Neural Rejuvenation: Improving Deep Network Training by Enhancing Computational Resource Utilization

Siyuan Qiao[1*] Zhe Lin[2] Jianming Zhang[2] Alan Yuille[1]
[1]Johns Hopkins University [2]Adobe Research
{siyuan.qiao, alan.yuille}@jhu.edu {zlin,jianmzha}@adobe.com

Abstract

In this paper, we study the problem of improving computational resource utilization of neural networks. Deep neural networks are usually over-parameterized for their tasks in order to achieve good performances, thus are likely to have underutilized computational resources. This observation motivates a lot of research topics, e.g. network pruning, architecture search, etc. As models with higher computational costs (e.g. more parameters or more computations) usually have better performances, we study the problem of improving the resource utilization of neural networks so that their potentials can be further realized. To this end, we propose a novel optimization method named Neural Rejuvenation. As its name suggests, our method detects dead neurons and computes resource utilization in real time, rejuvenates dead neurons by resource reallocation and reinitialization, and trains them with new training schemes. By simply replacing standard optimizers with Neural Rejuvenation, we are able to improve the performances of neural networks by a very large margin while using similar training efforts and maintaining their original resource usages. The code is available here: https://github.com/joe-siyuan-qiao/NeuralRejuvenation-CVPR19

1. Introduction

Deep networks achieve state-of-the-art performances in many visual tasks [9, 23, 42, 47]. On large-scale tasks such as ImageNet [53] classification, a common observation is that the models with more parameters, or more FLOPs, tend to achieve better results. For example, DenseNet [28] plots the validation error rates as functions of the number of parameters and FLOPs, and shows consistent accuracy improvements as the model size increases. This is consistent with our intuition that large-scale tasks require models with sufficient capacity to fit the data well. As a result, it is usually beneficial to train a larger model if the additional computational resources are properly utilized. However, previous work on network pruning [40, 64] already shows that many neural networks trained by SGD have unsatisfactory resource utilization. For instance, the number of parameters of a VGG [54] network trained on CIFAR [33] can be compressed by a factor of 10 without affecting its accuracy [40]. Such low utilization results in a waste of training and testing time, and restricts the models from achieving their full potentials. To address this problem, we investigate novel neural network training and optimization techniques to enhance resource utilization and improve accuracy.

Formally, this paper studies the following optimization problem. We are given a loss function $\mathcal{L}(f(x;\mathcal{A},\theta_{\mathcal{A}}),y)$ defined on data (x,y) from a dataset $\mathcal{D}$, and a computational resource constraint $\mathcal{C}$. Here, $f(x;\mathcal{A},\theta_{\mathcal{A}})$ is a neural network with architecture $\mathcal{A}$ and parameterized by $\theta_{\mathcal{A}}$. Let $c(\mathcal{A})$ denote the cost of using architecture $\mathcal{A}$ in f, *e.g.*, $c(\mathcal{A})$ can be the number of parameters in $\mathcal{A}$ or its FLOPs. Our task is to find $\mathcal{A}$ and its parameter $\theta_{\mathcal{A}}$ that minimize the average loss $\mathcal{L}$ on dataset $\mathcal{D}$ under the resource constraint $\mathcal{C}$, *i.e.*,

$$\begin{aligned} \mathcal{A},\theta_{\mathcal{A}} = \arg\min_{\mathcal{A},\theta_{\mathcal{A}}} \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\big(f(x_i;\mathcal{A},\theta_{\mathcal{A}}),y_i\big) \\ \textit{s.t.}\ \ c(\mathcal{A}) \le \mathcal{C} \end{aligned} \tag{1}$$

The architecture $\mathcal{A}$ is usually designed by researchers and fixed during minimizing Eq. 1, and thus the solution $\mathcal{A},\theta_{\mathcal{A}}$ will always meet the resource constraint. When $\mathcal{A}$ is fixed, $\theta_{\mathcal{A}}$ found by standard gradient-based optimizers may have neurons (*i.e.* channels) that have little effects on the average loss, removing which will save resources while maintaining good performance. In other words, $\theta_{\mathcal{A}}$ may not fully utilize all the resources available in $\mathcal{A}$. Let $\mathcal{U}(\theta_{\mathcal{A}})$ denote the computational cost based on $\theta_{\mathcal{A}}$'s actual utilization of the computational resource of $\mathcal{A}$, which can be measured by removing dead neurons which have little effect on the output. Clearly, $\mathcal{U}(\theta_{\mathcal{A}}) \le c(\mathcal{A})$. As previous work suggests [40], the utilization ratio $r(\theta_{\mathcal{A}}) = \mathcal{U}(\theta_{\mathcal{A}})/c(\mathcal{A})$ trained by standard SGD can be as low as 11.5%.

The low utilization motivates the research on network

*Work done while an intern at Adobe.

pruning [40, 64], *i.e.*, extracting the effective subnet $\mathcal{A}'$ from $\mathcal{A}$ such that $c(\theta_{\mathcal{A}'}) = \mathcal{U}(\theta_{\mathcal{A}})$. Although the utilization ratio $r(\theta_{\mathcal{A}'})$ is high, this is opposite to our problem because it tries to narrow the difference between $c(\mathcal{A})$ and $\mathcal{U}(\mathcal{A})$ by moving $c(\mathcal{A})$ towards $\mathcal{U}(\mathcal{A})$. By contrast, our objective is to design an optimization procedure $\mathcal{P}$ which enables us to find parameters $\theta_{\mathcal{A}} = \mathcal{P}(\mathcal{A}, \mathcal{L}, \mathcal{D})$ with a high $r(\theta_{\mathcal{A}})$. In other words, we are trying to move $\mathcal{U}(\mathcal{A})$ towards $c(\mathcal{A})$, which maximizes the real utilization of the constraint $\mathcal{C}$.

There are many reasons for low utilization ratio $r(\theta_{\mathcal{A}})$. One is bad initialization [14], which can be alleviated by parameter reinitialization for the spare resource. Another one is inefficient resource allocation [16], *e.g.*, the numbers of channels or the depths of blocks may not be configured properly to meet their real needs. Unlike the previous methods [16, 39] which search architectures by training a lot of networks, we aim to design an optimizer that trains *one* network only *once* and includes both resource *reinitialization* and *reallocation* for maximizing resource utilization.

In this paper, we propose an optimization method named Neural Rejuvenation (NR) for enhancing resource utilization during training. Our method is intuitive and simple. During training, as some neurons may be found to be useless (*i.e.* have little effect on the output), we revive them with new initialization and allocate them to the places they are needed the most. From a neuroscience perspective, this is to rejuvenate dead neurons by bringing them back to functional use [12] – hence the name. The challenges of Neural Rejuvenation are also clear. Firstly, we need a real-time resource utilization monitor. Secondly, when we rejuvenate dead neurons, we need to know how to reinitialize them and where to place them. Lastly, after dead neuron rejuvenation, survived neurons ($\mathcal{S}$ neurons) and rejuvenated neurons ($\mathcal{R}$ neurons) are mixed up, and how to train networks with both of them present is unclear.

Our solution is a plug-and-play optimizer, the codes of which will be made public. Under the hood, it is built on standard gradient-based optimizers, but with additional functions including real-time resource utilization monitoring, dead neuron rejuvenation, and new training schemes designed for networks with mixed types of neurons. We introduce these components as below.

Resource utilization monitoring Similar to [40, 64], we use the activation scales of neurons to identify utilized and spare computational resource, and calculate a real-time utilization ratio $r(\theta_{\mathcal{A}})$ during training. An event will be triggered if $r(\theta_{\mathcal{A}})$ is below a threshold T_r, and the procedure of dead neuron rejuvenation will take the control before the next step of training, after which $r(\theta_{\mathcal{A}})$ will go back to 1.

Dead neuron rejuvenation This component rejuvenates the dead neurons by collecting the unused resources and putting them back in $\mathcal{A}$. Similar to MorphNet [16], more spare resources are allocated to the layers with more $\mathcal{S}$ neurons. However, unlike MorphNet [16] which trains the whole network again from scratch after the rearrangement, we only reinitialize the dead neurons and then continue training. By taking the advantages of dead neuron reinitialization [14] and our training schemes, our optimizer is able to train *one* model only *once* and outperform the optimal network found by MorphNet [16] from lots of architectures.

Training with mixed neural types After dead neuron rejuvenation, each layer will have two types of neurons: $\mathcal{S}$ and $\mathcal{R}$ neurons. We propose two novel training schemes for different cases when training networks with mixed types of neurons. The first one is to remove the cross-connections between $\mathcal{S}$ and $\mathcal{R}$ neurons, and the second one is to use cross-attention between them to increase the network capacity. Sec. 3.3 presents the detailed discussions.

We evaluate Neural Rejuvenation on two common image recognition benchmarks, *i.e.* CIFAR-10/100 [33] and ImageNet [53] and show that it outperforms the baseline optimizer by a very large margin. For example, we lower the top-1 error of ResNet-50 [23] on ImageNet by 1.51%, and by 1.82% for MobileNet-0.25 [27] while maintaining their FLOPs. On CIFAR where we rejuvenate the resources to the half of the constraint and compare with the previous state-of-the-art compression method [40], we outperform it by up to 0.87% on CIFAR-10 and 3.39% on CIFAR-100.

2. Related Work

Efficiency of neural networks It is widely recognized that deep neural networks are over-parameterized [2, 11] to win the filter lottery tickets [14]. This efficiency issue is addressed by many methods, including weight quantization [10, 51], low-rank approximation [11, 34], knowledge distillation [26, 63] and network pruning [20, 22, 36, 38, 40, 45, 64, 65]. The most related method is network pruning, which finds the subnet that affects the outputs the most. Network pruning has several research directions, such as weight pruning, structural pruning, *etc*. Weight pruning focuses on individual weights [18, 20, 22, 36], but requires dedicated hardware and software implementations to achieve compression and acceleration [17]. Structural pruning identifies channels and layers to remove from the architecture, thus is able to directly achieve speedup without the need of specialized implementations [1, 25, 35, 41, 45, 59, 68]. Following [40, 64], we encourage channel sparsity by imposing penalty term to the scaling factors.

Different from these previous methods, Neural Rejuvenation studies the efficiency issue from a new angle: we aim to directly maximize the utilization by reusing spare computational resources. As an analogy in the context of lottery hypothesis [14], Neural Rejuvenation is like getting refund for the useless tickets and then buying new ones.

Cross attention In this work, we propose to use cross attention to increase the capacity of the networks without introducing additional costs. This is motivated by adding second-order transform [15, 32, 57] on multi-branch networks [23, 28, 50, 55, 56, 58]. Instead of using a geometric mean as in [57], we propose to use cross attention [21, 37] as the second-order term to increase the capacity. Attention models have been widely used in deep neural networks for a variety of vision and language tasks, such as object detection [3, 44, 48, 66], machine translation [4], visual question answering [8, 60], image captioning [61], *etc*. Unlike the previous attention models, our method uses one group of channels to generate attentions for the other channels, and our attention model is mainly used to increase capacity.

Architecture search Our objective formulated by Eq. 1 is similar to neural architecture search which approaches the problem by searching architecture $\mathcal{A}$ in a pre-defined space, and thus they need to train a lot of networks to find the optimal architecture. For example, NAS [69] uses reinforcement learning to find the architecture, [70] extends it by using a more structured search space, and [39] improves the search efficiency by progressively finding architectures. But their computational costs are very high, *e.g.*, [70] uses 2000 GPU days. There are more methods focusing on the search problem [5, 6, 13, 43, 46, 52, 67]. Different from architecture search, Neural Rejuvenation does not search $\mathcal{A}$ which requires hundreds of thousands of models to train, although it does change the architecture a little bit. Instead, our method is an optimization technique which trains models in just one training pass. The closest method is MorphNet [16] in that we both use linearly expanding technique to find resource arrangement. Yet, it still needs multiple training passes and does not rejuvenate dead neurons nor reuse partially-trained filters. We show direct comparisons with it and outperform it by a large margin.

Parameter reinitialization Parameter reinitialization is a common strategy in optimization to avoid useless computations and improve performances. For example, during the k-means optimization, empty clusters are automatically reassigned, and big clusters are encouraged to split into small clusters [7, 30, 31, 62]. Our method is reminiscent to this in that it also detects unsatisfactory components and reinitializes them so that they can better fit the tasks.

3. Neural Rejuvenation

Algorithm 1 presents a basic framework of Neural Rejuvenation which adds two new modules: resource utilization monitoring (Step 6) and dead neuron rejuvenation (Step 7 and 8) to a standard SGD optimizer. The training schemes are not shown here, which will be discussed in Sec. 3.3. We periodically set the Neural Rejuvenation flag on with a predefined time interval to check the utilization and rejuvenate dead neurons when needed. In the following subsections, we will present how each component is implemented.

Algorithm 1: SGD with Neural Rejuvenation

Input : Learning rate ϵ, utilization threshold T_r, initial architecture $\mathcal{A}$ and $\theta_{\mathcal{A}}$, and resource constraint $\mathcal{C}$

1 **while** *stopping criterion not met*:
2 Sample a minibatch $\{(x_1, y_1), ..., (x_m, y_m)\}$;
3 Compute gradient $g \leftarrow \frac{1}{m}\nabla \sum_i \mathcal{L}(f(x_i; \mathcal{A}, \theta_{\mathcal{A}}), y_i)$;
4 Apply update $\theta_{\mathcal{A}} = \theta_{\mathcal{A}} - \epsilon \cdot g$;
5 **if** *neural rejuvenation flag is on*:
6 Compute utilization ratio $r(\theta_{\mathcal{A}})$;
7 **if** $r(\theta_{\mathcal{A}}) < T_r$:
8 Rejuvenate dead neurons and obtain new $\mathcal{A}$ and $\theta_{\mathcal{A}}$ under resource constraint $\mathcal{C}$;
9 **return** Architecture $\mathcal{A}$ and its parameter $\theta_{\mathcal{A}}$;

3.1. Resource Utilization Monitoring

3.1.1 Liveliness of Neurons

We consider a convolutional neural network where every convolutional layer is followed by a batch normalization layer [29]. An affine transform layer with learnable parameters are also valid if batch normalization is not practical. For each batch-normalized convolutional layer, let $\mathcal{B} = \{u_1, ..., u_m\}$ be a mini-batch of values after the convolution. Then, its normalized output $\{v_1, ..., v_m\}$ is

$$v_i = \gamma \cdot \frac{u_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \beta, \ \ \forall i \in \{1, ..., m\}$$
$$\text{where } \mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} u_i \text{ and } \sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(u_i - \mu_{\mathcal{B}})^2 \tag{2}$$

Each neuron (*i.e.* channel) in the convolutional layer has its own learnable scaling parameter γ, which we use as an estimate of the liveliness of the corresponding neuron [40, 64]. As our experiments suggest, if a channel's scaling parameter γ is less than $0.01 \times \gamma_{\max}$ where $\gamma_{\max}$ is the maximum γ in the same batch-normalized convolution layer, removing it will have little effect on the output of f and the loss $\mathcal{L}$. Therefore, in all experiments shown in this paper, a neuron is considered dead if its scaling parameter $\gamma < 0.01 \times \gamma_{\max}$. Let $\mathcal{T}$ be the set of all the scaling parameters within the architecture $\mathcal{A}$. Similar to [40], we add a L1 penalty term on $\mathcal{T}$ in order to encourage neuron sparsity, *i.e.,* instead of the given loss function $\mathcal{L}$, we minimize the following loss

$$\mathcal{L}_\lambda = \mathcal{L}\big(f(x_i; \mathcal{A}, \theta_{\mathcal{A}}), y_i\big) + \lambda \sum_{\gamma \in \mathcal{T}} |\gamma| \tag{3}$$

where λ is a hyper-parameter.

3.1.2 Computing $r(\theta_{\mathcal{A}})$ by Feed-Forwarding

Here, we show how to compute the utilization ratio $r(\theta_{\mathcal{A}})$ based on the liveliness of the neurons in real time. We compute $r(\theta_{\mathcal{A}})$ by a separate feed-forwarding similar to that of function f. The computational cost of the feed-forwarding for $r(\theta_{\mathcal{A}})$ is negligible compared with that of f. We first rewrite the function f:

$$f(x) = (f_l \circ f_{l-1} \circ ... \circ f_1)(x) \tag{4}$$

where f_i is the i-th layer of the architecture $\mathcal{A}$. When computing $r(\theta_{\mathcal{A}})$, instead of passing the output of a layer computed from x to the next layer as input, each layer f_i will send a binary mask indicating the liveliness of its neurons. Let M_i^{in} denote the binary mask for the input neurons for layer f_i, and M_i^{out} denote the binary mask for its own neurons. Then, the effective number of parameters of f_i is $||M_i^{\text{in}}||_1 \cdot ||M_i^{\text{out}}||_1 \cdot K_w \cdot K_h$, if f_i is a convolutional layer with 1 group and no bias, and its computational cost is computed by $||M_i^{\text{in}}||_1 \cdot ||M_i^{\text{out}}||_1 \cdot K_w \cdot K_h \cdot O_w \cdot O_h$ following [23]. Here, K_w and K_h are the kernel size, and O_w and O_h are the output size. Note that the cost of f is the sum of the costs of all layers f_i; therefore, we also pass the effective computational cost and the original cost in feed-forwarding. After that, we are able to compute $U(\theta_{\mathcal{A}})$ and $c(\theta_{\mathcal{A}})$, and consequently $r(\theta_{\mathcal{A}})$. During the computation of $r(\theta_{\mathcal{A}})$, each layer will also keep a copy of the liveliness of the neurons of its previous layer. This information is used in the step of dead neural rejuvenation after $r(\theta_{\mathcal{A}}) < T_r$ is met. It also records the values of the scaling parameter γ of the input neurons. This is used for neural rescaling which is discussed in Sec. 3.2.

3.1.3 Adaptive Penalty Coefficient λ

The utilization ratio $r(\theta_{\mathcal{A}})$ will depend on the value of the sparsity coefficient λ as a larger λ tends to result in a sparser network. When $\lambda = 0$, all neurons will probably stay alive as we have a tough threshold $0.01 \times \gamma_{\max}$. As a result, Step 7 and 8 of Algorithm 1 will never get executed and our optimizer is behaving as the standard one. When λ goes larger, the real loss function $\mathcal{L}_\lambda$ we optimize will become far from the original loss $\mathcal{L}$. Consequently, the performance will be less unsatisfactory. Therefore, choosing a proper λ is critical for our problem, and we would like it to be automatic and optimized to the task and the architecture.

In Neural Rejuvenation, the value of λ is dynamically determined by the trend of the utilization ratio $r(\theta_{\mathcal{A}})$. Specifically, when the neural rejuvenation flag is on, we keep a record of the utilization ratio $r(\theta_{\mathcal{A}})^t$ after training for t iterations. After Δt iterations , we compare the current ratio $r(\theta_{\mathcal{A}})^t$ with the previous one $r(\theta_{\mathcal{A}})^{t-\Delta t}$. If $r(\theta_{\mathcal{A}})^t < r(\theta_{\mathcal{A}})^{t-\Delta t} - \Delta r$, we keep the current λ; otherwise, we increase λ by $\Delta\lambda$. Here, Δt, Δr and $\Delta\lambda$ are hyper-parameters. λ is initialized with 0. After Step 8 gets executed, λ is set back to 0.

It is beneficial to set λ in the above way rather than having a fixed value throughout the training. Firstly, different tasks and architectures may require different values of λ. The above strategy frees us from manually selecting one based on trial and error. Secondly, the number of iterations needed to enter Step 8 is bounded. This is because after λ gets large enough, each Δt will decrease the utilization ratio by at least Δr. Hence, the number of iterations to reach T_r is bounded by $(1-T_r)/\Delta r + O(1)$. In a word, this strategy automatically finds the value of λ, and guarantees that the condition $r(\theta_{\mathcal{A}}) < T_r$ will be met in a bounded number of training iterations.

3.2. Dead Neuron Rejuvenation

After detecting the liveliness of the neurons and the condition $r(\theta_{\mathcal{A}}) < T_r$ is met, we proceed to Step 8 of Algorithm 1. Here, our objective is to rejuvenate the dead neurons and reallocate those rejuvenated neurons to the places they are needed the most under the resource constraint $\mathcal{C}$. There are three major steps in dead neuron rejuvenation. We present them in order as follows.

Resource reallocation The first step is to reallocate the computational resource saved by removing all the dead neurons. The removal reduces the computational cost from $c(\mathcal{A})$ to $U(\theta_{\mathcal{A}})$; therefore, there is $c(\mathcal{A}) - U(\theta_{\mathcal{A}})$ available resource to reallocate. The main question is where to add this free resource back in $\mathcal{A}$. Let w_i denote the number of output channels of layer f_i in f, and w_i is reduced to w_i' by dead neuron removal. Let $\mathcal{A}'$ denote the architecture after dead neuron removal with w_i' output channels at layer f_i. Then, $c(\mathcal{A}') = U(\mathcal{A})$. To increase the computational cost of $\mathcal{A}'$ to the level of $\mathcal{A}$, our resource reallocation will linearly expand $w_i'' = \alpha \cdot w_i'$ by a shared expansion rate α across all the layers f_i, to build a new architecture $\mathcal{A}''$ with numbers of channels w_i''. The assumption here is that if a layer has a higher ratio of living neurons, this layer needs more resources, *i.e.* more output channels; by contrast, if a layer has a lower ratio, this means that more than needed resources were allocated to it in $\mathcal{A}$. This assumption is modeled by having a shared linear expansion rate α.

The resource reallocation used here is similar to the iterative squeeze-and-expand algorithm in MorphNet [16] for neural architecture search. The differences are also clear. Neural Rejuvenation models both dead neuron reinitialization, reallocation and training schemes to train just one network only once, while MorphNet is only interested in the numbers of channels of each layer that are optimal when trained from scratch and finds it by training many networks.

Parameter reinitialization The second step is to reinitialize the parameters of the reallocated neurons. Let $\mathcal{S}_{\text{in}}$

and $\mathcal{R}_{\text{in}}$ denote the input S (survived) neurons and R (rejuvenated) neurons, respectively, and $\mathcal{S}_{\text{out}}$ and $\mathcal{R}_{\text{out}}$ denote the output S neurons and R neurons, respectively. Then, the parameters W can be divided into four groups: $W_{\mathcal{S}\to\mathcal{S}}$, $W_{\mathcal{S}\to\mathcal{R}}$, $W_{\mathcal{R}\to\mathcal{R}}$, $W_{\mathcal{R}\to\mathcal{S}}$, which correspond to the parameters from $\mathcal{S}_{\text{in}}$ to $\mathcal{S}_{\text{out}}$, from $\mathcal{S}_{\text{in}}$ to $\mathcal{R}_{\text{out}}$, from $\mathcal{R}_{\text{in}}$ to $\mathcal{R}_{\text{out}}$ and from $\mathcal{R}_{\text{in}}$ to $\mathcal{S}_{\text{out}}$, respectively. During reinitialization, the parameters $W_{\mathcal{S}\to\mathcal{S}}$ are kept since they survive the dead neuron test. The parameters $W_{\mathcal{R}\to\mathcal{R}}$ are randomly initialized and their scaling parameters γ's are restored to the initial level. In order for the $\mathcal{S}$ neurons to keep their mapping functions after the rejuvenation, $W_{\mathcal{R}\to\mathcal{S}}$ is set to 0. We also set $W_{\mathcal{S}\to\mathcal{R}}$ to 0 as this initialization does not affect the performances as the experiments suggest.

Neural rescaling Recall that in order to encourage the sparsity of the neurons, all neurons receive the same amount of penalty. This means that not only the dead neurons have small scaling values, some $\mathcal{S}$ neurons also have scaling values that are very small compared with $\gamma_{\max}$ of the corresponding layers. As experiments in Sec. 4.2 show, this is harmful for gradient-based training. Our solution is to rescale those neurons to the initial level, *i.e.*, $|\gamma_i'| = \max\{|\gamma_i|, \gamma_0\}\ \forall i$, where γ_0 is the initial value for γ. We do not change the sign of γ. Note that rescaling takes all neurons into consideration, including $\mathcal{S}$ neurons with large scaling values ($|\gamma| \geq |\gamma_0|$), $\mathcal{S}$ neurons with small scaling values ($|\gamma| < |\gamma_0|$) and dead neurons ($|\gamma| \approx 0$). After neural rescaling, we adjust the parameters to restore the original mappings. For $\mathcal{S}$ neurons, let $s_i = \gamma_i'/\gamma_i$. In order for $\mathcal{S}$ neurons to keep their original mapping functions, we divide the parameters that use them by s_i. Experiments show that this leads to performance improvements.

3.3. Training with Mixed Types of Neurons

Let us now focus on each individual layer. After neural rejuvenation, each layer will have two types of input neurons, $\mathcal{S}_{\text{in}}$ and $\mathcal{R}_{\text{in}}$, and two types of output neurons, $\mathcal{S}_{\text{out}}$ and $\mathcal{R}_{\text{out}}$. For simplicity, we also use them to denote the features of the corresponding neurons. Then, by the definition of convolution, we have

$$\begin{aligned} \mathcal{S}_{\text{out}} &= W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}} + W_{\mathcal{R}\to\mathcal{S}} * \mathcal{R}_{\text{in}} \\ \mathcal{R}_{\text{out}} &= W_{\mathcal{S}\to\mathcal{R}} * \mathcal{S}_{\text{in}} + W_{\mathcal{R}\to\mathcal{R}} * \mathcal{R}_{\text{in}} \end{aligned} \tag{5}$$

where $*$ denote the convolution operation. $W_{\mathcal{R}\to\mathcal{S}}$ is set to 0 in reinitialization; therefore, $\mathcal{S}_{\text{out}} = W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}}$ initially, which keeps the original mappings between $\mathcal{S}_{\text{in}}$ and $\mathcal{S}_{\text{out}}$. In this subsection, we discuss how to train W.

The training of W depends on how much the network needs the additional capacity brought by the rejuvenated neurons to fit the data. When $\mathcal{S}$ neurons do not need this additional capacity at all, adding $\mathcal{R}$ neurons by Eq. 5 may not help because $\mathcal{S}$ neurons alone are already able to fit the data well. As a result, changing training scheme is necessary in this case in order to utilize the additional capacity. However, when $\mathcal{S}$ neurons alone have difficulties fitting the data, the additional capacity provided by $\mathcal{R}$ neurons will ease the training. They were found to be useless previously either because of improper initialization or inefficient resource arrangement, but now are reinitialized and rearranged. We present the detailed discussions as below.

When $\mathcal{S}$ does not need $\mathcal{R}$ Here, we consider the situation where the network capacity is bigger than necessary, and $\mathcal{S}$ neurons alone are able to fit the training data well. An example is training networks on CIFAR [33], where most of the modern architectures can reach 99.0% training accuracy. When adding $\mathcal{R}$ neurons into the architecture as in Eq. 5, since $\mathcal{S}$ neurons have already been trained to fit the data well, the gradient back-propagated from the loss will not encourage any great changes on the local mapping $(\mathcal{S}_{\text{in}}, \mathcal{R}_{\text{in}}) \to (\mathcal{S}_{\text{out}}, \mathcal{R}_{\text{out}})$. Therefore, keep modeling the computation as Eq. 5 may result in $\mathcal{R}_{\text{in}}$ neurons being dead soon and $\mathcal{R}_{\text{out}}$ producing redundant features.

The cause of the above problem is the existence of cross-connections between $\mathcal{R}$ neurons and $\mathcal{S}$ neurons, which provides short-cuts to $\mathcal{R}$. If we completely remove them, *i.e.*,

$$\mathcal{S}_{\text{out}} = W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}} \quad \mathcal{R}_{\text{out}} = W_{\mathcal{R}\to\mathcal{R}} * \mathcal{R}_{\text{in}} \tag{6}$$

then $\mathcal{R}$ neurons are forced to learn features that are new and ideally different. We use NR-CR to denote Neural Rejuvenation with cross-connections removed.

When $\mathcal{S}$ needs $\mathcal{R}$ Here, we assume that the capacity of $\mathcal{S}$ alone is not enough for fitting the training data. One example is training small networks on ImageNet dataset [53]. In this case, it is desirable to keep the cross-connections to increase the capacity. Experiments in Sec. 4.2 compare the performances of a simplified VGG network [54] on ImageNet, and show that Neural Rejuvenation with cross-connections kept and removed both improve the accuracies, but keeping cross-connections improves more.

Cross-attention between $\mathcal{S}$ and $\mathcal{R}$ We continue the discussion where we assume $\mathcal{S}$ needs the capacity of $\mathcal{R}$ and we keep the cross-connections. Then according to Eq. 5, the outputs from $\mathcal{S}_{\text{in}}$ and $\mathcal{R}_{\text{in}}$ are added up for $\mathcal{S}_{\text{out}}$, *i.e.*

$$\mathcal{S}_{\text{out}} = W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}} + W_{\mathcal{R}\to\mathcal{S}} * \mathcal{R}_{\text{in}} \tag{7}$$

Since the assumption here is that the model capacity is insufficient for fitting the training data, it would be better if we can increase the capacity not only by rejuvenating dead neurons, but also by changing Eq. 7 to add more capacity without using any more parameters nor resulting in substantial increases of computations (if any) compared with the convolution operation itself. As $W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}}$ is fixed, we focus on $W_{\mathcal{R}\to\mathcal{S}} * \mathcal{R}_{\text{in}}$. One way to increase capacity

is to use second-order response transform [57]. The original second-order response transform is defined on residual learning [23] by adding a geometric mean, *i.e.*

$$y = x + F(x) \Rightarrow y = x + F(x) + \sqrt{x \cdot F(x)} \quad (8)$$

For our problem, although Eq. 7 does not have residual connections, the outputs $W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}}$ and $W_{\mathcal{R}\to\mathcal{S}} * \mathcal{R}_{\text{in}}$ are added up as in residual learning; therefore, we can add a similar response transform to Eq. 7. Instead of adding a geometric mean which causes training instability [57], we propose to use cross attentions as shown in Eq. 9.

$$\mathcal{S}_{\text{out}} = W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}} + 2 \cdot \sigma(W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}}) W_{\mathcal{R}\to\mathcal{S}} * \mathcal{R}_{\text{in}} \quad (9)$$

Here, $\sigma(\cdot)$ denotes the Sigmoid function. Symmetrically, we add cross attentions to the output of $\mathcal{R}_{\text{out}}$, *i.e.*

$$\mathcal{R}_{\text{out}} = W_{\mathcal{R}\to\mathcal{R}} * \mathcal{R}_{\text{in}} + 2 \cdot \sigma(W_{\mathcal{R}\to\mathcal{R}} * \mathcal{R}_{\text{in}}) W_{\mathcal{S}\to\mathcal{R}} * \mathcal{S}_{\text{in}} \quad (10)$$

We use NR-CA to denote NR with cross attentions.

4. Experiments

In this section, we will show the experimental results that support our previous discussions, and present the improvements of Neural Rejuvenation on a variety of architectures.

4.1. Resource Utilization

We show the resource utilization of training ResNet-50 and ResNet-101 on ImageNet in Figure 1 when the sparsity term is added to the loss. In the figure, we show the plots of the parameter utilization and validation accuracy of the models with respect to the number of training epochs. Training such a model usually takes 90 epochs when the batch size is 256 or 100 epochs when the batch size is 128 [28]. In all the experiments, the sparsity coefficient λ is initialized with 0, Δt is set to one epoch, $\Delta r = 0.01$ and $\Delta\lambda = 5 \times 10^{-5}$. T_r is set to 0.5 unless otherwise stated.

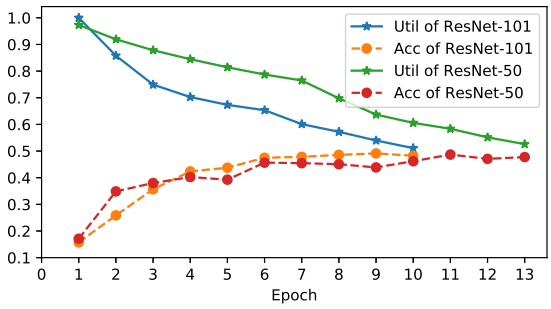

Figure 1. Parameter utilization and validation accuracy of ResNet-50 and ResNet-101 trained on ImageNet from scratch.

Fig. 1 shows two typical examples that convey the following important messages. (1) Training on large-scale dataset such as ImageNet cannot avoid the waste of the computational resources; therefore, our work is also valid for large-scale training. (2) It is easier to find dead neurons in larger models than in smaller models. This is consistent with our intuition that larger models increase the capacity and the risk of more resource wastes. (3) It does not take too long to reach the utilization threshold at 0.5. 10 epochs are enough for saving half of the resources for ResNet-101.

For ImageNet training, we set the neural rejuvenation flag on only for the first 30 epochs where the learning rate is 0.1. Since it usually takes 10-20 epochs for $r(\theta_{\mathcal{A}})$ to reach $T_r = 0.5$, there will be about 1 to 2 times that Step 8 in Algorithm 1 will get executed. To simplify the experiments, we only do one time of neural rejuvenation on ImageNet and reset the epoch counter to 0 afterwards. The training time with neural rejuvenation thus will be a little longer than the original training, but the increase will be less than 20% and experiments show that it is definitely worth it. For unlimited training time, Sec. 4.4 shows the performances on CIFAR with multiple times of Neural Rejuvenation.

4.2. Ablation Study on Neural Rejuvenation

To provide better understandings of Neural Rejuvenation applied on training deep networks, we present an ablation study shown in Table 1, which demonstrates the results of Neural Rejuvenation with different variations.

Method	Top-1	Top-5	Method	Top-1	Top-5
BL	32.13	11.97	BL-CA	31.58	11.46
NR-CR	31.40	11.53	NR-FS	31.26	11.37
NR	30.74	10.94	NR-BR	30.31	10.67
NR-CA	30.28	10.88	NR-CA-BR	29.98	10.58
NR-IP	31.35	11.45	NR+DSD	28.84	9.94

Table 1. Error rates of a simplified VGG-19 on ImageNet with $T_r = 0.25$ while maintaining the number of parameters. BL: baseline. BL-CA: baseline with cross attentions. NR-CR: NR with cross-connections removed. NR-FS: training $\mathcal{A}$ found by NR from scratch. NR: NR with cross-connections. NR-BR: NR with neural rescaling. NR-CA: NR with cross attentions. NR-CA-BR: NR with cross attentions and neural rescaling. NR-IP: NR without reallocation. NR+DSD: NR-CA-BR + DSD [19].

The network is a simplified VGG-19, which is trained on low-resolution images from ImageNet. The image size for training and testing is 128x128. We remove the last three fully-connected layers, and replace them with a global average pooling layer and one fully-connected layer. The resulted model has only 20.5M parameters. To further accelerate training, we replace the first convolutional layer with that in ResNet [23]. By applying all the changes, we can train one model with 4 Titan Xp GPUs in less than one day, which is fast enough for the purpose of ablation study.

Clearly, such a simplified model does not have sufficient capacity for fitting ImageNet. As we have discussed in

Architecture	Baseline				NR Params				NR FLOPs				Relative
	Params	FLOPs	Top-1	Top-5	Params	FLOPs	Top-1	Top-5	Params	FLOPs	Top-1	Top-5	Gain
DenseNet-121 [28]	7.92M	2.83G	25.32	7.88	8.22M	3.13G	24.50	7.49	7.28M	2.73G	24.78	7.56	-3.24%
VGG-16 [54]	37.7M	15.3G	24.26	7.32	36.4M	23.5G	23.11	6.69	21.5M	15.3G	23.71	7.01	-4.74%
ResNet-18 [23]	11.7M	1.81G	30.30	10.7	11.9M	2.16G	28.86	9.93	9.09M	1.73G	29.73	10.5	-4.75%
ResNet-34 [23]	21.8M	3.66G	26.61	8.68	21.9M	3.77G	25.77	8.10	20.4M	3.56G	25.45	8.04	-4.35%
ResNet-50 [23]	25.6M	4.08G	24.30	7.19	26.4M	3.90G	22.93	6.47	26.9M	3.99G	22.79	6.56	-6.21%
ResNet-101 [23]	44.5M	7.80G	22.44	6.21	46.6M	6.96G	21.22	5.76	50.2M	7.51G	20.98	5.69	-6.50%

Table 2. Error rates of deep neural networks on ImageNet validation set trained with and without Neural Rejuvenation. Each neural network has three sets of top-1 and top-5 error rates, which are baseline, Neural Rejuvenation with the number of parameters as the resource constraint (NR Params), and Neural Rejuvenation with FLOPs as resource constraint (NR FLOPs). The last column *Relative Gain* shows the best relative gain of top-1 error while maintaining either number of parameters or FLOPs.

Sec. 3.3, it is better to keep the cross connections for increasing the model capacity. As also demonstrated here, NR-CR improves the top-1 accuracy by 0.7% than the baseline, but is 0.7% behind NR where cross-connections are kept. We further show that cross attentions lower the top-1 error rates by roughly 0.5%, and neural rescaling further improves the accuracies. In the following experiments on ImageNet, we use NR-CA-BR for all the methods.

4.3. Results on ImageNet

Table 2 shows the performance improvements on ImageNet dataset [53]. ImageNet dataset is a large-scale image classification dataset, which contains about 1.28 million color images for training and 50,000 for validation. Table 2 lists some modern architectures which achieve very strong accuracies on such a challenging task. Previously, a lot of attention is paid to designing novel architectures that are more suitable for vision tasks. Our results show that in addition to architecture design and search, the current optimization technique still has a lot of room to improve. Our work focuses only on the utilization issues, but already achieves strong performance improvements.

Here, we briefly introduce the setting of the experiments for easy reproduction. All the models are trained with batch size 256 if the model can fit in the memory; otherwise, we set the batch size to 128. In total, we train the models for 90 epochs when the batch size is 256, and for 100 epochs if the batch size is 128. The learning rate is initialized as 0.1, and then divided by 10 at the 31^{st}, 61^{st}, and 91^{st} epoch.

For our task, we make the following changes to those state-of-the-art architectures. For VGG-16 [54], we add batch normalization layers after each convolutional layer and remove the last three fully-connected layers. After that, we add two convolutional layers that both output 4096 channels, in order to follow the original VGG-16 that has two fully-connected layers outputting the same amount of channels. After these two convolutional layers, we add a global average pooling layer, and a fully-connected layer that transforms the 4096 channels to 1000 channels for image classification. The resulted model has fewer number of parameters (138M to 37.7M), but with a much lower top-1 error rate (27 to 24.26). All the VGG-16 layers receive the sparsity penalty. For ResNet [23], all the convolutional layers except the ones that are added back to the main stream are taken into the consideration for neural rejuvenation. For DenseNet [28], due to the GPU memory and speed issue, we are only able to run DenseNet with 121 layers. We change it from pre-activation [24] to post-activation [23] to follow our assumption that each convolutional layer is directly followed by a batch normalization layer. This change yields a similar accuracy to the original one.

A quick observation of our results is that the models with stronger capacities actually have better improvements from Neural Rejuvenation. This is consistent with our discussion in Sec. 3.3 and the observation in Sec. 4.1. For large-scale tasks, the model capacity is important and larger models are more likely to waste more resources. Therefore, rejuvenating dead neurons in large models will improve more than doing that in small models where the resources are better utilized. In all models, DenseNet-121 is the hardest to find dead neurons, and thus has the smallest improvements. This may explain the model compactness discussed in their paper [28]. Moreover, VGG-16 with NR achieves 23.71% top-1 error with just 21.5M parameters, far better than [40] which achieves 36.66% top-1 error with 23.2M.

Architecture	BL [16]	MN [16]	BL*	NR
MobileNet-0.50	42.9	41.9	41.77	40.12
MobileNet-0.25	55.2	54.1	53.76	51.94

Table 3. Top-1 error rates of MobileNet [27] on ImageNet. The image size is 128x128 for both training and testing. The FLOPs are maintained in all the methods. BL: the baseline performances reported in [16], MN: MorphNet [16], BL*: our implementation of the baseline, and NR: Neural Rejuvenation.

Next, we show experiments on MobileNet-0.5 and 0.25 in Table 3. They are not included in Table 2 because their image size is 128x128 and the learning rate follows the

Architecture	Baseline		Network Slimming [40]		Neural Rejuvenation	
	C10 (Params)	C100 (Params)	C10 (Params)	C100 (Params)	C10 (Params)	C100 (Params)
VGG-19 [54]	5.44 (20.04M)	23.11 (20.08M)	5.06 (10.07M)	24.92 (10.32M)	4.19 (9.99M)	21.53 (10.04M)
ResNet-164 [23]	6.11 (1.70M)	28.86 (1.73M)	5.65 (0.94M)	25.61 (0.96M)	5.13 (0.88M)	23.84 (0.92M)
DenseNet-100-40 [28]	3.64 (8.27M)	19.85 (8.37M)	3.75 (4.36M)	19.29 (4.65M)	3.40 (4.12M)	18.59 (4.31M)

Table 4. Neural Rejuvenation for model compression on CIFAR [33]. In the experiments for ImageNet, the computational resources are kept when rejuvenating dead neurons. But here, we set the resource target of neural rejuvenation to the half of the original usage. Then, our Neural Rejuvenation becomes a model compressing method, and thus can be compared with the state-of-the-art pruning method [40].

cosine learning rate schedule starting from 0.1 [49]. MobileNet is designed for platforms with low computational resources. Our NR outperforms the previous method [16] and shows very strong improvements.

4.4. Results on CIFAR

The experiments on CIFAR have two parts. The first part is to use Neural Rejuvenation as a model compression method to compare with the previous state-of-the-arts when the model sizes are halved. The results are shown in Table 4. In the second part, we show the performances in Table 5 where we do Neural Rejuvenation for multiple times.

Model compression Table 4 shows the performance comparisons on CIFAR-10/100 datasets [33]. CIFAR dataset is a small dataset, with 50,000 training images and 10,000 test images. Unlike our experiments on ImageNet, here, we do not rejuvenate dead neurons to utilize all the available computational resource; instead, we set the resource target to $0.5 \times \mathcal{C}$ where $\mathcal{C}$ is the original resource constraint. In practice, this is done by setting $T_r = 0.25$ and rejuvenating the models to the level of $0.5 \times \mathcal{C}$. As a result, Neural Rejuvenation ends up training a model with only a half of the parameters, which can be compared with the previous state-of-the-art network pruning method [40].

Multiple NR Table 5 shows the performances of VGG-19 tested on CIFAR datasets without limiting the times of Neural Rejuvenation. The improvement trends are clear when the number of Neural Rejuvenation increases. The relative gains are 33.5% for CIFAR-10 and 13.8% for CIFAR-100.

# of NR	0	1	2	3	4	5
C10	5.44	4.19	4.03	3.79	3.69	3.62
C100	23.11	21.53	20.47	19.91	—	—

Table 5. Error rates of VGG-19 on CIFAR-10 (C10) and CIFAR-100 (C100) with different times of Neural Rejuvenation while maintaining the number of parameters.

Here, we introduce the detailed settings of the experiments. For VGG-19, we make the following changes because the original architecture is not designed for CIFAR. First, we remove all the fully-connected layers and add a global average pooling layer after the convolutional layers which is then followed by a fully-connected layer that produces the final outputs. Then, we remove the original 4 max-pooling layers and add 2 max-pooling layers after the 4^{th} and the 10^{th} convolutional layers for downsampling. These changes adapt the original architecture to CIFAR, and the baseline error rates become lower, *e.g.* from 6.66 to 5.44 on CIFAR-10 and from 28.05 to 23.11 on CIFAR-100. We make the same changes to DenseNet as for ImageNet. For ResNet-164 with bottleneck blocks, similar to our settings on ImageNet, we only consider the neurons that are not on the mainstream of the network for Neural Rejuvenation. Our method is NR-CR, which removes all the cross-connections. Table 4 shows that our Neural Rejuvenation can be used for training small models as well. Table 5 presents the potential of VGG-19 when trained with multiple times of Neural Rejuvenation. While maintaining the number of parameters, Neural Rejuvenation improves the performances by a very large margin.

5. Conclusion

In this paper, we study the problem of maximizing the resource utilization. To this end, we propose a novel method named Neural Rejuvenation, which rejuvenates dead neurons during training by reallocating and reinitializing them. Neural rejuvenation is composed of three components: resource utilization monitoring, dead neuron rejuvenation and training schemes for networks with mixed types of neurons. These components detect the liveliness of neurons in real time, rejuvenate dead ones when needed and provide different training strategies when the networks have mixed types of neurons. We test neural rejuvenation on the challenging datasets CIFAR and ImageNet, and show that our method can improve a variety of state-of-the-art network architectures while maintaining either their numbers of parameters or the loads of computations. In conclusion, Neural Rejuvenation is an optimization technique with a focus on the resource utilization, which improves the training of deep neural networks by enhancing the utilization.

Acknowledgements We gratefully acknowledge supports from NSF award CCF-1317376, a gift from Adobe, and a gift from YITU. SQ also thanks Wanyu Huang for support.

References

[1] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

[2] J. Ba and R. Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[3] J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*, 2014.

[4] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[6] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient architecture search by network transformation. AAAI, 2018.

[7] M. Caron, P. Bojanowski, A. Joulin, and M. Douze. Deep clustering for unsupervised learning of visual features. In *arXiv preprint arXiv:1807.05520*, 2018.

[8] K. Chen, J. Wang, L.-C. Chen, H. Gao, W. Xu, and R. Nevatia. Abc-cnn: An attention based convolutional neural network for visual question answering. *arXiv preprint arXiv:1511.05960*, 2015.

[9] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *International Conference on Learning Representations*, 2015.

[10] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[11] 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*, pages 1269–1277, 2014.

[12] Y. Dong and E. J. Nestler. The neural rejuvenation hypothesis of cocaine addiction. *Trends Pharmacol Sci*, 35(8):374–383, Aug 2014.

[13] T. Elsken, J.-H. Metzen, and F. Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

[14] J. Frankle and M. Carbin. The lottery ticket hypothesis: Training pruned neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[15] S. D. Goggin, K. M. Johnson, and K. E. Gustafson. A second-order translation, rotation and scale invariant neural network. In *Advances in neural information processing systems*, pages 313–319, 1991.

[16] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[17] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016*, pages 243–254. IEEE, 2016.

[18] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[19] S. Han, J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran, et al. Dsd: Dense-sparse-dense training for deep neural networks. *arXiv preprint arXiv:1607.04381*, 2016.

[20] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[21] Y. Hao, Y. Zhang, K. Liu, S. He, Z. Liu, H. Wu, and J. Zhao. An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge. In *Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 221–231, 2017.

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

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

[24] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *ECCV*, 2016.

[25] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 1398–1406, 2017.

[26] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[27] 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.

[28] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2017.

[29] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, 2015.

[30] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[31] A. Joulin, L. van der Maaten, A. Jabri, and N. Vasilache. Learning visual features from large weakly supervised data. In *ECCV*, pages 67–84. Springer, 2016.

[32] A. Kazemy, S. A. Hosseini, and M. Farrokhi. Second order diagonal recurrent neural network. In *Industrial Electronics, ISIE 2007.*, pages 251–256. IEEE, 2007.

[33] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[34] 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.

[35] V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2554–2564. IEEE, 2016.

[36] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[37] K.-H. Lee, X. Chen, G. Hua, H. Hu, and X. He. Stacked cross attention for image-text matching. *arXiv preprint arXiv:1803.08024*, 2018.

[38] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[39] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part I*, pages 19–35, 2018.

[40] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2755–2763, 2017.

[41] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, 2017.

[42] J. Mao, W. Xu, Y. Yang, J. Wang, and A. L. Yuille. Deep captioning with multimodal recurrent neural networks (m-rnn). *CoRR*, abs/1412.6632, 2014.

[43] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

[44] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[45] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[46] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[47] S. Qiao, C. Liu, W. Shen, and A. L. Yuille. Few-shot image recognition by predicting parameters from activations. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2018.

[48] S. Qiao, W. Shen, W. Qiu, C. Liu, and A. L. Yuille. Scalenet: Guiding object proposal generation in supermarkets and beyond. In *2017 IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29*, 2017.

[49] S. Qiao, W. Shen, Z. Zhang, B. Wang, and A. Yuille. Deep co-training for semi-supervised image recognition. In *European Conference on Computer Vision*, 2018.

[50] S. Qiao, Z. Zhang, W. Shen, B. Wang, and A. L. Yuille. Gradually updated neural networks for large-scale image recognition. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2018.

[51] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[52] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

[53] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[54] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[55] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[56] 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 conference on computer vision and pattern recognition*, pages 1–9, 2015.

[57] Y. Wang, L. Xie, C. Liu, S. Qiao, Y. Zhang, W. Zhang, Q. Tian, and A. Yuille. SORT: Second-Order Response Transform for Visual Recognition. *IEEE International Conference on Computer Vision*, 2017.

[58] Y. Wang, L. Xie, S. Qiao, Y. Zhang, W. Zhang, and A. L. Yuille. Multi-scale spatially-asymmetric recalibration for image classification. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[59] 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*, pages 2074–2082, 2016.

[60] H. Xu and K. Saenko. Ask, attend and answer: Exploring question-guided spatial attention for visual question answering. In *European Conference on Computer Vision*, pages 451–466. Springer, 2016.

[61] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[62] L. Xu, J. Neufeld, B. Larson, and D. Schuurmans. Maximum margin clustering. In *Advances in neural information processing systems*, pages 1537–1544, 2005.

[63] C. Yang, L. Xie, S. Qiao, and A. Yuille. Knowledge distillation in generations: More tolerant teachers educate better students. *AAAI*, 2018.

[64] J. Ye, X. Lu, Z. L. Lin, and J. Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *CoRR*, abs/1802.00124, 2018.

[65] 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. *Preprint at https://arxiv. org/abs/1711.05908*, 2017.

[66] Z. Zhang, S. Qiao, C. Xie, W. Shen, B. Wang, and A. L. Yuille. Single-shot object detection with enriched semantics. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 5813–5821, 2018.

[67] Z. Zhong, J. Yan, and C.-L. Liu. Practical network blocks design with q-learning. *arXiv preprint arXiv:1708.05552*, 2017.

[68] H. Zhou, J. M. Alvarez, and F. Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, pages 662–677. Springer, 2016.

[69] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[70] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2(6), 2017.