Exploiting Kernel Sparsity and Entropy for Interpretable CNN Compression

Yuchao Li[1#], Shaohui Lin[1#], Baochang Zhang[2], Jianzhuang Liu[3],
David Doermann[4], Yongjian Wu[5], Feiyue Huang[5], Rongrong Ji[1*]

[1]School of Information Science and Engineering, Xiamen University, China
[2]School of Automation Science and Electrical Engineering, Beihang University, China
[3]Huawei Noahs Ark Lab, China
[4]University at Buffalo, New York, USA
[5]BestImage, Tencent Technology (Shanghai) Co.,Ltd, China

xiamenlyc@gmail.com, shaohuilin007@gmail.com, bczhang@buaa.edu.cn, liu.jianzhuang@huawei.com,
doermann@buffalo.edu, littlekenwu@tencent.com, garyhuang@tencent.com, rrji@xmu.edu.cn

Abstract

Compressing convolutional neural networks (CNNs) has received ever-increasing research focus. However, most existing CNN compression methods do not interpret their inherent structures to distinguish the implicit redundancy. In this paper, we investigate the problem of CNN compression from a novel interpretable perspective. The relationship between the input feature maps and 2D kernels is revealed in a theoretical framework, based on which a kernel sparsity and entropy (KSE) indicator is proposed to quantitate the feature map importance in a feature-agnostic manner to guide model compression. Kernel clustering is further conducted based on the KSE indicator to accomplish high-precision CNN compression. KSE is capable of simultaneously compressing each layer in an efficient way, which is significantly faster compared to previous data-driven feature map pruning methods. We comprehensively evaluate the compression and speedup of the proposed method on CIFAR-10, SVHN and ImageNet 2012. Our method demonstrates superior performance gains over previous ones. In particular, it achieves $4.7\times$ *FLOPs reduction and* $2.9\times$ *compression on ResNet-50 with only a Top-5 accuracy drop of* 0.35% *on ImageNet 2012, which significantly outperforms state-of-the-art methods.*

1. Introduction

Deep convolutional neural networks (CNNs) have achieved great success in various computer vision tasks, including object classification [1, 2, 3, 4], detection [5, 6] and segmentation [7, 8]. However, deep CNNs typically require high computation overhead and large memory footprint, which prevents them from being directly applied on mobile or embedded devices. As a result, extensive efforts have been made for CNN compression and acceleration, including low-rank approximation [9, 10, 11, 12], parameter quantization [13, 14] and binarization [15]. One promising direction to reduce the redundancy of CNNs is network pruning [16, 17, 18, 19, 20], which can be applied to different elements of CNNs such as the weights, the filters and the layers.

Early works in network pruning [16, 17] mainly resort to removing less important weight connections independently with precision loss as little as possible. However, these un-

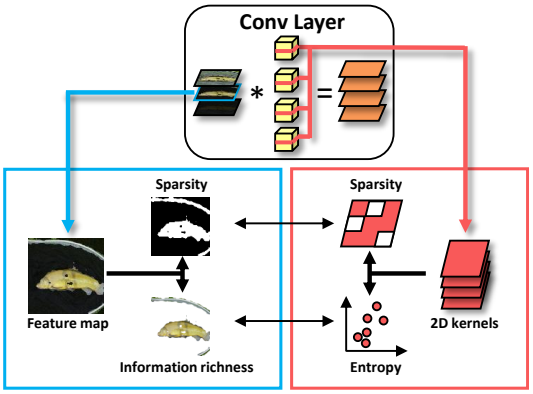

Figure 1. The relationship between an input feature map and its corresponding 2D kernels is investigated. We introduce the kernel sparsity and entropy (KSE) to represent the sparsity and information richness of the input feature maps.

[#]Equal contribution.
[*]Corresponding author.

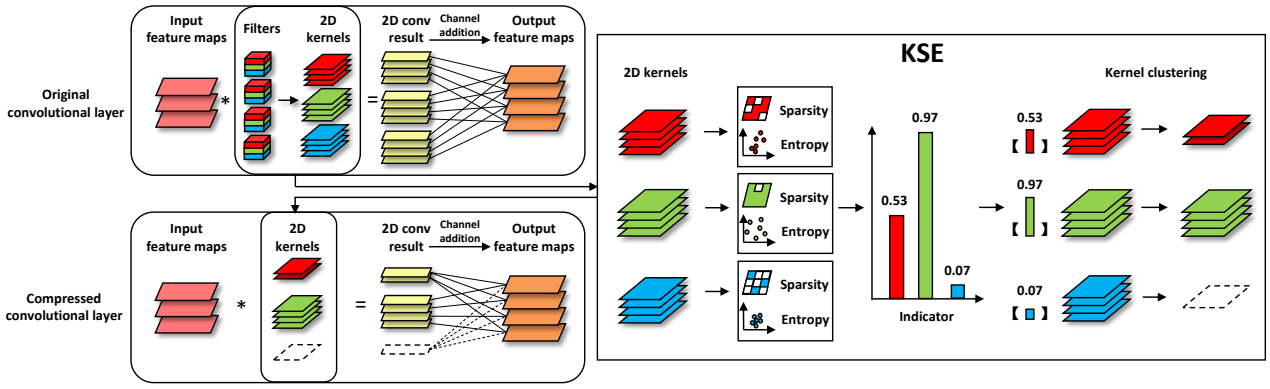

Figure 2. The framework of our method. The convolution operation is split into two parts, 2D convolution and channel fusion (addition). The 2D convolution is used to extract features from each input feature map, and the channel addition is used to obtain an output feature map by summing the intermediate results of the 2D convolution across all the input feature maps. In our KSE method, we first obtain the 2D kernels corresponding to an input feature map and calculate their sparsity and entropy as an indicator, which is further used to reduce the number of the 2D kernels by kernel clustering and generate a compact network.

structured pruning methods require specialized software or hardware designs to store a large number of indices for efficient speedup. Among them, filter pruning has received ever-increasing research attention. It can simultaneously reduce computation complexity and memory overhead by directly pruning redundant filters, and is well supported by various off-the-shelf deep learning platforms. For instance, Li *et al*. [21] proposed to remove unimportant filters based on the ℓ_1-norm. Molchanov *et al*. [22] calculated the effect of filters on the network loss based on a Taylor expansion. Those methods directly prune filters and their corresponding output feature maps in each convolutional layer, which may lead to dimensional mismatch in popular multi-branch networks, *e.g.,* ResNets [3] and ResNeXts [23]. For example, by removing the output feature maps in the residual mapping, the "add" operator cannot be implemented due to different output dimensions between the identity mapping and the residual mapping in ResNets. Instead, several channel pruning methods [24, 25] focus on the input feature maps in the convolutional layers, which do not modify the network architecture and operator when reducing the network size and FLOPs[1]. However, through directly removing the input feature maps, this approach typically has limited compression and speedup with significant accuracy drop.

In this paper, we investigate the problem of CNN compression from a novel *interpretable* perspective. We argue that interpreting the inherent network structure provides a novel and fundamental means to discover the implicit network redundancy. As investigated in network explanation [26, 27, 28], individual feature maps within and across different layers play different roles in the network. As an intuitive example, feature maps in different layers can be seen as hierarchical features, *e.g.,* features like simple structures in the bottom layers, and semantic features in the top layers. Even in the same layer, the importance of feature maps varies; the more information a feature map represents, the more important it is for the network. To this end, interpreting the network, especially the feature map importance, if possible, can well guide the quantization and/or pruning of the network elements.

We here have the first attempt to interpret the network structure towards fast and robust CNN compression. In particular, we first introduce the receptive field of a feature map to reveal the sparsity and information richness, which are the key elements to evaluate the feature map importance. Then, as shown in Fig. 1, the relationship between an input feature map and its corresponding 2D kernels is investigated, based on which we propose kernel sparsity and entropy (KSE) as a new indicator to efficiently quantitate the importance of the input feature maps in a *feature-agnostic* manner. Compared to previous data-driven compression methods [18, 19, 29], which need to compute all the feature maps corresponding to the entire training dataset to achieve a generalized result, and thus suffer from heavy computational cost for a large dataset, KSE can efficiently handle every layer in parallel in a data-free manner. Finally, we employ kernel clustering to quantize the kernels for CNN compression, and fine-tune the network with a small number of epochs.

The main contributions of our paper are three-fold:

- We investigate the problem of CNN compression from a novel *interpretable* perspective, and discover that the importance of a feature map depends on its sparsity and information richness.

[1]FLOPs: The number of floating-point operations

- Our method in Fig. 2 is *feature-agnostic* that only needs the 2D kernels to calculate the importance of the input feature maps, which differs from the existing data-driven methods based on directly evaluating the feature maps [18, 19, 29]. It can thus simultaneously handle all the layers efficiently in parallel.

- Kernel clustering is proposed to replace the common kernel pruning methods [30, 31], which leads to a higher compression ratio with only slight accuracy degradation.

We demonstrate the advantages of KSE using two widely-used models (ResNets and DenseNets) on three datasets (CIFAR-10, SVHN and ImageNet 2012). Compared to the state-of-the-art methods, KSE achieves superior performance. For ResNet-50, we obtain $4.7\times$ FLOPs reduction and $2.9\times$ compression with only 0.35% Top-5 accuracy drop on ImageNet 2012. The compressed DenseNets achieve much better performance than other compact networks (*e.g.,* MobileNet V2 and ShuffleNet V2) and auto-searched networks (*e.g.,* MNASNet and PNASNet).

The rest of this paper is organized as follows: In Section 2, we review the related work. Then we interpret the feature map importance in Section 3 and describe our KSE method in Section 4. In Section 5, we report the quantitative results of KSE. Finally, we conclude this paper in Section 6.

2. Related Work

In this section, we briefly review related work of network pruning for CNN compression that removes redundant parts, which can be divided into unstructured pruning and structured pruning.

Unstructured pruning is to remove unimportant weights independently. Han *et al*. [16, 17] proposed to prune the weights with small absolute values, and store the sparse structure in a compressed sparse row or column format. Yang *et al.* [32] proposed an energy-aware pruning approach to prune the unimportant weights layer-by-layer by minimizing the error reconstruction. Unfortunately, these methods need a special format to store the network and the speedup can only be achieved by using specific sparse matrix multiplication in special hardware.

By contrast, structured pruning directly removes structured parts (*e.g.,* kernels, filters or layers) to simultaneously compress and speedup CNNs and is well supported by various off-the-shelf deep learning libraries. Yoon *et al*. [20] proposed a group sparsity regularization that exploits correlations among features in the network. Hu *et al*. [18] computed the Average Percentage of Zeros (APoZ) of each filter, which is equal to the percentage of zero values in the output feature map corresponding to the filter. Recently, Luo *et al*. [19] pruned the filters that have a minimal impact on the next convolutional layer. He *et al*. [29] proposed a LASSO regression based channel selection, which uses least square reconstruction to prune filters. Differently, Lin *et al*. [33] proposed a global and dynamic training algorithm to prune unsalient filters. Although filter pruning approaches in [18, 19, 20, 29, 33] can reduce the memory footprint, they encounter a dimensional mismatch problem for the popular multi-branch networks, *e.g.,* ResNets [3]. Our method differs from all the above ones, which reduces the redundancy of 2D kernels corresponding to the input feature maps and do not modify the output of a convolutional layer to avoid the dimensional mismatch.

Several channel pruning methods [24, 25] are more suitable for the widely-used ResNets and DenseNets. These methods remove unimportant input feature maps of convolutional layers, and can avoid dimensional mismatch. For example, Liu *et al*. [24] imposed ℓ_1-regularization on the scaling factors on the batch normalization to select unimportant feature maps. Huang *et al*. [25] combined the weight pruning and group convolution to sparsify networks. These channel pruning methods obtain a sparse network based on a complex training procedure that requires significant cost of offline training. Unlike these methods, our approach reduces the 2D convolutional kernels that extract features on the input feature maps in a *feature-agnostic* manner, instead of directly removing them, which is able to achieve finer-grained compression more faster.

Besides the network pruning, our work is also related to some other methods [34, 35, 36]. Wu *et al*. [34] quantified filters in convolutional layers and weight matrices in fully-connected layers by minimizing the reconstruction error. However, they do not consider the different importance of input feature maps, and instead used the same number of quantification centroids. Tung *et al*. [35] combined weight pruning and quantization to compress CNN, which is an unstructured compression that needs specialized software and hardware designs. Son *et al*. [36] compressed the network by reconstructing it from a small set of spatial convolutional kernels. However, it can only be used for 3×3 kernels and cannot compress the 1×1 ones. In contrast, our KSE method can be applied to all layers.

Note that our approach can be further integrated with other strategies to obtain more compact networks, such as low-rank approximation [10, 11, 12] and compact architecture design [37, 38].

3. Interpretation of Feature Maps

Towards identifying feature map importance on the network, the work in [39] uses bilinear interpolation to scale the feature maps up to the resolution of the input image. Then, a threshold determined by a top quantile level is used to obtain the receptive field of the feature maps, which can be regarded as a binary mask. Following this principle, as

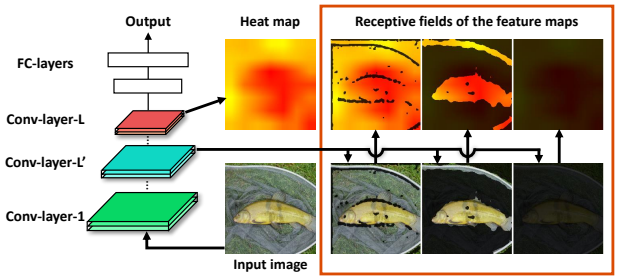

Figure 3. Visualization of the receptive fields (in the resolution of the input image) of three feature maps in the network.

shown in Fig. 3, for the input feature maps from the same convolutional layer, we compute their corresponding receptive fields on the original input image. These receptive fields indicate the different information contained in these feature maps. The visualization results in the lower part of the red box in Fig. 3 can be interpreted as an indicator of the feature map importance, where the left one is most important among the three and the right one can be ignored.

To quantify such an interpretation, we first compute the heat map of the original input image, which represents the information distribution [40]. We use the output feature maps of the last convolutional layer in the network, and add them together on the channel dimension. Then we scale the sumed feature map to the resolution of the input image by bilinear interpolation. Each pixel value $H_{i,j}$ in the heat map H represents the importance of this pixel in the input image. Finally, we compute the receptive field of a feature map on the heat map. As shown in Fig. 3, the red part in a receptive field can quantify the interpretation of the corresponding feature map. From this analysis, the information contained in a feature map can be viewed as the sum of the products between the elements of the mask and the heat map:

$$\sum_{i=1}^{H_{im}} \sum_{j=1}^{W_{im}} H_{i,j} M_{i,j}, \tag{1}$$

where H_{im} and W_{im} denote the resolution (height and width) of the input image, and M is the binary mask generated by the feature map. Eq. 1 can be rewritten as:

$$\mathbb{I}\{M = 1\}\overline{\mathrm{H}}, \tag{2}$$

where $\mathbb{I}\{M = 1\}$ is the number of the elements in M with value 1, which can be viewed as the area of the receptive field that depends on the sparsity of the feature map, and $\overline{\mathrm{H}}$ is the average of all entry $H_{i,j}$ whose corresponding element in M is 1. The heat map represents the information distribution in the input image. The higher the value of an element in H is, the more information this element contains. Therefore, $\overline{\mathrm{H}}$ can represent the information richness in the feature map. Eq. 2 indicates that the importance of a feature map depends on its sparsity and information richness. However, if we simply use Eq. 2 to compute the importance of each feature map, it suffers from heavy computation cost, since we need to compute all the feature maps with respect to the entire training dataset to obtain a comparative generalized result.

4. Proposed Method

To handle the above issue, we introduce the kernel sparsity and entropy (KSE), which serves as an indicator to represent the sparsity and information richness of input feature maps in a feature-agnostic manner. It is generic, and can be used to compress fully-connected layers by treating them as 1×1 convolutional layers.

Generally, a convolutional layer transforms an input tensor $\mathcal{X} \in \mathbb{R}^{C \times H_{in} \times W_{in}}$ into an output tensor $\mathcal{Y} \in \mathbb{R}^{N \times H_{out} \times W_{out}}$ by using the filters $\mathcal{W} \in \mathbb{R}^{N \times C \times K_h \times K_w}$. Here, C is the number of the input feature maps, N is the number of the filters, and K_h and K_w are the height and width of a filter, respectively. The convolution operation can be formulated as follows:

$$Y_n = \sum_{c=1}^{C} W_{n,c} * X_c, \tag{3}$$

where $*$ represents the convolution operation, and X_c and Y_c are the channels (feature maps) of $\mathcal{X}$ and $\mathcal{Y}$, respectively. For simplicity, the biases are omitted for easy presentation. For an input feature map X_c, we call the set $\{W_{n,c}\}_{n=1}^{N}$ the corresponding 2D kernels of X_c.

4.1. Kernel Sparsity

We measure the sparsity of an input feature map (*i.e.* $\mathbb{I}\{M = 1\}$) by calculating the sparsity of its corresponding 2D kernels, *i.e.,* the sum of their ℓ_1-norms $\sum_n |W_{n,c}|$. Although these kernels do not participate in generating the input feature map, we show below that their sparsity is related to it in the pre-trained network.

During training, the update of the 2D kernel $W_{n,c}$ depends on the gradient $\frac{\partial L}{\partial W_{n,c}}$ and the weight decay $R(W_{n,c})$:

$$\begin{aligned} W_{n,c}^{(t+1)} &= W_{n,c}^{(t)} - \eta \frac{\partial L}{\partial W_{n,c}^{(t)}} - \frac{\partial R(W_{n,c}^{(t)})}{\partial W_{n,c}^{(t)}} \\ &= W_{n,c}^{(t)} - \eta \frac{\partial L}{\partial Y_n^{(t)}} X_c^{(t)} - \frac{\partial R(W_{n,c}^{(t)})}{\partial W_{n,c}^{(t)}}, \end{aligned} \tag{4}$$

where L represents the loss function and η is the learning rate. If the input feature map $X_c^{(t)}$ is sparse, the kernel's

gradient is relatively small, and then the update formula becomes:

$$W_{n,c}^{(t+1)} \approx W_{n,c}^{(t)} - \frac{\partial R(W_{n,c}^{(t)})}{\partial W_{n,c}^{(t)}}. \tag{5}$$

Note that $|W_{n,c}^{(t+1)}| \to 0$ with the iterations if $R(W_{n,c}^{(t)})$ is defined based on the ℓ_2-norm, which may make the kernel being sparse [41]. Thus, the kernel corresponding to the sparse input feature map may be sparse during training. Therefore, for the c-th input feature map, we define its sparsity as:

$$s_c = \sum_{n=1}^{N} |W_{n,c}|. \tag{6}$$

To the best of our knowledge, we are the first to build the relation between the input feature maps (rather than the output feature maps) and the kernels in terms of sparsity. Note that this sparsity relation has been verified in our experiments shown in Section 5.2.

4.2. Kernel Entropy

For a convolutional layer, if the convolution results from a single input feature map are more diversified, this feature map contains more information and should be maintained in the compressed network. In this case, the distribution of the corresponding 2D kernels is more complicated. Therefore, we propose a new concept called kernel entropy to statistically measure the information richness of an input feature map $\overline{\mathrm{H}}$.

We first construct a nearest neighbor distance matrix A_c for the 2D kernels corresponding to the c-th input feature map. For each row i and col j, if $W_{i,c}$ and $W_{j,c}$ are "close"[2], *i.e.,* $W_{j,c}$ is among the k nearest neighbours of $W_{i,c}$, then $A_{c_{i,j}} = \|W_{i,c} - W_{j,c}\|$, and $A_{c_{i,j}} = 0$ otherwise. We set k to 5 empirically. Then we calculate the density metric of each kernel by the number of instances located in the neighborhood of this kernel:

$$dm(W_{i,c}) = \sum_{j=1}^{N} A_{c_{i,j}}. \tag{7}$$

The larger the density metric of $W_{i,c}$ is, the smaller the density of $W_{i,c}$ is, *i.e.,* the kernel is away from the others, and the convolution result using $W_{i,c}$ becomes more different. Hence, we define the kernel entropy to measure the complexity of the distribution of the 2D kernels:

[2]Let $W_{i,c}^{'}$ and $W_{j,c}^{'}$ be two vectors formed by the elements of $W_{i,c}$ and $W_{j,c}$, respectively. Then the distance between $W_{i,c}$ and $W_{j,c}$ is defined as the distance between $W_{i,c}^{'}$ and $W_{j,c}^{'}$.

$$e_c = -\sum_{i=1}^{N} \frac{dm(W_{i,c})}{d_c} log_2 \frac{dm(W_{i,c})}{d_c}, \tag{8}$$

where $d_c = \sum_{i=1}^{N} dm(W_{i,c})$. The smaller the kernel entropy is, the more complicated the distribution of the 2D kernels is, and the more diverse the kernels are. In this case, the features extracted by these kernels have greater difference. Therefore, the corresponding input feature map provides more information to the network.

4.3. Definition of the KSE Indicator

As discussed in Section 3, the feature map importance depends on two parts, the sparsity and the information richness. Upon this discovery, we combine the kernel sparsity and entropy to measure the overall importance of an input feature map by:

$$v_c = \frac{s_c}{1 + \alpha e_c}, \tag{9}$$

where α is a parameter to control the balance between the sparsity and entropy, which is set to 1 in this work. We call v_c the KSE indicator that measures the interpretability and importance of the input feature map. We further use the min-max normalization to rescale the indicators to $[0, 1]$ based on all the input feature maps in a convolutional layer.

4.4. Kernel Clustering

To compress kernels, previous channel pruning methods divide channels into two categories based on their importance, *i.e.,* important or unimportant ones. Thus, for an input feature map, its corresponding 2D kernels are either all kept or all deleted, which is a coarse compression. In our work, through clustering, we develop a fine-grained compression scheme to reduce the number of the kernels where after pruning, the number is an integer between 0 and N (but not just 0 or N as in the previous methods).

First, we decide the number of kernels required for their corresponding c-th input feature map as:

$$q_c = \begin{cases} 0, & \lfloor v_c G \rfloor = 0, \\ N, & \lceil v_c G \rceil = G, \\ \left\lceil \frac{N}{2^{G - \lfloor v_c G \rfloor + T}} \right\rceil, & \text{otherwise}, \end{cases} \tag{10}$$

where G controls the level of compression granularity. A larger G results in a finer granularity. N is the number of the original 2D kernels and T is a hyper-parameter to control the compression and acceleration ratios.

Second, to guarantee each output feature map contains the information from most input feature maps, we choose to cluster, rather than pruning, the 2D kernels to reduce the kernel number. It is achieved simply by the k-means algorithm with the number of cluster centroids equal to

q_c. Thus, the c-th input feature map generates q_c centroids (new 2D kernels) $\{B_{i,c} \in \mathbb{R}^{K_h \times K_w}\}_{i=1}^{q_c}$ and an index set $\{I_{n,c} \in \{1, 2, ..., q_c\}\}_{n=1}^{N}$ to replace the original 2D kernels $\{W_{n,c} \in \mathbb{R}^{K_h \times K_w}\}_{n=1}^{N}$. For example, $I_{1,c} = 2$ denotes that the first original kernel is classified to the second cluster $B_{2,c}$. When $q_c = 0$, the c-th input feature map is considered as unimportant, and all its corresponding kernels are pruned. In the other extreme case where $q_c = N$, this feature map is considered as most important for the convolutional layer, and all its corresponding kernels are kept.

There are three steps in our training procedure. (i) Pretrain a network on a dataset. (ii) Compress the network (in all the convolutional and fully-connected layers) using the method proposed above and obtain q_c, $B_{i,c}$ and $I_{n,c}$. (iii) Fine-tune the compressed network for a small number of epochs. During the fine-tuning, we only update the cluster centroids.

In inference, our method accelerates the network by sharing the 2D activation maps extracted from the input feature maps to reduce the computation complexity of convolution operations. Note that here the activation maps (yellow planes in Fig. 2) are not the output feature maps (orange planes in Fig. 2). As shown in Fig. 2, we split the convolutional operation into two parts, 2D convolution and channel fusion. In 2D convolution, the responses from each input feature map are computed simultaneously to generate the 2D activation maps. The c-th input feature map corresponds to the q_c 2D kernels, which generates q_c 2D activation maps $Z_{i,c} = B_{i,c} * X_c$. Then, in the channel addition, we compute Y_n by summing their corresponding 2D activation maps:

$$Y_n = \sum_{c=1}^{C} Z_{I_{n,c},c}. \tag{11}$$

5. Experiments

We have implemented our method using Pytorch [42]. The effectiveness validation is performed on three datasets, CIFAR-10, Street View House Numbers (SVHN), and ImageNet ILSVRC 2012. CIFAR-10 has 32×32 images from 10 classes. The training set contains 50,000 images and the test set contains 10,000 images. The SVHN dataset has 73,257 32×32 color digit images in the training set and 26,032 images in the test set. ImageNet ILSVRC 2012 consists of 1.28 million images for training and 50,000 images for validation over 1,000 classes.

All networks are trained using stochastic gradient descent (SGD) with momentum set to 0.9. On CIFAR-10 and SVHN, we respectively train the networks for 200 and 20 epochs using the mini-batch size of 64. The initial learning rate is 0.01 and is multiplied by 0.1 at 50% of the total number of epochs. On ImageNet, we train the networks for 21 epochs with the mini-batch size of 64, and the initial learning rate is 0.001 which is divided by 10 at epoch 7 and 14. Because the first layer has only three input channels and the last layer is a classifier in ResNet and DenseNet, we do not compress the first and the last layers of the networks.

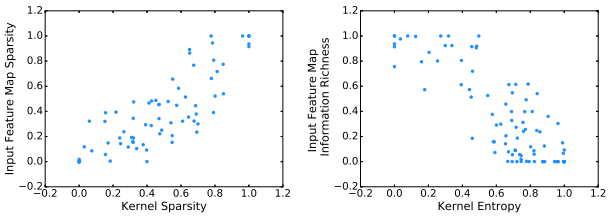

Figure 4. Left: Relationship between the sparsity of the input feature maps and their corresponding kernel sparsity. Right: Relationship between the information richness of the input feature maps and their corresponding kernel entropy.

5.1. Compression and Acceleration Ratios

In this section, we analyze the compression and acceleration ratios. For a convolutional layer, the size of the original parameters is $N \times C \times K_h \times K_w$ and each weight is assumed to require 32 bits. We store q_c cluster centroids, each weight of which again requires 32 bits. Besides, each index per input feature map takes $log_2 q_c$ bits. The size of each centroid is $K_h \times K_w$, and each input feature map needs N indices for the correspondence. Therefore, we can calculate the compression ratio r_{comp} for each layer by:

$$r_{comp} = \frac{NCK_hK_w}{\sum_c (q_c K_h K_w + N \frac{log_2 q_c}{32})}. \tag{12}$$

We can also accelerate the compressed network based on the sharing of the convolution results. As mentioned in Section 4.4, we compute all the intermediate features from the input feature maps, and then add the corresponding features for each output feature map. The computation is mainly consumed by convolution operations. Thus, the theoretical acceleration ratio r_{acce} of each convolutional layer is computed by:

$$r_{acce} \simeq \frac{NCH_{out}W_{out}K_hK_w}{\sum_c q_c H_{out}W_{out}K_hK_w} = \frac{NC}{\sum_c q_c}. \tag{13}$$

We can also calculate the compression and acceleration ratios on fully-connected layers by treating them as 1×1 convolutional layers.

5.2. Relationship between Input Feature Maps and their Corresponding Kernels

We calculate the sparsity $\mathbb{I}\{M = 1\}$ and information richness $\overline{\mathrm{H}}$ of the input feature maps, and their corresponding kernel sparsity and entropy. The relationships are shown

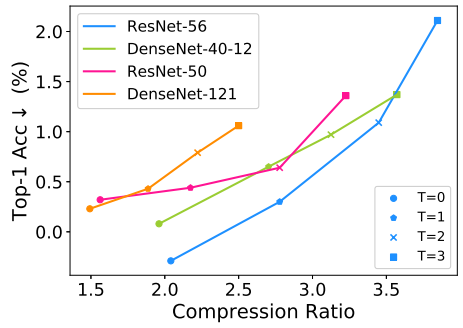

Figure 5. Sensitivity analysis in various networks. The horizontal axis shows the increased compression ratio compared to the original network as T increases. The vertical axis shows the increased Top-1 classification error rate compared to the original network as T increases.

in Fig. 4 where the feature maps are a random subset of all the feature maps from ResNet-56 on CIFAR-10. We can see that the sparsity of the input feature maps and the kernel sparsity increase simultaneously, while the information richness of the input feature maps decreases as the kernel entropy increases.

We can further use the Spearman correlation coefficient ρ to quantify these relationships:

$$\rho = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2 \sum_i (y_i - \overline{y})^2}}, \tag{14}$$

where $\overline{x}$ and $\overline{y}$ are the averages of the random variables x and y, respectively. The correlation coefficient for the relationship on the left of Fig. 4 is 0.833, while the correlation coefficient for the relationship on the right of Fig. 4 is -0.826. These values confirm the positive correlation for the first and the negative correlation for the second.

5.3. Ablation Study

The effective use of KSE, is related to G and T. We select ResNet-56 and DenseNet-40 on CIFAR-10, and ResNet-50 and DenseNet-121 on ImageNet2012 to evaluate them. Moreover, we analyze three different indicators.

5.3.1 Sensitivity Analysis of T

In our method, we use T to control the compression and acceleration ratios, which determines the number of 2D kernels after compression. Larger T results in fewer kernels and higher compression and acceleration ratios. Therefore, we explore the sensitivity of different networks on different datasets to guide the parameter setting of T. In the experiments in this sub-section, we set G to 4 for all the networks, because with $G = 4$, we can have the best overall trade-off

Model	FLOPs (r_{acce})	#Param. (r_{comp})	Top-1 Acc%
ResNet-56$_{baseline}$	125M(1.0×)	0.85M(1.0×)	93.03
ResNet-56-pruned-A [21]	112M(1.1×)	0.77M(1.1×)	93.10
ResNet-56-pruned-B [21]	90M(1.4×)	0.73M(1.2×)	93.06
ResNet-56-pruned [29]	62M(2.0×)	-	91.80
KSE (G=4)	60M(2.1×)	0.43M(2.0×)	**93.23**
KSE (G=5)	50M((2.5×)	0.36M(2.4×)	**92.88**

Table 1. Results of ResNet-56 on CIFAR-10. In all tables and figures, M/B means million/billion.

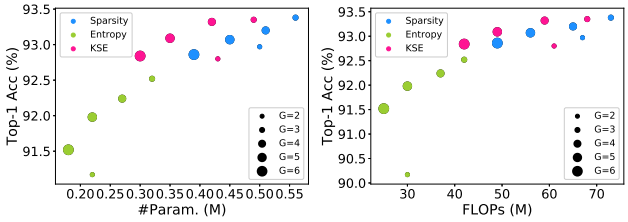

Figure 6. Compression granularity and indicator analysis with ResNet-56 on CIFAR-10.

between the Top-1 accuracy and the compression and acceleration ratios. As shown in Fig. 5, small networks are more sensitive to kernel reduction. Thus, we set T to 0 for ResNet-56, DenseNet-40-12 and DenseNet-100-12 on both CIFAR-10 and SVHN, and set T to 1 for ResNet-50 and all the DenseNets on ImageNet 2012.

5.3.2 Effect of the Compression Granularity G

In our kernel clustering, we use G to control the level of compression granularity. The results of different G are shown in Fig. 6. Note that in this sub-section, only the red solid circles in Fig. 6 are concerned. When $G = 2$, the 2D kernels corresponding to the c-th feature map are divided into two categories: $q_c = 0$ or $q_c = N$ for pruning or keeping all the kernels. It is a coarse-grained pruning method. As G increases, q_c achieves various different values, which means to compress the 2D kernels in a fine-grained manner. In addition, the compression and acceleration ratios are also increased. Compared to the coarse-grained pruning, the fine-grained pruning achieves much better results. For example, when $G = 4$, it achieves the same compression ratio as $G = 2$ with 0.52% Top-1 accuracy increase.

5.3.3 Indicator Analysis

In this paper, we have proposed three concepts, kernel sparsity, kernel entropy, and KSE indicator. In fact, all of them can be used as an indicator to judge the importance of the feature maps. We next evaluate these three indicators on ResNet-56, with the proposed kernel clustering for compression. As shown in Fig. 6, compared to the indicators

Model	FLOPs (r_{acce})	#Param. (r_{comp})	Top-1 Acc%
DenseNet-40$_{baseline}$	283M(1.0×)	1.04M(1.0×)	94.81
DenseNet-40 (40%) [24]	190M(1.5×)	0.66M(1.6×)	94.81
DenseNet-40 (70%) [24]	120M(2.4×)	0.35M(3.0×)	94.35
KSE (G=3)	170M(1.7×)	0.63M(1.7×)	**94.81**
KSE (G=6)	115M(2.5×)	0.39M(2.7×)	**94.70**
DenseNet-BC-100$_{baseline}$	288M(1.0×)	0.75M(1.0×)	95.45
DenseNet-PC128N [36]	212M(1.4×)	0.50M(1.5×)	95.43
CondenseNet^{light}-94 [25]	122M(2.4×)	0.33M(2.3×)	95.00
KSE (G=3)	159M(1.8×)	0.45M(1.7×)	**95.49**
KSE (G=6)	103M(2.8×)	0.31M(2.4×)	**95.08**

Table 2. Results of DenseNet on CIFAR-10.

Model	FLOPs (r_{acce})	#Param. (r_{comp})	Top-1 Acc%
DenseNet-40$_{baseline}$	283M(1.0×)	1.04M(1.0×)	98.17
DenseNet-40 (40%) [24]	185M(1.5×)	0.65M(1.6×)	98.21
DenseNet-40 (60%) [24]	134M(2.1×)	0.44M(2.4×)	98.19
KSE (G=4)	147M(1.9×)	0.49M(2.1×)	**98.27**
KSE (G=5)	130M(2.2×)	0.42M(2.5×)	**98.25**

Table 3. Results of DenseNet-40-12 on SVHN.

of kernel sparsity and kernel entropy, the KSE indicator achieves the best results for different compression granularities. This is due to the fact that the kernel sparsity only represents the area of the receptive field of a feature map, and the density entropy of the 2D kernels only expresses the position information of a receptive field, which alone are not as effective as KSE to evaluate the importance of a feature map.

5.4. Comparison with State-of-the-Art Methods

CIFAR-10. We compare our method with [21, 29] on ResNet-56, and with [24, 25, 36] on DenseNet. For ResNet-56, we set G to two values (4 and 5) to compress the network. For DenseNet-40-12 and DenseNet-BC-100-12, we set G to another two values (3 and 6). As shown in Table 1 and Table 2, our method achieves the best results, compared to the filter pruning methods [21, 29] on ResNet-56 and the channel pruning methods [24, 25] on DenseNet. Moreover, our method also achieves better results than other kernel clustering methods [36] on DenseNet-BC-100-12. Compared to Son *et al.* [36], our KSE method can not only compress the 3×3 convolutional layers, but also the 1×1 convolutional layers to obtain a more compact network.

SVHN. We also evaluate the performance of KSE on DenseNet-40-12 on SVHN. We set G to two values, 4 and 5. As shown in Table 3, our method achieves better performance than the channel pruning method [24]. For example, compared to He *et al.* [24], we obtain 0.06% increase in Top-1 accuracy (98.25% *vs.* 98.19%) with higher compression and acceleration ratios (2.5× and 2.2× *vs.* 2.4× and 2.1×).

ImageNet2012. We set G to 4 and 5, and compare our

Model	FLOPs (r_{acce})	#Param. (r_{comp})	Top-1 Acc%	Top-5 Acc%
ResNet-50$_{baseline}$	4.10B(1.0×)	25.56M(1.0×)	76.15	92.87
GDP-0.6 [33]	1.88B(2.2×)	-	71.19	90.71
ResNet-50(2×) [29]	2.73B(1.5×)	-	72.30	90.80
ThiNet-50 [19]	1.71B(2.4×)	12.38M(2.0×)	71.01	90.02
ThiNet-30 [19]	1.10B(3.7×)	8.66M(3.0×)	68.42	88.30
KSE (G=5)	1.09B(3.8×)	10.00M(2.6×)	**75.51**	**92.69**
KSE (G=6)	0.88B(4.7×)	8.73M(2.9×)	**75.31**	**92.52**

Table 4. Results of ResNet-50 on ImageNet2012.

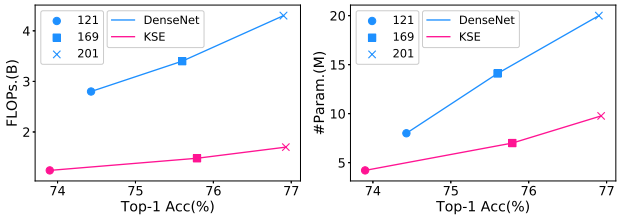

Figure 7. Parameter and FLOPs amount comparison between the original DenseNets and the compressed DenseNets by our KSE.

KSE method with three state-of-the-art methods [19, 29, 33]. As show in Table 4, Our method achieves the best performance with only a decrease of 0.35% in Top-5 accuracy by an factor of 2.9× compression and 4.7× speedup. These state-of-the-art methods perform worse mainly because they use dichotomy to compress networks, *i.e.,* prune or keep the filters/channels, which leads to the loss of some important information in the network. Besides, filter pruning methods like [19, 33] cannot be applied to some convolutional layers due to the dimensional mismatch problem on ResNets with a multi-branch architecture.

For DenseNets, we set G to 4, and compress them with three different numbers of layers, DenseNet-121, DenseNet-169, and DenseNet-201. As shown in Fig. 7, the compressed networks by KSE is on par with the original networks, but achieves almost 2× parameter compression and speedup.

Recently, many compact networks [37, 38] have been proposed to be used on mobile and embedded devices. In addition, auto-search algorithms [43, 44] have been proposed to search the best network architecture by reinforcement learning. We compare the compressed DenseNet-121 and DenseNet-169 by KSE with these methods [25, 37, 38, 43, 44]. In Fig. 8. 'A' represents $G = 4$ and 'B' represents $G = 3$. We use KSE to compress DenseNets, which achieves more compact results. For example, we obtain 73.03% Top-1 accuracy with only 3.37M parameters on ImageNet 2012. Our KSE uses different numbers of 2D kernels for different input feature maps to do convolution, which reduces more redundant kernels, compared to the complicated auto-search algorithms which only use

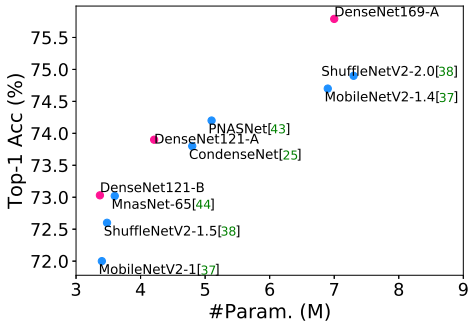

Figure 8. Comparison of the compressed DenseNets (red circles) by our KSE and other compact networks (blue circles).

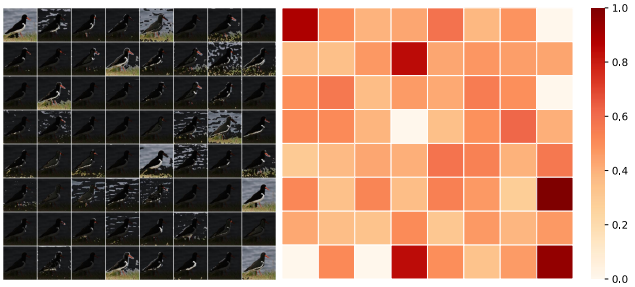

Figure 9. Visualization of the input feature maps and their KSE indicator values at the *Block1-Unit1-Conv2* layer of ResNet-50.

the traditional convolutional layer. Besides, the widely-used depth-wise convolution on MobileNet or ShuffleNet may cause significant information loss, due to only one 2D kernel is used to extract features from each feature map.

5.5. Visualization Analysis

We visualize the input feature maps and the corresponding KSE indicator values at the *Block1-Unit1-Conv2* layer of ResNet-50 to reveal their connection. As shown in Fig. 9, the input image contains a bird. When the value of the indicator is smaller, its corresponding feature map provides less information of the bird. On the contrary, when the value is close to 1, the feature map has both the bird and the background information. Therefore, our KSE method can accurately discriminate the feature maps, and effectively judge their importance.

6. Conclusion

In this paper, we first investigate the problem of CNN compression from a novel *interpretable* perspective and discover that the sparsity and information richness are the key elements to evaluate the importance of the feature maps. Then we propose kernel sparsity and entropy (KSE) and combine them as an indicator to measure this importance in a *feature-agnostic* manner. Finally, we employ kernel clustering to reduce the number of kernels based on the KSE indicator and fine-tune the compressed network in a few epochs. The networks compressed using our approach achieve better results than state-of-the-art methods.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems (NIPS)*, 2012. 1

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 2, 3

[4] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. *Computer Vision and Pattern Recognition (CVPR)*, 2017. 1

[5] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *Computer Vision and Pattern Recognition (CVPR)*, 2017. 1

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *Neural Information Processing Systems (NIPS)*, 2015. 1

[7] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *Computer Vision and Pattern Recognition (CVPR)*, 2015. 1

[8] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *European Conference on Computer Vision (ECCV)*, 2018. 1

[9] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *International Conference on Learning Representations (ICLR)*, 2014. 1

[10] Shaohui Lin, Rongrong Ji, Chao Chen, Dacheng Tao, and Jiebo Luo. Holistic cnn compression via low-rank decomposition with knowledge transfer. *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*, 2018. 1, 3

[11] Shaohui Lin, Rongrong Ji, Xiaowei Guo, Xuelong Li, et al. Towards convolutional neural networks compression via global error reconstruction. *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018. 1, 3

[12] Wei Wen, Cong Xu, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Coordinating filters for faster deep neural networks. *International Conference on Computer Vision (ICCV)*, 2017. 1, 3

[13] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry

Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *Computer Vision and Pattern Recognition (CVPR)*, 2017. 1

[14] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *Computer Vision and Pattern Recognition (CVPR)*, 2016. 1

[15] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *European Conference on Computer Vision (ECCV)*, 2016. 1

[16] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016. 1, 3

[17] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Neural Information Processing Systems (NIPS)*, 2015. 1, 3

[18] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016. 1, 2, 3

[19] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. *International Conference on Computer Vision (ICCV)*, 2017. 1, 2, 3, 8

[20] Jaehong Yoon and Sung Ju Hwang. Combined group and exclusive sparsity for deep neural networks. *International Conference on Machine Learning (ICML)*, 2017. 1, 3

[21] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *International Conference on Learning Representations (ICLR)*, 2016. 2, 7, 8

[22] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *CoRR, abs/1611.06440*, 2016. 2

[23] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[24] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. *International Conference on Computer Vision (ICCV)*, 2017. 2, 3, 8

[25] Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. *Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 3, 8

[26] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John E Hopcroft. Convergent learning: Do different neural networks learn the same representations? *Neural Information Processing Systems (NIPS)*, 2015. 2

[27] Ari S Morcos, David GT Barrett, Neil C Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. *International Conference on Learning Representations (ICLR)*, 2018. 2

[28] Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. Revisiting the importance of individual units in cnns via ablation. *arXiv preprint arXiv:1806.02891*, 2018. 2

[29] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. *International Conference on Computer Vision (ICCV)*, 2017. 2, 3, 7, 8

[30] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Neural Information Processing Systems (NIPS)*, 2016. 3

[31] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the regularity of sparse structure in convolutional neural networks. *Neural Information Processing Systems (NIPS)*, 2017. 3

[32] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[33] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. Accelerating convolutional networks via global & dynamic filter pruning. *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018. 3, 8

[34] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. *Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[35] Frederick Tung and Greg Mori. Clip-q: Deep network compression learning by in-parallel pruning-quantization. *Computer Vision and Pattern Recognition (CVPR)*, 2018. 3

[36] Sanghyun Son, Seungjun Nah, and Kyoung Mu Lee. Clustering convolutional kernels to compress deep neural networks. *European Conference on Computer Vision (ECCV)*, 2018. 3, 8

[37] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *Computer Vision and Pattern Recognition (CVPR)*, 2018. 3, 8

[38] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *European Conference on Computer Vision (ECCV)*, 2018. 3, 8

[39] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. *Computer Vision and Pattern Recognition (CVPR)*, 2017. 3

[40] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. *Computer Vision and Pattern Recognition (CVPR)*, 2016. 4

[41] Baochang Zhang, Alessandro Perina, Vittorio Murino, and Alessio Del Bue. Sparse representation classification with manifold constraints transfer. *Computer Vision and Pattern Recognition (CVPR)*, 2015. 5

[42] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. *Neural Information Processing Systems (NIPS) Workshop*, 2017. 6

[43] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *European Conference on Computer Vision (ECCV)*, 2018. 8

[44] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018. 8