NISP: Pruning Networks using Neuron Importance Score Propagation

Ruichi Yu[1,2] Ang Li[3*] Chun-Fu Chen[2] Jui-Hsin Lai[5†] Vlad I. Morariu[4*]
Xintong Han[1] Mingfei Gao[1] Ching-Yung Lin[2] Larry S. Davis[1]
[1]University of Maryland, College Park [2]IBM T. J. Watson Research
[3]DeepMind [4]Adobe Research [5]JD.com
{richyu, morariu, xintong, mgao, lsd}@umiacs.umd.edu anglili@google.com
{chenrich, chingyung}us.ibm.com, larry.lai@jd.com

Abstract

To reduce the significant redundancy in deep Convolutional Neural Networks (CNNs), most existing methods prune neurons by only considering statistics of an individual layer or two consecutive layers (e.g., prune one layer to minimize the reconstruction error of the next layer), ignoring the effect of error propagation in deep networks. In contrast, we argue that it is essential to prune neurons in the entire neuron network jointly based on a unified goal: minimizing the reconstruction error of important responses in the "final response layer" (FRL), which is the second-to-last layer before classification, for a pruned network to retrain its predictive power. Specifically, we apply feature ranking techniques to measure the importance of each neuron in the FRL, and formulate network pruning as a binary integer optimization problem and derive a closed-form solution to it for pruning neurons in earlier layers. Based on our theoretical analysis, we propose the Neuron Importance Score Propagation (NISP) algorithm to propagate the importance scores of final responses to every neuron in the network. The CNN is pruned by removing neurons with least importance, and then fine-tuned to retain its predictive power. NISP is evaluated on several datasets with multiple CNN models and demonstrated to achieve significant acceleration and compression with negligible accuracy loss.

1. Introduction

CNNs require a large number of parameters and high computational cost in both training and testing phases. Recent studies have investigated the significant redundancy in deep networks [6] and reduced the number of neurons and filters [3, 12, 21, 24] by pruning the unimportant ones. However, most current approaches that prune neurons and filters consider only the statistics of one layer (*e.g.*, prune neurons with small magnitude of weights [21, 12]), or two consecutive layers [24] to determine the "importance" of a neuron. These methods prune the "least important" neurons layer-by-layer either independently [12] or greedily [21, 24], without considering all neurons in different layers jointly.

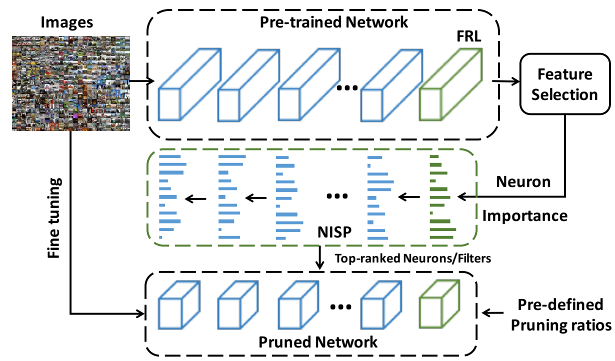


Figure 1. We measure the importance of neurons in the final response layer (FRL), and derive Neuron Importance Score Propagation (NISP) to propagate the importance to the entire network. Given a pre-defined pruning ratio per layer, we prune the neurons/filters with lower importance score. We finally fine-tune the pruned model to recover its predictive accuracy.

One problem with such methods is that neurons deemed unimportant in an early layer can, in fact, contribute significantly to responses of important neurons in later layers. Our experiments (see Sec.4.4) reveal that greedy layer-by-layer pruning leads to significant reconstruction error propagation, especially in deep networks, which indicates the need for a global measurement of neuron importance across different layers of a CNN.

To address this problem, we argue that it is essential for a pruned model to retain the most important responses of the second-to-last layer before classification ("final re-

*This work was done while the author was at University of Maryland.
†This work was done while the author was at IBM

sponse layer" (FRL)) to retrain its predictive power, since those responses are the direct inputs of the classification task (which is also suggested by feature selection methods, *e.g.*, [28]). We define the importance of neurons in early layers based on a **unified goal**: *minimizing the reconstruction errors of the responses produced in FRL.* We first measure the importance of responses in the FRL by treating them as features and applying some feature ranking techniques (*e.g.*, [28]), then propagate the importance of neurons backwards from the FRL to earlier layers. We prune only nodes which have low propagated importance (*i.e.*, those whose removal does not result in large propagated error). From a theoretical perspective, we formulate the network pruning problem as a binary integer programming objective that minimizes the weighted ℓ^1 distance (proportional to the importance scores) between the original final response and the one produced by a pruned network. We obtain a closed-form solution to a relaxed version of this objective to infer the importance score of every neuron in the network. Based on this solution, we derive the *Neuron Importance Score Propagation* (NISP) algorithm, which computes all importance scores recursively, using only one feature ranking of the final response layer and one backward pass through the network as illustrated in Fig. 1.

The network is then pruned based on the inferred neuron importance scores and fine-tuned to retain its predictive capability. We treat the pruning ratio per layer as a pre-defined hyper-parameter, which can be determined based on different needs of specific applications (*e.g.*, FLOPs, memory and accuracy constraints). The pruning algorithm is generic, since feature ranking can be applied to any layer of interest and the importance scores can still be propagated. In addition, NISP is not hardware specific. Given a pretrained model, NISP outputs a smaller network of the same type, which can be deployed on the hardware devices designed for the original model.

We evaluate our approach on MNIST [20], CIFAR10 [18] and ImageNet [5] using multiple standard CNN architectures such as LeNet [20], AlexNet [19], GoogLeNet [31] and ResNet [13]. Our experiments show that CNNs pruned by our approach outperform those with the same structures but which are either trained from scratch or randomly pruned. We demonstrate that our approach outperforms magnitude-based and layer-by-layer pruning. A comparison of the theoretical reduction of FLOPs and number of parameters of different methods shows that our method achieves faster full-network acceleration and compression with lower accuracy loss, *e.g.*, our approach loses 1.43% accuracy on Alexnet and reduces FLOPs by 67.85% while Figurnov *et al.* [11] loses more (2%) and reduces FLOPs less (50%). With almost zero accuracy loss on ResNet-56, we achieve a 43.61% FLOP reduction, significantly higher than the 27.60% reduction by Li *et al.* [21].

1.1. Contribution

We introduce a generic network pruning algorithm which formulates the pruning problem as a binary integer optimization and provide a closed-form solution based on final response importance. We present NISP to efficiently propagate the importance scores from final responses to all other neurons. Experiments demonstrate that NISP leads to full-network acceleration and compression for all types of layers in a CNN with small accuracy loss.

2. Related Work

There has been recent interest in reducing the redundancy of deep CNNs to achieve acceleration and compression. In [6], the redundancy in the parameterization of deep learning models has been studied and demonstrated. Cheng *et al.* [2] exploited properties of structured matrices and used circulant matrices to represent FC layers, reducing storage cost. Han *et al.* [12] studied the weight sparsity and compressed CNNs by combining pruning, quantization, and Huffman coding. Sparsity regularization terms have been use to learn sparse CNN structure in [22, 32, 30]. Miao *et al.* [25] studied network compression based on float data quantization for the purpose of massive model storage.

To accelerate inference in convolution layers, Jaderberg *et al.* [15] constructed a low rank basis of filters that are rank-1 in the spatial domain by exploiting cross-channel or filter redundancy. Liu *et al.* [23] imposed a scaling factor in the training process and facilitated one channel-level pruning. Figurnov *et al.* [11] speeded up the convolutional layers by skipping operations in some spatial positions, which is based on loop perforation from source code optimization. In [7, 34, 17], low-rank approximation methods have been utilized to speed up convolutional layers by decomposing the weight matrix into low-rank matrices. Molchanov *et al.* [26] prune CNNs based on Taylor expansion.

Focusing on compressing the fully connected (FC) layers, Srinivas *et al.* [29] pruned neurons that are similar to each other. Yang *et al.* [33] applied the "Fastfood" transform to reparameterize the matrix-vector multiplication of FC layers. Ciresan *et al.* [3] reduced the parameters by randomly pruning neurons. Chen *et al.* [1] used a low-cost hash function to randomly group connection weights into hash buckets and then fine-tuned the network with back-propagation. Other studies focused on fixed point computation rather than exploiting the CNN redundancy [4, 27]. Another work studied the fundamental idea about transferring knowledge from a large model or an ensemble model to a small one by using the class probabilities produced by the cumbersome model as "soft targets" for training the small model [14]. Our method prunes a pre-trained network and requires a fast-converging fine-tuning process, rather than re-training a network from scratch. To measure the importance of neurons in a CNN, the exact solution is very hard to obtain given the complexity of nonlinearity. Some previous works [8, 9, 10] approximate it using 2nd-order Taylor expansion. Our work is a different approximation based on the Lipschitz continuity of a neural network.

Most similar to our approach, Li *et al.* [21] pruned filters by their weight magnitude. Luo *et al.* [24] utilized statistics information computed from the next layer to guide a greedy layer-by-layer pruning. In contrast, we measure neuron importance based not only on a neuron's individual weight but also the properties of the input data and other neurons in the network. Meanwhile, instead of pruning layer-by-layer in greedy fashion under the assumption that one layer can only affect its next layer, which may cause error propagation, we measure the importance across the entire network by propagating the importance from the final response layer.

3. Our Approach

An overview of NISP is illustrated in Fig. 1. Given a trained CNN, we first apply a feature ranking algorithm on this final response layer and obtain the importance score of each neuron. Then, the proposed NISP algorithm propagates importance scores throughout the network. Finally, the network is pruned based on the importance scores of neurons and fine-tuned to recover its accuracy.

3.1. Feature Ranking on the Final Response Layer

Our intuition is that the final responses of a neural network should play key roles in full network pruning since they are the direct inputs of the classification task. So, in the first step, we apply feature ranking on the final responses.

It is worth noting that our method can work with any feature selection that scores features *w.r.t.* their classification power. We employ the recently introduced filtering method Inf-FS [28] because of its efficiency and effectiveness on CNN feature selection. Inf-FS utilizes properties of the power series of matrices to efficiently compute the importance of a feature with respect to all the other features, *i.e.*, it is able to integrate the importance of a feature over all paths in the affinity graph[1].

3.2. Neuron Importance Score Propagation (NISP)

Our goal is to decide which intermediate neurons to delete, given the importance scores of final responses, so that the predictive power of the network is maximally retained. We formulate this problem as a binary integer programming (optimization) and provide a closed-form approximate solution. Based on our theoretical analysis, we develop the *Neuron Importance Score Propagation* algorithm to efficiently compute the neuron importance for the whole network.

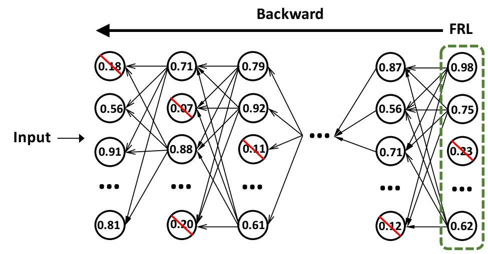


Figure 2. We propagate the neuron importance from the final response layer (FRL) to previous layers, and prune bottom-ranked neurons (with low importance scores shown in each node) given a pre-defined pruning ratio per layer in a single pass. The importance of pruned neurons (with backslash) is not propagated.

3.2.1 Problem Definition

The goal of pruning is to remove neurons while minimizing accuracy loss. Since model accuracy is dependent on the final responses, we define our objective as minimizing the weighted distance between the original final responses and the final responses after neurons are pruned of a specific layer. In following, we use bold symbols to represent vectors and matrices.

Most neural networks can be represented as a nested function. Thus, we define a network with depth n as a function $F^{(n)} = f^{(n)} \circ f^{(n-1)} \circ \cdots \circ f^{(1)}$. The l-th layer $f^{(l)}$ is represented using the following general form,

$$f^{(l)}(\mathbf{x}) = \sigma^{(l)}(\mathbf{w}^{(l)}\mathbf{x} + \mathbf{b}^{(l)}), \tag{1}$$

where $\sigma^{(l)}$ is an activation function and $\mathbf{w}^{(l)}$, $\mathbf{b}^{(l)}$ are weight and bias, and f(n) represents the "final response layer". Networks with branch connections such as the skip connection in ResNet can be transformed to this representation by padding weights and merging layers.

We define the *neuron importance score* as a non-negative value *w.r.t.* a neuron, and use $\mathbf{s}_l$ to represent the vector of neuron importance scores in the l-th layer. Suppose N_l neurons are to be kept in the l-th layer after pruning; we define the *neuron prune indicator* of the l-th layer as a binary vector $\mathbf{s}_l^*$, computed based on neuron importance scores $\mathbf{s}_l$ such that $s_{l,i}^* = 1$ if and only if $s_{l,i}$ is among top N_l values in $\mathbf{s}_l$.

3.2.2 Objective Function

The motivation of our objective is that the difference between the responses produced by the original network and the one produced by the pruned network should be minimized w.r.t. important neurons. Let $F^{(n)}$ be a neural network with n layers. Suppose we have a dataset of M sam-

[1]Details of the method are introduced in [28] and its codes taken from `https://www.mathworks.com/matlabcentral/fileexchange/54763-infinite-feature-selection-2016`.

ples, and each is represented using $\mathbf{x}_0^{(m)}$. For the m-th sample, we use $\mathbf{x}_l^{(m)}$ to represent the response of the l-th layer (which is the input to the $(l+1)$-th layer). The final output of the network is $\mathbf{x}_n^{(m)}$ and its corresponding non-negative neuron importance is $\mathbf{s}_n$. We define

$$G^{(i,j)} = f^{(j)} \circ f^{(j-1)} \circ \cdots \circ f^{(i)} \tag{2}$$

as a sub-network of $F^{(n)}$ starting from the i-th layer to the j-th layer. Our goal is to compute for the l-th layer the neuron prune indicator $\mathbf{s}_l^*$ so that the influence of pruning the l-th layer on the important neurons of the final response is minimized. To accomplish this, we define an optimization objective w.r.t. the l-th layer neuron prune indicator, *i.e.*,

$$\arg\min_{\mathbf{s}_l^*} \sum_{m=1}^{M} \mathcal{F}(\mathbf{s}_l^* | \mathbf{x}_l^{(m)}, \mathbf{s}_n; G^{(l+1,n)}) \,, \tag{3}$$

which is accumulated over all samples in the dataset. The objective function for a single sample is defined as

$$\mathcal{F}(\mathbf{s}_l^* | \mathbf{x}, \mathbf{s}_n; F) = \langle\, \mathbf{s}_n,\ |F(\mathbf{x}) - F(\mathbf{s}_l^* \odot \mathbf{x})|\, \rangle \,, \tag{4}$$

where $\langle \cdot, \cdot \rangle$ is dot product, $\odot$ is element-wise product and $|\cdot|$ is element-wise absolute value. The solution to Eq. 3 indicates which neurons should be pruned in an arbitrary layer.

3.2.3 Solution

The network pruning problem can be formulated as a binary integer program, finding the optimal neuron prune indicator in Eq. 3. However, it is hard to obtain efficient analytical solutions by directly optimizing Eq. 3. So we derive an upper bound on this objective, and show that a sub-optimal solution can be obtained by minimizing the upper bound. Interestingly, we find a feasible and efficient formulation for the importance scores of all neurons based on this sub-optimal solution.

Recall that the k-th layer is defined as $f^{(k)}(\mathbf{x}) = \sigma^{(k)}(\mathbf{w}^{(k)}\mathbf{x} + \mathbf{b}^{(k)})$. We assume the activation function $\sigma^{(k)}$ is Lipschitz continuous since it is generally true for most of the commonly used activations in neural networks such as Identity, ReLU, sigmoid, tanh, PReLU, *etc*. Then we know for any $\mathbf{x}, \mathbf{y}$, there exists a constant $C_\sigma^{(k)}$ such that $|\sigma^{(k)}(\mathbf{x}) - \sigma^{(k)}(\mathbf{y})| \le C_\sigma^{(k)} |\mathbf{x} - \mathbf{y}|$. Then it is easy to see

$$|f^{(k)}(\mathbf{x}) - f^{(k)}(\mathbf{y})| \le C_\sigma^{(k)} |\mathbf{w}^{(k)}| \cdot |\mathbf{x} - \mathbf{y}| \,, \tag{5}$$

where $|\cdot|$ is the element-wise absolute value. From Eq. 2, we see that $G^{(i,j)} = f^{(j)} \circ G^{(i,j-1)}$. Therefore, we have,

$$\begin{aligned} &|G^{(i,j)}(\mathbf{x}) - G^{(i,j)}(\mathbf{y})| \\ &\quad \le C_\sigma^{(j)} |\mathbf{w}^{(j)}| |G^{(i,j-1)}(\mathbf{x}) - G^{(i,j-1)}(\mathbf{y})| \,. \end{aligned} \tag{6}$$

Applying Eq. 5 and Eq. 6 repeatedly, we have, $\forall i \le j \le n$,

$$|G^{(i,n)}(\mathbf{x}) - G^{(i,n)}(\mathbf{y})| \le C_\Sigma^{(i,n)} \mathbf{W}^{(i,n)} |\mathbf{x} - \mathbf{y}|, \tag{7}$$

where $\mathbf{W}^{(i,j)} = |\mathbf{w}^{(j)}||\mathbf{w}^{(j-1)}| \cdots |\mathbf{w}^{(i)}|$, and $C_\Sigma^{(i,j)} = \prod_{k=i}^{j} C_\sigma^{(k)}$. Substituting $\mathbf{x} = \mathbf{x}_l^{(m)}, \mathbf{y} = \mathbf{s}_l^* \odot \mathbf{x}_l^{(m)}, i = l+1$ into Eq. 7, we have

$$\begin{aligned} &|G^{(l+1,n)}(\mathbf{x}_l^{(m)}) - G^{(l+1,n)}(\mathbf{s}_l^* \odot \mathbf{x}_l^{(m)})| \\ &\quad \le C_\Sigma^{(l+1,n)} \mathbf{W}^{(l+1,n)} |\mathbf{x}_l^{(m)} - \mathbf{s}_l^* \odot \mathbf{x}_l^{(m)}| \,. \end{aligned} \tag{8}$$

Since $\mathbf{s}_n$ is a non-negative vector,

$$\begin{aligned} &\mathcal{F}(\mathbf{s}_l^* | \mathbf{x}_l^{(m)}, \mathbf{s}_n; G^{(l+1,n)}) \\ &\quad = \langle \mathbf{s}_n, |G^{(l+1,n)}(\mathbf{x}_l^{(m)}) - G^{(l+1,n)}(\mathbf{s}_l^* \odot \mathbf{x}_l^{(m)})| \rangle \quad (9) \\ &\quad \le \langle \mathbf{s}_n, C_\Sigma^{(l+1,n)} \mathbf{W}^{(l+1,n)} |\mathbf{x}_l^{(m)} - \mathbf{s}_l^* \odot \mathbf{x}_l^{(m)}| \rangle \quad (10) \\ &\quad = C_\Sigma^{(l+1,n)} \langle \mathbf{W}^{(l+1,n)\intercal} \mathbf{s}_n, (\mathbf{1} - \mathbf{s}_l^*) \odot |\mathbf{x}_l^{(m)}| \rangle \,. \quad (11) \end{aligned}$$

Let us define $\mathbf{r}_l = \mathbf{W}^{(l+1,n)\intercal} \mathbf{s}_n$; then

$$\begin{aligned} &\textstyle\sum_{m=1}^{M} \mathcal{F}(\mathbf{s}_l^* | \mathbf{x}_l^{(m)}, \mathbf{s}_n; G^{(l+1,n)}) \\ &\quad \le C_\Sigma^{(l+1,n)} \textstyle\sum_{m=1}^{M} \langle \mathbf{r}_l, (\mathbf{1} - \mathbf{s}_l^*) \odot |\mathbf{x}_l^{(m)}| \rangle \quad (12) \\ &\quad \le C_\Sigma^{(l+1,n)} \textstyle\sum_{m=1}^{M} \sum_i r_{l,i} (1 - s_{l,i}^*) |x_{l,i}^{(m)}| \quad (13) \\ &\quad = C_\Sigma^{(l+1,n)} \textstyle\sum_i r_{l,i} (1 - s_{l,i}^*) \sum_{m=1}^{M} |x_{l,i}^{(m)}| \,. \quad (14) \end{aligned}$$

Since $|\mathbf{x}_{l,i}^{(m)}|$ is bounded, there must exist a constant C_x such that $\sum_{m=1}^{M} |x_{l,i}^{(m)}| \le C_x, \forall i$. Thus, we have

$$\sum_{m=1}^{M} \mathcal{F}(\mathbf{s}_l^* | \mathbf{x}_l^{(m)}, \mathbf{s}_n; F^{(l+1)}) \le C \sum_i r_{l,i} (1 - s_{l,i}^*), \tag{15}$$

where $C = C_\Sigma^{(l+1,n)} C_x$ is a constant factor.

Eq. 15 reveals an upper-bound of our objective in Eq. 3. Thus, we minimize this upper-bound, *i.e.*,

$$\arg\min_{\mathbf{s}_l^*} \sum_i r_{l,i} (1 - s_{l,i}^*) \Leftrightarrow \arg\max_{\mathbf{s}_l^*} \sum_i s_{l,i}^* r_{l,i} \,. \tag{16}$$

The optimal solution to Eq.16 is sub-optimal with respect to the original objective in Eq. 3, however it still captures the importance of neurons. It is easy to see that if we keep N_x neurons in the l-th layer after pruning, then the solution to Eq. 16 is that $s_{l,i}^* = 1$ if and only if $r_{l,i}$ is among the highest N_x values in $\mathbf{r}_l$. According to the definition of neuron prune indicator in Sec. 3.2.1, $\mathbf{r}_l = \mathbf{W}^{(l+1,n)\intercal} \mathbf{s}_n$ is a feasible solution to the importance scores of the l-th layer response. This conclusion can be applied to every layer in the network. Based on this result, we define the neuron importance of a network as follows.

Definition 1 (Neuron importance score). *Given a neural network $F^{(n)}$ containing n layers and the importance score $\mathbf{s}^{(n)}$ of the last layer response, the importance score of the k-th layer response can be computed as*

$$\mathbf{s}_k = |\mathbf{w}^{(k+1)}|^\intercal |\mathbf{w}^{(k+2)}|^\intercal \cdots |\mathbf{w}^{(n)}|^\intercal \mathbf{s}_n, \qquad (17)$$

where $\mathbf{w}^{(i)}$ is the weight matrix of the i-th layer.

An important property of neuron importance is that it can be computed recursively (or propagated) along the network.

Proposition 2 (Neuron importance score propagation). *The importance score of the k^{th} layer response can be propagated from the importance score of the $(k+1)^{th}$ layer by*

$$\mathbf{s}_k = |\mathbf{w}^{(k+1)}|^\intercal \mathbf{s}_{k+1}, \qquad (18)$$

where $\mathbf{w}^{(k+1)}$ is the weight matrix of the $(k+1)^{th}$ layer.

3.2.4 Algorithm

We propose the *Neuron Importance Score Propagation* (NISP) algorithm (shown in Fig. 2) based on Proposition 2. Initially, we have the importance score of every neuron in the final response layer of the network. Definition 1 shows that the importance score of every other layer in the network is directly correlated with the importance of the final response. However, instead of computing the importance expensively using Definition 1, we see from Eq. 18 that the importance score of a lower layer can be propagated directly from the adjacent layer above it. An equivalent form of Eq. 18 is

$$s_{k,j} = \sum_i |w_{i,j}^{(k+1)}| s_{k+1,i}, \qquad (19)$$

where $s_{k,j}$ is the importance score of the j-th neuron in the k-th layer response.

We conclude from Eq. 19 that the importance of a neuron is a weighted sum of all the subsequent neurons that are directly connected to it. This conclusion also applies to normalization, pooling and branch connections in the network (*i.e.*, a layer is directly connected with multiple layers)[2]. The NISP algorithm starts with the importance in FRL and repeats the propagation (Eq. 19) to obtain the importance of all neurons in the network with a single backward pass (Fig. 1).

3.3. Pruning Networks Using NISP

Given target pruning ratios for each layer, we propagate the importance scores, compute the prune indicator of neurons based on their importance scores and remove neurons with prune indicator value 0. The importance propagation and layer pruning happens jointly in a single backward pass, and the importance of a pruned neuron is not propagated to any further low-level layers. For fully connected layers, we prune each individual neuron. For convolution layers, we prune a whole channel of neurons together. The importance score of a channel is computed as the summation of the importance scores of all neurons within this channel[2].

4. Experiments

We evaluate our approach on standard datasets with popular CNN networks. We first compare to *random pruning* and *training-from-scratch* baselines to demonstrate the effectiveness of our method. We then compare to two other baselines, *magnitude-based pruning* and *layer-by-layer pruning* to highlight the contributions of feature ranking and neuron importance score propagation, respectively. Finally, we benchmark the pruning results and compare to existing methods such as [11, 17, 30, 21].

4.1. Experimental Setting

We conduct experiments on three datasets, MNIST [20], CIFAR10 and ImageNet [5], for the image classification task. We evaluate using five commonly used CNN architectures: *LeNet* [20], *Cifar-net*[3], *AlexNet* [19], *GoogLeNet* [31] and *ResNet* [13].

All experiments and time benchmarks are obtained using Caffe [16]. The hyper-parameter of Inf-FS is a loading coefficient $\alpha \in [0, 1]$, which controls the influence of variance and correlation when measuring the importance. We conduct PCA accumulated energy analysis (results shown in the supplementary material) as suggested in [34] to guide our choice of pruning ratios.

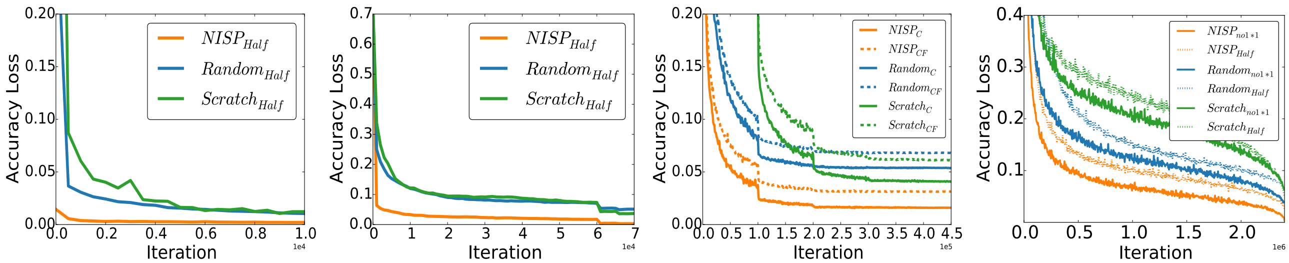
4.2. Comparison with Random Pruning and Train-from-scratch Baselines

We compare to two baselines: (1) randomly pruning the pre-trained CNN and then fine-tuning, and (2) training a small CNN with the same number of neurons/filters per layer as our pruned model from scratch. We use the same experimental settings for our method and baselines except for the initial learning rate. For training from scratch, we set the initial learning rate to the original one, while for fine-tuning tasks (both NISP and random pruning), the initial learning rate is reduced by a factor of 10.

LeNet on MNIST: We prune half of the neurons in FC layers and half of the filters in both convolution layers in Fig. 3(a). Our method is denoted as $NISP_{Half}$, while the baseline methods that prune randomly or train from scratch are denoted as $Random_{Half}$ and $Scratch_{Half}$. Our method outperforms the baselines in three aspects. First, for fine-tuning (after pruning), unlike the baselines, our method has very small accuracy loss at iteration 0; this implies that it retains

[2]See supplementary material for more details and proofs.

[3]https://code.google.com/p/cuda-convnet/.



(a) MNIST (b) CIFAR10 (c) ImageNet: AlexNet (d) ImageNet: GoogLeNet

Figure 3. Learning curves of random pruning and training from scratch baselines and NISP using different CNNs on different datasets. The pruning ratio of neurons and filters is 50%. Networks pruned by NISP (orange curves) converge the fastest with the lowest accuracy loss.

the most important neurons, pruning only redundant or less discriminative ones. Second, our method converges much faster than the baselines. Third, our method has the smallest accuracy loss after fine-tuning. For LeNet on MNIST, our method only decreases 0.02% top-1 accuracy with a pruning ratio of 50% as compared to the pre-pruned network.

Cifar-net on CIFAR10: The learning curves are shown in Fig. 3(b). Similar to the observations from the experiment for LeNet on MNIST, our method outperforms the baselines in the same three aspects: the lowest initial loss of accuracy, the highest convergence speed and the lowest accuracy loss after fine-tuning. Our method has less than 1% top-1 accuracy loss with 50% pruning ratio for each layer.

AlexNet on ImageNet: To demonstrate that our method works on large and deep CNNs, we replicate experiments on AlexNet with a pruning ratio of 50% for all convolution layers and FC layers (denoted as $NISP_{CF}$ when we prune both conv and FC layers). Considering the importance of FC layers in AlexNet, we compare one more scenario in which our approach only prunes half of the filters but without pruning neurons in FC layers (denoted as $NISP_C$). We reduce the initial learning rate by a factor of 10, then fine-tune 90 epochs and report top-5 accuracy loss. Fig. 3(c) shows that for both cases (pruning both convolution and FC layers and pruning only convolution layers), the advantages we observed on MNIST and CIFAR10 still hold. Layer-wise computational reduction analysis that shows the full-network acceleration can be found in supplementary materials.

GoogLeNet on ImageNet: We denote the reduction layers in an inception module as "Reduce", and the 1×1 convolution layer without reduction as "1×1". We use the quick solver from Caffe in training. We conduct experiments between our method and the baselines for 3 pruning strategies: (*Half*) pruning all convolution layers by half; (*noReduce*) pruning every convolution layer except for the reduction layers in inception modules by half; (*no1x1*) pruning every convolution layer by half except the 1×1 layers in inception modules. We show results for two of them in Fig. 3(d), and observe similar patterns to the experiments on other CNN networks[4]. For all GoogLeNet experiments, we train/fine-tune for 60 epochs and report top-5 accuracy loss.

4.3. Feature Selection v.s. Magnitude of Weights

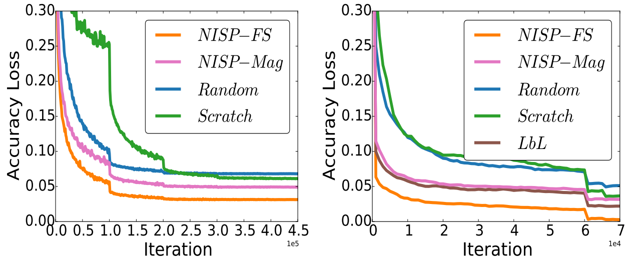
How to define neuron importance is an open problem. Besides using feature ranking to measure neuron importance, other methods [21, 24, 12] measure neuron importance by magnitude of weights. To study the effects of different criteria to determine neuron importance, we conduct experiments by fixing other parts of NISP and only comparing the pruning results with different measurements of importance: 1. using feature selection method in [28] (NISP-FS) and 2. considering only magnitude of weights (NISP-Mag). For the Magnitude-based pruning, the importance of a neuron in the final response layer equals the absolute sum of all weights connecting the neuron with its previous layer. To compare only the two metrics of importance, we rank the importance of neurons in the final response layer based on the magnitude of their weight values, and propagate their importance to the lower layers. Finally, we prune and fine-tune the model in the same way as the NISP method.

For the "NISP-Mag" baseline, we use both AlexNet and Cifar-net architectures. The learning curves of those baselines are shown in Fig. 4. We observe that "NISP-FS" yields much smaller accuracy loss with the same pruning ratio than "NISP-Mag", but "NISP-Mag" still outperforms the random pruning and train-from-scratch baselines, which shows the effectiveness of NISP with different measurement of importance. In the remainder of this paper, we employ the feature ranking method proposed in [28] in NISP.

4.4. NISP v.s. Layer-by-Layer Pruning

To demonstrate the advantage of the NISP's importance propagation, we compare with a pruning method that conducts feature ranking on every layer to measure the neuron importance and prune the unimportant neurons of each layer independently. All other settings are the same as NISP. We call this method "Layer-by-Layer" (LbL) pruning.

[4]See supplementary materials for the results of *noReduce*.



(a) AlexNet on ImageNet (b) Cifar-net on CIFAR10

Figure 4. Comparison with layer-by-layer (LbL) and magnitude based (Mag) pruning baselines. We prune 50% of neurons and filters in all layers for both CNNs. NISP-FS outperforms NISP-Mag and LbL in terms of prediction accuracy.

One challenge for the "LbL" baseline is that the computational cost of measuring neuron importance on each layer is huge. So we choose a small CNN structure trained on the CIFAR10 dataset. Fig. 4(b) shows that although the "LbL" method outperforms the baselines, it performs much worse than NISP in terms of the final accuracy loss with the same pruning ratio, which shows the need for measuring the neuron importance across the entire network using NISP.

To further study the advantage of NISP over layer-by-layer pruning, we define the Weighted Average Reconstruction Error (WARE) to measure the change of the important neurons' responses on the final response layer after pruning (without fine-tuning) as:

$$\text{WARE} = \frac{\sum_{m=1}^{M}\sum_{i=1}^{N} s_i \cdot \frac{|\hat{y}_{i,m} - y_{i,m}|}{|y_{i,m}|}}{M \cdot N}, \tag{20}$$

where M and N are the number of samples and number of retained neurons in the final response layer; s_i is the importance score; $y_{i,m}$ and $\hat{y}_{i,m}$ is the response on the m^{th} sample of the i^{th} neuron before/after pruning.

We design different Cifar-net-like CNNs with different numbers of Conv layers, and apply NISP and LbL pruning with different pruning ratios. We report the WARE on the retained neurons in the final response layer ("ip1" layer in Cifar-net-like CNNs) in Fig. 5. We observe that: 1. As network depth increases, the WARE of the LbL-pruned network dramatically increases, which indicates the error propagation problem of layer-by-layer pruning, especially when the network is deep, and suggests the need for a global pruning method such as NISP; 2. The WARE of the LbL method becomes much larger when the pruning ratio is large, but is more stable when using NISP to prune a network; 3. NISP methods always reduce WARE on the retained neurons compared to LbL. The small reconstruction errors on the important neurons in the final response layer obtained by NISP provides a better initialization for fine-tuning, which leads to much lower accuracy loss of the pruned network.

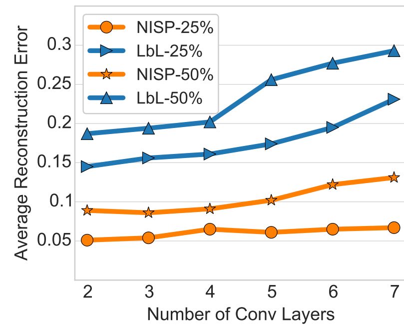


Figure 5. Weighted Average Reconstruction Error (WARE) on the final responses without fine-tuning: we set pruning ratios as 25% and 50% and evaluate the WARE on the final responses of models with different depths pruned using NISP or LbL. It is clear that networks pruned by NISP have the lowest reconstruction errors.

	Model	Accu.↓%	FLOPs↓%	Params.↓%
AlexNet on ImageNet	NISP-A	**1.43**	**67.85**	33.77
	Perforated [11]	2.00	50.00	-
	NISP-B	**0.97**	**62.69**	1.96
	Tucker [17]	1.70	62.55	-
	NISP-C	**0.54**	**53.70**	2.91
	Learning [30]	1.20	48.19	-
	NISP-D	0.00	40.12	47.09
GoogLeNet on ImageNet	NISP	**0.21**	**58.34**	**33.76**
	Tucker [17]	0.24	51.50	31.88
ResNet on CIFAR10	NISP-56	0.03	**43.61**	**42.60**
	56-A [21]	**-0.06**[5]	10.40	9.40
	56-B [21]	-0.02	27.60	13.70
	NISP-110	0.18	**43.78**	**43.25**
	110-A [21]	**0.02**	15.90	2.30
	110-B [21]	0.23	38.60	32.40
ResNet on ImageNet	NISP-34-A	**0.28**	27.32	27.14
	NISP-34-B	0.92	**43.76**	**43.68**
	Res34 [21]	1.06	24.20	-
	NISP-50-A	**0.21**	27.31	27.12
	NISP-50-B	0.89	**44.01**	**43.82**
	Res50 [24]	0.84	36.79	33.67

Table 1. Compression Benchmark. [Accu.↓%] denotes the absolute accuracy loss; [FLOPs↓%] denotes the reduction of computations; [Params.↓%] demotes the reduction of parameter numbers;

4.5. Comparison with Existing Methods

We compare our method with existing pruning methods on AlexNet, GoogLeNet and ResNet, and show results in Table 1.

We show benchmarks of several pruning strategies in Table 1, and provide additional results in the supplementary materials. In Table 1, for AlexNet, the pruning ratio is 50%. NISP-A denotes pruning all Conv layers; NISP-B

[5] A negative value here indicates an improved model accuracy.

denotes pruning all Conv layers except for Conv5; NISP-C denotes pruning all Conv layers except for Conv5 and Conv4; NISP-D means pruning Conv2, Conv3 and FC6 layers. For GoogLeNet, we use the similar the pruning ratios of the 3×3 layers in [17], and we prune 20% of the reduce layers. Our method is denoted as "NISP".

To compare theoretical speedup, we report reduction in the number of multiplication and the number of parameters following [17] and [11], and denote them as [FLOPs↓%] and [Params.↓%] in the table. Pruning a CNN is a trade-off between efficiency and accuracy. We compare different methods by fixing one metric and comparing the other.

On AlexNet, by achieving smaller accuracy loss (1.43% ours vs. 2.00% [11]), our method NISP-A manages to reduce significantly more FLOPs (67.85%) than the one in [11] (50%), denoted as "Perforate" in the table; compare to the method in [30] (denoted as "Learning"), our method NISP-C achieves much smaller accuracy loss (0.54% ours vs. 1.20%) and prunes more FLOPs (53.70% ours vs. 48.19%). We manage to achieve 0 accuracy loss and reduce over 40% FLOPs and 47.09% parameters (NISP-D). On GoogLeNet, Our method achieves similar accuracy loss with larger FLOPs reduction (58.34% vs. 51.50%) Using ResNet on Cifar10 dataset, with top-1 accuracy loss similar to [21] (56-A, 56-B. 110-A and 110-B), our method reduces more FLOPs and parameters.

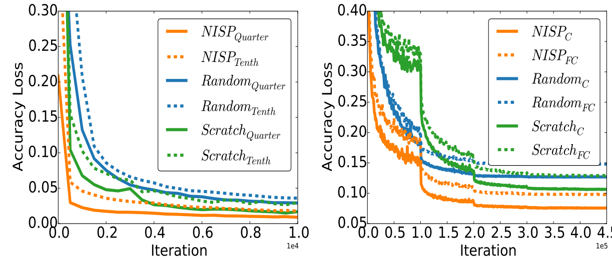
We also conduct our ResNet experiments on ImageNet [5]. We train a ResNet-34 and a ResNet-50 for 90 epochs. For both ResNet models, we prune 15% and 25% of filters for each layer (denote as "NISP-X-A" and "NISP-X-B" ("X" indicates the ResNet model) in Table 1), and obtain 27-44% FLOPs and parameter reduction with tiny top-1 accuracy loss, which shows superior performance when compared with the state-of-the-art methods [21, 24].

4.6. Additional Analysis

Below, we provide case studies and ablation analysis to help understand the proposed NISP pruning algorithm.

Similar Predictive Power of Networks Before/After Pruning. To check whether the pruned network performs similarly with the original network, we compare the final classification results of the original AlexNet and the pruned one with fine-tuning using the ILSVRC2012 validation set. 85.9% of the top 1 predictions of the two networks agree with each other, and 95.1% top 1 predictions of the pruned network can be found in the top 5 predictions of the original network. The above experiments show that the network pruned by NISP performs similarly with the original one.

Sensitivity of pruning ratios. The selection of per-layer pruning ratios given a FLOPs budget is a challenging open problem with a large search space. Due to time limitation, we either choose a single pruning ratio for all layers or replicate the pruning ratios of baseline methods (*e.g.*, [17]), and



(a) LeNet Prune 75% and 90%   (b) AlexNet Prune 75%

Figure 6. Evaluations for different pruning ratios (a) LeNet: pruning 75% and 90%, (b) AlexNet: pruning 75%. CNNs pruned by NISP converge fastest with the lowest accuracy loss.

NISP achieves smaller accuracy loss, which shows the effectiveness of NISP. In practice, if time and GPU resources permit, one can search the optimal hyper-parameters by trying different pruning ratio combinations on a validation set.

We also evaluate NISP with very large pruning ratios. We test on pruning ratios of 75% (denoted as *Quarter* in the figures) and 90% using LeNet (Fig. 6(a)) (denoted as *Tenth*) for both Conv and FC layers. For AlexNet (Fig. 6(b)), we test on pruning ratios of 75% (*Quarter*) for both convolution and FC layers, and we test two pruning strategies: (1) prune 75% of neurons in FC layers and filters in Conv layers, denoted as *FC*; and (2) only prune 75% of the convolution filters without pruning FC layers, denoted as *C*.

The above experiments show that NISP still outperforms all baselines significantly with large pruning ratios, in terms of both convergence speed and final accuracy.

5. Conclusion

We proposed a generic framework for network compression and acceleration based on identifying the importance levels of neurons. Neuron importance scores in the layer of interest (usually the last layer before classification) are obtained by feature ranking. We formulated the network pruning problem as a binary integer program and obtained a closed-form solution to a relaxed version of the formulation. We presented the Neuron Importance Score Propagation algorithm that efficiently propagates the importance to every neuron in the whole network. The network is pruned by removing less important neurons and fine-tuned to retain its predicative capability. Experiments demonstrated that our method effectively reduces CNN redundancy and achieves full-network acceleration and compression.

Acknowledgement

The research was partially supported by the Office of Naval Research under Grant N000141612713: Visual Common Sense Reasoning for Multi-agent Activity Prediction and Recognition.

References

[1] W. Chen, J. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick". In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2285–2294, 2015.

[2] 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 *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2857–2865, Dec 2015.

[3] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, IJCAI'11, pages 1237–1242, 2011.

[4] M. Courbariaux, Y. Bengio, and J. David. Training deep neural networks with low precision multiplications. In *ICLR Workshop*, 2015.

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

[6] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. D. Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems 26 (NIPS)*, pages 2148–2156. Curran Associates, Inc., 2013.

[7] 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 27 (NIPS)*, pages 1269–1277. 2014.

[8] B. H. et al. Second order derivatives for network pruning: Optimal brain surgeon. In *NIPS*. 1993.

[9] P. M. et al. Pruning convolutional neural networks for resource efficient transfer learning. *CoRR*, abs/1611.06440, 2016.

[10] Y. L. C. et al. Optimal brain damage. In *NIPS*, 1990.

[11] M. Figurnov, A. Ibraimova, D. P. Vetrov, and P. Kohli. Perforatedcnns: Acceleration through elimination of redundant convolutions. In *Advances in Neural Information Processing Systems 29 (NIPS)*, pages 947–955. 2016.

[12] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[14] J. D. G. Hinton and O. Vinyals. Distilling the knowledge in a neural network. In *NIPS 2014 Deep Learning Workshop*, 2014.

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

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM International Conference on Multimedia*, MM'14, pages 675–678, New York, NY, USA, 2014. ACM.

[17] Y. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shi. Compression of deep convolutional neural networks for fast and low power mobile applications. In *International Conference on Learning Representations (ICLR)*, 2016.

[18] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

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

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent signal processing*, pages 306–351. IEEE Press, 2001.

[21] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2017.

[22] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy. Sparse convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 806–814, June 2015.

[23] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

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

[25] H. Miao, A. Li, L. S. Davis, and A. Deshpande. Towards unified data and lifecycle management for deep

learning. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 571–582, April 2017.

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

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

[28] G. Roffo, S. Melzi, and M. Cristani. Infinite feature selection. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4202–4210, 2015.

[29] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 31.1–31.12. BMVA Press, 2015.

[30] S. Srinivas and R. V. Babu. Learning the architecture of deep neural networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 104.1–104.11, September 2016.

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[32] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems 29 (NIPS)*, pages 2074–2082. 2016.

[33] Z. Yang, M. Moczulski, M. Denil, N. d. Freitas, A. Smola, L. Song, and Z. Wang. Deep fried convnets. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1476–1483, Dec 2015.

[34] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.