Cascaded Projection: End-to-End Network Compression and Acceleration

Breton Minnehan
Rochester Institute of Technology
blm2144@rit.edu

Andreas Savakis
Rochester Institute of Technology
andreas.savakis@rit.edu

Abstract

We propose a data-driven approach for deep convolutional neural network compression that achieves high accuracy with high throughput and low memory requirements. Current network compression methods either find a low-rank factorization of the features that requires more memory, or select only a subset of features by pruning entire filter channels. We propose the Cascaded Projection (CaP) compression method that projects the output and input filter channels of successive layers to a unified low dimensional space based on a low-rank projection. We optimize the projection to minimize classification loss and the difference between the next layer's features in the compressed and uncompressed networks. To solve this non-convex optimization problem we propose a new optimization method of a proxy matrix using backpropagation and Stochastic Gradient Descent (SGD) with geometric constraints. Our cascaded projection approach leads to improvements in all critical areas of network compression: high accuracy, low memory consumption, low parameter count and high processing speed. The proposed CaP method demonstrates state-of-the-art results compressing VGG16 and ResNet networks with over 4× reduction in the number of computations and excellent performance in top-5 accuracy on the ImageNet dataset before and after fine-tuning.

1. Introduction

The compression of deep neural networks is gaining attention due to the effectiveness of deep networks and their potential applications on mobile and embedded devices. The powerful deep networks developed today are often overparameterized [9] and require large amounts of memory and computational resources [3]. Thus, efficient network compression, that reduces the number of computations and memory required to process images, enables the broader application of deep neural networks.

Methods for network compression can be categorized into four types, based on quantization, sparsification, factorization and pruning. In this work we leverage the advantages of factorization and pruning methods, as they are the most popular. Quantization methods accelerate deep networks and reduce storage by using mixed precision arithmetic and hashing codes [4, 6, 13]. However most of them require mixed precision arithmetic, which is not always available on standard hardware. Sparsification methods eliminate individual connections between nodes that have minimal impact on the network, however, they are not well suited for current applications because most neural network libraries are not optimized for sparse convolution operations and fail to achieve significant speedup.

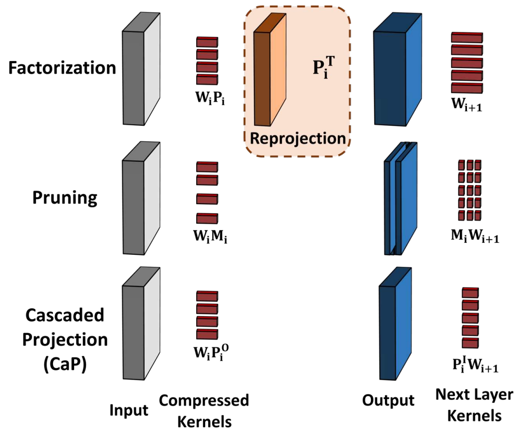

Figure 1. Visual representation of network compression methods on a single CNN layer. Top row: Factorization compression with a reprojection step that increases memory. Middle row: Pruning compression where individual filters are removed. Bottom row: Proposed CaP method which forms linear combinations of the filters without requiring reprojection.

Factorization methods [10, 29, 33, 55] reduce computations by factorizing the network kernels, often by splitting large kernels into a series of convolutions with smaller filters. These methods have the drawback of increasing memory consumption due to the intermediate convolution operations. Such memory requirements pose a problem for mobile applications, where network acceleration is needed most. Pruning methods [13, 19, 35, 37, 39, 44, 54, 56] compress layers of a network by removing entire convolutional filters and the corresponding channels in the filters of the next layer. They do not require feature map reprojection, however they discard a large amount of information when eliminating entire filter channels.

In this paper, we propose the Cascaded Projection (CaP) compression method which combines the superior reconstruction ability of factorization methods with the multi-layer cascaded compression of pruning methods. Instead of selecting a subset of features, as is done in pruning methods, CaP forms linear combinations of the original features that retain more information. However, unlike factorization methods, CaP brings the kernels in the next layer to low dimensional feature space and therefore does not require additional memory for reprojection.

Figure 1 provides a visual representation of the differences between the three methods: factorization (top row) reprojects to higher dimensional space and increases memory, pruning (middle row) masks filters and eliminates their channels, and our proposed CaP methods (bottom row) combines filters to a smaller number without reprojecting. Our results demonstrate that by forming filters based on linear combinations instead of pruning with a mask, more information is kept in the filtering operations and better network classification accuracy is achieved. The primary contributions of this paper are the following:

1. We propose the CaP compression method that finds a low dimensional projection of the feature kernels and cascades the projection to compress the input channels of the kernels in the next layers.
2. We introduce proxy matrix projection backpropagation, the first method to optimize the compression projection for each layer using end-to-end training with standard backpropagation and stochastic gradient descent.
3. Our optimization method allows us to use a new loss function that combines the reconstruction loss with classification loss to find a better solution.
4. The CaP method is the first to simultaneously optimize the compression projection for all layers of residual networks.
5. Our results illustrate that CaP compressed networks achieve state-of-the-art accuracy while reducing the network's number of parameters, computational load and memory consumption.

2. Related Work

The goal of network compression and acceleration is to reduce the number of parameters and computations performed in deep networks without sacrificing accuracy. Early work in network pruning dates back to the 1990's [14]. However, the area did not gain much interest until deep convolutional networks became common [31, 32, 43] and the redundancy of network parameters became apparent [9]. Recent works aim to develop smaller network architectures that require fewer resources [20, 25, 42].

Quantization techniques [4, 6, 13, 28] use integer or mixed precision arithmetic only available on state-of-the-art GPUs [38]. These methods reduce the computation time and the amount of storage required for the network parameters. They can be applied in addition to other methods to further accelerate compressed networks, as was done in [30].

Network sparsification [36], sometimes referred to as unstructured pruning, reduces the number of connections in deep networks by imposing sparsity constraints. The work in [21] proposed recasting the sparsified network into separate groups of operations where the filters in each layer are only connected to a subset of the input channels. In [52] k-means clustering is used to encourage similarity between features to aid in compression. However, these methods require training the network from scratch which is not practical or efficient.

Filter factorization methods reduce computations at the cost of increased memory load for storing intermediate feature maps. Initial works focused on factorizing the three-dimensional convolutional kernels into three separable one-dimensional filters [10, 29]. In [33] CP-decomposition is used to decompose the convolutional layers into five layers with lower complexity. More recently [55] performed a channel decomposition that found a projection of the convolutional filters in each layer such that the asymmetric reprojection error was minimized.

Channel pruning methods [35, 37, 39, 44, 56] remove entire feature kernels for network compression. In [13] kernels are pruned based on their magnitudes, under the assumption that kernels with low magnitudes provide little information to the network. Li *et al*. [35] suggested a similar pruning technique based on kernel statistics. He *et al*. [19] proposed pruning filters based on minimizing the reconstruction error of each layer. Luo *et al*. [37] further extended the concepts in [19] to prune filters that have minimal impact on the reconstruction of the next layer. Yu *et al*. [54] proposed Neuron Importance Score Propagation (NISP) to calculate the importance of each neuron based on its contribution to the final feature representation and prune feature channels that provide minimal information to the final feature representation.

Other recent works have focused less on finding the optimal set of features to prune and more on finding the optimal amount of features to remove from each layer of the network. This is important to study because the amount of pruning performed in each layer is often set arbitrarily or through extensive experimentation. In [53, 54] the authors propose automatic pruning architecture methods based on statistical measures. In [18, 24] methods are proposed which use reinforcement learning to learn an optimal network compression architecture. Additional work has been done to reduce the number of parameters in the final layers of deep networks [5], however the fully connected layer only contributes a small fraction of the overall computations.

3. Cascaded Projection Methodology

In this section we provide an in depth discussion of the CaP compression and acceleration method. We first introduce projection compression when applied to a single layer, and explain the relationship between CaP and previous filter factorization methods [55]. One of the main goals of CaP compression is eliminating the feature reprojection step performed in factorization methods. To accomplish this, CaP extends the compression in the present layer to the inputs of the kernels in the next layer by projecting them to the same low dimensional space, as shown in Figure 2. Next we demonstrate that, with a few alterations, the CaP compression method can perform simultaneous optimization of the projections for all of the layers in residual networks [15]. Lastly we present the core component of the CaP method, which is our new end-to-end optimization method that optimizes the layer compression projections using standard back-propagation and stochastic gradient descent.

3.1. Problem Definition

In a convolutional network, as illustrated in the top row of Fig. 2, the i^{th} layer takes as input a 4-Tensor $\mathbf{I_i}$ of dimension $(n{\times}c_i{\times}h_i{\times}w_i)$, where n is the number of images (mini-batch size) input into the network, c_i is the number channels in the input and w_i and h_i are the height and width of the input. The input is convolved with a set of filters $\mathbf{W_i}$ represented as a 4-Tensor with dimensions $(c_{i+1}{\times}c_i{\times}k{\times}k)$, where c_{i+1} is the number of kernels and k is the spatial dimensions of the kernels, generally 3 pixels. In many networks, there is an additional bias, $\mathbf{b_i}$, of dimension $(c_{i+1} \times 1 \times 1 \times 1)$, that is added to each channel of the output. More formally, the convolution operation for layer i of a CNN is given as:

$$\mathbf{O_i} = \mathbf{I_i} * \mathbf{W_i} + \mathbf{b_i} \tag{1}$$

where $(*)$ is the convolution operator. The input to the next layer is calculated by applying a nonlinearity to the output as $\mathbf{I_{i+1}} = G(\mathbf{O_i})$, where $G(\cdot)$ is often a ReLU [40].

Network compression aims to reduce the number of filters so that the classification accuracy of the network is minimally impacted. In this work we find a projection $\mathbf{P_i}$ that maps the features to a lower dimensional space by minimizing both the reconstruction error and the classification loss, as described in the rest of this section.

3.2. Single Layer Projection Compression

We first present how projection based compression is used to compress a single layer of a network. To compress layer i, the output features are projected to low dimensional representation of rank r using an orthonormal projection matrix $\mathbf{P_i}$ represented as a 4-Tensor of dimension $(c_{i+1} \times r \times 1 \times 1)$. The optimal projection, $\mathbf{P_i^*}$ for layer i, based on minimizing the reconstruction loss is given as:

$$\mathbf{P_i^*} = \underset{\mathbf{P_i}}{\operatorname{argmin}} \left\|\mathbf{O_i} - (\mathbf{I_i} * \mathbf{W_i} * \mathbf{P_i} + \mathbf{b_i} * \mathbf{P_i}) * \mathbf{P_i^T}\right\|_F^2 \tag{2}$$

where $\|\cdot\|_F^2$ is the Frobenious norm.

Inspired by [55], we alter our optimization criteria to minimize the reconstruction loss of the input to the next layer. This results in the optimization:

$$\mathbf{P_i^*} = \underset{\mathbf{P_i}}{\operatorname{argmin}} \left\|G(\mathbf{O_i}) - G((\mathbf{I_i} * \mathbf{W_i} * \mathbf{P_i} + \mathbf{b_i} * \mathbf{P_i}) * \mathbf{P_i^T})\right\|_F^2 \tag{3}$$

The inclusion of the nonlinearity makes this a more difficult optimization problem. In [55] the problem is relaxed and solved using Generalized SVD [12, 49, 50]. Our Cascaded Projection method is based on the end-to-end approach described next.

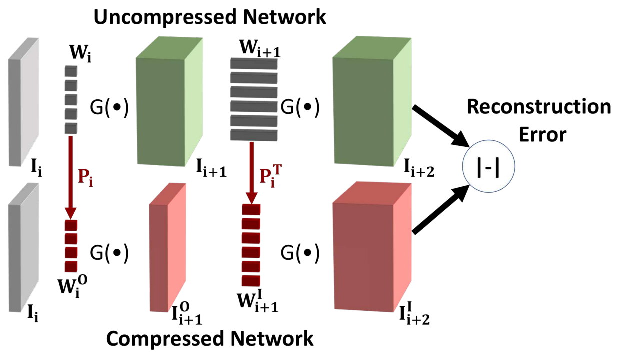

Figure 2. Visual representation of the compression of a CNN layer using the CaP method to compress the filters $\mathbf{W_i}$ and $\mathbf{W_{i+1}}$ in the current and next layers using projections $\mathbf{P_i}$ and $\mathbf{P_i^T}$ respectively. The reconstruction error in the next layer is computed after the nonlinearity $G(\cdot)$.

3.3. Cascaded Projection Compression

Factorization methods, including the single layer projection compression discussed above, are inefficient due to the additional convolution operations required to reproject the features to high dimensional space. Pruning methods avoid reprojection by removing all channels associated with the pruned filters. CaP takes a more powerful approach that forms linear combination of the kernels by projecting without the extra memory requirements of factorization methods. Following the diagram in Figure 2, we consider two successive convolutional layers, labeled i and $i+1$, with kernels $\mathbf{W_i}$, $\mathbf{W_{i+1}}$ and biases $\mathbf{b_i}$, $\mathbf{b_{i+1}}$ respectively. The input to layer i is $\mathbf{I_i}$, while the output of layer $i+1$ is the input to layer $i+2$, denoted by $\mathbf{I_{i+2}}$ and given below.

$$\mathbf{I_{i+2}} = G(G(\mathbf{I_i} * \mathbf{W_i} + \mathbf{b_i}) * \mathbf{W_{i+1}} + \mathbf{b_{i+1}}) \quad (4)$$

After substituting our compressed representation with reprojection for layer i in the above we get:

$$\mathbf{I_{i+2}} = G(G((\mathbf{I_i} * \mathbf{W_i} * \mathbf{P_i} + \mathbf{b_i} * \mathbf{P_i}) * \mathbf{P_i^T}) * \mathbf{W_{i+1}} + \mathbf{b_{i+1}}) \quad (5)$$

To avoid reprojecting the low dimensional features back to higher dimensional space with $\mathbf{P_i^T}$, we seek two projections. The first $\mathbf{P_i^O}$ which captures the optimal lower dimensional representation of the features in the current layer, and the second $\mathbf{P_i^I}$ which pulls the kernels of the next layer down to lower dimensional space. This formulation leads to an optimization problem over the projection operators:

$$\begin{aligned}\{\mathbf{P_i^I}^*, \mathbf{P_i^O}^*\} = \underset{\mathbf{P_i^I}, \mathbf{P_i^O}}{\operatorname{argmin}} \|\mathbf{I_{i+2}} - G(G((\mathbf{I_i} * \mathbf{W_i} * \mathbf{P_i^O} \\ + \mathbf{b_i} * \mathbf{P_i^O})) * \mathbf{P_i^I} * \mathbf{W_{i+1}} + \mathbf{b_{i+1}})\|_F^2\end{aligned} \quad (6)$$

To make the problem tractable, we enforce two strong constraints on the projections. We require that they are orthonormal and transposes of each other: $\mathbf{P_i^I} = (\mathbf{P_i^O})^T$. For the remainder of this work we replace $\mathbf{P_i^O}$ and $\mathbf{P_i^I}$ with $\mathbf{P_i}$ and $\mathbf{P_i^T}$, respectively. These constraints make the optimization problem more feasible by reducing the parameter search space to a single projection operator for each layer.

$$\begin{aligned}\mathbf{P_i^*} = \underset{\mathbf{P_i}, \mathbf{P_i} \in \mathbb{O}^{\mathbf{n} \times \mathbf{m}}}{\operatorname{argmin}} \|\mathbf{I_{i+2}} - G(G((\mathbf{I_i} * \mathbf{W_i} * \mathbf{P_i} \\ + \mathbf{b_i} * \mathbf{P_i})) * \mathbf{P_i^T} * \mathbf{W_{i+1}} + \mathbf{b_{i+1}})\|_F^2\end{aligned} \quad (7)$$

We solve the optimization of a single projection operator for each layer using a novel data-driven optimization method for projection operators discussed in Section 3.6.

3.3.1 Kernel Compression and Relaxation

Once the projection optimization is complete, we replace the kernels and biases in the current layer with their projected versions $\mathbf{W_i^O} = \mathbf{W_i} * \mathbf{P_i}$ and $\mathbf{b_i^O} = \mathbf{b_i} * \mathbf{P_i}$ respectively. We also replace the kernels in the next layer with their input compressed versions $\mathbf{W_{i+1}^I} = \mathbf{P_i^T} * \mathbf{W_{i+1}}$. Thus,

$$\mathbf{I_{i+2}} = G(G((\mathbf{I_i} * \mathbf{W_i^O} + \mathbf{b_i^O})) * \mathbf{W_{i+1}^I} + \mathbf{b_{i+1}}) \quad (8)$$

Figure 2 depicts how the filters $\mathbf{W_{i+1}^I}$ in the next layer are compressed using the projection $\mathbf{P_i^T}$ and are therefore smaller than the kernels in the original network. Utilizing the compressed kernels $\mathbf{W_i^O}$ and $\mathbf{W_i^I}$ results in twice the speedup over traditional factorization methods for all compressed intermediate layers (other than first and last layers).

Following kernel projection, we perform an additional round of training in which only the compressed kernels are optimized. We refer to this step as kernel relaxation because we are allowing the kernels to find a better optimal solution after our projection optimization step.

3.4. Mixture Loss

A benefit of gradient based optimization is that a loss function can be altered to minimize both reconstruction and classification error. Previous methods have focused on either reconstruction error minimization [19, 37] or classification [54] based metrics when pruning each layer. We propose using a combination of the standard cross entropy classification loss, L_{Class}, and the reconstruction loss L_R, shown in Figure 2. The reconstruction loss for layer i is given as:

$$\begin{aligned}L_R(i) = \|\mathbf{I_{i+2}} - G(G((\mathbf{I_i} * \mathbf{W_i} * \mathbf{P_i} \\ + \mathbf{b_i} * \mathbf{P_i})) * \mathbf{P_i^T} * \mathbf{W_{i+1}} + \mathbf{b_{i+1}})\|_F^2\end{aligned} \quad (9)$$

The mixture loss used to optimize the projections in layer i is given as

$$L(i) = L_R(i) + \gamma L_{Class} \quad (10)$$

where γ is a mixture parameter that allows adjusting the impact of each loss during training. By using a combination of the two losses we obtain a compressed network that maintains classification accuracy while having feature representations for each layer which contain the maximal amount of information from the original network.

3.5. Compressing Multi-Branch Networks

Multi-branch networks are popular due to their excellent performance and come in a variety of forms such as the Inception networks [46, 47, 45], Residual networks (ResNets) [15] and Dense Networks (DenseNets) [22] among others. We primarily focus on applying CaP network compression to ResNets, but our method can be integrated with other multi-branch networks. We select the ResNet architecture for two reasons. First, ResNets have a proven record of producing state-of-the art results [15, 16]. And second, the skip connections work well with network compression, as they allow propagating information through the network regardless of the compression process within the individual layers.

Our CaP modification for ResNet compression is illustrated in Figure 3. In our approach, we do not alter the structure of the residual block outputs, therefore we do not compress the outputs of the last convolution layers in each residual block, as was done by [37]. In [35, 54] pruning is performed on the residual connections, but we do not affect them, because pruning these layers has a large negative impact on the network's accuracy.

We calculate the reconstruction error in ResNets at the outputs of each residual block, as shown in Fig. 3, in contrast to single branch networks where we calculate the reconstruction error at the next layer as shown in Fig. 2. By calculating the reconstruction error after the skip connections, we leverage the information in the skip connections in our projection optimization.

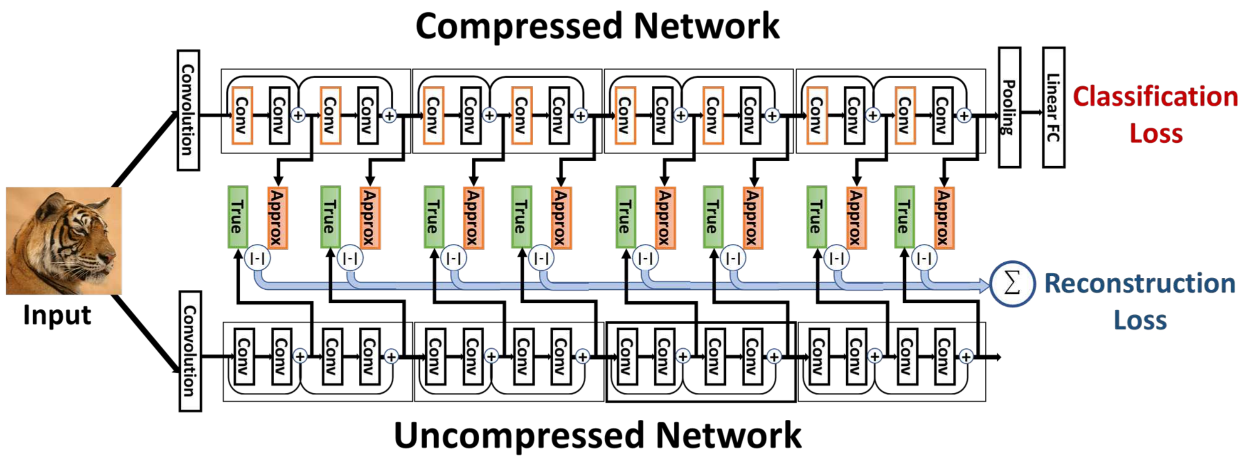

Figure 3. Illustration of simultaneous optimization of the projections for each layer of the ResNet18 network using a mixture loss that includes the classification loss and the reconstruction losses in each layer for intermediate supervision. We do not alter the structure of the residual block outputs, therefore we do not affect residual connections and we do not compress the outputs of the last convolution layers in each residual block.

3.5.1 Simultaneous Layer Compression

Most network compression methods apply a greedy layer-wise compression scheme, where one layer is compressed or pruned at a time. However, this layer-by-layer approach to network compression can lead to sub-optimal results [54]. We now present a version of CaP where all layers are simultaneously optimized. This approach allows the latter layers to help guide the projections of the earlier layers and minimize the total reconstruction error throughout the network.

In our experiments, we found that simultaneous optimization of the projection matrices has the risk of becoming unstable when we compress more than one layer in each residual block. To overcome this problem we split the training of the projections in residual blocks with more than one compressible layer into two rounds. In the first round, the projections for the odd layers are optimized, and in the second round the even layer projections are optimized.

Additionally, we found that using the reconstruction loss at the final layers did not provide enough supervision to the network. We therefore introduced deep supervision for each layer by minimizing the sum of normalized reconstruction losses for each layer, given by:

$$\underset{\mathbf{P_i}\in\mathbf{P}}{\operatorname{argmin}} \sum_{i=0}^{N} L_R(i) + \gamma L_{Class} \tag{11}$$

where $\mathbf{P_i}$ is the projection for the i^{th} layer, and N is the total number of layers. We outline our approach to finding a solution for the above optimization using iterative backpropagation next.

3.6. Back-Propagated Projection Optimization

In this section we present an end-to-end Proxy Matrix Projection (PMaP) optimization method, which is an iterative optimization of the projection using backpropagation with Stochastic Gradient Descent (SGD). The proposed method efficiently optimizes the network compression by combining backpropagation with geometric constraints.

In our framework, we restrict the projection operators to be orthogonal and thus satisfy $\mathbf{P_i}^T\mathbf{P_i} = \mathbf{I}$. The set of $(n \times m)$ real-valued orthogonal matrices $\mathbb{O}^{\mathbf{n\times m}}$, forms a smooth manifold known as a Grassmann manifold. There are several optimization methods on Grassmann manifolds, most of which include iterative optimization and retraction methods [7, 1, 48, 2, 51].

With CaP compression, the projection for each layer is dependent on the projections in all previous layers adding dependencies in the optimization across layers. Little work had been done in the field of optimization over multiple dependent Grassmann manifolds. Huang *et al.* [23] impose orthogonality constraints on the weights of a neural network during training using a method for backpropagation of gradients through structured linear algebra layers developed in [26, 27]. Inspired by these works, we utilize a similar approach where instead of optimizing each projection matrix directly, we use a proxy matrix $\mathbf{X_i}$ for each layer i and a transformation $\Phi(\cdot)$ such that $\Phi(\mathbf{X_i}) = \mathbf{P_i}$.

We obtain the transformation $\Phi(\cdot)$ that projects each proxy matrix $\mathbf{X_i}$ to the closest location on the Grassmann manifold by performing Singular Value Decomposition (SVD) on $\mathbf{X_i}$, such that $\mathbf{X_i} = \mathbf{U_i}\mathbf{\Sigma_i}\mathbf{V_i^T}$, where $\mathbf{U_i}$ and $\mathbf{V_i^T}$ are orthogonal matrices and $\mathbf{\Sigma_i}$ is the matrix of singular values. The projection to the closest location on the Grassmann manifold is performed as $\Phi(\mathbf{X_i}) = \mathbf{U_i}\mathbf{V_i^T} = \mathbf{P_i}$.

During training, the projection matrix $\mathbf{P_i}$ is not updated directly; instead the proxy parameter $\mathbf{X_i}$ is updated based on the partial derivatives of the loss with respect to $\mathbf{U_i}$ and $\mathbf{V_i}$, $\frac{\partial L}{\partial \mathbf{U_i}}$ and $\frac{\partial L}{\partial \mathbf{V_i}}$ respectively. The partial derivative of the loss L with respect to the proxy parameter $\mathbf{X_i}$ was derived in [26, 27] using the chain rule and is given by:

$$\frac{\partial L}{\partial \mathbf{X_i}} = \mathbf{U_i}\left\{ 2\mathbf{\Sigma_i}\left(\mathbf{K_i^T} \circ \left(\mathbf{V_i^T}\frac{\partial L}{\partial \mathbf{V_i}}\right)\right)_{sym} + \frac{\partial L}{\partial \mathbf{\Sigma_i}}\right\}\mathbf{V_i^T} \tag{12}$$

where $\circ$ is the Hadamard product, $\mathbf{A_{sym}}$ is the symmetric part of matrix A given as $\mathbf{A_{sym}} = \frac{1}{2}(\mathbf{A^T} + \mathbf{A})$. Since $\Phi(\mathbf{X_i}) = \mathbf{U_i}\mathbf{V_i^T}$, the loss does not depend on the matrix $\mathbf{\Sigma_i}$. Thus, $\frac{\partial L}{\partial \mathbf{\Sigma_i}} = 0$, and equation (12) becomes:

$$\frac{\partial L}{\partial \mathbf{X_i}} = \mathbf{U_i}\left\{ 2\mathbf{\Sigma_i}\left(\mathbf{K_i^T} \circ \left(\mathbf{V_i^T}\frac{\partial L}{\partial \mathbf{V_i}}\right)\right)_{sym}\right\}\mathbf{V_i^T} \tag{13}$$

The above allows us to optimize our compression projection operators for each layer of the network using backpropagation and SGD. Our method allows for end-to-end network compression using standard deep learning frameworks for the first time.

4. Experiments

We first perform experiments on independent layer compression of the VGG16 network to investigate how each layer responds to various levels of compression. We then perform a set of ablation studies on the proposed CaP algorithm to determine the impact for each step of the algorithm on the final accuracy of the compressed network. We compare CaP to other state-of-the-art methods by compressing the VGG16 network to have over 4× fewer floating point operations. Finally we present our experiments with varying levels of compression of ResNet architectures, with 18 or 50 layers, trained on the CIFAR10 dataset.

All experiments were performed using PyTorch 0.4 [41] on a work station running Ubuntu 16.04. The workstation had an Intel i5-6500 3.20GHz CPU with 15 GB of RAM and a NVIDIA Titan V GPU.

4.1. Layer-wise Experiments

In these experiment we investigate how each layer of the network is affected by increasing amounts of compression. We perform filter compression using CaP for each layer independently, while leaving all other layers uncompressed. We considered a range of compression for each layer, from 5% to 99%, and display the results in Figure 4. This plot shows two trends. Firstly the reconstruction error does not increase much until 70% compression, indicating that a large portion of the parameters in each layer

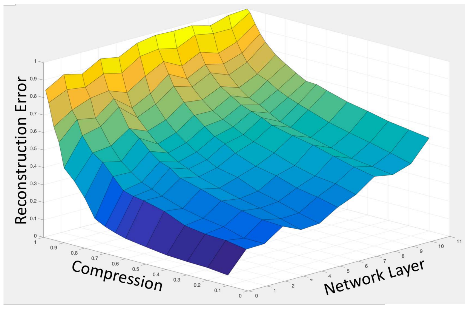

Figure 4. Plot of the reconstruction error (vertical axis) for the range of compression (left axis) for each layer of the network (right axis). The reconstruction error is lower when early layers are compressed.

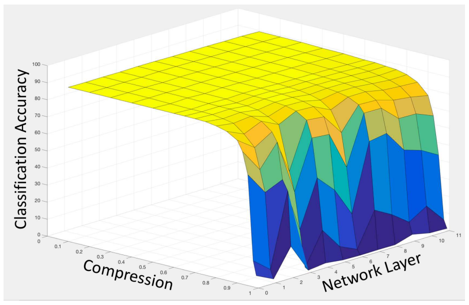

Figure 5. Plot of the classification accuracy (vertical axis) for the range of compression (left axis) for each layer of the network (right axis). The classification accuracy remains unaffected for large amounts of compression in a single layer anywhere in the network.

are redundant and could be reduced without much loss in accuracy. The second trend is the increase in reconstruction error for each level of compression for the deeper layers of the network (right axis).

In Figure 5 we plot the network accuracy resulting from each level of compression for each layer. The network is relatively unaffected for a large range of compression, despite the fact that there is a significant amount of reconstruction error introduced by the compression shown in Figure 4.

4.2. CaP Ablation Experiments

We ran additional experiments to determine the contribution of the projection optimization and kernel relaxation steps of our algorithm. We first trained the ResNet18 network on the CIFAR100 dataset and achieved a baseline accuracy of 78.23%. We then compressed the network to 50% of the original size using only parts of the CaP method to assess the effects of different components. We present these results in Table 1.

We also trained a compressed version ResNet18 from scratch for 350 epochs, to provide a baseline for the compressed ResNet18 network. When only projection compression is performed on the original ResNet18 network, there was a drop in accuracy of 1.58%. This loss in classification accuracy decreased to 0.76% after kernel relaxation. In contrast, when the optimized projections are replaced with random projections and only kernel relaxation training is performed, there is a 1.96% drop in accuracy, a 2.5 times increase in classification error. These results demonstrate that the projection optimization is an important aspect of our network compression algorithm, and the combination of both steps outperforms training the compressed network from scratch.

ResNet18 Network Variation	Accuracy
ResNet18 Uncompressed (upper bound)	78.23
Compressed ResNet18 from Scratch	77.22
CaP Compression with Projection Only	76.65
CaP with Random Proj. & Kernel Relax	76.27
CaP with Projection & Kernel Relax	**77.47**

Table 1. Network compression ablation study of the CaP method compressing the ResNet18 Network trained on the CIFAR100 dataset. (Bold numbers are best).

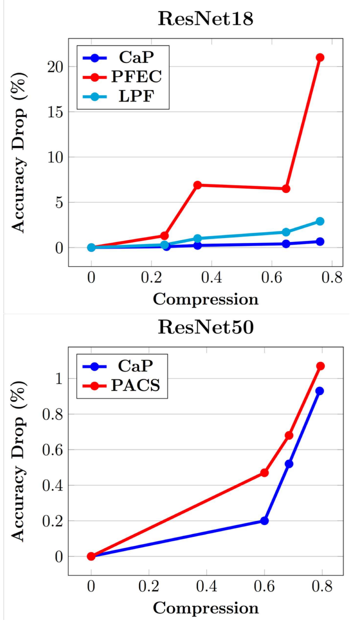

Figure 6. Classification accuracy drop on CIFAR10, relative to baseline, of compression methods (CaP, PCAS [53], PFEC [35] and LPF [24]) for a range of compression levels on ResNet18 (Top) and ResNet50 (Bottom).

4.3. ResNet Compression on CIFAR 10

We perform two sets of experiments using ResNet18 and ResNet50 trained on the CIFAR10 dataset [31]. We compress 18 and 50 layer ResNets with varying levels of compression and compare the relative drop in accuracy of CaP with other state-of-the-art methods [53, 35, 24]. We plot the drop in classification accuracy for ResNet18 and ResNet50 in Figure 6. For both networks, the CaP method outperforms the other methods for the full range of compression.

ResNet	Method	FT	FLOPs %	Acc. / Base
56	PFEC [35]	N	72.4	91.31 / 93.04
	CP [19]	N	50.0	90.90 / 92.80
	SFP [17]	N	47.4	92.26 / 93.59
	AMC [18]	N	50.0	90.1 / 92.8
	CaP	N	50.2	**92.92** / 93.51
	PFEC [35]	Y	72.4	93.06 / 93.04
	NISP [54]	Y	57.4	(-0.03) *
	CP [19]	Y	50.0	91.80 / 92.80
	SFP [17]	Y	47.4	93.35 / 93.59
	AMC [18]	Y	50.0	91.9 / 92.8
	DCP [56]	Y	35.0	**93.7** / 93.6
	CaP	Y	50.2	93.22 / 93.51
110	PFEC [35]	N	61.4	92.94 / 93.53
	MIL [11]	N	65.8	93.44 / 93.63
	SFP [17]	N	59.2	93.38 / 93.68
	CaP	N	50.1	**93.95**/ 94.29
	PFEC [35]	Y	61.4	93.30 / 93.53
	NISP [54]	Y	56.3	(-0.18) *
	SFP [17]	Y	59.2	93.86 / 93.68
	CaP	Y	50.1	**94.14**/ 94.29

Table 2. Comparison of CaP with pruning and factorization based methods using ResNet56 and ResNet110 trained on CIFAR10. FT denotes fine-tuning. (Bold numbers are best). * Only the relative drop in accuracy was reported in [54] without baseline accuracy.

In Table 2, we present classification accuracy of ResNet56 and ResNet110 with each residual block compressed to have 50% fewer FLOPs using CaPs. We compare the results obtained by CaP with those of [17, 18, 35, 54, 19] where the networks have been subjected to similar compression ratios. We report accuracy results with and without fine-tuning and include the baseline performance for comparison.

Results with fine-tuning are generally better, except in cases when there is over-fitting. However, fine-tuning for a long period of time can hide the poor performance of a compression algorithm by retraining the network filters away from the compression results. The results of the CaP method without fine-tuning are based on projection optimization and kernel relaxation on the compressed filters with reconstruction loss, while the fine-tuning results are produced with an additional round of training based on mixture loss for all of the layers in the network.

Method	Parameters	Memory (Mb)	FLOPs	GPU Speedup	Top-5 Accuracy / Baseline
VGG16 [43] (Baseline)	14.71M	3.39	30.9B	1	89.9
Low-Rank [29]	-	-	-	1.01*	80.02 / 89.9
Asym. [55]	**5.11M**	3.90	3.7B	1.55*	86.06 / 89.9
Channel Pruning [19]	7.48M	1.35	**6.8B**	2.5*	82.0 / 89.9
CaP (based on [19] arch)	7.48M	1.35	**6.8B**	3.05	86.57 / 90.38
CaP Optimal	7.93M	**1.11**	**6.8B**	**3.44**	**88.23** / 90.38

Table 3. Network compression results of pruning and factorization based methods without fine-tuning. The top-5 accuracy of the baseline VGG16 network varies slightly for each of the methods due to different models and frameworks. (Bold numbers are best). Results marked with * were obtained from [19].

Method	Mem. (Mb)	FLOPs	Top-5 Acc. / Baseline
VGG16 [43]	3.39	30.9B	89.9
Scratch [19]	1.35	6.8B	88.1
COBLA [34]	4.21	7.7B	88.9 / 89.9
Tucker [30]	4.96	**6.3B**	**89.4** / 89.9
CP [19]	1.35	6.8B	88.9 / 89.9
ThiNet-2 [37]	1.44	6.7B	88.86 / 90.01
CaP	**1.11**	6.8B	89.39 / 90.38

Table 4. Network compression results of pruning and factorization based methods with fine-tuning. (Bold numbers are best).

4.4. VGG16 Compression with ImageNet

We compress the VGG16 network trained on ImageNet2012 [8] and compare the results of CaP with other state-of-the-art methods. We present two sets of results, without fine-tuning and with fine-tuning, in Tables 3 and 4 respectively. Fine-tuning on ImageNet is time intensive and requires significant computation power. This is a hindrance for many applications where users do not have enough resources to retrain a compressed network.

In Table 3 we compare CaP with factorization and pruning methods, all without fine-tuning. As expected, factorization methods suffer from increased memory load due to their additional intermediate feature maps. The channel pruning method in [19] has a significant reduction in memory consumption but under-performs the factorization method in [55] without fine-tuning. We present two sets of results for the CaP algorithm, each with different levels of compression for each layer. To match the architecture used in [19] we compressed layers 1-7 to 33% of their original size, and filters in layers 8-10 to 50% of their original size, while the remaining layers are left uncompressed . We also used the CaP method with a compression architecture that was selected based on our layer-wise training experiments. The results in Table 3 demonstrate that the proposed CaP compression achieves higher speedup and higher classification accuracy than the factorization or pruning methods.

In Table 4 we compare CaP with state-of-the-art network compression methods, all with fine-tuning. The uncompressed VGG16 results are from [43]. We include results from training a compressed version of VGG16 from scratch on the ImageNet dataset as reported in [19]. We compare CaP with the results of two factorization methods [34, 30] and two pruning methods [19], [37]. Both factorization methods achieve impressive classification accuracy, but this comes at the cost of increased memory consumption. The pruning methods reduce both the FLOPs and the memory consumption of the network, while maintaining high classification accuracy. However, they rely heavily on fine-tuning to achieve high accuracy. We lastly provide the results of the CaP compression optimized at each layer. Our results demonstrate that the CaP algorithm gives state-of-the-art results, has the largest reduction in memory consumption, and outperforms the pruning methods in terms of top-5 accuracy.

5. Conclusion

In this paper, we propose cascaded projection, an end-to-end trainable framework for network compression that optimizes compression in each layer. Our CaP approach forms linear combinations of kernels in each layer of the network in a manner that both minimizes reconstruction error and maximizes classification accuracy. The CaP method is the first in the field of network compression to optimize the low dimensional projections of the layers of the network using backpropagation and SGD, using our proposed Proxy Matrix Projection optimization method.

We demonstrate state-of-the-art performance compared to pruning and factorization methods, when the CaP method is used to compress standard network architectures trained on standard datasets. A side benefit of the CaP formulation is that it can be performed using standard deep learning frameworks and hardware, and it does not require any specialized libraries for hardware for acceleration. In future work, the CaP method can be combined with other methods, such as quantization and hashing, to further accelerate deep networks.

References

[1] P. A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[2] P. A. Absil and J. Malick. Projection-like retractions on matrix manifolds. *SIAM Journal on Optimization*, 22(1):135–158, 2012.

[3] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[4] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *Proceedings of the International Conference on Machine Learning (ICML) (ICML)*, pages 2285–2294, 2015.

[5] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2857–2865, 2015.

[6] M. Courbariaux, Y. Bengio, and J.-P. David. Training deep neural networks with low precision multiplications. In *Proceedings of the International Conference on Machine Learning (ICML) Workshop*, 2014.

[7] J. P. Cunningham and Z. Ghahramani. Linear dimensionality reduction: survey, insights, and generalizations. *Journal of Machine Learning Research*, 16(1):2859–2900, 2015.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[9] M. Denil, B. Shakibi, L. Dinh, N. De Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2148–2156, 2013.

[10] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1269–1277, 2014.

[11] X. Dong, J. Huang, Y. Yang, and S. Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[12] J. C. Gower, G. B. Dijksterhuis, et al. *Procrustes problems*, volume 30. Oxford University Press on Demand, 2004.

[13] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2015.

[14] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems (NIPS*, pages 164–171, 1993.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, pages 630–645. Springer, 2016.

[17] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[18] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[19] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[20] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[21] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[22] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, page 3, 2017.

[23] L. Huang, X. Liu, B. Lang, A. W. Yu, Y. Wang, and B. Li. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. *arXiv preprint arXiv:1709.06079*, 2017.

[24] Q. Huang, K. Zhou, S. You, and U. Neumann. Learning to prune filters in convolutional neural networks. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 709–718, 2018.

[25] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[26] C. Ionescu, O. Vantzos, and C. Sminchisescu. Matrix Backpropagation for Deep Networks with Structured Layers. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.

[27] C. Ionescu, O. Vantzos, and C. Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *arXiv preprint arXiv:1509.07838*, 2015.

[28] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *arXiv preprint arXiv:1712.05877*, 2017.

[29] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2014.

[30] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast

and low power mobile applications. In *International Conference on Learning Representations (ICLR)*, 2016.

[31] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems (NIPS)*, 25:1097–1105, 2012.

[33] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[34] C. Li and C. Richard Shi. Constrained optimization based low-rank approximation of deep neural networks. In *Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, pages 732–747, 2018.

[35] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2016.

[36] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pages 806–814, 2015.

[37] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[38] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter. Nvidia tensor core programmability, performance & precision. *arXiv preprint arXiv:1803.04014*, 2018.

[39] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[40] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

[41] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[42] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[43] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[44] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2015.

[45] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[46] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE onference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[47] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[48] H. D. Tagare. Notes on optimization on stiefel manifolds. Technical report, Technical report, Yale University, 2011.

[49] Y. Takane and H. Hwang. Regularized linear and kernel redundancy analysis. *Computational Statistics & Data Analysis*, 52(1):394–405, 2007.

[50] Y. Takane and S. Jung. Generalized constrained redundancy analysis. *Behaviormetrika*, 33(2):179–192, 2006.

[51] Z. Wen and W. Yin. A feasible method for optimization with orthogonality constraints. *Mathematical Programming*, 142(1-2):397–434, 2013.

[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 *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5363–5372, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018.

[53] K. Yamamoto and K. Maeno. Pcas: Pruning channels with attention statistics. *arXiv preprint arXiv:1806.05382*, 2018.

[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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[55] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1984–1992, 2015.

[56] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems 31*, pages 875–886. 2018.