preference level becomes larger, which indicates deeper CLs have stronger classes activation divergence. (2) For the *VGG-16* and the *AlexNet* model, layer class activation preference level become larger starting from CL4-1 and CL4 respectively. This preliminary analysis justifies our pruning scheme design.

Cluster-level: As there are a large number of independent clusters distributed in each layer, varied functionality strength also presents in different filter clusters. Therefore, a carefully-managed cluster-level pruning is also necessary to maintain the optimal model accuracy.

Considering the filter cluster volume and class activation importance, we define the functionality importance of one cluster c in layer I_l as:

$$L_{c,l} = \alpha \cdot length(C_c^l) \times \beta \cdot S_{c,l}, \tag{8}$$

where the $length(C_c^l)$ calculates the cluster volume size. The design heuristic is as follows: We first select the unneeded filter clusters based on their cluster label. For these unneeded clusters, two factors are taken into consideration. First, the cluster with larger cluster volume counts for more classification contribution to the unneeded classes. Second, the cluster with a larger standard deviation among unneeded classes

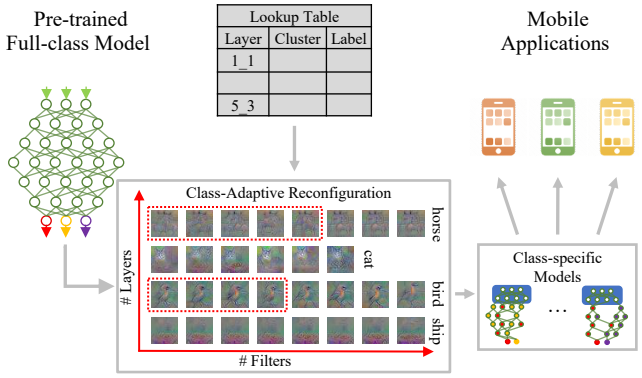


Fig. 8: Class-adaptive CNN Reconfiguration Framework.

should have certain stronger functionalities towards one or more unneeded class targets. Thus, we normalize two factors into the same scale and take these two factors into importance consideration. And finally, the clusters with larger $L_{c,l}$ would be first pruned since their functionality mainly focuses on unnecessary class targets. Although pruning unneeded clusters will also affect the ability to classify needed classes only if the reconfigured model needs to distinguish between the needed classes and unneeded classes at the same time. Since our reconfigured model is target to classify the needed classes, so pruning the pruning unneeded clusters will not affect the performance of the reconfigured model.

B. *Class-adaptive Model Reconfiguration*

With the defined layer-wise class activation preference level L_l and cluster-wise class activation preference level $L_{c,l}$, we can conduct our proposed class-adaptive model reconfiguration framework for local class-specific model deployment.

Fig. 8 shows the detailed implementation of our class-adaptive reconfiguration. Based on the filter clustering through filter visualization and class activation preference identification, the filters are associated with certain class targets. Such CNN interpretation results for model configurations are stored in a look-up table (LUT) with the general model online. When the local mobile devices request certain CNN with dedicated classification targets, the general model can be intuitively reconfigured according to the LUT for local deployment.

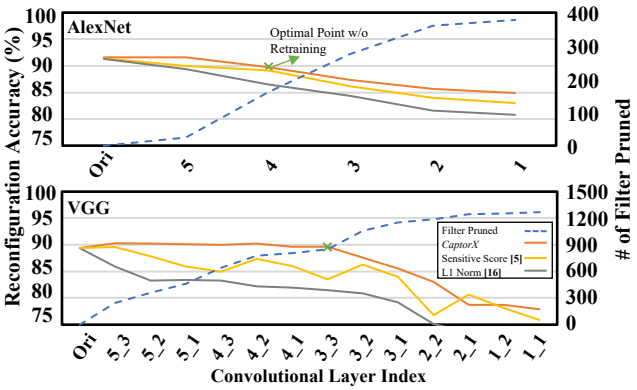


Fig. 9: Class Adaptive Model Reconfiguration.

Specifically, *CaptorX* have great reconfiguration flexibility, which can reconfigure a pre-trained full-class model into single class-specific models or with certain class combinations. In the reconfiguration, *CaptorX* finely prunes the unneeded filters cluster from the deep layers to shallow layers. The first several layers (*i.e.*, shared layers indicated by the blue rectangle in Fig.8) extract the basic features are shared by all the class-specific models. Meanwhile, class-specific meta-components in the deep layer are preserved for dedicated class targets.

For example, Fig. 9 shows the accuracy drop and pruned filters when the pre-trained full-class *AlexNet* and *VGG-16* are reconfigured to 5 class-specific model on *CIFAR10*. According to the LUT, *CaptorX* prunes all the unneeded clusters from deep CLs to shallow CLs and examine the classification accuracy. As a result, *CaptorX* can prune 851 unneeded filters in total from CL5-3 to CL3-3 without accuracy drop in *VGG-16*, marked by the green "$\times$", which indicates the optimal reconfiguration setting without model retraining. The *AlexNet* model has a more compact network structure, but as shown in the Fig. 9, *CaptorX* can still prune many unneeded filters in *AlexNet* model with optimal accuracy maintained. Therefore, the reconfigured class-specific models can be directly used for application deployment, enabling outstanding reconfiguration flexibility and feasibility.

We also compared our reconfiguration method with previous works [9] and [21] by pruning same amount of filter in each layer. Guo *et al.* [9] proposed to select filters based on the sensitivity analysis. Different with our method, they pruning filter without clustering filters, but the filters are ranked by the sensitivity score. As shown in the Fig. 9, our method can maintain higher accuracy when pruning same amount of filters. While, in the [21], Li *et al.* [9] pruning filter that are unimportant in general (*i.e.*, L1 norm ranking of filter weights). We can see that the accuracy drop is much larger without considering the filter's class preference.

VI. Distributed Model Collaboration

In this section, we apply *CaptorX* in a distributed manner and demonstrate an efficient distributed collaborative training application built upon our reconfiguration framework. In this application, the reconfigured class-specific models can be further trained on distributed mobile devices. Meanwhile, with our collaboration policy, the class-adaptive models can be collected back and further improve the performance of the central full-class model.

A. Further Performance Improvement by Local Training

By aforementioned class-adaptive reconfiguration, the class-specific lightweight models can be directly deployed to mobile devices for inference. Furthermore, leveraging the local device computation capability, we can prune the pre-trained full-class model more aggressively to achieve more computation reduction. After the reconfigured class-specific light-weight model was deployed to the local devices, the filter clusters in the class-specific layers can be quickly trained locally (*i.e.*, conducting on-device learning).

TABLE II: Different Local Training Configs. for 5 Classes.

Configuration	0	100	300	500	700	1k	8k
Tune FC layer	83.8	84.3	85.6	84.9	85.1	84.1	92.1
Tune Class-Specific & FC layer	**83.8**	**84.5**	**85.7**	**85.1**	**85.6**	**85.8**	**93.3**
Tune All layer	83.8	84.0	84.7	84.5	83.7	84.1	91.4

In *CaptorX*, different class-specific models are deployed onto distributed mobile devices with shared layers and class-specific meta-components. Therefore, we need certain training regulation for the distributed collaboration so as to avoid training divergence.

Specifically, our local training regulation freezes the filter weights of the first several layers (the shared layers learned) of the class-specific models, and trains the deep layers (class-specific layers). Such a design motivation is because the first several layers capture basic features like edges, lines, and colors, which are not very specific to a particular classification target. To prove our regulation's effectiveness, Table II shows the on-devices training results of a local 5 class model under different training regulations. In the first regulation setting (Tune FC layer), the shared layers and class-specific layers are frozen while only the final classifier are trained. In the second regulation setting (Tune Class-specific & FC layer), only the shared layers are frozen. The class-specific layers and final classifier are trained. In the third regulation setting (Tune All layers), all the layers are trained. As shown in the Table II, we can achieve best classification accuracy with 8K retraining iterations when freezing the shared layers and training the class-specific and fully-connected layers. However, training all layers demonstrate the lowest classification accuracy. Thus, in our distributed collation we only train the class-specific and fully-connected layers.

Meanwhile, to show the generality of our local training framework, Fig. 10 compares the local training process when we deploy the class-adaptive *VGG-16* with different number of classification targets. In the Fig. 10 (a), each mobile device was deployed a single class target model. While in the Fig. 10 (b), the pre-trained *VGG-16* are reconfigured into 2 to 10 classes models. We can see that: (1) Both fine-tuning processes can quickly recover reconfigured class-adaptive model's accuracy. (2) And when the local device decoupled with single class to perform one-vs-all classification, the model can achieve up to 98% classification accuracy. (3) With more class targets in the class-adaptive models, the accuracy will be closed to the pre-trained model with optimal accuracy. This demonstrates the effectiveness of our local training policy, which could utilize distributed mobile devices' training capability to improve the local model accuracy.

B. Distributed Model Collaboration Policy

With the local class-specific models well trained, *CaptorX* is also able to collaboratively improve the central full-class model. The underlying foundation of this scheme is to collect shared layers and different class-specific meta-components back to the central model.

Figure 11 shows the full-class model aggregation process in distributed collaboration. In the training process, each mobile device can communicate and share their class-specific

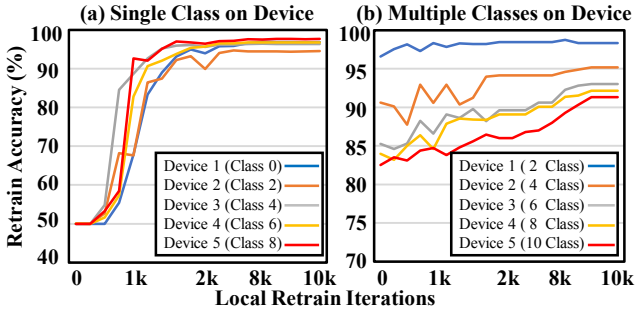


Fig. 10: Class Adaptive Local Training.
(a) Each mobile device was deployed a single-class model.
(b) Each mobile device was deployed a multiple-class model.

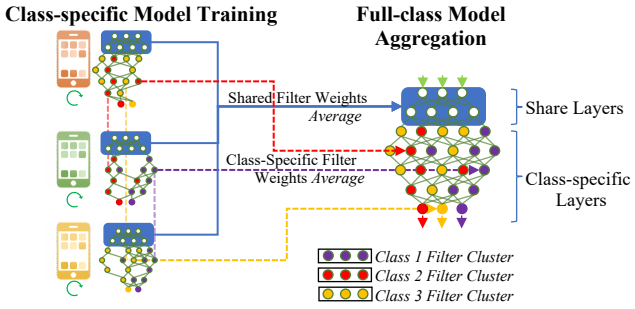


Fig. 11: Full-class Model Aggregation by Filter Cluster Averaging.

updated weight parameters with the central full-class model. Therefore, the central model doesn't require the local training data from the distributed devices. After each training round (E epochs), each mobile device would also download the most updated weight parameters from the central model, and thus indirectly collaborate and achieve consensus with other device during the collaborative training. As a result, all local participants would collaboratively produce a more accurate full-class model which is better than any local model produced by their standalone training.

Specifically, algorithm 2 shows our distributed model collaboration policy utilizes the weight average policy as used in the the Federated Learning [39]. First, the *CaptorX* cloud needs the initialization of class-specific models by model reconfiguration to start the training. According to LUT and classification tasks, each local mobile device was sent the shared layers w_{sl} and the selected class-specific filter clusters $w_{cls(t)}$. Second, each local device trains the class-specific model on the local data and calculates the updates of the filter clusters in parallel. Since our distributed models are heterogeneous, our weight averaging policy are thus different from previous ones: (1) We average the shared layers' parameters for all local models since they are shared by all models; (2) For the class-specific filter clusters, we average such filter clusters only among models who share this class. Finally, the cloud collects the parameters of all local devices, aggregates the updates with the average of all corresponding filter clusters, then updates the cloud full-class model.

Algorithm 2 Distributed Model Collaboration with Class-specific Filter Cluster Average

1: ***CaptorX* On-Device Training:**
for each round 1, ..., t in T; **do**
for each mobile device 1, ..., k in K; **do**
Receive $w_{sl}(t)$ and $w_{cls}(t)$ from *CaptorX* Cloud;
Set $w^k_{cls}(t) = w_{cls}(t)$;
Set $w^k_{sl}(t) = w_{sl}(t)$;
Perform local update and obtain $w^k_{sl}(t)$ and $w^k_{cls}(t)$;
Send $w^k_{sl}(t)$ and $w^k_{cls}(t)$ to *CaptorX* Cloud;

2: ***CaptorX* Cloud Aggregation:**
Reconfigure the pre-trained full-class CNN according to LUT;
Send shared layers $w_{sl}(t)$ and selected class-specific clusters $w_{cls}(t)$ to local devices;
repeat
Receive $w^k_{sl}(t)$ and $w^k_{cls}(t)$ from each local devices k;
Average shared weights and all identical filter clusters;
Send averaged parameters to local devices;
return Enhanced Cloud Central Model;

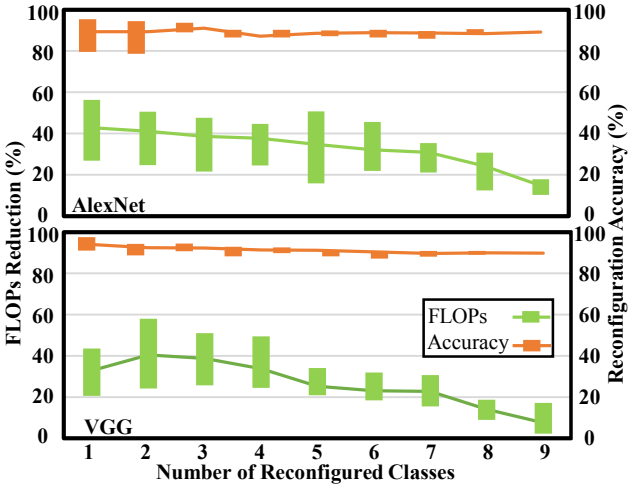


Fig. 12: Computation Load reduction and Classification Accuracy with Different Subset of Class Targets.

VII. Experiments and Evaluation

In this section, a series of experiments are conducted to evaluate the effectiveness of the proposed *CaptorX* framework from three perspectives: (1) Class-adaptive model reconfiguration without local training. (3) Class-adaptive model performance improvement with local training. (3) Distributed model collaboration evaluation.

A. Experiment Setup

To enable the class-adaptive CNN model reconfiguration framework, we implement the model reconfiguration of *CaptorX* on a desktop PC with dual NVIDIA GTX 1080. For the mobile deployment, we implement the *CaptorX* on Google Nexus 5X mobile phones to demonstrate its reconfiguration feasibility and efficiency.

Datasets We select three commonly used computer vision datasets, namely *CIFAR10*, *CIFAR100* and *ImageNet10*, containing a small, and a large number of class targets, representing an easy, and a difficult classification task correspondingly.

- CIFAR10: This dataset contains 50K training images and 10K testing images belonging to 10 class targets.

TABLE III: Comparison between the original 10-class *VGG-16* model and a reconfigured 5-class *VGG-16* model on reduced percentage of FLOPs from the pruned filters.

Layers	Before		After		
	#Filters	#FLOPs	#Filters	#FLOPs	FLOPs Pruned
CL1_1	64	1.84e+06	64	1.84e+06	0%
CL1_2	64	3.78e+07	64	3.78e+07	0%
CL2_1	128	1.89e+07	128	1.89e+07	0%
CL2_2	128	3.78e+07	89	2.63e+07	30.47%
CL3_1	256	1.89e+07	231	1.19e+07	37.23%
CL3_2	256	3.78e+07	203	2.70e+07	28.44%
CL3_3	256	3.78e+07	245	2.87e+07	24.10%
CL4_1	512	1.89e+07	424	1.50e+07	20.74%
CL4_2	512	3.78e+07	473	2.89e+07	23.49%
CL4_3	512	3.78e+07	343	2.34e+07	38.10%
CL5_1	512	9.44e+06	383	4.73e+06	49.88%
CL5_2	512	9.44e+06	387	5.55e+05	43.45%
CL5_3	512	9.44e+06	344	4.79e+05	49.21%
Liner	10	5.13e+03	5	1.75e+02	66.37%
Total	4234	3.13e+08	3383	2.34e+08	**25.21%**

- CIFAR100: This dataset contains 50K training images and 10K testing images belonging to 100 class targets.
- ImageNet10: This dataset is a subset of the ILSVRC ImageNet. It contains 13K training images and 5K testing images belonging to top 10 most popular classification categories based on the popularity ranking provided by the official ImageNet website.

CNN Models Three representative CNN models, namely *AlexNet*, *VGG-16*, and *MobileNet*, are used as implementation targets with task-adaptive reconfiguration. To get the baseline accuracy for each network, we train each model from scratch.

- *AlexNet*: This model has the same number of layer and filter as the original *AlexNet* that is designed for the ImageNet datasets. While we modified the all original filters' size to 3×3 while keeping the number of filters in each layer as the same as the original model, which can be trained with the CIFAR10 datasets and achieve 90.6% classification accuracy.
- *VGG-16*: This model contains 13 convolutional layers, which is widely used in various computer vision tasks. This model can achieve 93.1%, 72.5%, and 92.8% classification accuracy on the *CIFAR10*, *CIFAR100* and *ImageNet10* datasets respectively.
- *MobileNet*: The MobileNet is a family of mobile-first computer vision models [2]. We adopt the *MobileNetV1* for our class-adaptive model reconfiguration. This model can achieve 67.3% and 85.7% classification accuracy on the *CIFAR100* and *ImageNet10* datasets respectively.

Since our framework relies on the output feature maps to calculate the class activation preference – γ_{c,y_j}, we use the training images for both the model reconfiguration and the class-adaptive model training. All accuracy numbers are measured by the testing images. By using mobile support from the *PyTorch* and Android studio development platform, we are able to deploy the class-specific models to mobile devices.

B. *Model Reconfiguration without Local Training*

In this section, we first evaluate our class-adaptive model reconfiguration framework without local training capability.

1) *Classification Accuracy Evaluation*

We evaluate the classification accuracy in different class target number for *VGG-16* and *AlexNet*, respectively. Fig. 12 shows the classification accuracy and computation load reduction when the CNN model is reconfigured to class-specific model from 1 class to 9 class targets. For each test case with certain class number, we randomly choose 10 different combinations of classes to test the pruned models.

As shown in Fig. 12, for the different number of class targets, the length of orange bars demonstrate classification accuracy of 10 repeated experiment. The length of green bars shows the computation load (FLOPs) reduction under different class target number. The average value is demonstrated by the solid lines. We can observe that: (1) Even with 40% of the FLOPs reduction, the reconfigured 2 and 3-class model on average still achieve an optimal accuracy for both *VGG-16* and *AlexNet*. (2) Models for different combinations of classes with the optimal accuracy maintained vary in FLOPs reduction, especially when the network is reconfigured to a small number of class target. For instance, the FLOPs reduction in a reconfigured 2-classes *VGG-16* model varies from 24% to 58%. That is because different class depends on the different subset of clusters. Some classes depend on fewer clusters which can achieve more aggressively FLOPs reduction.

2) *Computation Load Reduction*

To clearly illustrate the model reconfiguration setting and computation load reduction from each layer, we present a reconfigured 5-class *VGG-16* model on CIFAR10 as a case study. We compared a pre-trained 10-class *VGG-16* model with our reconfigured 5-class *VGG-16* model in FLOPs reduction percentage layer by layer. The original architecture and reconfigured architecture of the *VGG-16* model is shown in the Table III. The convolutional layers of *VGG1-6* can be divided into 5 groups and more filers are contained in the higher groups. Since the higher convolutional layers demonstrate stronger exclusive activation preference, more filters can be removed in the higher layer, which benefits for our framework.

We can see that *CaptorX* achieves 25.21% FLOPs reduction in total with optimal accuracy maintained. Moreover, the FLOPs reduction from the lower CLs is relatively small, and there are even no pruned filters in the first three CLs. That is because the filters in the first few layers extracted the basic features, and have more balanced class activation preference on different class targets.

3) *Computation Resource Reduction Evaluation*

Table IV shows the average energy consumption and inference time per sample under the different number of class targets. For the reconfiguration configuration, D_F_r mens the last D convolutional layers are reconfigured (class-specific layers) with $F_r\%$ of class related filters are preserved. Since this reconfiguration does not require model retraining to compensate the accuracy drop, we preserve all class related filters. We can see that: (1) For energy consumption, when

TABLE IV: Average Computation Resource Reduction Per Sample Under Different Class Targets w/o Retraining.

	Classes	1	2	3	4	5	6	7	8	9	10		1	2	3	4	5	6	7	8	9	10
VGG-16	Configuration ($D\_F_r\%$)	7_100	7_100	7_100	7_100	7_100	7_100	7_100	7_100	7_100	-	AlexNet	3_100	3_100	3_100	3_100	3_100	3_100	3_100	3_100	3_100	-
	FLOPs Pruned	32.9%	40.53%	38.79%	33.79%	25.29%	23.01%	22.71%	14.13%	7.49%	0%		42.83%	41.08%	38.61%	37.62%	34.75%	32.13%	30.83%	24.13%	14.79%	0%
	Energy(mJ)	1351	1170	1280	1346	1460	1510	1532	1651	1755	1885		752	772	802	816	857	902	919	977	1070	1205
	Energy Saved	28.3%	37.9%	32.1%	28.6%	22.5%	19.9%	18.7%	12.4%	6.9%	-		37.6%	35.9%	33.4%	32.2%	28.9%	25.1%	23.7%	18.9%	11.2%	-
	Inference Time(ms)	44.8	47.2	50.4	53.3	60.6	61.7	62.1	67.6	72.9	80.7		26.2	27.6	29.3	29.4	30.7	32.8	33.1	35.1	38.1	42.4
	Time Saved	44.5%	41.5%	37.5%	31.4%	24.9%	23.5%	23.0%	16.2%	9.7%	-		38.2%	34.9%	30.8%	30.7%	27.6%	22.6%	21.9%	17.2%	10.1%	-

*VGG Baseline: Accuracy (93.1%), Latency (80.7 ms), Energy (1885.4 mJ), Memory (149.2 MB).
*AlexNet Baseline: Accuracy (90.1%), Latency (42.4 ms), Energy (1250.7 mJ), Memory (157.4 MB).
*($D\_F_r\%$): last D layers are reconfigured (class-specific layers) with $F_r\%$ class related filters are preserved.

TABLE V: Average Computation Resource Reduction Per Sample of MobiNetV1 Under Different Class Targets w/o Retraining.

	Classes	10	15	20	25	30	35	40	45	50	100		1	2	3	4	5	6	7	8	9	10
Cifar100	Configuration ($D\_F_r\%$)	8_100	8_100	8_100	8_100	8_100	8_100	8_100	8_100	8_100	-	ImageNet10	8_100	8_100	8_100	8_100	8_100	8_100	8_100	8_100	8_100	-
	FLOPs Pruned	28.12%	26.21%	25.54%	24.53%	23.23%	22.31 %	19.22%	17.11%	16.81%	0%		23.94%	22.17%	16.24%	14.41%	10.23%	9.51%	7.92%	6.13%	5.32%	0%
	Energy(mJ)	201.2	204.7	217.4	218.5	219.6	222.5	226.7	230.2	231.8	265.8		255.5	259.8	274.4	280.9	291.9	292.6	302.6	308.8	312.4	324.7
	Energy Saved	20.8%	20.0%	15.5%	13.5%	10.1%	19.9%	6.8%	4.9%	3.8%	-		21.3%	20.0%	15.5%	13.5%	10.1%	9.9%	6.8%	4.9%	3.8%	-
	Inference Time(ms)	15.2	15.3	15.5	15.6	15.7	15.8	15.9	16.0	16.3	18.3		192.4	200.3	206.9	208.6	216.9	219.1	226.0	227.9	230.8	238.4
	Time Saved	17.2%	16.6%	15.1%	14.9%	14.2%	13.4%	12.9%	12.4%	11.2%	-		19.3%	16.0%	13.4%	12.5%	9.0%	8.1%	5.2%	4.4%	3.2%	-

*MobileNetV1 CIFAR100 Baseline: Accuracy (67.3%), Latency (18.3 ms), Energy (265.8 mJ), Memory (35.2 MB).
*MobileNetV1 ImageNet10 Baseline: Accuracy (85.2%), Latency (238.4 ms), Energy (324.7 mJ), Memory (63.4 MB).
*($D\_F_r\%$): last D layers are reconfigured (class-specific layers) with $F_r\%$ class related filters are preserved.

the class number choose from 1 to 9, *CaptorX* can achieve at most 37.9% energy consumption reduction for *VGG-16* and 37.6% for *AlexNet*. (2) For the inference time, when the class number choose from 1 to 9, *CaptorX* can save at most 44.5% time for *VGG-16* and 38.2% time for *AlexNet*. Therefore, *CaptorX* can be well deployed on the mobile device and save both energy and inference time to satisfy various resource-constrained mobile scenarios.

Table V shows the average energy consumption and inference time per sample for *MobiNetV1* on CIFAR100 and ImageNet10. Since MobileNet spends most portion of computation time and parameters in 1 $\times$1 convolutions [2], our reconfiguration mainly focus on the 1$\times$1 stand convolutional layers rather than depth-wise convolutional layers. We can see that: (1) For the *CIFAR100*, when the class number choose from 50 to 10, CaptorX can achieve at most 28.12% energy consumption reduction and 20.8% for inference time reduction. (2) For the *ImageNet10*, when the class number choose from 9 to 1, CaptorX can achieve at most 21.3%energy consumption reduction and 19.3% inference time reduction.

C. *Model Reconfiguration with Local Training*

In this section, we further evaluate class-adaptive model performance acceleration with local training. By leveraging the local device computation capability, we can prune the pre-trained full-class model more aggressively to achieve more computation reduction.

1) *Classification Accuracy Evaluation*

We evaluate the classification accuracy in different class reconfiguration for *AlexNet* and *VGG-16* respectively. Fig. 13 shows the classification accuracy when the CNN model is reconfigured to different class targets (*CIFAR10*: 2 to 10, and *ImageNet10*: 2 to 10, *CIFAR100*: 10 to 50). Same as the evaluation without model retraining, for certain number of class target, we evaluate average inference accuracy of the 10 random combinations.

As shown in Fig. 13, for different number of class targets, we demonstrate the classification accuracy of our proposed reconfiguration method, training from scratch and baseline accuracy. The training from scratch is based on a original full-size model but trained on the specific number class targets. The original accuracy is the original pre-trained model accuracy on the specific number class targets.

We can observe that: (1) Compared with the the training from scratch, our method can achieve comparable accuracy. However, our method only requires only one-tenth retraining epochs to reconfigure the pre-trained model to any small models. (2) Our reconfigured model can achieve higher ac-

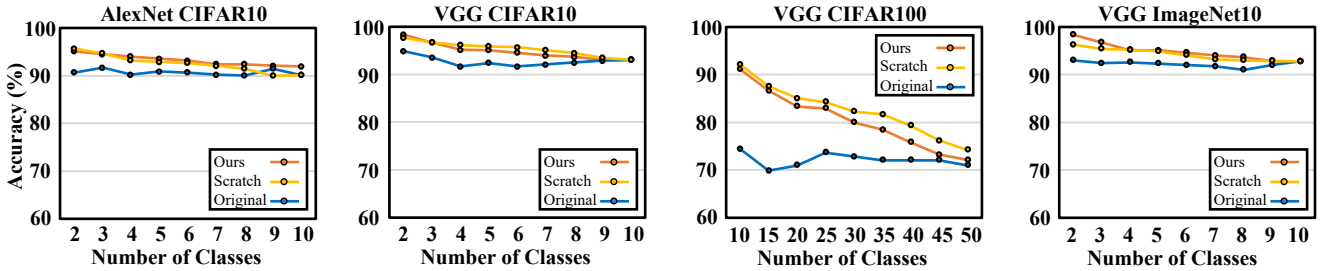


Fig. 13: Classification Accuracy Evaluation with Different Subset of Class Targets. **Ours:** Class-adaptive Model Reconfiguration with local Training; **Scratch:** Training Class-adaptive Model from Scratch; **Original:** Pre-trained Full-class Model Testing on Subset of Classes.

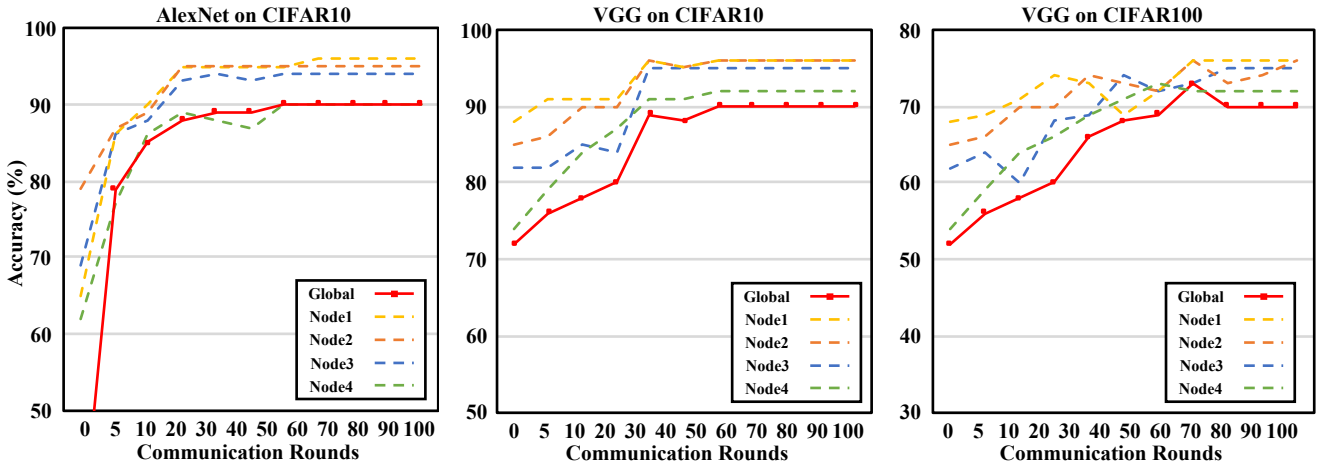


Fig. 14: Distributed Model Collaboration Evaluation.
Solid Line: Accuracy of Aggregated Full-class Model; **Dash Line:** Accuracy of Class-specific Models on Device Training.

curacy than the full-class model (original) testing accuracy under all different subset of classes, especially when the CNN model is reconfigured with a small number of class target. For instance, 2-classes *VGG-16* on the *CIFAR10* model can achieve up to 98.35%. Note that the full-class model classification is much harder than the reconfigured model. The accuracy improvement is mainly benefit from the specific utilization scenario.

*2) **Computation Resource Reduction Evaluation***

In this section, we will further evaluate the computation resource reduction in terms of inference latency, energy consumption, and memory occupation in mobile devices.

Table VI shows the computation resource reduction per sample under the different number of class targets for the *AlexNet* and *VGG-16* on *CIFAR10*. For the *AlexNet*, the optimal class adaptive configuration is D=3, F_r = 4%. For the *VGG-16*, the the optimal class adaptive configuration is D=7, F_r = 4%. We can see that: (1) For energy consumption, when the class targets choose from 2 to 10, *CaptorX* can achieve at most 41.2% energy consumption reduction for *AlexNet* and 61.4% for *VGG-16*. (2) For the inference time, when the class number choose from 2 to 10, *CaptorX* can save at most 66.5% time for *AlexNet* and 42.6% time for *VGG-16*. (3) For the memory occupation, when the class number choose from 2 to 10, *CaptorX* can save at most 19.5% time for *AlexNet* and 72.5% time for *VGG-16*. (4) Note that, when the number of class target is 10, our methods can be viewed as tradition model compressing without class-adaptive reconfiguration. However, our method can still significantly reduce the computation resource consumption.

Table VII shows the computation resource reduction per sample under the different number of class targets for the *VGG-16* on *CIFAR100* and *ImageNet10*. With more class composition (*i.e.*, *CIFAR100*), the filters clusters in meta-component becomes less. The optimal class adaptive configuration is D=6, F_r = 1%. With more input image complexity (*i.e.*, *ImageNet10*), each meta-component needs more filter clusters to extract input features. As a result, the the optimal class adaptive configuration is D=6, F_r = 10%. We can see that: (1) When the class targets choose from 10 to 50, *CaptorX* can achieve at most 59.4% energy consumption reduction, 38.9% inference time reduction and 70.6% memory reduction for *VGG-16* on *CIFAR100*. (2) When the class targets choose from 2 to 10, *CaptorX* can achieve at most 32.1% energy consumption reduction, 37.1% inference time reduction and 6.9% memory reduction for *VGG-16* on *ImageNet10*.

Therefore, *CaptorX* can be well deployed on the mobile device and save both energy and inference time to satisfy various resource-constrained mobile scenarios.

TABLE VI: Average Computation Resource Reduction for *AlexNet* and *VGG-16* on *CIFAR10* with Retraining.

	Classes	2	3	4	5	6	7	8	9	10		2	3	4	5	6	7	8	9	10
AlexNet on CIFAR10	Configuration (D_F_r%)	3_4%	3_4%	3_4%	3_4%	3_4%	3_4%	3_4%	3_4%	3_4%	VGG-16 on CIFAR10	7_4%	7_4%	7_4%	7_4%	7_4%	7_4%	7_4%	7_4%	7_4%
	FLOPs Pruned (%)	70.86	69.75	68.64	67.53	66.42	65.31	64.20	63.09	61.98		50.13	49.60	49.07	48.54	48.01	47.48	46.49	46.42	45.89
	Energy(mJ)	320.74	324.61	328.49	332.36	336.23	340.10	343.97	347.84	351.72		140.76	141.15	141.54	141.93	142.31	142.70	143.09	143.48	143.87
	Energy Saved (%)	41.2	40.5	39.8	39.1	38.4	37.6	36.9	36.3	35.6		61.4	61.3	61.2	61.1	61.0	60.9	60.8	60.7	60.6
	Inference Time(ms)	72.35	74.6	76.86	79.11	81.36	83.61	85.87	88.12	90.37		50.42	50.82	51.21	51.61	52.0	52.4	52.8	53.19	53.59
	Time Saved (%)	66.5	65.5	64.4	63.4	62.3	61.3	60.2	59.2	58.2		42.6	42.1	41.7	41.2	40.8	40.3	39.8	39.4	38.9
	Memory(MB)	131.66	132.03	132.41	132.78	133.15	133.52	133.9	134.27	134.64		40.96	41.32	41.68	42.04	42.4	42.76	43.12	43.48	43.84
	Memory Saved (%)	19.5	16.1	15.8	15.6	15.4	15.1	14.9	14.7	14.5		72.5	72.3	72.0	71.8	71.5	71.3	71.0	70.8	70.6

*AlexNet Baseline: Accuracy (90.1%), Latency (216.2 ms), Energy (545.7 mJ), Memory (157.4 MB).
*VGG Baseline: Accuracy (93.1%), Latency (87.84 ms), Energy (365.4 mJ), Memory (149.2 MB).
*(D_F_r%): last D layers are reconfigured(class-specific layers) with F_r% class related filters are preserved.

TABLE VII: Average Computation Resource Reduction for *VGG-16* on *CIFAR100* and *ImageNet10* with Retraining.

	Classes	10	15	20	25	30	35	40	45	50
VGG-16 on CIFAR100	Configuration ($D\_F_r\%$)	7_4%	7_4%	7_4%	7_4%	6_1%	6_1%	6_1%	6_1%	6_1%
	FLOPs Pruned (%)	45.91	43.26	40.60	37.94	37.43	37.15	36.86	36.57	36.28
	Energy(mJ)	148.12	148.4	148.69	148.97	149.25	149.54	149.82	150.10	150.39
	Energy Saved (%)	59.4	59.3	59.3	59.2	59.1	59.1	59.0	58.9	58.8
	Inference Time(ms)	53.59	55.57	55.57	55.57	60.01	60.24	60.47	60.7	60.94
	Time Saved (%)	38.9	36.5	36.5	36.5	31.5	31.3	31.2	30.6	30.6
	Memory(MB)	43.84	45.64	47.44	49.24	48.22	48.74	49.26	49.79	50.31
	Memory Saved(%)	70.6	69.4	67.9	66.1	66.8	66.5	67.0	64.0	66.4

*VGG Baseline: Accuracy (72.15%), Latency (87.8 ms), Energy (365.5 mJ), Memory (149.5 MB).

*($D\_F_r\%$): last D layers are reconfigured(class-specific layers) with $F_r\%$ class related filters are preserved.

	Classes	2	3	4	5	6	7	8	9	10
VGG-16 on ImagenNet10	Configuration ($D\_F_r\%$)	6_10%	6_10%	6_10%	6_10%	6_10%	6_10%	6_10%	6_10%	6_10%
	FLOPs Pruned (%)	37.23	36.30	35.38	34.45	33.52	32.59	31.66	30.74	29.80
	Energy(mJ)	1736.2	1759.9	1783.6	1807.3	1831.0	1854.6	1878.3	1902.0	1925.7
	Energy Saved (%)	32.1	32.2	30.3	29.3	28.4	27.5	26.5	25.6	24.7
	Inference Time(ms)	924.6	938.2	951.8	965.4	979.0	992.6	1006.2	1019.8	1033.4
	Time Saved (%)	37.1	38.0	35.3	34.3	33.4	32.5	31.5	29.72	29.04
	Memory(MB)	238.18	241.26	244.35	247.43	250.51	253.59	256.68	259.76	262.84
	Memory Saved(%)	6.9	5.8	4.6	3.5	2.1	0.9	-	-	-

*VGG Baseline: Accuracy (92.8%), Latency (1470.1 ms), Energy (2558.2 mJ), Memory (255.93 MB).

D. Distributed Model Collaboration Evaluation

In this section, we further evaluate the effectiveness of our full-class model aggregation scheme. Two implementation scenarios are designed in this part. (1) The *CIFAR10* training dataset is deployed to 4 mobile devices, and the two CNN model are reconfigured into 4 class-specific model that each has two three random class targets. (2) We randomly select 35 class targets from *CIFAR100*, and the total 100 class targets are deployed on 4 mobile devices.

Fig. 14 shows the full-class model aggregation evaluation for the *AlexNet* and VGG on the *CIFAR10* and *CIFAR100* dataset. For the *AlexNet*, the last 3 convolutional layers are reconfigured with 4% of class related filters for each classification targets. For the *VGG-16*, the last 7 convolutional layers are reconfigured with 4% of class related filters for each classification targets. In our distribute training, we set E=1, which means in each communication round the local devices train the reconfigured model on their local for 1 epoch.

As shown in Fig. 14, the horizontal axis indicates the communication round between the cloud server and each local mobile devices, and the vertical axis indicates the local and global training accuracy. We can see that, different local devices demonstrate different classification performance on their local training data due to different class targets combination. However, for the *AlexNet* model on *CIFAR10*, the central model can achieve 90.2% classification accuracy under 60 communication round. For the *VGG-16* model on *CIFAR10*, the global model can achieve 90.5% classification accuracy under 50 communication round. After 50 communication round, the *VGG-16* model *CIFAR100* can achieve 70.1% classification accuracy while reducing up to 37.58% communication cost compared to traditional distributed learning methods without model reconfiguration.

Therefore, *CaptorX* can be well deployed on the mobile device for distributed training scenario. The local private data doesn't need to upload to the cloud server while the global model can still achieve optimal performance as the centralized trained model.

E. Comparison with Related Works

In this section, we compared our proposed *CaptorX* with other class-level reconfiguration methods in terms of classification accuracy in the table VIII. We can see that our proposed method outperforms the Tree-CNN in in both *CIFAR10* and *CIFAR100* datasets. In addition, the model design in the Tree-CNN is not fixed, it is computationally expensive and time-consuming in designing and training new models [43]. However, our proposed *CaptorX* can quickly reconfigure the pre-trained model into any small models. For the HSD-CNN, SaiRam *et al.* measured the activation difference on each class target by applying a certain amount of perturbation to each filter [10]. They trained CNN models into class-specific tree structures, which can be reconfigured to a different subset of class targets. However, when the number of class targets become larger, the computation increased significantly. Hence, the HSD-CNN only reconfigure the pre-trained CNN with relatively small subset of classification targets. Also, our method still outperforms HSD-CNN in *CIFAR100* dataset.

VIII. CONCLUSION

In this paper, we proposed *CaptorX* – a class-adaptive CNN reconfiguration framework for high-performance mobile computing. *CaptorX* can reconfigure the general pre-trained full-class CNN model to class-specific CNN models for dedicated mobile applications. In this framework, we identify the convolutional filters' exclusive activation preference to specific class targets by CNN visualization, and propose a novel CNN reconfiguration method based on the filters' functionality. With our layer-wise and cluster-wise model reconfiguration optimization scheme, the reconfigured class-specific CNN models can be well offloaded to mobile devices for optimal computation efficiency and utilization feasibility. In addition, we can also leverage the local computation capability to further improve the local model computation efficiency. Furthermore, *CaptorX* framework was implemented in a distributed collaboration setting, which can collaboratively update a central model for inference and future reconfiguration.

In our future work, we will further examine the close combination of the neural network visualization and the mobile system features for more performance escalation.

TABLE VIII: Comparison of our proposed *CaptorX* framework with other CNN class-oriented reconfiguration methods.

Datasets	CIFAR10	CIFAR100						
Classes	10	10	11	13	16	20	30	50
Tree-CNN [43]	86.24	85.2	*	*	*	82	79.2	69.0
HSD-CNN [10]	93.33	84.9	85	85.8	72.8	*	*	*
CaptorX	92.1	91.1	90.1	87.3	85.3	83.3	80.0	72.1

*Not reported in the original paper.

REFERENCES

[1] R. Girshick and *et al*, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. of Computer Vision and Pattern Recognition*, 2014.

[2] A. G. Howard and *et al*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[3] F. N. Iandola, , and *et al*, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[4] F. Yu, , and *et al*, "Rein the robuts: Robust dnn-based image recognition in autonomous driving systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[5] "Monotag - Image Recognition!" Download Best Mobile Apps in Appcrawlr." [Online]. Available: appcrawlr.com/android/Monotag-image-recognition

[6] H. Cai, C. Gan, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[7] Z. Xu and *et al*, "Directx: Dynamic resourceaware cnn reconfiguration framework for real-time mobile applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[8] Z. Qin and *et al*, "Captor: a class adaptive filter pruning framework for convolutional neural networks in mobile applications," in *Proc. of Asia and South Pacific Design Automation Conference*, 2019.

[9] J. Guo and M. Potkonjak, "Pruning convnets online for efficient specialist models," in *Proc. of Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017.

[10] K. SaiRam and *et al*, "Hsd-cnn: Hierarchically self decomposing cnn architecture using class specific filter sensitivity analysis," *arXiv preprint arXiv:1811.04406*, 2018.

[11] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Network dissection: Quantifying interpretability of deep visual representations," in *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2017.

[12] P. Wang and J. Cheng, "Accelerating convolutional neural networks for mobile applications," in *Proc. of ACM on Multimedia Conference*, 2016.

[13] Z. Xu and *et al*, "Direct: Resource-aware dynamic model reconfiguration for convolutional neural network in mobile systems," in *International Symposium on Low Power Electronics and Design*, 2018.

[14] T.-J. Yang and *et al*, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[15] Y. LeCun, "LeNet-5, Convolutional Neural Networks," 2015. [Online]. Available: http://yann.lecun.com/exdb/lenet

[16] A. Krizhevsky and *et al*, "Imagenet classification with deep convolutional neural networks," in *Proc. of Advances in Neural Information Processing Systems*, 2012, pp. 1098–1105.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[18] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proc. of International Conference on Learning Representations*, 2014.

[19] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proc. of International Conference on Learning Representations*, 2016.

[20] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. of Neural Information Processing Systems*, 2015.

[21] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. of International Conference on Learning Representations*, 2017.

[22] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. of International Conference on Computer Vision*, 2017.

[23] Z. Qin and *et al*, "Functionality-oriented convolutional filter pruning," in *The British Machine Vision Conference*, 2019.

[24] F. Yu, , and *et al*, "Dc-cnn: computational flow redefinition for efficient cnn through structural decoupling," in *Workshop on Compact Deep Neural Networks with Industrial Applications*, 2018.

[25] Z. Qin, , and *et al*, "Demystifying neural network filter pruning," in *Workshop on Compact Deep Neural Networks with Industrial Applications*, 2018.

[26] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proc. of British Machine Vision Conference (BMVC)*, 2014.

[27] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. of Neural Information Processing Systems (NIPS)*, 2014.

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

[29] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proc. of European Conference on Computer Vision (ECCV)*, 2016.

[30] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. of Neural Information Processing Systems (NIPS)*, 1990.

[31] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *Proc. of Neural Networks*, 1993.

[32] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *Proc. of International Conference on Learning Representations (ICLR)*, 2016.

[33] Z. Qin and *et al*, "How convolutional neural network see the world: A survey of convolutional neural network visualization methods," *Mathematical Foundations of Computing*, 2018.

[34] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Object detectors emerge in deep scene cnns," *arXiv preprint arXiv:1412.6856*, 2014.

[35] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," in *Proc. of International Conference on Machine Learning Workshops*, 2015.

[36] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," *University of Montreal*, 2009.

[37] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[38] M. Li and *et al*, "Scaling distributed machine learning with the parameter server," in *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, 2014.

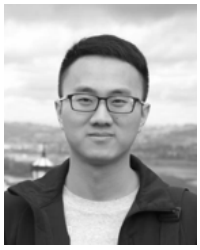
[39] H. B. McMahan and *et al*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[40] F. Yu, , and *et al*, "Heterogeneous federated learning," in *arXiv preprint arXiv:2008.06767*, 2020.

[41] J. Mao, Z. Qin, Z. Xu, K. W. Nixon, X. Chen, H. Li, and Y. Chen, "Adalearner: An adaptive distributed mobile learning system for neural networks," in *Proc. of International Conference on Computer-Aided Design*. IEEE, 2017.

[42] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[43] D. Roy, P. Panda, and K. Roy, "Tree-cnn: a hierarchical deep convolutional neural network for incremental learning," *arXiv preprint arXiv:1802.05800*, 2018.



Zhuwei Qin received the M.S. degree from Oregon State University, Corvallis, USA in 2017, and received the Ph.D. degree from the ECE Department at the George Mason University, Fairfax, USA, in 2020. He is currently an Assistant Professor in the School of Engineering of San Francisco State University.

His research interests are in the broad area of efficient mobile computing, deep learning acceleration, distributed edge computing, and interpretable deep learning.

Fuxun Yu received the B.S. degree from Harbin Institute of Technology, China, in 2017. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering at George Mason University under the supervision of Prof. Xiang Chen.

His current research directions include deep learning security, adversarial attacks and defenses on neural network, high-performance deep neural network on mobile devices, and interpretability and explainability of deep learning.

Zirui Xu received the B.S. and M.S. degree from Beijing Jiaotong University, Beijing, China, in 2014 and in 2017, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering at George Mason University under the supervision of Prof. Xiang Chen.

His current research interests include high performance mobile computing system, neural network model optimization, and mobile intelligent application robustness and security.

ChenChen Liu received the M.S. degree from Peking University, China in 2013, and received the Ph.D. degree from the ECE Department at the University of Pittsburgh in 2017. She is currently an Assistant Professor in the Department of Computer Science and Electrical Engineering of the University of Maryland, Baltimore County.

Her current research interests include brain-inspired computing system and security, integrated circuits design, and emerging nonvolatile memory technologies.

Xiang Chen received his M.S. and Ph.D. degree from the ECE Department at the University of Pittsburgh in 2012 and 2016, respectively.

He is currently an Assistant Professor in the Department of the Computer Engineering of the George Mason University and the founder of the Intelligence Fusion Laboratory. His research interests are in the low-power mobile system, high-performance mobile computing, machine learning, and secure computing system. In the past years of research, he has published more than 30 papers in the top international conferences and journals and received many best paper nominations and other awards. He also stays in close cooperation with not only academic society (such as Duke, UCSB, PITT, Syracuse, Tsinghua, HKUST, CityU, etc.), but also industries, such as the research labs of HP, Samsung, MSRA, Marvell, Amazon, and Apple.