Filter Distillation for Network Compression

Xavier Suau*
Apple
xsuaucuadros@apple.com

Luca Zappella*
Apple
lzappella@apple.com

Nicholas Apostoloff
Apple
napostoloff@apple.com

Abstract

In this paper we introduce Principal Filter Analysis (PFA), an easy to use and effective method for neural network compression. PFA exploits the correlation between filter responses within network layers to recommend a smaller network that maintain as much as possible the accuracy of the full model. We propose two algorithms: the first allows users to target compression to specific network property, such as number of trainable variable (footprint), and produces a compressed model that satisfies the requested property while preserving the maximum amount of spectral energy in the responses of each layer, while the second is a **parameter-free** *heuristic that selects the compression used at each layer by trying to mimic an ideal set of uncorrelated responses. Since PFA compresses networks based on the correlation of their responses we show in our experiments that it gains the additional flexibility of adapting each architecture to a specific domain while compressing. PFA is evaluated against several architectures and datasets, and shows considerable compression rates without compromising accuracy, e.g., for VGG-16 on CIFAR-10, CIFAR-100 and ImageNet, PFA achieves a compression rate of 8x, 3x, and 1.4x with an accuracy* **gain** *of 0.4%, 1.4% points, and 2.4% respectively. Our tests show that PFA is competitive with state-of-the-art approaches while removing adoption barriers thanks to its practical implementation, intuitive philosophy and ease of use.*

1. Introduction

Despite decades of research, the design of deep neural networks (DNNs) is often an empirical process. Practitioners frequently make design choices, such as the number of layers, types of layer, number of filters per layer, etc., based on intuition or brute-force search. Nevertheless, the strong performance of DNNs, together with GPU advances, have led to a growing popularity of these techniques in both academia and industry. Recent studies have unveiled some intrinsic properties of DNNs. For example, there is a consensus that depth can accelerate learning, and that wider layers help with optimization [3, 31, 41, 10]. However, in practical applications, the size of these networks is often a limiting factor when deploying on devices with constrained storage, memory, and computation resources.

Another observed DNN property is that the responses of a layer exhibit considerable correlation [13], inspiring the idea of learning decorrelated filters [9, 43]. [9, 43] propose a modified loss function to encourage decorrelation during training and show that accuracy improves with decorrelated filters. However, such algorithms focus on training and do not address network compression. *Our hypothesis is that layers that exhibit high correlation in* ***filter responses*** *could learn equally well using a smaller number of filters.*

Principal Filter Analysis (PFA) draws from the recent findings by letting the user start with an over-parametrized network and then leverages the intra-layer correlation to reduce the network size after training. PFA analyzes a trained network and is agnostic to the training methodology and the loss function. Inference is performed on a dataset, and the correlation within the responses of each layer is used to provide a compression recipe. A new, smaller architecture based on this recipe can then be fine-tuned.

We propose two closed-form algorithms based on spectral energy analysis for suggesting the number of filters to remove in a layer:

PFA-En uses Principal Component Analysis (PCA) [23] to allow a user to specify the proportion of the energy in the original response that should be preserved in each layer; since this operation is extremely fast, the user can alternatively provide a network property, such as footprint (or FLOPs), and different energy thresholds can be iteratively tested until the requested property is satisfied.

PFA-KL is a parameter-free approach that balances the trade-off between compression and accuracy change. PFA-KL uses Kullback-Leibler (KL) divergence [26] to quantify the divergence between the current set of responses and an ideal set of uncorrelated responses in

*Equal contributor

order to identify the number of redundant filters.

Based on the PFA recommendation, filters that produce maximally correlated responses are removed and the network is fine-tuned. As shown in Sec. 4.1, using popular convolutional networks and datasets such as VGG-16 on CIFAR-10, CIFAR-100 and ImageNet, PFA achieves a compression rate of 8x, 3x, and 1.4x with an accuracy **gain** of 0.4%, 1.4% points, and 2.4% respectively. Our tests show that PFA is competitive with state-of-the-art approaches while providing the unique advantage of being *practical to implement, intuitive to understand, and easy to use*. Since PFA exploits the correlation of the responses, its recommendations become specific for a given domain. This specialization makes PFA suitable also for the task of simultaneous compression and domain adaptation, as shown in Sec. 4.3.

2. Related work

The field of network compression encompasses a wide range of techniques that can be grouped into the following families: quantization, knowledge distillation, tensor factorization and network pruning.

Quantization algorithms compress networks by reducing the number of bits used to represent each weight [51, 16, 42, 20, 52].

Knowledge distillation [22] aim to create a simpler model that mimics the output of a more complex model. Variations on this concept include [5, 4, 44, 7].

Tensor factorization algorithms exploit the redundancy present in convolution layers by replacing the original tensors with a sequence of smaller or sparser counterparts that produce similar responses [14, 27, 24, 8, 35, 50, 55, 2, 39].

Network pruning is a family of techniques that compress networks by iteratively removing connections based on the salience of their weights. Early work, like Optimal Brain Damage [28] and Optimal Brain Surgeon [18], targeted fully connected networks. Recent work can be divided into two sub-families: *sparse pruning* [17, 47, 49, 48, 1, 6, 57, 11], where individual neurons are removed, and *structured pruning* [29, 21, 33, 36, 54], where entire filters are removed.

PFA falls within the family of structured network pruning. Some of these techniques (e.g., [29]) require user defined parameters that are hard to choose and whose effect on the footprint is difficult to predict (see Sec. 4.2 for a more detailed discussion). Others also require modification of the loss function (e.g., [30]). In contrast, PFA-En has only one and intuitive parameter, which is the proportion of the response energy to be preserved at each layer, and PFA-KL is parameter-free. Furthermore, instead of learning the saliency of the filters during training by modifying the loss function, PFA estimates it after training without requiring knowledge of the training details. This makes PFA applicable to any trained network, without the need to know its loss function or training regime.

Within the structured pruning family, there are approaches based on singular value decomposition (SVD) [53, 37, 40], where a new set of filters are obtained by projecting the original weights onto a lower dimensional space. Techniques that make compression decisions based on *weights*, rather than the *responses*, cannot take into account the specificity of the task. PFA differs from these methods because SVD is performed on the responses of the layers rather than on the filter weights, and no projection is done. This is particularly important for domain adaption applications, where a trained network is specialized for a different task. As shown in Sec. 4.3, PFA derives different architectures from the same initial model when the responses are obtained from different tasks (i.e., datasets).

Some methods also reason on the layer responses [29, 35, 39]. These techniques aim to find a smaller set of filters that minimize the reconstruction error of the feature maps or the response output. PFA has a different philosophy: it uses the concept of correlation within the responses to identify redundancy within a layer. In practice, this means that PFA can compress all layers simultaneously, while the majority of the techniques that use responses need to operate on one layer at the time.

Finally, PFA is orthogonal to the quantization, tensor factorization and distillation methods, and could be used as a complementary strategy to further compress neural networks.

3. Principal Filter Analysis

In this section, the PFA-En and PFA-KL algorithms are described in detail. Both algorithms share the idea of exploiting correlations between responses in convolutional layers and neurons in fully connected layers to obtain a principled recommendation for network compression.

3.1. Definitions

PFA is inherently data driven and thus takes advantage of a dataset $\{\mathbf{X}_i\} \in \mathbb{R}^{M \times I}$ where $\mathbf{X}_i$ is the i^{th} input data sample, M is the number of samples in the dataset, and I is the input dimensionality. Typically, this dataset is the data used to train the network, but it can also be a representative set that covers the distribution of inputs likely to be encountered. Without loss of generality, we assume that the input data are images:$\{\mathbf{X}_i\} \in \mathbb{R}^{M \times H \times W \times C}$, where H is the image height, W is the image width and C is the number channels.

Let $\mathbf{T}_i^{[\ell]} \in \mathbb{R}^{1 \times H^{[\ell]} \times W^{[\ell]} \times C^{[\ell]}}$ be the output tensor produced by a given layer ℓ of a network on the i^{th} input sample. Any operation in the network is considered a layer (e.g., batch normalization, ReLU, etc.). In this work, we analyze the output of convolutional and fully connected layers.

For a convolutional layer ℓ, let $\mathbf{W}^{[\ell]} \in$

$\mathbb{R}^{f_h^{[\ell]} \times f_w^{[\ell]} \times C^{[\ell-1]} \times C^{[\ell]}}$ be the set of $C^{[\ell]}$ trainable filters with a kernel of size $f_h^{[\ell]} \times f_w^{[\ell]} \times C^{[\ell-1]}$. Therefore, we can formally express $\mathbf{T}_i^{[\ell]}$ produced by the convolutional layer as $\mathbf{T}_i^{[\ell]} = \mathbf{W}^{[\ell]} * \mathbf{T}_i^{[\ell-1]}$ with $\mathbf{T}_i^{[0]} = \mathbf{X}_i$, where $*$ denotes the convolution operator. We omit the bias term to improve readability.

We define the response vector $\mathbf{a}_i^{[\ell]} \in \mathbb{R}^{C^{[\ell]}}$ of a given layer ℓ with respect to an input $\mathbf{X}_i$ to be the spatially max-pooled and flattened tensor $\mathbf{T}_i^{[\ell]}$ (i.e., max-pooling over the dimensions $H^{[\ell]}$ and $W^{[\ell]}$). For fully-connected layers, $\mathbf{W}^{[\ell]} \in \mathbb{R}^{C^{[\ell-1]} \times C^{[\ell]}}$, with $C^{[\ell]}$ being the number of neurons in layer ℓ. The output tensor is $\mathbf{T}_i^{[\ell]} = \mathbf{W}^{[\ell]} \mathbf{T}_i^{[\ell-1]}$, and since no pooling is required, we define the response vector as $\mathbf{a}_i^{[\ell]} = \mathbf{T}_i^{[\ell]} \in \mathbb{R}^{C^{[\ell]}}$.

Let $\mathbf{A}^{[\ell]} = [\mathbf{a}_1^{[\ell]}, \dots, \mathbf{a}_M^{[\ell]}]^\top \in \mathbb{R}^{M \times C^{[\ell]}}$ be the matrix of responses of a generic layer ℓ given a dataset with M samples. Given $\mathbf{A}^{[\ell]}$ we can compute its covariance matrix $\in \mathbb{R}^{C^{[\ell]} \times C^{[\ell]}}$, and extract its eigenvalues $\boldsymbol{\lambda}^{[\ell]} \in \mathbb{R}^{C^{[\ell]}}$, *sorted in descending order and normalized to sum to 1*.

3.2. Compression recipes

The set $\boldsymbol{\lambda}^{[\ell]}$ provides insight into the correlation of the responses produced by layer ℓ. *The closer $\boldsymbol{\lambda}^{[\ell]}$ is to a uniform distribution, the more decorrelated the response of the filters and the more uniform their contribution to the overall response energy*. Conversely, the closer $\boldsymbol{\lambda}^{[\ell]}$ is to a Dirac δ-distribution, the more correlated the filters. Our hypothesis is that layers that exhibit high correlation could learn equally well using a smaller number of filters.

We present two strategies that use $\boldsymbol{\lambda}^{[\ell]}$ and produce a recipe with the goal of maximizing compression by reducing correlation. Let a recipe $\boldsymbol{\Gamma} = \{\gamma^{[\ell]}\}$, with $\gamma^{[\ell]} \in (0, 1]$, be the set of compression factors applied to each of the L layers included in the analysis. For example, $\gamma^{[3]} = 0.6$ means that we keep 60% of the filters in layer 3.

Up to now the recipes only indicate *how many* filters each layer should have. Once the correct number of filters has been determined, we continue to choose *which* filters should be kept. We call this *filter selection* and we outline it in Sec. 3.2.1.

PFA-En: energy-based recipe. PCA can be used for dimensionality reduction by performing a linear mapping to a lower dimensional space that maximizes the variance of the data in this space. This can be accomplished by extracting the eigenvectors and eigenvalues of the covariance matrix. The original data is then reconstructed using the minimum number of eigenvectors that correspond to the eigenvalues that sum up to the desired energy factor τ. Inspired by this strategy, we propose to keep the minimum set of filters such that a fraction of response energy greater or equal to a user defined energy, τ, is preserved. We define the energy at a given compression ratio for a layer as $\mathcal{E}(\gamma^{[\ell]}) = \sum_{k=1}^{\lceil \gamma^{[\ell]} \cdot C^{[\ell]} \rceil} \lambda_k^{[\ell]}$, and we propose to re-architect the network according to the following recipe:

$$\boldsymbol{\Gamma}^\star_{\mathcal{E}}(\tau) = \{\min \gamma^{[\ell]}\} \quad \text{s.t.} \quad \mathcal{E}(\gamma^{[\ell]}) \geq \tau, \quad \forall \ell. \tag{1}$$

The parameter τ provides the user with the ability to guide the compression ratio.

PFA-En has the advantage of being tightly connected to well-established dimensionality reduction techniques based on PCA, is simple to implement, and uses a single, highly intuitive parameter. Furthermore, since evaluating the size of a model (or its FLOPs) obtained at different energy thresholds is easy and fast, it is straightforward to replace the parameter τ with the desired footprint $\mathcal{F}$ (or FLOPs) after compression by solving iteratively the optimization: $\boldsymbol{\Gamma}^\star_{foot}(\mathcal{F}) = \text{argmax}_\tau \, foot(\boldsymbol{\Gamma}^\star_{\mathcal{E}}(\tau)) \leq \mathcal{F}$, where $foot(\cdot)$ is a function that returns the footprint of a model given a recipe. Being able to specify a target footprint instead of an energy threshold gives PFA-En an even greater appeal for practical use cases.

PFA-KL: KL divergence-based recipe. We propose an alternative formulation to obtain a recipe $\boldsymbol{\Gamma}^\star_{\text{KL}}$, based on the KL divergence. This formulation is a heuristic that frees PFA from the use of any parameter. As previously mentioned, a set $\boldsymbol{\lambda}^{[\ell]}$ similar to a uniform distribution implies an uncorrelated response of the filters in layer ℓ. Therefore, the further $\boldsymbol{\lambda}^{[\ell]}$ is from a flat distribution the more layer ℓ can be compressed.

Let us define $\boldsymbol{u}^{[\ell]} \in \mathbb{R}^{C^{[\ell]}} \sim \cup[1, C^{[\ell]}]$ as the *desired* uniform (i.e., flat) distribution (no correlation between filters), and $\boldsymbol{d} = \text{Dirac}()$ as the *worst case* distribution (all filters are perfectly correlated). We can measure the dissimilarity of the actual set, $\boldsymbol{\lambda}^{[\ell]}$, from the *desired* distribution, $\boldsymbol{u}^{[\ell]}$, as the empirical KL divergence $\text{KL}(\boldsymbol{\lambda}^{[\ell]}, \boldsymbol{u}^{[\ell]})$. The upper bound of which is given by $u_{\text{KL}} = \text{KL}(\boldsymbol{d}, \boldsymbol{u}^{[\ell]})$, while the lower bound is 0. Note that the KL divergence is not symmetric, however, since $\boldsymbol{d}$ has only one point of support, u_{KL} can only be computed in one direction. Also note that one could replace the KL divergence with any dissimalarity measure between distributions, such as χ^2 or the Wasserstein metric [45].

Intuitively, when the actual set of eigenvalues is identical to the ideal distribution (i.e., no correlation found) then we would like to preserve all filters. Conversely, when the actual set of eigenvalues is identical to the worst case distribution (i.e., all filters are maximally correlated) then one single filter would be sufficient. The proposed KL divergence-based recipe is a mapping $\psi : [0, u_{\text{KL}}] \mapsto (0, 1]$; a divergence close to the upper bound results in a strong compression and a divergence close to the lower bound results in a milder

compression:

$$\mathbf{\Gamma}^{\star}_{\mathrm{KL}} = \left\{\psi\big(\mathrm{KL}(\boldsymbol{\lambda}^{[\ell]}, \boldsymbol{u}^{[\ell]}), u_{\mathrm{KL}}\big)\right\}, \quad \forall \ell. \tag{2}$$

In this work, we use a simple linear mapping $\psi(x, u_{KL}) = 1 - {}^{x}/_{u_{KL}}$. Other mappings were explored, leading to different degrees of compression; however, we have observed that a linear mapping produces good results that generalize well across networks.

3.2.1 Filter selection

The recipes produced by PFA-En and PFA-KL provide the number of filters, $F^{[\ell]} = \lceil \gamma^{[\ell]} \cdot C^{[\ell]} \rceil$, that should be kept in each layer, but do not indicate which filters should be kept. Once a new compressed architecture is created the question becomes how to initialize it. One option is to initialize it at random. In this case, it does not matter which filters are chosen. An alternative is to select which filters to keep and use their values for initialization, with the intuition (verified in our experiments) that the use of previously trained filters will improve convergence and, for the same given training budget, lead to better accuracy than random initialization. We do this by removing those filters in each layer that are maximally correlated. For each filter in a given layer we compute the ℓ_1-norm of the Pearson's correlation coefficients [38] with all the other filters , and remove the filter with the largest norm. If more filters need to be removed, we update the coefficients by removing those that correspond to the previously selected filter, and iterate until the desired number of filters has been removed. In the rare, but theoretically possible, case in which two filters have the same ℓ_1-norm we choose the one with the highest individual correlation coefficient.

4. Experiments

4.1. Quantitative comparison

To evaluate PFA, we apply it to several architectures and datasets, and we compare results to the state of the art. We compare PFA to another method, the filter pruning approach (FP)[29], that like PFA belongs to the family of structured pruning algorithms. We also extend the comparison to other families even if algorithms in those families tend to be more complex and computationally demanding. We compare against sparse pruning algorithms, such as the network slimming approach (NS) [30], and the variational information bottleneck approach (VIB) [11]. We also provide a comparison against a tensor facorization method: the filter group approximation approach (FGA) [39].

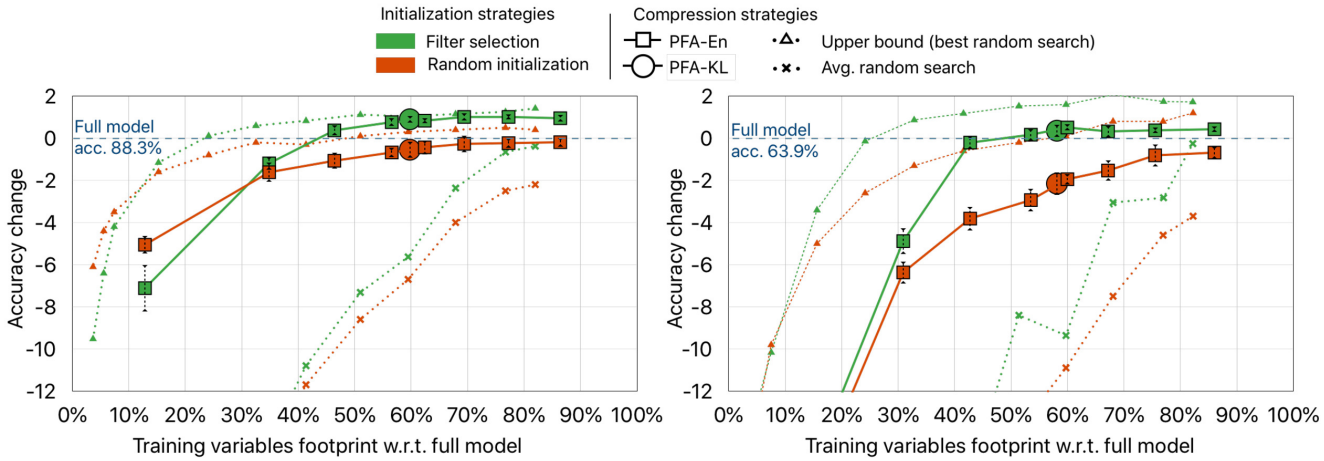
For the comparison, we focus on the compression ratio and the accuracy change, measured in percentage points (pp), obtained by the compressed architectures. This enables comparing various techniques in the same plot, even if the accuracy of each original architecture is slightly different because of different training strategies used. In App. B and C, we provide the exact accuracy of the full and compressed models, footprint, FLOPs, and all the hyper-parameters used to train the models.

Ablation studies and random compression. In this set of experiments we assess the impact of the PFA compression strategies separately from the impact of the initialization of the pruned network, i.e., the filter selection strategy. In addition, we try to understand how PFA compares to randomly compressed networks and how close its solution is to the optimal architecture (empirically defined).

In order to be able to repeat this experiment many times we adopt a small convolutional network that we refer to as SimpleCNN (see App. C.1 for the exact specification of the network and the training hyper-parameters). Results are shown on CIFAR-10 and CIFAR-100 [25]. The full model is obtained by training using 10 random initializations – we choose the initialization that leads to the highest test accuracy and perform inference on the training set to obtain the responses ($\mathbf{A}^{[\ell]}$) at each layer needed for the PFA analysis. We analyze all layers in parallel (one-shot, as opposed to an iterative approach which would also be applicable to PFA and likely to lead to even better results, but it would not be a fair comparison with all other state-of-art algorithms) to obtain PFA recipes. PFA-En is computed for the following energy values: $\tau \in \{0.8, 0.85, 0.93, 0.95, 0.96, 0.97, 0.98, 0.99\}$, whereas PFA-KL is parameter-free and is computed once per baseline network.

To evaluate different initialization strategies (i.e., random vs filter selection), after creating the compressed architecture according to a PFA recipe we perform two types of fine-tuning. First, we retrain from scratch with 10 different random weight initializations. Second, we retrain 10 times using *filter selection*, and fine-tune the compressed network starting from the weights of the selected filters. We report the mean and standard deviation of the accuracy of each of these 10 models. It is important to note that the retraining is done without hyper-parameter tuning (we use the same parameters used to train the full model). While this is a sub-optimal strategy, it removes any ambiguity on how well the parameters were tuned for the full model compared to the compressed networks. In practice, one could expect to attain even better results if parameter sweeping was performed on the compressed networks.

An *empirical upper bound* of the accuracy at different footprints is obtained by randomly choosing how many filters to remove at each layer, and by repeating this process a sufficient number of times. The best result at each footprint can be considered an empirical upper bound for that architecture and footprint. On the other hand, the result averaged across all random searches is representative of how easy



(a) SimpleCNN on CIFAR-10 (b) SimpleCNN on CIFAR-100

Figure 1. Results of different SimpleCNN compressed networks. Accuracy change in the y axis is reported in percentage points (error bars show the standard deviation of multiple runs). Note how: (1) all PFA solutions lie close to the upper bound while random pruning severely degrades accuracy; (2) in most cases filter selection strategy is better than random initalization.

(or difficult) it is to randomly compress a network without hurting its accuracy. In these experiments we trained **300** randomly pruned architectures for each footprint.

Results are reported in Fig. 1. The first notable remark (for both datasets and initialization strategies) is the considerable gap between the upper bound (up-facing triangles) and the average random search (crosses): this indicates, unsurprisingly, that random pruning is not an effective strategy. The second remark is that there exist smaller architectures derived from the base model that can perform even better than the full model. We attribute this result to the potential of a smaller model to generalize better.

PFA-En (squares) and PFA-KL (circles) are consistently better than the mean random search and very close to the empirical upper bound. The use of filter selection (green) improves in all approaches (even random search) compared to random initialization (brown). Notably, when using PFA with filter selection the accuracy for footprints above 40% becomes even better than that of the full model.

Interestingly, at the 30% footprint mark a random initialization for PFA-En appears to be better than the use of filter selection. It is possible that when keeping an extremely small number of filters, the starting point provided by the filter selection becomes a local minimum that is difficult to escape. For thin layers in relatively small architectures (like SimpleCNN), a random initialization may give more room for exploration during the learning phase.

Overall, we have found that the filter selection strategy converges faster during training and performs consistently better across different architectures and datasets, hence, from now on we will only report results using PFA with filter selection.

CIFAR-10 and CIFAR-100. We repeated the experiments above on known architectures and compared our results with state-of-the-art techniques.

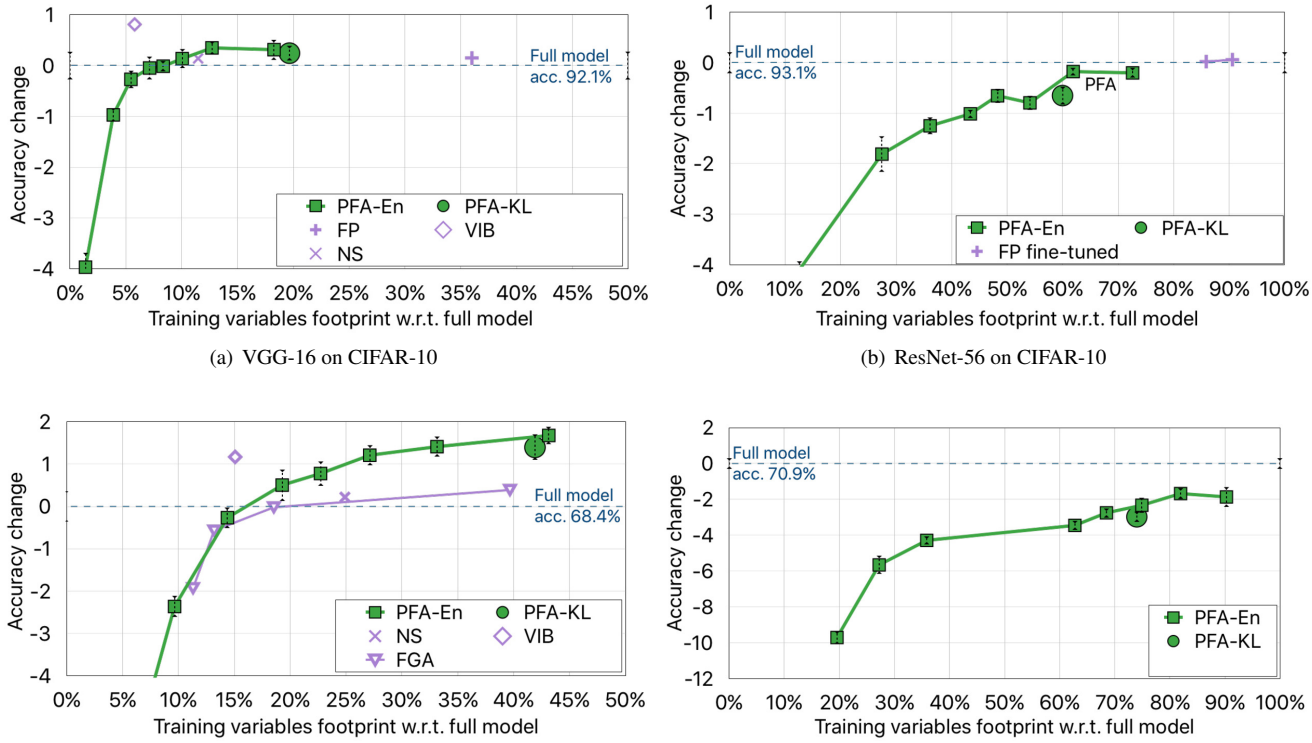
The architectures used are VGG-16 [46] (version proposed by [56] for CIFAR) and ResNet-56 [19], with padding in the skip-connections (refer to App. C.2 for the training hyper-parameters and for a detailed explanation on how we handle skip-connections with padding). We compare the results of PFA with those reported by FP, VIB[1], FGA[2], and NS (after a single iteration for a direct comparison with the other methods).

As shown in Fig. 2, results are consistent irrespective of the architecture and dataset: PFA is comparable to or does better than more complex techniques that require more computational resources such as NS and FGA. At comparable footprints, VIB achieves a slightly higher accuracy than PFA (around 1 pp), but again at higher computational cost.

ImageNet. For the experiments on ImageNet [12], we train and compress one baseline for each architecture. We retrain the models obtained by PFA 3 times (using the same hyper parameters used to train the full model) and report the mean and standard deviation of the Top-1 accuracy (we also provide the Top-5 accuracy in the App. B).

The architectures used are VGG-16 [32] (fully convolutional) and ResNet-34, with projection in the skip-connections (refer to App. C.3 for a detailed explanation

[1]Error of the original full models kindly provided by the authors of VIB.
[2]Number of trainable variables kindly provided by the authors of FGA.



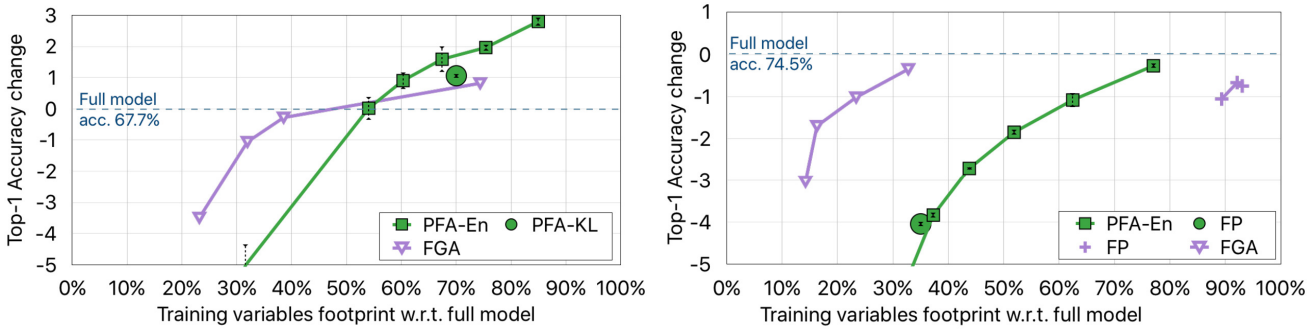
(a) VGG-16 on CIFAR-10

(b) ResNet-56 on CIFAR-10

(c) VGG-16 on CIFAR-100

(d) ResNet-56 on CIFAR-100

Figure 2. Results for VGG-16 and ResNet-56 on CIFAR-10 and CIFAR-100. Accuracy change in the y axis is reported in percentage points. Note how the accuracy obtained by PFA is comparable to the state of the art, and how PFA works across different architectures.



(a) VGG-16 on ImageNet

(b) ResNet-34 on ImageNet

Figure 3. Results on ImageNet. Accuracy change in the y axis is reported in percentage points. On VGG-16 PFA is comparable to FGA. On ResNet-34, PFA is better than FP but FGA, in this specific experiment, achieves an even stronger compression.

on how we handle skip-connections with projections).

Results are shown in Fig. 3. On VGG-16, PFA achieves better accuracy than FGA[3], at comparable model sizes, until a size of 50% of the full model, after which FGA maintains a better accuracy. On ResNet-34 PFA achieves both a better accuracy and stronger compression than FP. In this experiment, FGA is extremely efficient and outperforms other techniques, which is different than what we observed in previous experiments, where PFA performed comparably (VGG-16 with ImageNet) or better (VGG-16 with CIFAR-100) than FGA.

We have shown that PFA works consistently across different architectures and datasets. In general, PFA is comparable to the state of the art even without training hyper-parameters search, and contrary to most state-of-the-art algorithms it does not require tuning of its own parameters.

[3]Number of trainable variables kindly provided by the authors of FGA.

On the complexity and scalability of PFA The complexity of PFA (excluding the inference step), with respect to number of filters and dataset size, is dominated by the the PCA analysis which, for a given layer, is $O(mn^2 + n^3)$, with n being the number of filters, and m the number of samples. For example, for ImageNet, m=1.2M, and assuming a VGG-16 architecture with layers of size $n = 64$, 128, 256, 512, and 4096, the time to compute PFA per layer is roughly 1.24s, 2.8s, 4.6s, 9.3s, and 127.5s respectively (single CPU @ 2.30GHz). The complexity of the filter selection depends only on the layer size. In the worst case the complexity is $O(rn^2)$, where r is the number of filters to remove.

Considering that PFA has to run only once at the end of the training step, the time consumed by PFA is negligible compared to the whole training time. In exchange for this negligible additional time, PFA provides the long-term benefit of a smaller footprint and faster inference, which, in the lifetime of a deployed network, including re-training when new data becomes available, will quickly surpass the time initially required by PFA.

Once the eigenvalue set $\boldsymbol{\lambda}^{[\ell]}$ is computed for all layers, generating PFA recipes with different energy values is extremely fast. Hence, the threshold in the PFA-En strategy can conveniently be replaced with the target model size or FLOPs. PFA-En can then be computed iteratively with decreasing energy thresholds until the requested size is achieved. This seemingly small change in the interaction with the user is a great benefit since it is often difficult to relate algorithms parameters to practical characteristics (such as the size) of the final model.

4.2. Discussion

All state-of-the-art techniques analyzed achieve great results in term of maintaining accuracy and reducing memory footprint and FLOPs. In general, even without training hyper-parameter search, PFA yields competitive results. We believe, however, that the biggest advantage of PFA is its simplicity and efficacy compared to other techniques.

Often state-of-the-art algorithms are not adopted because of the high friction required for their application. For example, VIB requires the user to modify the network to perform a sampling step during the forward pass, FGA requires an optimization problem to be solved for each layer to decompose a convolutional layer into a group of smaller operations that approximate the output of the full layer, and NS requires the training protocol to be modified to induce sparsity in the full model. PFA is based on an intuitive idea: remove filters that produce correlated responses. This makes its implementation, application and adoption straightforward.

PFA does not need to modify any loss function, unlike NS and VIB, a potential second barrier to adoption. This makes it attractive because known hyper-parameters can be used for the full model, and also makes PFA deployable as a service: given a full model and a dataset, PFA can provide an initialized smaller model (without needing to know the loss function). Furthermore, while intuitively one might expect that techniques that modify the loss function should obtain better results (since the compression aspect is included in the optimization) our experiments did not show a consistent benefit compared to PFA.

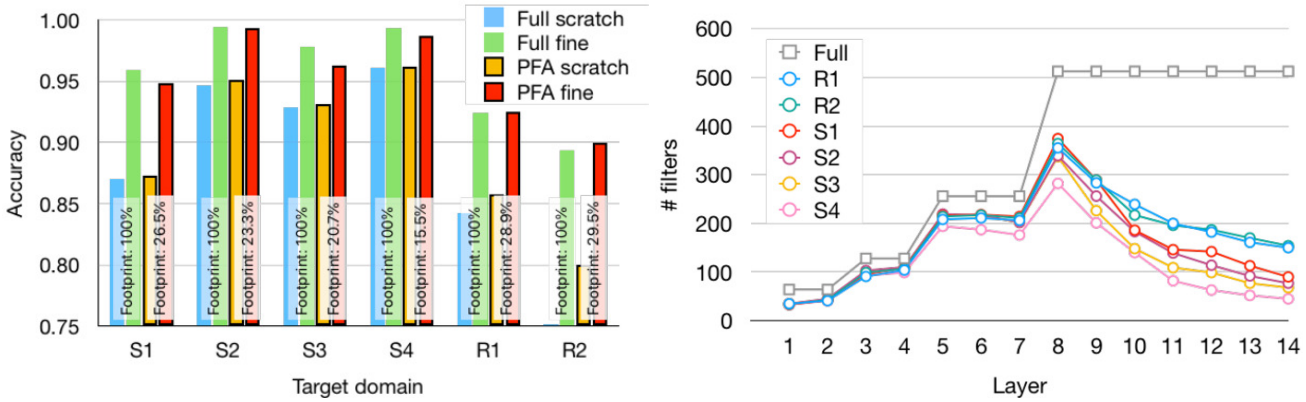
All state-of-the-art techniques analyzed require user defined parameters that need additional tuning and are difficult to relate the FLOPs or the size of the compressed model. FP needs a threshold to decide if the ℓ_1-norm of a filter is small enough to be pruned. This process is non-trivial and requires the user to choose the compression thresholds based on a pre-analysis that provides insight on the sensitivity of each layer to pruning. NS has two crucial parameters: the weight of the sparsity regularizer, and the percentage of filters to be pruned. The weight has a direct impact on the induced sparsity, however, there is no intuitive way to set this parameter and it needs to be tuned for each architecture and dataset. In addition, setting the same percentage of filters to be pruned at each layer for the whole network ignores the relative effect of those filters on the accuracy and the footprint of the network. VIB requires a parameter to control the influence of the information bottleneck term, which is related to the compression achieved. FGA requires a parameter that defines the compression ratio for each layer. In both algorithms, the tuning is different depending both on the network-dataset combination and the layer depth. From the results there seems to be no intuitive way to set this parameter other than by trial and error. In contrast, PFA requires a single intuitive parameter (for example the desired model size in PFA-En), or it is parameter-free (PFA-KL).

Lastly, compared to techniques based on weight analysis, such as FP, PFA is based on the responses of a layer. This means that different datasets used for the PFA analysis leads to different and specialized models, as we will describe in Sec. 4.3, which makes PFA a suitable candidate for the task of simultaneous compression and domain adaptation.

4.3. Simultaneous compression and domain adaptation using PFA

By compressing networks based on their responses, rather than their weights, the compressed networks become specialized for the target domain at hand: depending on the dataset used to generate the responses the compressed architecture will change. In this section, we show some examples of how PFA modifies the same original architecture differently to adapt to different target datasets, while taking advantage of the original training.

Let us denote the initial domain used for training as $\mathcal{D}_A$; in this test $\mathcal{D}_A$ is CIFAR-100. We denote the domain used for PFA as $\mathcal{D}_Z$. We generate different $\mathcal{D}_Z$s by randomly sampling classes out of the original 100 classes contained



(a) Domain adaptation from CIFAR-100

(b) PFA recipes starting from CIFAR-100

Figure 4. Domain adaption from CIFAR-100. (a) *PFA fine* matches the accuracy of *Full fine* while using architectures more than 4x smaller. *PFA fine* significantly outperforms the full model trained from scratch *Full scratch*. The vertical percentage labels show the PFA compression ratio. In (b) recipes for VGG-16 trained on CIFAR-100 using PFA-KL with data from different target domains. Note how PFA exploits the knowledge of the target domain, creating different recipes adapted to the task complexity.

in CIFAR-100. We generate two targets $\mathcal{D}_Z$ of 10 classes each (R1 and R2), and four targets $\mathcal{D}_Z$ of 2 classes each (S1, S2, S3, and S4). Refer to App. A for a detailed explanation of the target domains used, as well as experiments adapting from CIFAR-10 to the same $\mathcal{D}_Z$.

For each adaptation $\mathcal{D}_A \rightarrow \mathcal{D}_Z$ we run the following experiments using a VGG-16 model:

- **Full scratch**: Train from scratch on domain $\mathcal{D}_Z$.
- **Full fine**: Train from scratch on domain $\mathcal{D}_A$ and fine-tune on $\mathcal{D}_Z$.
- **PFA scratch**: Train from scratch on domain $\mathcal{D}_A$, run PFA-KL on domain $\mathcal{D}_Z$ and train the compressed architecture from scratch on $\mathcal{D}_Z$.
- **PFA fine**: Train from scratch on domain $\mathcal{D}_A$, run PFA-KL on domain $\mathcal{D}_Z$ and train the compressed architecture using filter selection on $\mathcal{D}_Z$.

The results in Fig. 4(a) show how the *PFA fine* strategy (red bars) performs similarly to the full fined tuned model (*Full fine*, green bars), while obtaining models more than 4 times smaller. Moreover, the *PFA fine* strategy significantly outperforms the full model trained from scratch on the target domain (*Full scratch*, blue bars).

The compressed architectures generated by PFA, Fig. 4(b), are different depending on the complexity of the final task. Note how PFA obtains architectures with more filters for the 10 class subsets (R1 and R2) than for the 2 class subset (S1, S2, S3, and S4). Even among the 2 class subset, there is a small variation in the final architecture, reflecting the different level of difficulty to distinguish between the two target classes.

These results show how by analyzing the responses rather than the weights, PFA is able to compress a network while specializing it to different domains.

5. Conclusions

Two effective, and yet easy to implement techniques for the compression of neural networks using Principal Filter Analysis are presented: PFA-En and PFA-KL. These techniques exploit the correlation of filter responses within layers to compress networks without compromising accuracy. PFA can be applied to the output response of any layer with no knowledge of the training procedure or the loss function. Our tests show that PFA is competitive with state-of-the-art approaches while removing adoption barriers thanks to its practical implementation, intuitive philosophy and ease of use. PFA-KL is parameter free, and PFA-En has only a single intuitive parameter: the energy to be preserved in each layer or a desired network characteristic (such as a target model size or FLOPs).

The flexibility of PFA makes it applicable to a wide variety of architectures that we would like to investigate in future: recurrent neural networks, models with attention, and even word embedding.

References

[1] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *NIPS*, pages 3180–3189, 2017.

[2] J. M. Alvarez and M. Salzmann. Compression-aware training of deep networks. volume abs/1711.02638, 2017.

[3] S. Arora, N. Cohen, and E. Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *NIPS*, 2018.

[4] L. J. Ba and R. Caurana. Do deep nets really need to be deep? In *NIPS*, 2014.

[5] C. Bucilua, R. Caruana, and A. Niculescu-Mizil. Model compression. In *SIGKDD*, pages 535–541, 2006.

[6] M. A. Carreira-Perpinan and Y. Idelbayev. "learning-compression" algorithms for neural net pruning. In *CVPR*, 2018.

[7] T. Chen, I. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. In *ICLR*, 11 2016.

[8] T. Cheng, X. Tong, W. Xiaogang, and W. E. Convolutional neural networks with low-rank regularization. In *ICLR*, 2016.

[9] M. Cogswell, F. Ahmed, R. B. Girshick, L. Zitnick, and D. Batra. Reducing overfitting in deep networks by decorrelating representations. In *ICLR*, 2016.

[10] G. Cohen, G. Sapiro, and R. Giryes. DNN or k-nn: That is the generalize vs. memorize question. *CoRR*, 2018.

[11] B. Dai, C. Zhu, B. Guo, and D. Wipf. Compressing neural networks using the variational information bottleneck. In *ICML*, 2018.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[13] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas. Predicting parameters in deep learning. In *NIPS*, pages 2148–2156, 2013.

[14] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, pages 1269–1277, 2014.

[15] M. A. et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[16] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[17] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. In *NIPS*, 2016.

[18] B. Hassibi, D. G. Stork, and G. Wolff. Optimal brain surgeon: Extensions and performance comparisons. In *NIPS*, pages 263–270, 1993.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[20] X. He and J. Cheng. Learning compression from limited unlabeled data. In *ECCV*, 2018.

[21] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

[22] G. Hinton, J. Dean, and O. Vinyals. Distilling the knowledge in a neural network. In *NIPS*, pages 1–9, 03 2014.

[23] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441 and 498–520, 1933.

[24] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[25] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, 2009.

[26] S. Kullback and R. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[27] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *ICLR*, 2014.

[28] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *NIPS*, pages 598–605, 1990.

[29] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

[30] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[31] R. Livni, S. Shalev-Shwartz, and O. Shamir. On the computational efficiency of training neural networks. In *NIPS*, 2014.

[32] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[33] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

[34] M. Ma, H. Pouransari, D. Chao, S. Adya, S. A. Serrano, Y. Qin, D. Gimnicher, and D. Walsh. Democratizing production-scale distributed deep learning. *CoRR*, abs/1811.00143, 2018.

[35] M. Masana, J. van de Weijer, L. Herranz, A. D. Bagdanov, and J. M. Álvarez. Domain-adaptive deep network compression. In *ICCV*, 2017.

[36] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. In *ICLR*, 2017.

[37] P. Nakkiran, R. Alvarez, R. Prabhavalkar, and C. Parada. Compressing deep neural networks using a rank-constrained topology. pages 1473–1477, 2015.

[38] K. Pearson. Notes on regression and inheritance in the case of two parents. In *Royal Society of London*, volume 58, pages 240–242, 1895.

[39] B. Peng, W. Tan, Z. Li, S. Zhang, D. Xie, and S. Pu. Extreme network compression via filter group approximation. In *ECCV*, 2018.

[40] R. Prabhavalkar, O. Alsharif, A. Bruguier, and I. McGraw. On the compression of recurrent neural networks with an application to LVCSR acoustic modeling for embedded speech recognition. In *ICASSP*, pages 5970–5974, 2016.

[41] N. Quynh and H. Matthias. The loss surface of deep and wide neural networks. In *ICML*, 2017.

[42] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[43] P. Rodríguez, J. Gonzàlez, G. Cucurull, J. M. Gonfaus, and F. X. Roca. Regularizing cnns with locally constrained decorrelations. In *ICLR*, 2017.

[44] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2015.

[45] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *ICCV*, 1998.

[46] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[47] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. In *BMVC*, pages 31.1–31.12, 2015.

[48] F. Tung and G. Mori. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *CVPR*, 2018.

[49] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *NIPS*, pages 2074–2082, 2016.

[50] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li. Coordinating filters for faster deep neural networks. In *ICCV*, 2017.

[51] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *CVPR*, 2016.

[52] J. Wu, Y. Wang, Z. Wu, Z. Wang, A. Veeraraghavan, and Y. Lin. Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions. In *ICML*, 2018.

[53] J. Xue, J. Li, and Y. Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *ICASSP*, pages 6359–6363, 2013.

[54] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis. Nisp: Pruning networks using neuron importance score propagation. In *CVPR*, 2018.

[55] X. Yu, T. Liu, X. Wang, and D. Tao. On compressing deep models by low rank and sparse decomposition. In *CVPR*, pages 7370–7379, 2017.

[56] S. Zagoruyko. 92.45% on cifar-10 in torch. `https://github.com/szagoruyko/cifar.torch`, 2015.

[57] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *ECCV*, 2018.