CAPTOR: A Class Adaptive Filter Pruning Framework for Convolutional Neural Networks in Mobile Applications

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 online with vast classification targets to cover various utilization contexts. However, only partial classes are practically tested due to individual mobile user preference and application specificity. Thus the unneeded classes cause considerable computation and communication cost. In this work, we propose *CAPTOR* – a class-level reconfiguration framework for Convolutional Neural Networks (CNNs). By identifying the class activation preference of convolutional filters through feature interest visualization and gradient analysis, *CAPTOR* can effectively cluster and adaptively prune the filters associated with unneeded classes. Therefore, *CAPTOR* enables class-level CNN reconfiguration for network model compression and local deployment on mobile devices. Experiment shows that, *CAPTOR* can reduce computation load for *VGG-16* by up to 40.5% and 37.9% energy consumption with ignored loss of accuracy. For *AlexNet*, *CAPTOR* also reduces computation load by up to 42.8% and 37.6% energy consumption with less than 3% loss in accuracy.

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 Networks (CNNs) have been widely adopted by those applications for classification tasks [1,2]. However, their potential is highly restrained by limited local mobile computation resource. Therefore, most applications are still depending on online CNN training and testing with external infrastructures, which significantly compromises the local computation resource utilization and offline accessibility [3,4].

Many previous works have been proposed to compress and accelerate CNNs for local computation on mobile devices: Han *et al.* removed the weights with small values to reduce model size [5]. Li *et al.* ranked the convolutional filters with their absolute values to expel the insignificant filters for better computation efficiency [6]. Jaberberg *et al.* decomposed a convolutional layer matrix into multiple shrieked ones to reduce the matrix operation loads [7]. Efficient and effective as they are, most of them focus on network compression targeting weight, filter, and layer levels, while the class-level oriented optimization is highly overlooked.

Despite the lack of research, the class-level compression is specifically necessary for mobile applications. For current CNN utilization, the networks are generally trained online with vast classification targets to cover various utilization contexts (*e.g.* *ImageNet* with 1000 targets). However, only partial classes are practically needed in mobile application scenarios. For example, we analyzed the "Top 100 Intelligent Android Applications" [8] and found over 22% applications only focus on one kind of classification target (*e.g.* faces, handwriting, books, *etc.*) [9]. Therefore, a large amount of well-trained classes are actually unneeded due to individual mobile user preference and task specificity. Such a class-level redundancy introduces considerable computation and communication cost.

In this work, we propose *CAPTOR* – a class adaptive reconfiguration framework for Convolutional Neural Networks (CNNs) in mobile applications. Through CNN visualization analysis, we found that convolutional filters may have dedicated class activation preference, and the filters with similar preference can be well clustered. As shown in Fig. 1, *CAPTOR* adaptively prunes the filter clusters associated with unneeded classes and reconfigures a general CNN trained online to application specific model with dedicated class targets. Such a class-level compressed CNN can be well offloaded to local mobile computation for optimal efficiency and accessibility.

In this work, we have the following contributions:

- We analyzed the class preference similarity of a CNN's convolutional filters through feature interest visualization. An k-means based algorithm is also proposed to group filters with similar interests into independent clusters;
- We defined the class activation preference label of each filter cluster by evaluating backward-propagation gradients in terms of class activation impact;
- We proposed a class oriented filter pruning framework, which prunes the filter clusters associated with the unneeded classes according to specific classification task.

Experiment shows that, *CAPTOR* can reduce computation load for *VGG-16* by up to 40.5% and 37.9% energy consumption with no loss of accuracy. For *AlexNet*, *CAPTOR* also reduces computation load by up to 42.8% and 37.6% energy consumption with less than 3% loss in accuracy.

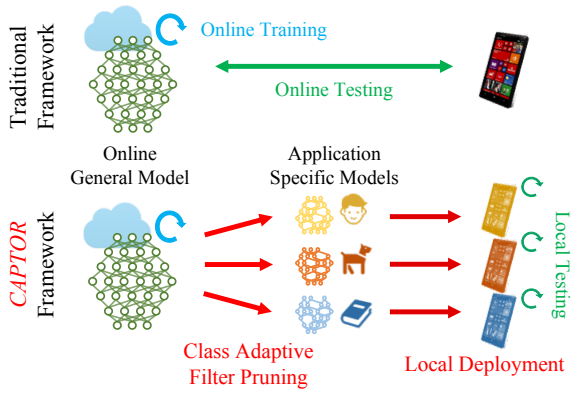

Fig. 1. Class Adaptive Reconfiguration for Mobile CNNs

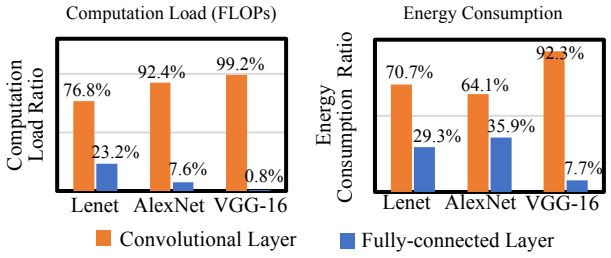

Fig. 2. CNNs Computation Load and Energy Consumption

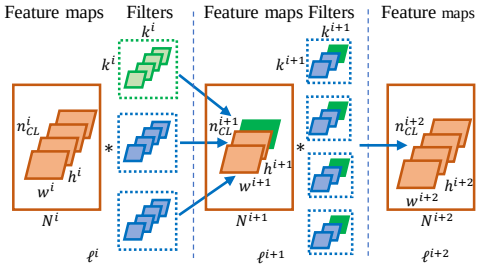

Fig. 3. Illustration of CNNs Filter Pruning

II. Preliminary

A. Mobile CNN Computation Consumption Analysis

The sophisticated CNN structure consists of multiple convolutional layers (CLs), pooling layers (PLs), and fully-connected layers (FLs). Such a structure enables outstanding performance, but at a cost of intensive computation load and energy consumption. Specifically, 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 [10].

Fig. 2 analyzes the computation load and energy consumption of CLs and FLs with of three representative CNNs on a Google Nexus 5X, *i.e. LeNet* [11], *AlexNet* [12], and *VGG-16* [13]. Fig. 2 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 network becomes deeper from *LeNet* to *VGG-16*, the cost of CLs also significantly increases.

While mobile storage space issue is relative ignorable nowadays [14], 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. Convolutional Filter Pruning

As CLs cause the most computation cost, many CLs oriented optimization works have been proposed, such as weight sparsity [5, 15], filter pruning [6], layer decomposition [7], *etc.* Considering the filters are the most fundamental and interpretable component of a CL, we mainly leverage the filter pruning scheme as the primary optimization approach.

Fig. 3 demonstrates the mechanism of the convolutional filter pruning scheme with three CLs. Given a CL of l^i, it is consist of a set of filters, namely $F \in R^{k^i \times k^i \times n^i_{CL}}$ (represented by blue and green blocks). Each filter also contains multiple convolutional kernels of $K \in R^{k^i \times k^i}$. The convolutional operation generates and passes feature maps between layers (*i.e.* $N^i \in R^{h^i \times w^i \times n^i}$ and $N^{i+1} \in R^{h^{i+1} \times w^{i+1} \times n^{i+1}}$ as the input and output of l^i). The convolutional filter pruning scheme aims to prune certain "insignificant" filters in the layer of l^i (represented by green block). As a filter in the layer of l^i is pruned, its connected output feature maps and the corresponding convolutional kernels in layer l^{i+1}'s filters are also pruned. Meanwhile, the filter number in l^{i+1} remains the same as well as the output feature map size of $N^{i+2} \in R^{h^{i+2} \times w^{i+2} \times n^{i+2}}$ in l^{i+2}. Therefore the pruned filters and feature maps could effectively reduce certain computation load.

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

$$W_{CL_{i_{th}}} = n^{i+1}_{CL} \times (n^i_{CL} \times (k^i)^2) \times (h^{i+1} \times w^{i+1}), \quad (1)$$

where the n^i is total amount of filters and output feature maps.

By pruning each filter in i^{th} CL, the computation load reduction can be formulated as:

$$\begin{aligned} W^-_{CL_{i_{th}}} =& (n^i_{CL} \times (k^i)^2) \times (h^{i+1} \times w^{i+1}) + \\ & n^{i+1}_{CL} \times (k^{i+1})^2 \times (h^{i+2} \times w^{i+2}). \end{aligned} \quad (2)$$

In this work, we found that the filters may have dedicated class activation preference, and we consider the filters associated with unneeded classes as "insignificant" ones to prune. In the next section, we will illustrate the identification process of the class activation preference of each convolutional filter.

III. Class Activation Preference of Convolutional Filters

To achieve class adaptive filter pruning, we define the class activation preference label of each convolution filter by evaluating the class activation impact, which is measured by the backward-propagation gradients. Considering the massive volume of convolutional filters, we first introduce a cluster scheme based on CNN visualization analysis, which groups the filters based on their preference similarity in terms of feature interest.

A. Visualization Analysis of Filter Feature Interest

As the fundamental feature extract components, each convolutional filter has dedicated sensitivity for different input features. From the CNN interpretation perspective, such filter characteristic is defined as feature interest. And many works have been proposed to analyze the feature interest to interpret the CNN inner mechanism. Among those works, the CNN visualization is considered as the most effective approach, which is widely adopted for filter feature interest analysis in an intuitive and qualitative approach [16, 17].

In this work, we adopt the Activation Maximization technique (AM) [18], one of the most effective CNN visualization method, to analyze the feature interest of convolutional filters. In AM, the features interest of a convolutional filter can be qualitatively represented by a graphic pattern of $V(F^l_i)$, which indicates the most sensitive feature patterns to the filter. The generation $V(F^l_i)$ can be considered as a synthesizing an input image to the network that delicately maximizes the activation (*i.e* output feature map) of i_{th} filter F^l_i in the layer of l. Mathematically, this synthesis process can be formulated as:

$$V(F^l_i) = \underset{X}{\operatorname{argmax}} A^l_i(X), \qquad X \leftarrow X + \eta \cdot \frac{\partial A^l_i(X)}{\partial X}, \quad (3)$$

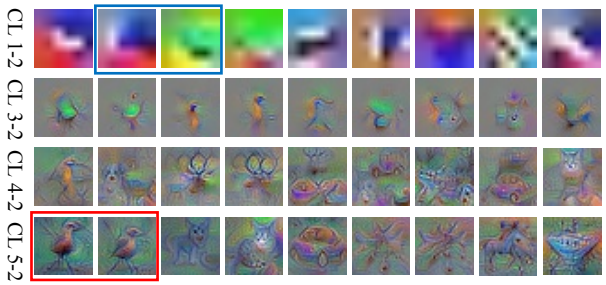

Fig. 4. The Visualized Feature Interest Patterns from Different Layers

where, $A_i^l(X)$ is the activation of F_i^l from an input image X, η is the gradient ascent step size.

When X has the highest activation of $A_i^l(X)$, it is considered as the visualized $V(F_i^l)$. To approach the highest activation, X is initialized as an image of random noise. Then each of its pixels changed iteratively by the guidance of gradients of $\partial A_i^l(X)/\partial X$. Eventually, the visualized $V(F_i^l)$ contains the filter's most preferred input features, and therefore represents the filters' feature interests. To interpret all the filters' feature interest, this process is applied to all filters repeatedly.

Fig. 4 illustrates some AM visualized filter feature interest patterns, which come from 4 CLs trained with *VGG-16* model and *Cifar-10* dataset [19]. From Fig. 4, we can see that: (1) The graphic patterns of feature interest in lower layers (*e.g.* CL1-2) tend to be simpler (*e.g.* edges and lines), which indicates fundamental feature extraction without dedicated class activation preference. (2) As the layer deepens, the complexity and variation of the visualized feature interest patterns increase. The visualized feature interest patterns start to synthesize shapes and even objects. (3) In even higher layers (*e.g.* CL4-2 and CL5-3), clear class object patterns emerge, demonstrating filters' intuitive class activation preference. For example, object patterns of bird, car, and cat can be easily found in the visualized feature interest patterns from CL5-2. (4) Moreover, many similar graphic patterns also present in each convolutional layer, which indicates the corresponding filter has similar feature interest and can be well clustered.

Based on the observation, we found that many filters may share dedicated class activation preferences *w.r.t.* feature interest. To further prepare those filters for class adaptive pruning, we also cluster them based on their feature interest similarity.

B. *Filter Clustering based on Feature Interest Similarity*

The visualized filters' feature interest patterns demonstrate certain class activation preference and similarity. To cluster the filters according to their feature interest similarity, we take an image analysis approach with the visualized graphic patterns.

As illustrated in Algorithm 1, we first utilize the AM to obtain the visualized filter feature interest pattern of each convolutional filter. Then, we adopt *k*-means algorithm with pixel-level *Euclidean-distance* on the visualized patterns of all convolutional filters in each individual layer. To determine proper cluster amount in each layer, we perform a grid search from one to half of the total filter number. With more cluster amount, smaller pattern differences are taken into consideration, which might overwhelm the similarity clustering. To prevent excessive cluster amount, the grid search selects the clustering condition that most filters are clustered into least clusters. And the non-clustered filters are grouped in a special cluster, which is considered to have unique features and won't participate the later class adaptive filter pruning.

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 c in range (1, $I_l/2$) **do** 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

C. *Clusters' Class Activation Preference Label based on Gradient Analysis*

Based on the visualized filter feature interest analysis, we clustered the convolutional filters in each CL based on their feature interest similarity. And from the observation from Fig. 4, we found that the feature interests may demonstrate certain class activation preference. Therefore, we define the class activation preference label for filter clusters to associate each of them with a dedicated class target.

To quantitatively define the cluster label, we evaluate the class activation impact of each filter cluster by examining the backward-propagation gradients. And the gradients demonstrate the impact of each convolutional filter to a given class:

$$\gamma_{i,y_j} = \frac{1}{N}\sum_{n=1}^{N}\left\|\frac{\partial P(y_j)}{\partial A_i(x_n)}\right\|, \tag{4}$$

where the $P(y_j)$ is the output probability of a sample image n corresponding to the class y_j, and the $A_i(x_n)$ is the activation (output feature map) of filter 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 i, which indicates the activation preference of filter i to class y_i. We further select $N = 1000$ random test samples from *Cifar-10* dataset (100 samples from each class) for gradient analysis. The averaged value of γ_{i,y_j} can be considered the cluster's activation preference to a certain class. Through our preliminary experiment, we found that each filter in the same cluster has similar class activation preference, which further justifies our analysis and clustering scheme.

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 preference label:

$$\gamma_{c,y_j} = \frac{1}{I}\sum_{i=1}^{I}\gamma_{i,y_j}. \tag{5}$$

Therefore, the class activation preference labels effectively associate filters with classes for adaptive pruning.

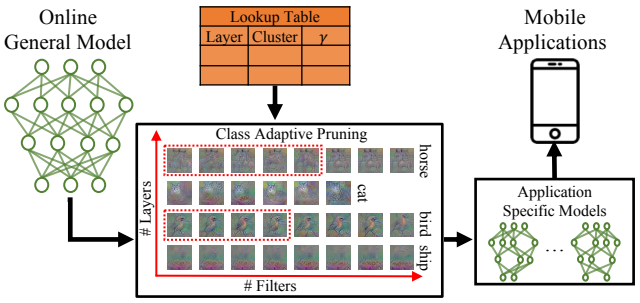

Fig. 5. Implementation of Class Adaptive Filter Pruning Framework

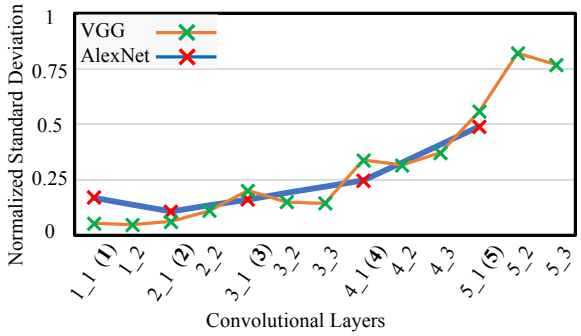

Fig. 6. The Evaluation of Class Activation Preferences Level of Different Convolutional Layers in *VGG-16* and *AlexNet* model.

IV. *CAPTOR*: Class Adaptive Filter Pruning

After filter clustering and class preference labeling, we will dive into the details of the class adaptive filter pruning.

A. *CAPTOR Framework Overview*

Fig. 5 shows the implementation of our class adaptive filter pruning framework. Given a pre-trained general CNN online, we are supposed to quickly reconfigure this CNN to specialized models for dedicated mobile applications. Based on the filter clustering through filter visualization and class activation preference labeling with gradient analysis, the filters are associated with certain class targets. For each class (or class combination), *CAPTOR* finely prunes the filters online regarding cluster and layer level schemes to achieve the optimal class-level filter pruning configuration with desired compression rate and accuracy level. The filter pruning configurations are stored in a look-up table (LUT) with the general model online. When the node mobile devices request certain neural network with dedicated classification targets, the general model can be intuitively compressed according to the LUT and shipped to the mobile device for local deployment.

B. *Cluster Level Filter Pruning*

As there is a large number of independent clusters distributed in each layer, certain variances also present in different filter clusters. Therefore cluster adaptive pruning order is necessary for the cluster level pruning. Considering the filter cluster volume and class activation preference, we define the class activation preference level of one cluster c in layer l as:

$$L_{c,l} = \alpha \cdot length(C_c^l) \times \beta \cdot S_{c,l}, \quad S_{c,l} = \sqrt{\frac{1}{J}\sum_{j=1}^{J}(\gamma_{c,y_j} - \mu)^2}, \quad (6)$$

where the $length(C_c^l)$ calculates the cluster volume size, and $S_{c,l}$ is the standard deviation of class activation preference on different classes. The cluster with more stronger class activation preferences will have a larger standard deviation among different classes, while larger cluster volume demonstrates more classification contribution. Taking these two factors into consideration, we set empirical factors (i.e. α and β) for the clusters in each layer. For the unmatched clusters, the cluster with larger $L_{c,l}$ will be first pruned in each layer.

C. *Layer Level Filter Pruning*

Through the feature interest visualization, we found the filters demonstrate stronger class activation preferences in higher CLs. *CAPTOR* is supposed to start filter pruning from the higher CLs to lower CLs. To quantitatively evaluate the class activation preference differences, we define the class activation preferences level of one CL l as:

$$L_l = \sqrt{\frac{1}{C}\sum_{c=1}^{C} S_{c,l}}, \quad (7)$$

where C is the total cluster amount in l. L_l is the average standard deviation of all cluster activation preference on different class targets.

Fig. 6 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 increasing, the layer class activation preference level become larger, which indicates higher CLs have stronger classes activation preference. (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.

D. *Pruning Algorithm Overview*

With the help of cluster activation preference level $L_{c,l}$ and layer level class preference level L_l, we can achieve our proposed class adaptive filter pruning framework. Given a pre-trained online general model and dedicated class targets, our framework can quickly prune the general model to specialized model by referencing the LUT. Meanwhile, no further model retraining process is applied although they have been widely adopted by many other works. Our framework iteratively prunes the filter cluster and test the model accuracy. To

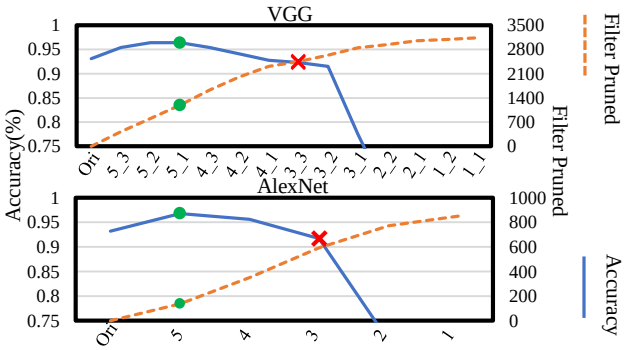

Fig. 7. Single Class Adaptive Filter Pruning

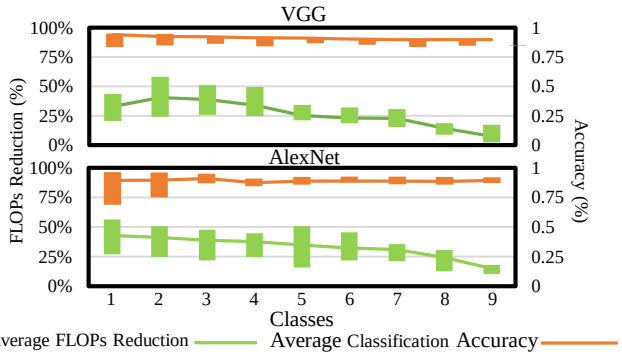

Fig. 8. Computation Load reduction and Classification Accuracy under Different Class Targets

balance the classification accuracy and the computation workload reduction, our framework prunes the filter clusters with maintained original classification accuracy or slightly accuracy drop to gain more computation load reduction.

E. *Pruning Algorithm Analysis with Single Class Target*

From the perspective of algorithm design, we conduct a preliminary effectiveness analysis with a case study to prune the network to only one classification target. As shown in the Fig. 7, the pre-trained *VGG-16* and *AlexNet* are reconfigured to single Truck class in *Cifar-10*. According to the LUT, *CAPTOR* prunes all the Truck-unmatched clusters from higher CLs to lower CLs and examine the classification accuracy. Interestingly, the classification accuracy can be further improved at the first few layers pruning. For instance, the truck classification accuracy of *VGG-16* model can be improved by 3%, marked by the green dot, when 1200 unmatched filters are pruned from CL5-3 to CL5-1. *CAPTOR* can prune 2300 unmatched filters in total from CL5-3 to CL3-3 with only 0.3% accuracy drop, marked by the red "×". Compared with the *VGG-16* model, the *AlexNet* model has a more compact network structure. However, as shown in the Fig. 7, *CAPTOR* can still prune many unmatched filters in *AlexNet* model with optimal accuracy maintained.

The single class reconfiguration is built on the assumption that the CNN is used for explicit data with a specific class or the under test data demonstrate certain class consistency that can be estimated for dedicated adaptation. With our proposed class adaptive pruning framework, *CAPTOR* is expected to effectively optimize CNNs with faster pruning speed and achieve optimal computation efficiency. In later sections, we will further evaluate the practical performance of *CAPTOR* on different class targets in perspectives of computation load, energy consumption and inference time on mobile devices.

V. Implementation and Experiment

In this section, a series of experiments are conducted to evaluate the effectiveness of proposed *CAPTOR* framework through three perspectives: (1) The layer-wise computation load reduction analysis; (2) The specific model accuracy under different class targets; (3) The energy consumption and inference time evaluation.

A. *Experiment Setup*

Two CNN models, namely *VGG-16* and *AlexNet*, are used as test targets with class adaptive reconfiguration. Google Nexus 5X is selected as the test platform. The original models are trained on *CAFFE*, and with the Android support, we are able to deploy the models to mobile phones. We trained the *VGG-16* model on the *Cifar-10* dataset that can achieve 90.2% classification accuracy. For the *AlexNet* model implementation, 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 *Cifar-10* dataset and achieve 90.6% classification accuracy.

B. *Computation Load Reduction*

We first evaluate the computation load reduction on the reduced percentage of FLOPs from the pruned filters. To clearly illustrate the computation load reduction from each layer, we present a reconfigured 5-class *VGG-16* model as a case study.

The architecture of the *VGG-16* model is shown in the Table I. The CLs of *VGG-16* can be divided into 5 groups and more filers are contained in the higher groups. Since the higher CLs demonstrate stronger class impact, more filters can be pruned in the higher layer, which benefits for our framework. We compared the original 10-class *VGG-16* model with our reconfigured 5-class *VGG-16* model in FLOPs reduction percentage from the pruned filters layer by layer. We can see that *CAPTOR* can achieve 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.

C. *Classification Accuracy Evaluation*

We further evaluate the classification accuracy in different class reconfiguration for *VGG-16* and *AlexNet* respectively. Fig. 8 shows the classification accuracy and computation load reduction when the CNN model is reconfigured to 1 to 9 class targets. For each test case, we randomly choose 10 different combinations of classes to test the pruned models.

As shown in Fig. 8, for the different number of class targets, we demonstrate the FLOPs reduction of all 10 repeated experiment by the green color bar and the classification accuracy by the orange bar. 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.

TABLE I
COMPARISON BETWEEN THE ORIGINAL 10-CLASS VGG MODEL AND A PRUNED 5-CLASS VGG 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%**

D. *Energy Consumption and Inference Time Evaluation*

By using mobile support from CAFFE and Android Studio Development platform, we are able to port our reconfigured models to Nexus 5X. For energy consumption, we use Monsoon power monitor to measure it, and the inference time is measured by checking the mobile system time.

Table II shows the average energy consumption and inference time per sample under the different number of class targets. We can see that: (1) For energy consumption, when the class number choose from 1 to 9, *CAPTOR* 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, *CAPTOR* can save at most 44.5% time for *VGG-16* and 38.2% time for *AlexNet*. Therefore, *CAPTOR* can be well deployed on the mobile device and save both energy and inference time to satisfy various resource-constrained mobile scenarios.

VI. CONCLUSION

In this paper, we proposed *CAPTOR* – a class adaptive filter pruning framework to prune the pre-trained CNNs to specialized networks for dedicated mobile applications. In this framework, we analyzed the feature interest of convolutional filters by utilizing CNNs visualization and collected the filters with similar feature interest into the independent cluster. By performing the gradients analysis, we assigned a class label to each cluster. *CAPTOR* can adaptively prune the clusters with the unmatched label for computation efficiency.

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.

REFERENCES

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

[2] 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.

[3] P. Wang and J. Cheng, "Accelerating convolutional neural networks for mobile applications," in *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 2016, pp. 541–545.

[4] J. Hauswald and *et al*, "A hybrid approach to offloading mobile image classification," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, 2014, pp. 8375–8379.

[5] S. Han and *et al*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[6] H. Li and *et al*, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[7] M. Jaderberg and *et al*, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[8] "Best Android apps for: "image recognition" ." [Online]. Available: http://appcrawlr.com/android-apps/best-apps-image-recognition

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

[10] T.-J. Yang and *et al*, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the the Computer Vision and Pattern Recognition*, 2017.

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

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

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

[14] M. Lin and *et al*, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[15] S. Han and *et al*, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[16] Z. Qin and *et al*, "How convolutional neural network see the world-a survey of convolutional neural network visualization methods," *arXiv preprint arXiv:1804.11191*, 2018.

[17] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[18] J. Yosinski and *et al*, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.

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

TABLE II
AVERAGE ENERGY CONSUMPTION AND INFERENCE TIME REDUCTION PER SAMPLE UNDER DIFFERENT CLASS TARGETS.

	Classes	1	2	3	4	5	6	7	8	9	10		1	2	3	4	5	6	7	8	9	10
VGG-16	FLOPs Pruned	32.9%	40.53%	38.79%	33.79%	25.29%	23.01%	22.71%	14.13%	7.49%	0%	AlexNet	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%	-