Supplementary Materials of NISP: Pruning Networks using Neuron Importance Score Propagation

Ruichi Yu[1] Ang Li[3*] Chun-Fu Chen[2] Jui-Hsin Lai[5†] Vlad I. Morariu[4*]
Xintong Han[1] Mingfei Gao[1] Ching-Yung Lin[6†] Larry S. Davis[1]
[1]University of Maryland, College Park [2]IBM T. J. Watson Research
[3]DeepMind [4]Adobe Research [5]JD.com [6]Graphen.ai
{richyu, xintong, mgao, lsd}@umiacs.umd.edu, anglili@google.com
chenrich@us.ibm.com, larry.lai@jd.com, morariu@adobe.com, cylin@graphen.ai

Despite their impressive predictive power on a wide range of tasks [6, 3, 5, 8], the redundancy in the parameterization of deep learning models has been studied and demonstrated [2]. We present NISP to efficiently propagate the importance scores from final responses to all other neurons to guide network pruning to achieve acceleration and compression of a deep network. In the supplementary materials, we show details on how to propagate neuron importance from the final response layer, and some additional experiments.

1. Neuron Importance Score Propagation (NISP)

Given the importance of a neuron, we first identify the positions in the previous layer that are used as its input, then propagate the importance to the positions proportional to the weights. We only propagate the importance of the selected feature extractors to the previous layers and ignore the pruned ones. The NISP process can be divided into three classes: from a 1-way tensor to a 1-way tensor, e.g. between FC layers; from a 1-way tensor to a 3-way tensor, e.g., from an FC layer to a conv/pooling layer; from a 3-way tensor to a 3-way tensor, e.g., from a pooling layer to a conv layer.

We simplify NISP by ignoring the propagation of bias.

2. NISP: from 1-way tensor to 1-way tensor

Given an FC layer with M input neurons and N output neurons, the N-*by*-1 importance vector ($\mathbf{S}$) of the output feature is $\mathbf{S_{FC_{out}}} = [S_{FC_{out}1}, S_{FC_{out}2} \dots S_{FC_{out}N}]^{\mathrm{T}}$. We use $\mathbf{W_{FC}} \in \mathbb{R}^{M \times N}$ to denote the weights of the FC layer. The importance vector of the input neurons is:

$$\mathbf{S_{FC_{in}}} = |\mathbf{W_{FC}}| \cdot \mathbf{S_{FC_{out}}} \ , \tag{1}$$

where $|\cdot|$ is element-wise absolute value.

3. NISP: from 1-way tensor to 3-way tensor

Given an FC layer with a 3-way tensor as input and N output neurons, the input has a size of $X \times X \times C$, where X is the spatial size and C is the number of input channels. The input can be the response of a convolutional layer or a pooling layer. We use $\mathbf{W_{FC}} \in \mathbb{R}^{(X^2 \times C) \times N}$ to denote the weights of the FC layer. The flattened importance vector $\mathbf{S_{in}} \in \mathbb{R}^{(X^2 \times C) \times 1}$ of the input tensor is:

$$\mathbf{S_{in}} = |\mathbf{W_{FC}}| \cdot \mathbf{S_{FC_{out}}}. \tag{2}$$

4. NISP: from 3-way tensor to 3-way tensor

4.0.1 Convolution Layer.

We derive NISP for a convolutional layer, which is the most complicated case of NISP between 3-way tensors. NISP for pooling and local response normalization (LRN) can be derived similarly.

For a convolutional layer with the input 3-way tensor $\mathbf{conv_{in}} \in \mathbb{R}^{X \times X \times N}$ and output tensor $\mathbf{conv_{out}} \in \mathbb{R}^{Y \times Y \times F)}$, the filter size is k, stride is s and the number of padded pixels is p. During the forward propagation, convolution consists of multiple inner products between a kernel $\mathbf{k}_f \in \mathbb{R}^{k \times k \times N}$, and multiple corresponding receptive cubes to produce an output response. Fixing input channel n and output channel f, the spatial convolutional kernel is $\mathbf{k}_{fn}$. For position i in the nth channel of the input tensor, the corresponding response of the output channel f at position i is defined as Equation 3:

$$R_f(i) = \sum_n \mathbf{k}_{fn} \cdot \mathbf{in}(i), \tag{3}$$

where $\mathbf{in}(i)$ is the corresponding 2-D receptive field. Given the importance cube of the output response $\mathbf{S_{out}} \in$

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

†This work was done while the author was at IBM.

Algorithm 1 NISP: convolutional layer

1: **Input** : weights of the conv layer $\mathbf{W} \in \mathbb{R}^{X \times X \times N \times F}$
2: , flattened importance of the fth output channel
3: $\mathbf{S}_{out}^{f} \in \mathbb{R}^{1 \times (X \times X)}$
4: **for** n in 1 . . . N **do**
5: **for** f in 1 . . . F **do**
6: $\mathbf{k}_{fn} \leftarrow |\mathbf{W}[:,:,n,f]|$
7: Construct $\mathbf{BP}_{conv}^{fn}$ as (5) and (6)
8: $\mathbf{S}_{in}^{fn} \leftarrow \mathbf{S}_{out}^{f} \cdot \mathbf{BP}_{conv}^{fn}$
9: $\mathbf{S}_{in}^{n} \leftarrow \sum_f \mathbf{S}_{in}^{fn}$
10: $\mathbf{S}_{in} \leftarrow [\mathbf{S}_{in}^{1}, \mathbf{S}_{in}^{2} \ldots, \mathbf{S}_{in}^{N}]$
11: end

$\mathbb{R}^{Y \times Y \times F}$, we use a similar linear computation to propagate the importance from the output response to the input:

$$S_n(i) = \sum_f \mathbf{k}_{fn} \cdot \mathbf{S}_{out}(i), \tag{4}$$

where $S_n(i)$ is the importance of position i in the nth input channel, and $\mathbf{S}_{out}(i)$ is the corresponding 2-D matrix that contains the output positions whose responses come from the value of that input position during forward propagation. We propagate the importance proportionally to the weights as described in Algorithm 1.

The propagation matrices used in algorithm 1 are defined in (5) and (6)

$$\mathbf{BP}_{conv}^{fn} = \begin{bmatrix} \mathbf{b}_1^{fn} \ldots & \mathbf{b}_j^{fn} & \ldots \mathbf{b}_k^{fn} & & & \\ & \mathbf{b}_1^{fn} \ldots & \mathbf{b}_j^{fn} & \ldots \mathbf{b}_k^{fn} & & \\ & & \vdots & & & \\ & & & \mathbf{b}_1^{fn} \ldots & \mathbf{b}_j^{fn} & \ldots \mathbf{b}_k^{fn} \end{bmatrix}, \tag{5}$$

where $\mathbf{b}_c^i$ is the building block of size $Y \times X$ defined as:

$$\mathbf{b}_i^{fn} = \begin{bmatrix} \mathbf{k}_{fn}[i,1] \ldots & \ldots \mathbf{k}_{fn}[i,k] & & \\ & \mathbf{k}_{fn}[i,1] \ldots & \ldots \mathbf{k}_{fn}[i,k] & \\ & & \vdots & \\ & & \mathbf{k}_{fn}[i,1] \ldots & \ldots \mathbf{k}_{fn}[i,k] \end{bmatrix}, \tag{6}$$

Equation 4 implies that the propagation of importance between 3-way tensors in convolutional layers can be decomposed into propagation between 2-D matrices. Fixing the input channel n and the output channel f, the input layer size is $X \times X$ and the output size is $Y \times Y$. Given the flattened importance vector $\mathbf{S_{out}}^f \in \mathbb{R}^{1 \times (Y \times Y)}$ of the output layer, the propagation matrix $\mathbf{BP}_{conv}^{fn} \in \mathbb{R}^{(Y \times Y) \times (X \times X)}$ is used to map from $\mathbf{S_{out}}^f$ to the importance of input layer $\mathbf{S_{in}}^{fn} \in \mathbb{R}^{1 \times (X \times X)}$. $\mathbf{BP}_{conv}^{fn}(i,j) \neq 0$, implies that the ith position in the output layer comes from a convolution operation with the jth position in the input layer, and we propagate the importance between the two positions. We use a $Y \times X$ matrix $\mathbf{b}_i^{fn}$ to represent the mapping between a row in the output layer to the corresponding row in the input layer. In each row of $\mathbf{b}_i^{fn}$, there are k non-zeros since each position in the output layer is obtained from a region with width k of the input layer. The non-zeros of each row of $\mathbf{b}_i^{fn}$ are the ith row of the convolutional kernel $\mathbf{k}_{fn}$. The offset of the beginning of the weights in each row is the stride s. The entire propagation matrix $\mathbf{BP}_{conv}^{fn}$ is a block matrix with each submatrix being a $Y \times X$ matrix of either $\mathbf{b}_i^{fn}$ or a zero matrix. Each row of $\mathbf{BP}_{conv}^{fn}$ has $\mathbf{b}_1^{fn}$ to $\mathbf{b}_k^{fn}$ because the height of a convolutional kernel is k. The offset of the beginning of the $\mathbf{b}$s in each row of $\mathbf{BP}_{conv}^{fn}$ is the stride s. We use the case when $X = 4, Y = 2, k = 3, s = 1$ as an example shown in Figure 1.

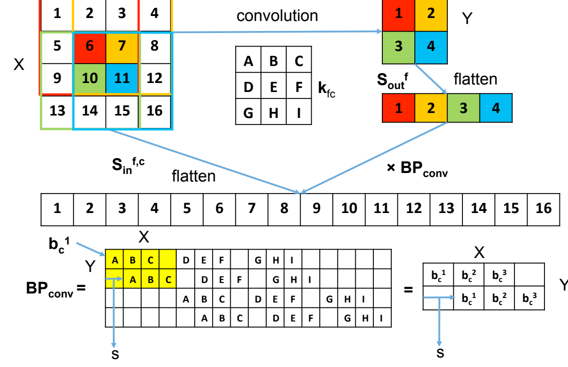


Figure 1. importance propagation: Convolutional layer. $X = 4, Y = 2, k = 3, s = 1$. Fixing the fth input channel and c^{th} output channel, the upper-left X-by-X grid is the corresponding input feature map, and the upper-right Y-by-Y grid is the output map after convolution is applied. $\mathbf{k}_{fc}$ is the corresponding 2D convolutional kernel. Given the flattened importance vector for the output feature map $\mathbf{S_{out}}^{f,c}$, we use $\mathbf{BP}_{conv}$ to propagate the importance and obtain $\mathbf{S_{in}}^{f,c}$, which contains the importance of the input feature map. The structure of $\mathbf{BP}_{conv}$ is determined by the kernel size k and stride s.

4.0.2 Pooling Layer.

Assume a pooling layer with input tensor of size $X \times X \times F$ and output size $Y \times Y \times F$. The pooling filter size is k and the stride is s. The basic idea of most pooling techniques is the same: use a fixed 2-dimensional filter to abstract local responses within each channel independently. For example, in max pooling each output response consists of the max of $k \times k$ values from the input responses. Due to the large variance of input data, it is safe to assume a uniform distribution on which value within the receptive field is the largest is a uniform distribution. Consequently, for an output response

location, the contributions from the corresponding $k \times k$ values of the input response are equal. Since pooling is a spatial operation that does not cross channels, we can propagate the importance of each channel independently. Given a flattened importance vector of a channel f $\mathbf{S_{out}}^{\mathbf{f}} \in \mathbb{R}^{1\times(Y\times Y)}$ of the output 3-way tensor, the flattened importance vector of the input tensor is calculated as:

$$\mathbf{S_{in}}^{\mathbf{f}} = \mathbf{S_{out}}^{\mathbf{f}} \cdot \mathbf{BP}_{pooling}, \quad (7)$$

where $\mathbf{BP}_{pooling}$ is the back-propagation matrix of size $Y^2 \times X^2$ defined as:

$$\mathbf{BP}_{pooling} = \begin{bmatrix} \mathbf{b}_p \ldots & \mathbf{b}_p & \ldots \mathbf{b}_p & & \\ & \mathbf{b}_p \ldots & \mathbf{b}_p & \ldots \mathbf{b}_p & \\ & & \vdots & & \\ & & \mathbf{b}_p \ldots & \mathbf{b}_p & \ldots \mathbf{b}_p \end{bmatrix}, \quad (8)$$

where $\mathbf{b}_p$ is the building block of size $Y \times X$ defined as:

$$\mathbf{b}_p = \begin{bmatrix} 1 \ldots & 1 & \ldots 1 & & \\ & 1 \ldots & 1 & \ldots 1 & \\ & & \vdots & & \\ & & 1 \ldots & 1 & \ldots 1 \end{bmatrix}, \quad (9)$$

Consider one channel with input size $X \times X$ and the output size $Y \times Y$. Given the flattened importance vector $\mathbf{S_{out}}^{f} \in \mathbb{R}^{1\times(Y\times Y)}$ of the output layer, the propagation matrix $\mathbf{BP}_{pooling} \in \mathbb{R}^{(Y\times Y)\times(X\times X)}$ is used to map from $\mathbf{S_{out}}^{f}$ to the importance of input layer $\mathbf{S_{in}}^{f} \in \mathbb{R}^{1\times(X\times X)}$. If $\mathbf{BP}_{pooling}(i,j) = 1$, the ith position in the output layer comes from a pooling operation and involves the jth position in the input layer, so we propagate the importance between the two positions. We use a $Y \times X$ matrix $\mathbf{b}_p$ to represent the mapping between a row in the output layer to the corresponding row in the input layer. In each row of $\mathbf{b}_p$, there are k $1's$ since each element in the output layer is pooled from a region with width k of the input layer. The offset of the beginning of the $1's$ is the stride s. The entire propagation matrix $\mathbf{BP}_{pooling}$ is a block matrix with each submatrix being a $Y \times X$ matrix of either $\mathbf{b}_p$ or a zero matrix. Each row of $\mathbf{BP}_{pooling}$ has k $\mathbf{b}_p s$ because the height of pooling filter is k. The offset of the beginning of the k $\mathbf{b}_p$s is the stride s. The ones in $\mathbf{b}_p$ will be normalized by the number of positions covered by a pooling filter (the same for LRN layers shown below). The other elements are all zeros. We use the case that $X = 4, Y = 2, k = 2, s = 2$ as an example shown in Figure 2.

4.0.3 Local Response Normalization Layer.

Krizhevsky *et al.* [4] proposed Local Response Normalization (LRN) to improve CNN generalization. For cross-channel LRN, sums over adjacent kernel maps at the same spatial position produce a response-normalized activation at that position. Since LRN is a non-linear operation, it is intractable to conduct exact importance propagation between the input and output tensors. One way to approximate propagation is to assume the kernel maps at one spatial position contribute equally to the response at that position of the output tensor when considering the large variance of the input data. Then, given the $X \times X \times N$ importance tensor for the response of a LRN layer with local_size $= l$, which is the number of adjacent kernel maps summed for a spatial position, considering all N channels of a spatial position (i,j), the importance vector of that spatial position is $\mathbf{S}_{out}^{ij} \in \mathbb{R}^{1\times N}$. The corresponding importance vector of the input $\mathbf{S}_{in}^{ij} \in \mathbb{R}^{1\times N}$ is:

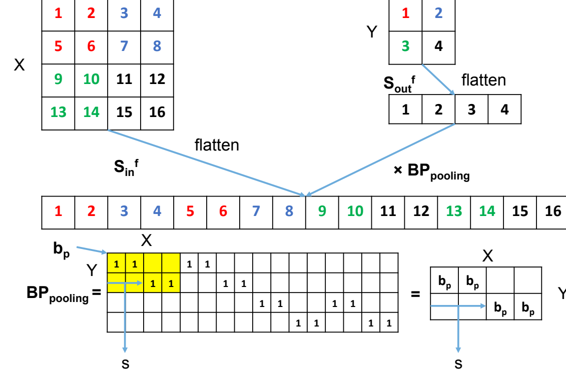


Figure 2. NISP: Pooling layer. $X = 4, Y = 2, k = 2, s = 2$. The upper-left X-by-X grid is the fth feature map of the input channel, and the upper-right Y-by-Y grid is the output channel after pooling is applied. Given the importance vector $\mathbf{S_{out}}^{f}$, we use $\mathbf{BP}_{pooling}$ to propagate the importance and obtain $\mathbf{S_{in}}^{f}$, which contains the importance of each position of the input feature map. The structure of $\mathbf{BP}_{pooling}$ relates to the kernel size k and stride s.

$$\mathbf{S}_{in}^{ij} = \mathbf{S}_{out}^{ij} \cdot \mathbf{BP}_{LRN}, \quad (10)$$

where $\mathbf{BP}_{LRN} \in \mathbb{R}^{N\times N}$ is defined as:

$$\mathbf{BP}_{LRN} = \begin{bmatrix} 1\,1 \cdots 1 & & & & \\ 1\,1 \cdots 1\,1 & & & & \\ \vdots & & & & \\ 1\,1 \cdots 1\,1 \cdots & & & & \\ \;\;1 \cdots 1\,1 \cdots & 1 & & & \\ \;\;\;\;\cdots & 1 & 1 & & \\ \;\;\;\;1\,1 \cdots & \cdots & \cdots & & \\ \;\;\;\;\;\;1 \cdots & 1 & 1 & \cdots 1 & \\ & 1 & 1 & \cdots 1\,1 & \\ & \vdots & \vdots & \vdots & \\ & 1 & 1 & \cdots 1\,1 & \\ & & 1 & \cdots 1\,1 & \end{bmatrix}. \quad (11)$$

For a cross-channel LRN, the output response tensor has the same shape as the input. For a spatial position (i,j) of the output tensor, given its importance vector $\mathbf{S}_{out}^{ij}$, we construct a $N \times N$ symmetric matrix $\mathbf{BP}_{LRN}$ to propagate its importance to the corresponding input vector$\mathbf{S}_{in}^{ij}$ at that

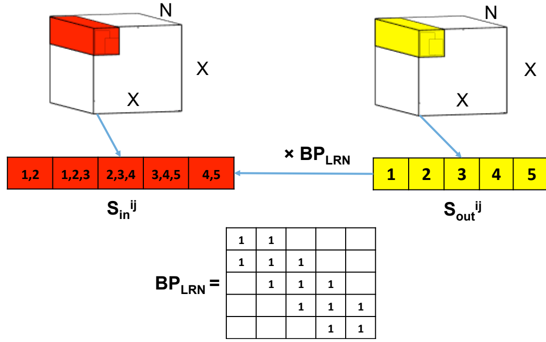


Figure 3. Importance propagation: LRN layer (cross-channel). $l = 3, N = 5$. The red vector is the cross-channel vector at spatial position (i, j) of the input tensor, and the yellow vector is the cross-channel vector at the same position of the output tensor after LRN is applied. Given the $\mathbf{S_{out}}^{ij}$, we use $\mathbf{BP}_{LRN}$ to propagate the importance and obtain $\mathbf{S_{in}}^{ij}$, which contains the importance of each position of the input feature map. The structure of $\mathbf{BP}_{LRN}$ relates to the local size l and number of channels N.

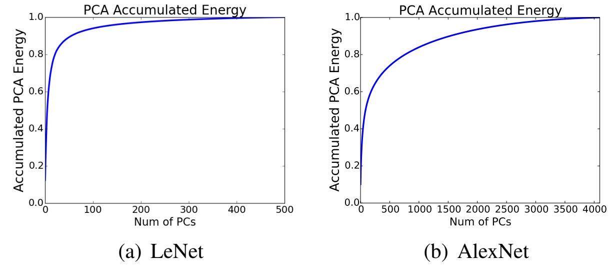
position. Since the center of the LRN operation is at position (i, j), the operation will cover $\frac{l+1}{2}$ positions to the left and right. When the operation is conducted on the positions at the center of the vector $\mathbf{S}_{out}^{ij}$ (from column $\frac{l+1}{2}$ to N-$\frac{l+1}{2}+1$), the operation covers l cross-channel position so that the corresponding columns in $\mathbf{BP}_{LRN}$ have l 1's. When the LRN operation is conducted at the margin of the vector, there are missing cross-channel positions so that from column $\frac{l+1}{2}$ to column 1 (similar for the right-bottom corner), the 1's in the corresponding column of $\mathbf{BP}_{LRN}$ decreases by 1 per step from the center to the margin. We use the case when $l = 3, N = 5$ as an example of LRN layer with cross-channel in Figure 3.

For within-channel LRN, following our equal distribution assumption, the importance can be propagated similarly as in a pooling layer.

5. Experiments

6. PCA Accumulated Energy Analysis

One way to guide the selection of pruning ratio is the PCA accumulated energy analysis [9] on the responses of a pre-pruned layer. The PCA accumulated energy analysis shows how many PCs it needs for that layer to capture the majority of variance of the samples, which implies a proper range of how many neurons/kernels we should keep for that layer. We show the PCA accumulated energy analysis results on the last FC layers before the classification part for LeNet (ip1) and AlexNet (fc7) in Figure 4(a) and 4(b). By setting variance threshold as 0.95, 120 out of 500 PCs are required for LeNet, 2234 out of 4096 PCS are required for AlexNet to capture the variance.



(a) LeNet (b) AlexNet

Figure 4. PCA accumulated energy analysis: LeNet on MNIST (a) and AlexNet on ImageNet (b). The y axis measures the PCA accumulated energy. The x axis shows the number of PCs.

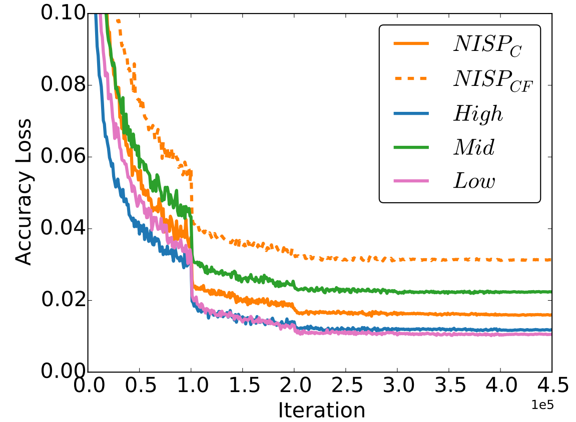


Figure 5. Learning Curves of AlexNet on ImageNet: The subscript CF means we prune both convolutional kernels and neurons in FC layers, and C means we only prune convolutional kernels. *High*, *Mid* and *Low* mean we prune the entire CNN except for the high/middle/low level convolutional layers (Conv4-Conv5, Conv3 and Conv1-Conv2 respectively).

7. Experiments on AlexNet: Convolutional Layers v.s. FC Layers

From the experiments in the main paper, we found that FC layers have significant influence on accuracy loss, model size and memory usage. To exploit the impact of pruning FC layers and convolutional layers, we conduct experiments on pruning half of the neurons in FC layers and some convolutional layers using ImageNet [1]]. We categorize the 5 convolutional layers into three-level feature extractors: low (Conv1-Conv2 layers), middle (Conv3 layer) and high (Conv4-Conv5 layers). Figure 5 displays learning curves and shows that although FC layers are important in AlexNet, powerful local feature extractors (more kernels in convolutional layers) can compensate the loss from pruning neurons in FC layers, or even achieve better predictive power (High and Low curves).

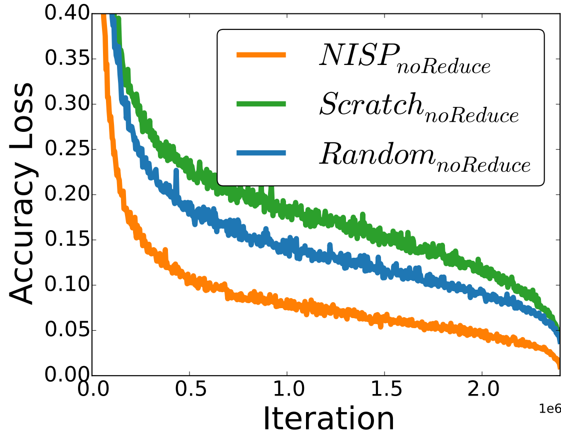


Figure 6. Learning Curves of GoogLeNet on ImageNet: The pruning ratio is 50%. We prune all layers but the reduction layers in the inception modules. importance based pruning method converges much faster and can achieve the smallest accuracy loss.

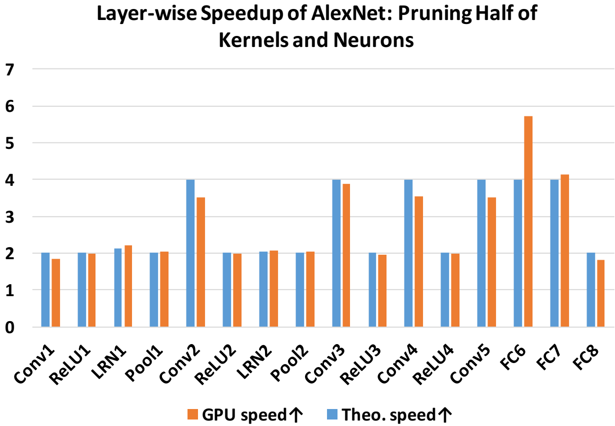


Figure 7. Full-Network Acceleration of AlexNet: Pruning Half of the Kernels and Neurons.

8. Experiments on GoogLeNet

The learning curves for "no_Reduce" of GoogLeNet [7] is shown in Figure 6. We observe that our importance based pruning method leads to better initialization, faster convergence and smaller final accuracy loss.

9. Layer-wise Improvements

In our experiments of AlexNet on Titan X, the empirical computation time for the intermediate layers (all layers except for convolutional layers and FC layers) accounts for 17% of the entire testing time; therefore, those layers must be considered as well while designing an acceleration method. One of our advantages over existing methods is that all layers in the network can be sped up due to the fact that the data volume or feature dimension at every layer is reduced. For example, by pruning kernels in convolutional layers, we reduce the number of both output channels of the current layer and input channels of the next layer. In theory, given a pruning ratio of 50%, except for the first layer whose input channels cannot be pruned, all of the convolutional layers can be sped up by $4\times$. The intermediate pooling, non-linearity and normalization layers have a theoretical speedup ratio of around $2\times$. The layer-wise acceleration ratios (both theoretical and empirical) of our method when the pruning ratio is 50% for both convolutional layers and FC layers are shown in Figure 7. We observe that the theoretical and empirical speedup are almost the same for pooling, non-linearity and normalization.

References

[1] 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.

[2] 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.

[3] X. Han, Z. Wu, Z. Wu, R. Yu, and L. S. Davis. Viton: An image-based virtual try-on network. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[4] 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.

[5] A. Li, J. Sun, J. Y.-H. Ng, R. Yu, V. I. Morariu, and L. S. Davis. Generating holistic 3d scene abstractions for text-based image retrieval. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[6] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[7] 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 Con-*

ference on Computer Vision and Pattern Recognition (CVPR), 2015.

[8] R. Yu, A. Li, V. I. Morariu, and L. S. Davis. Visual relationship detection with internal and external linguistic knowledge distillation. *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[9] 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.