Summary of Pruning

Chen Shangyu

May 14, 2019

Contents

1 Element-Wise Pruning

Neural network pruning is long studied dated back to 20th. [9] formulated the final loss variation due to the prunning of given elements. With the assumption of local identity and independence between element prunning. Formulation can be simplified as a quactical objective function involving a dignoal hessian.

$$\delta loss = \sum_i g_i \delta \mathbf{w}_i + \frac{1}{2} \sum_i h_{ii} \delta \mathbf{w}_i^2 + \sum_{i \neq j} h_{ij} \delta \mathbf{w}_i \delta \mathbf{w}_j \tag{1}$$

where g_i is the gradient of loss w.r.t $\mathbf{w}_i$, and h_{ij} are the elements of Hessian matrix. With the simplification that off-dignoal elements in Hessian matrix and gradient can be neglected, Eq.(1) can be reduced to:

$$\delta loss = \frac{1}{2} \sum_i h_{ii} \delta \mathbf{w}_i^2 \tag{2}$$

Eq.(1) is the criterion of weights prunning.

[4] expanded [9] with weights update after one element is pruned, with objective is changed to measure the difference of layer output after prunning:

$$\delta E = (\frac{\partial E}{\partial \mathbf{w}})^\top \delta \mathbf{w} + \frac{1}{2} \delta \mathbf{w}^\top \mathbf{H} \delta \mathbf{w} \tag{3}$$

Minimizing Eq.(3) with the constraint that weights of index q is pruned leads to weights compensation and corresponding loss as:

$$\begin{aligned} \delta \mathbf{w} &= -\frac{\mathbf{w}_q}{[\mathbf{H}^{-1}]_{qq}} \mathbf{H}^{-1} e_q \\ L_q &= \frac{1}{2} \frac{\mathbf{w}_q^2}{[\mathbf{H}^{-1}]_{qq}} \end{aligned} \tag{4}$$

According to Eq.(4), [4] updated the rest of weights after prunning and attained an more accurate loss due to weights pruning.

Entering the new century with deep learning explosion, The huge volumn of parameters in modern neural network makes [9] and [4] intractable. [3] (LWC) pioneered element prunning in deep neural network. It proposed to prune elements by its absolute value: it deleted those weights who approach to zero and remained the rest, which held the assumption that larger weights contribute more to the performance of neural network.

Inspired by such straightforward and effective prunning, [2] incorporated LWC into network training to prune weights in an incremental way. Specifically, during the training of deep neural network, it occasionally pruned weight according to their absolute value. Then it continued to train with pruned ones fixed as zero. Those pruned weights can be restored in latter retrain.

1.1 Learning sparse neural networks via sensitivity-driven regularization [15]

A method to push insensitive weights toward 0 gradually, then a thresholding is applied as pruning. The criterion of sensitivity is measured by gradient.

- Quantify the output sensitivity to parameters.
- Introduce a regularization that gradually lowers the absolute value of parameters with low sensitivity.
- Finally, fraction of the parameters approach zero and are eventually set to zero by simple thresholding.

1.1.1 Sensitivity

$$\Delta y_k \approx \Delta w_{n,i} \frac{\partial y_k}{\partial w_{n,i}} \tag{5}$$

A weighted sum of the variation in all output elements:

$$S(\mathbf{y}, w_{n,i}) = \sum_{k=1}^{C} \alpha_k |\Delta y_k| = |\Delta w_{n,i}| \sum_{k=1}^{C} \alpha_k |\frac{\partial y_k}{\partial w_{n,i}}| \tag{6}$$

Two types of weighted coefficients: The factors α_k are therefore taken as the elements in the one-hot encoding for the desired output $\mathbf{y}^*$:

$$S^{spec}(\mathbf{y}, \mathbf{y}^*, w_{n,i}) = \sum_{k=1}^{C} y^* |\frac{\partial y_k}{\partial w_{n,i}}| \tag{7}$$

1.1.2 Bounded Insensitivity

:

$$\bar{S}_b(\mathbf{y}, w_{n,i}) = \max \left[0, \underbrace{1 - S(\mathbf{y}, w_{n,i})}_{\bar{S}(\mathbf{y}, w_{n,i})} \right] \in [0,1] \tag{8}$$

The higher $\bar{S}_b(\mathbf{y}, w_{n,i})$, the less important the weight is.

1.1.3 Update Rule

:

$$w_{n,i}^t := w_{n,i}^{t-1} - \eta \frac{\partial L}{\partial w_{n,i}^{t-1}} - \lambda w_{n,i}^{t-1} \bar{S}_b(\mathbf{y}, w_{n,i}) \tag{9}$$

if the loss function is near a minimum, then the first correction term is very small. Weights with higher $\bar{S}_b(\mathbf{y}, w_{n,i})$ will be moved towards zero in proportion to λ.

1.1.4 Regularization

Overall regularization term as a sum over all parameters is defined, with integration of each term over $w_{n,i}$:

$$R(\theta) = \sum_i \sum_n R_{n,i}(w_{n,i}) = \sum_i \sum_n \left(\int w_{n,i} \bar{S}_b(\mathbf{y}, w_{n,i}) dw_{n,i} \right) \tag{10}$$

For ReLu activation, (10) reduced to:

$$R_{n,i}(w_{n,i}) = \frac{w_{n,i}^2}{2} \bar{S}(\mathbf{y}, w_{n,i}) \tag{11}$$

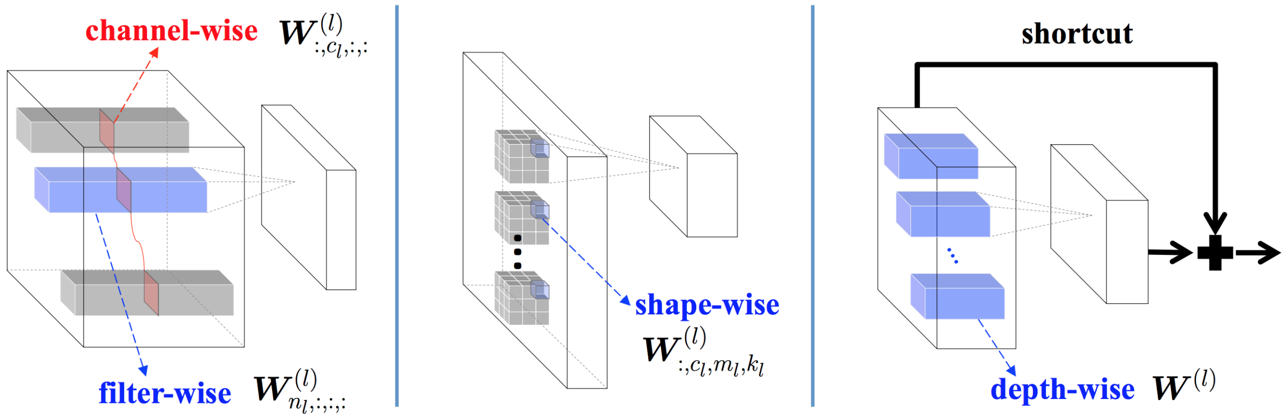

channel-wise $W^{(l)}_{:,c_l,:,:}$
filter-wise $W^{(l)}_{n_l,:,:,:}$
shape-wise $W^{(l)}_{:,c_l,m_l,k_l}$
shortcut
depth-wise $W^{(l)}$

Figure 1: Four main types of structure prunning in convolution layers. [16]

1.2 To prune, or not to prune: exploring the efficacy of pruning for model compression

2 Structure Pruning

Fig.1 demonstrates the four main types of structure prunning in convolution layer. Suppose the original convolution weights' dimension as $[c_{in} \times c_{out} \times w \times h]$.

- **Filter-wise** prunning reduces the entire filter, which results in pruned convolution weights' dimension as $[c_{in} \times c_{out}^{'} \times w \times h]$, where $c_{out}^{'} < c_{out}$.
- **Channel-wise** prunning can be regarded as "cut out of slices in convolution weights": $[c_{in}^{'} \times c_{out} \times w \times h]$, where $c_{in}^{'} < c_{in}$.
- **Shape-wise** methods prune weights in the same position among all filters, making a different shape of filters.
- **Layer-wise** methods prune the whole layer.

2.1 Learning Structured Sparsity in Deep Neural Networks

[16] proposed to prune entire structure of deep neural network to improve memory locality, which can accelerate computation than individual prunning with low locality. It come up with 4 schemes to prune: 1) Filter-wise 2) Channel-wise 3) Shape-wise and 4) Layer-wise. Pruning is performed by imposing $L_{2,1}$ norm in corresponding scheme then training.

2.2 Extreme Network Compression via Filter Group Approximation

3 Filter Pruning

Among the various methods, filter prunning in convolution layer is a popular research topic since each filter takes effect independently. Intuitively, prunning a whole filter brings less harm to overall performance. Besides, these methods lead to significant and obvious memory saving and acceleration due to network slim.

3.1 Pruning filters for efficient convnets

Similar with [3], [11] pruned entire filter in convolution layers by measuring the sum of their absolute value.

3.2 Network trimming: A data-driven neuron pruning approach towards efficient deep architectures

[7] recorded neurons the non-activate percentage during a batch of data to measure whether to prune this neuron.

4 Channel Pruning

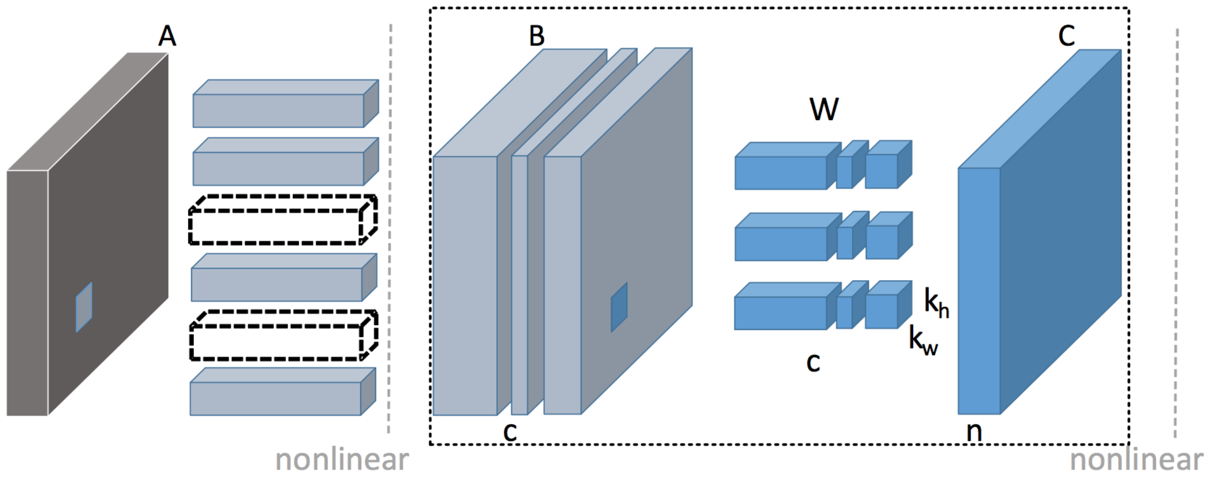


Figure 2: Pruning channels: prunning channels in layer k results in prunning filters in previous layer. Figure is cited from [6]

Fig.2 demonstrates how channel prunning is conducted and how it effects the before and latter layers. Some "slices" of all filters in W results in the invalidation of some input feature maps, thus back propagate to the filters in previous layers. As we can observe, the number of filters c^k_{out} in layer k is same as the number of channels c^{k+1}_{in} in layer $k+1$, prunning certains channels / filters in layer $k+1$ / k results in the corresponding filters / channels in layer k / $k+1$.

4.1 Learning efficient convolutional networks through network slimming

[13] introduced a scaling factor γ for each channel. By training entire neural network with γ, filters with less importance is neglected with $\gamma = 0$.

4.2 Channel pruning for accelerating very deep neural networks

[6] proposed a two-step layer-wise method: it firstly selected representative channels and pruned redundant ones by a LASSO regression based method. Then, it minimized the reconstruction

loss. The whole process is formulated as:

$$\begin{aligned}\arg\min_{\beta,\mathbf{W}} & \frac{1}{N}||\mathbf{Y}-\sum_{i=1}^{c}\beta_i\mathbf{X}_i\mathbf{W}_i|| \\ \text{s.t.} \quad & ||\beta||_0 \le \epsilon\end{aligned} \tag{12}$$

For each layer, after c channels are selected by β, it adjusts the remaining $\mathbf{W}$ to compensate pruned error in layer output. It alternatively optimizes β and $\mathbf{W}$.

4.3 Thinet: A filter level pruning method for deep neural network compression

[14] and [6] solved the same problem but with different approaches. As a comparsion, [14] transformed the problem of pruning filter in layer i as a input channel combination problem to minimize reconstruction error in layer $i+1$. Specifically, it formulated the problem as:

$$\arg\min_{S}\sum_{i=1}^{m}(\hat{y}-\sum_{j\in S}\hat{x}_{i,j})^2 \tag{13}$$

where m is number of instance, S is the selected channels. It simplifies Eq.(13) as:

$$\arg\min_{S}\sum_{i=1}^{m}(\sum_{j\in T}\hat{x}_{i,j})^2 \tag{14}$$

where T is the complementary set of S. However, Solving Eq.(14) is still NP hard, thus it used a greedy strategy by adding one element to T at a time, and choosing the channel leading to the smallest objective value.
After prunning it will further minimize the reconstruction error (Eq.(13)) by weighing the channels, which can be defined as:

$$\mathbf{W}^* = \arg\min_{\mathbf{W}}\sum_{i=1}^{m}(\sum_{j\in T}\hat{y}_i-\mathbf{W}_{i,j}^{\top}\hat{x}_i)^2 \tag{15}$$

4.4 Discrimination-aware Channel Pruning for Deep Neural Networks [17]

- This paper aims at pruning-based training: training (distillation) while layer-wisely pruning.
- To measure the discriminative power of channels in intermediate layers by introducing:
  - Layer-Wise Output Reconstruction Loss.
  - Early Prediction (Discrimination-aware Loss).
  - Final Prediction Loss.

- Learn early predictors θ by $\mathcal{F}_s^p$.
- Finetune pruned network M with $\mathcal{L}_f$.
- Select channel by $\mathbf{G}=\frac{\partial\mathbf{L}}{\partial\mathbf{W}}$, where $\mathbf{L}=\mathcal{L}_M+\mathcal{L}_s^p+\mathcal{L}_f$.

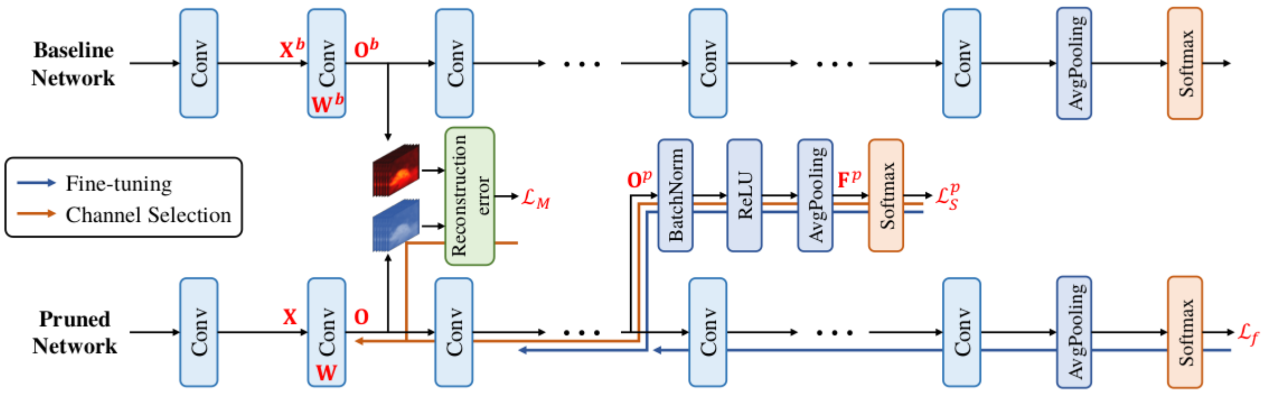


Figure 3: Illustration of discrimination-aware channel pruning.

4.4.1 Greedy Algorithm for Channel Selection

By introducing P losses $\{\mathcal{L}_S^p\}_{p=1}^P$, for each layer during P and $P+1$:

- First remove all channels and select channels.
- Add channels to set $\mathcal{A}$ according to their gradient $\mathbf{G}$. Update the added channels by:

$$\min \mathcal{L}(\mathbf{W}) = \mathcal{L}_M(\mathbf{W}) + \lambda \mathcal{L}_S^p(\mathbf{W}). \qquad \text{s.t.} \mathbf{W}_{\mathcal{A}^c} = 0 \tag{16}$$

- Until

$$\frac{|\mathcal{L}(\mathbf{W}^{t-1}) - \mathcal{L}(\mathbf{W}^t)|}{\mathcal{L}(\mathbf{W}^0)} \leq \epsilon \tag{17}$$

4.5 Squeeze-and-excitation networks [8]

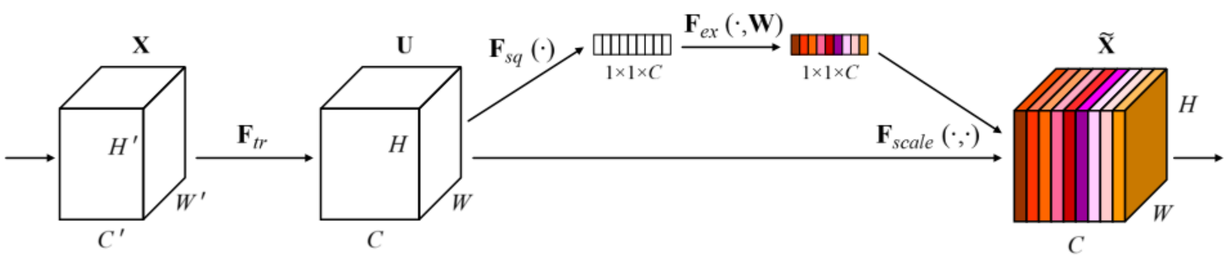


Figure 4: A Squeeze-and-Excitation block features

- SE-Net aims at adaptively recalibrating channel-wise feature responses by explicitly modeling interdependencies between channels.
- SE-Net takes action in the output features $\mathbf{U} \in \mathbb{R}^{H\times W\times C}$, with a global average pooling for each channel:

$$z_c = \mathbf{F}_{sq}(\mathbf{U}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i,j) \tag{18}$$

- Then a learnable excitation operation is conducted:

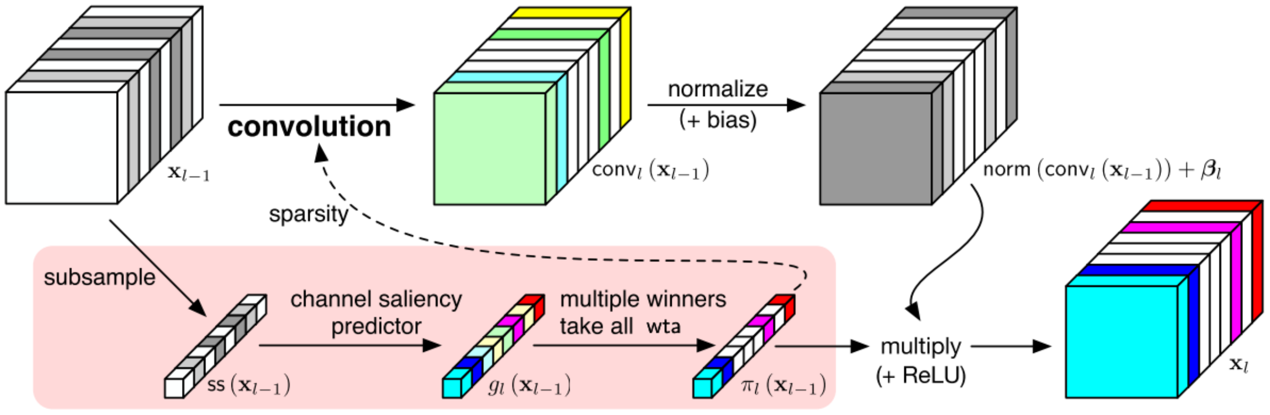
$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{z})) \tag{19}$$

  where δ refers to ReLU, $\mathbf{W}_1 \in \mathbb{R}^{\frac{C}{r} \times C}$, $\mathbf{W}_2 \in \mathbb{R}^{C \times \frac{C}{r}}$, which is a bottleneck with two FC layers around non-linearity.

- Finally the output $\mathbf{U}$ is rescaled as:

$$\tilde{\mathbf{x}}_c = s_c \times \mathbf{U}_c \tag{20}$$

4.6 Dynamic Channel Pruning: Feature Boosting and Suppression [1]



convolution
$\mathbf{x}_{l-1}$
normalize
(+ bias)
$\text{conv}_l(\mathbf{x}_{l-1})$
$\text{norm}(\text{conv}_l(\mathbf{x}_{l-1})) + \boldsymbol{\beta}_l$
sparsity
subsample
channel saliency predictor
multiple winners take all wta
$\text{ss}(\mathbf{x}_{l-1})$
$g_l(\mathbf{x}_{l-1})$
$\pi_l(\mathbf{x}_{l-1})$
multiply
(+ ReLU)
$\mathbf{x}_l$

Figure 5: A high level view of a convolutional layer with FBS

- The auxiliary components (in red) predict the importance of each **output** channel based on the **input** features, and amplify the output features accordingly, formally:

$$\hat{f}_l(\mathbf{x}_{l-1}) = \pi_l(\mathbf{x}_{l-1}) \cdot \text{conv}(\mathbf{x}_{l-1}, \theta_l) \tag{21}$$

  where $\pi_l(\mathbf{x}_{l-1}) = \text{wta}_{\lceil dC_l \rceil}(g_l(\mathbf{x}_{l-1}))$, $\text{wta}_k(\mathbf{z})$ is a k-winners-take-all function, i.e. it returns a tensor identical to $\mathbf{z}$, except that we zero out entries in $\mathbf{z}$ that are smaller than the k largest entries in **absolute magnitude**.

- Channel saliency predictor $g_l(\mathbf{x}_{l-1})$ is defined as:

$$\begin{aligned} g_l(\mathbf{x}_{l-1}) &= \text{FC}(\text{ss}(\mathbf{x}_{l-1}), \mathbf{W}_l) \\ \text{ss}(\mathbf{x}_{l-1}) &= \frac{1}{H \times W}[s(\mathbf{x}_{l-1}^1), s(\mathbf{x}_{l-1}^2), ..., s(\mathbf{x}_{l-1}^C)] \end{aligned} \tag{22}$$

  $s(\cdot)$ is chosen to be l_1 norm.

4.7 Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration [5]

4.8 Exploiting Kernel Sparsity and Entropy for Interpretable CNN Compression [12]

4.9 Structured Pruning of Neural Networks with Budget-Aware Regularization [10]

4.10 Comparison

Method	Network	CR (%)	FP Acc (%)	Drop (%)
[11]	VGG16	36	93.25	-0.15
Weight-based	ResNet56	86.3	93.04	-0.02
	ResNet110	67.6	93.53	0.23

Table 1: Experiments of filter-wise Pruning on CIFAR10

Method	Network	CR (%)	Top1/Top5 (%)	Drop Top1/Top5(%)
ThiNet [14]	VGG16	95.01 6.01	68.34/88.44	-1.46/-1.09 1/0.52

Table 2: Experiments of filter-wise Pruning on ImageNet

5 Skipping & Early Stop

5.1 SkipNet: Learning Dynamic Routing in Convolutional Networks

This work proposed a modified residual network to selectively skip convolutional blocks based on the activations of the previous layer. Whether to skip is learnt by RL. Denoted $\mathbf{x}_i$ as inputs,

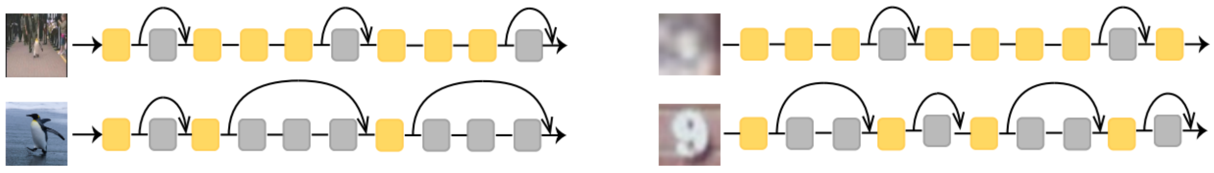


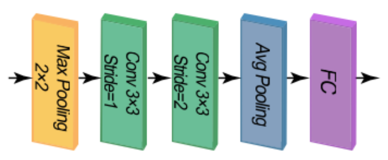
Figure 6: The SkipNet learns to skip convolutional layers on a per-input basis. More layers are executed for challenging images (top) than easy images (bottom)

$F^i(\mathbf{x}_i)$ be the output of the i^{th} layer or group of layer, then the output of the gated layer (or group of layers) as:

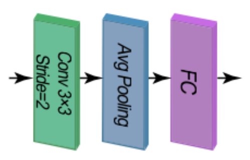
$$\mathbf{x}_{i+1} = G^i(\mathbf{x}_i)F^i(\mathbf{x}_i) + (1 - G^i(\mathbf{x}_i))\mathbf{x}_i \tag{23}$$

5.1.1 Gating Network Design

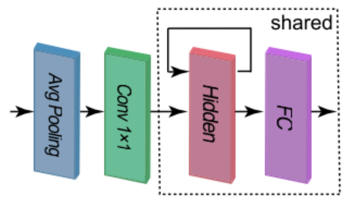
- Supervised Pre-training Optimizing
- Skipping Policy Learning with Hybrid RL



(a) FFGate-I



(b) FFGate-II



(c) RNNGate

6 Data-Free Pruning

6.1 DAC: Data-free Automatic Acceleration of Convolutional Networks

7 RL-based Pruning

7.1 N2N Learning: Network to Network Compression via Policy Gradient Reinforcement Learning

7.2 ADC: Automated Deep Compression and Acceleration with Reinforcement Learning

Reference

[1] Xitong Gao, Yiren Zhao, Lukasz Dudziak, Robert Mullins, and Cheng-zhong Xu. Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*, 2018.

[2] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

[3] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[4] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[5] Yang He, Ping Liu, Ziwei Wang, and Yi Yang. Pruning filter via geometric median for deep convolutional neural networks acceleration. *CVPR*, 2019.

[6] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, page 6, 2017.

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

[8] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 7, 2017.

[9] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990.

[10] Carl Lemaire, Andrew Achkar, and Pierre-Marc Jodoin. Structured pruning of neural networks with budget-aware regularization. *CVPR*, 2019.

[11] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[12] Yuchao Li, Shaohui Lin, Baochang Zhang, Jianzhuang Liu, David Doermann, Yongjian Wu, Feiyue Huang, and Rongrong Ji. Exploiting kernel sparsity and entropy for interpretable cnn compression. *CVPR*, 2019.

[13] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2755–2763. IEEE, 2017.

[14] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, 2017.

[15] Enzo Tartaglione, Skjalg Lepsøy, Attilio Fiandrotti, and Gianluca Francini. Learning sparse neural networks via sensitivity-driven regularization. In *Advances in Neural Information Processing Systems*, pages 3879–3889, 2018.

[16] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. 2016.

[17] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 881–892, 2018.