This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

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.

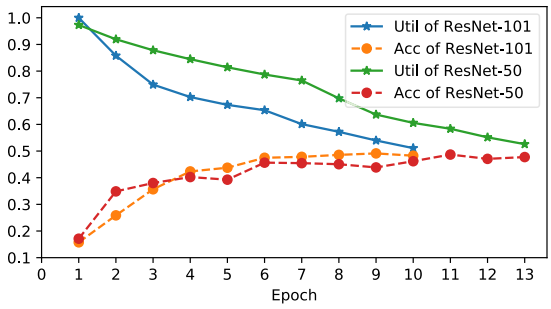

Figure 1. Parameter utilization and validation accuracy of ResNet-50 and ResNet-101 trained on ImageNet from scratch.

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.

CVPR2019 Paper Translation

姓名：贾沛东

学号：2017303072

班号：10011708

神经再生: 通过提高资源利用率来改善深度网络训练

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

摘要

在这篇论文中，我们探索了提高神经网络计算资源利用率的问题。深度的神经网络总是在任务中引入过多参数去达到一个很好的表现，因此更倾向于降低了计算资源的利用率。这个发现激发了很多的研究话题，例如网络修剪，结构搜索等等。因为有着更高的计算花费的模型（比如说更多的参数或者说更多的计算力）经常会有更好的效果，我们探究了提高神经网络的资源利用率的问题从而使他们的潜质被更远地实现。最终，我们提出了一种新颖的优化算法，叫做神经再生。就像这个名字所暗示的那样，我们的方法检测死神经元并且实时地进行计算资源利用率，通过对资源的重新分配和重新初始化来进行重生死的神经元，之后用新的训练方案训练他们。通过利用神经再生简单地替换标准的优化器，当我们使用类似的训练技巧并且保持他们的初始资源使用时，我们能够很大幅度地提高神经网络的表现。代码可以在这里获得：https://github.com/joe-siyuan-qiao/NeuralRejuvenation-CVPR19。

1. 引言

深度网络在很多视觉任务中达到了最好的表现 [9, 22, 41, 46]。在例如ImageNet [52] 分类这样的大型任务中，一个共性的发现是，拥有更多参数或者有更多计算力,往往趋于达到更好的结果。例如，DenseNet将验证错误率绘制为参数和计算力的函数，并且展示出随着模型大小增加时准确率相应地提升。这与大型任务要求足够能力的模型来很好地拟合数据是一致的。结果是，当训练更大的模型时若附加的计算资源被合理地使用是很有益处的。然而，之前的网络修剪工作 [39, 63]已经表示许多被SGD训练过的网络有着不充分的资源利用率。举例来说，在CIFAR [32]数据集上训练的VGG网络 [53]的参数的数量可以压缩百分之10而不影响他的准确度 [39]。如此低的利用率导致了训练和测试时间的浪费，并且限制模型达到其最好的潜力。为了解决这个问题，我们探索了新颖的神经网络训练和优化技术来提高资源利用率以及准确率。

正式地讲，这篇论文研究了以下的优化问题。我们给出了一个在从数据集$\mathcal{D}$获得的数据(x,y)上定义的损失函数$\mathcal{L}(f(x;\mathcal{A},a_{\mathcal{A}}),y)$，以及一个计算资源约束$\mathcal{C}$。其中，$f(x;\mathcal{A},\theta_{\mathcal{A}})$是有着$\mathcal{A}$架构并有着参数$\theta_{\mathcal{A}}$的神经网络。让$c(\mathcal{A})$表示在函数f中使用架构$\mathcal{A}$的花费，例如$c(\mathcal{A})$可以是$\mathcal{A}$的参数数量或者是他的计算力。我们的任务是去寻找一个架构$\mathcal{A}$，并且它的参数$\theta_{\mathcal{A}}$使得其能达到在数据集$\mathcal{D}$上在资源约束$\mathcal{C}$下的最小平均损失$\mathcal{L}$，例如，

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

架构$\mathcal{A}$通常被研究者们设计并且在最小化 Eq. 1时调整。从而使得解法$\mathcal{A}$，$\theta_{\mathcal{A}}$往往能满足资源约束。当$\mathcal{A}$被调整时，被标准的基于梯度的优化器发现的$\theta_{\mathcal{A}}$可能会有神经元（例如通道）对平均的损失影响很小，所以在保持好的表现并且能节省资源时会被移除。换句话说，$\theta_{\mathcal{A}}$可能没有完全利用$\mathcal{A}$中可以利用的全部资源。让$\mathcal{U}(\theta_{\mathcal{A}})$表示$\mathcal{A}$中基于$\theta_{\mathcal{A}}$的实际计算资源的花费，这可以通过在移除对输出影响小的死神经元时

*在Adobe实习期间完成的工作.

被测量出来。很显然，$\mathcal{U}(\theta_{\mathcal{A}}) <= c(\mathcal{A})$。之前的工作显示，被标准的SGD训练的利用率$r(\theta_{\mathcal{A}}) = \mathcal{U}(\theta_{\mathcal{A}}/c(\mathcal{A})$只能达到很低的11.5%。

较低的利用率推动着关于网络修剪的研究，例如，从$\mathcal{A}$中提取的有效子网络$\mathcal{A}'$使得$c(\theta_{\mathcal{A}'} = \mathcal{U}(\theta_(\mathcal{A}))$。虽然利用率$r(\theta_(\mathcal{A}'))$是很高的，但是这和我们的问题是相反的因为它尝试着去通过使得$c(\mathcal{A}$靠近$\mathcal{U}(\mathcal{A}$从而减少$c(\mathcal{A}$和$\mathcal{U}(\mathcal{A}$的差距。相反地，我们的目的是设计一个优化过程$\mathcal{P}$使得我们能够找到有着高的$r(\ theta_(\mathcal{A}'))$的参数$\theta_(\mathcal{A}) = \mathcal{P}(\mathcal{A}, \mathcal{L}, \mathcal{D})$。换句话说，我们尝试着使得$\mathcal{U}(\mathcal{A}$靠近$c(\mathcal{A}$，这可以在满足约束$\mathcal{C}$的条件下最大化实际的利用率。

有很多原因造成很低的利用率$r(\theta_(\mathcal{A}'))$。一个原因是较差的初始化，这可以通过对剩余的资源的参数重新初始化得到缓解。另外一个是不高效的资源分配，例如通道的数量或者是块的数量可能不会被合适地配置以适应他们真实的需求。不像之前训练很多网络来搜索结构的方法，我们目标是设计一个只训练一个网络一次的优化器，它能够包括资源的重新初始化和重新分配，从而能够最大化资源的利用率。

在这篇论文中，我们提出了一个叫做神经再生的优化方法，为了在训练中提高资源的利用率。我们的方法是直观简单的。在训练中，当有一些神经元被发现是无用的(例如对结果有着很小的影响)，我们复活他们，为了重新初始化并且分配到最需要他们的地方。从神经科学的角度来讲，这就是重生死的神经元通过把他们重新用到函数中-来自它的名字。神经再生的挑战是很清晰的。第一个，我们需要一个实时的资源利用监测器。第二个，当我们重生死的神经元时，我们需要知道如何重新初始化他们并且在哪里重新放置他们。最后一个，在死的神经元重置以后,幸存的神经元($\mathcal{S}$个神经元）以及被重生的神经元($\mathcal{R}$个神经元)被混合，并且目前如何去训练他们的网络是不明晰的。

我们的解决方法是一个插入并训练的优化器，关于这个的代码将会被公开。在外壳之下，它是基于标准的基于梯度的优化器的，但是增加了其他功能，包括实时的资源利用率的监控，死的神经元的重生，以及对有着混合类型的神经元的网络的设计的新的训练方式。我们将在下面介绍这些组件。

资源利用率监控 和[40,64]类似，我们使用神经元的激活量表来识别被利用的和空闲的计算资源，并且在训练过程中计算实时的利用率$r(\theta_(\mathcal{A}))$。当$r(\theta_(\mathcal{A}))$小于一个阈值$\mathcal{T}_\gamma$时会触发一个事件，并且死神经元的重生的过程会在下一步的训练步骤前获得控制权，在这个之后$r(\theta_(\mathcal{A}))$会慢慢回到1。

重生死神经元 这个组件通过收集不用的资源并把它们重新放回$\mathcal{A}$来重生死神经元。类似于MorphNet[16]，更多的空余的资源被重新分配给有更多$\mathcal{S}$神经元的层。然而，不像MorphNet，再重新分配后从scratch重新训练整个网络，我们只是重新初始化死神经元然后继续训练。通过利用死神经元的重新初始化以及我们的训练方案的优势，我们的优化器可以只训练模型一次，并且拥有MorphNet发现的很多结构上表现更好的网络。

训练混合类型的神经元 在死神经元复活后，每一层都将有两个类型的神经元：$\mathcal{S}$和$\mathcal{R}$两个类型的神经元。当训练拥有混合类型的神经元的网络时，我们为不同的情况提出了两种新颖的训练方案。第一是在$\mathcal{S}$和$\mathcal{R}$神经元中移除交叉连接，第二个是使用在它们之间使用交叉注意来提高网络能力。3.3小节展示了详细的讨论。

我们在两个通用的图像识别数据集上对神经再生进行了评估。例如，CIFAR-10/100以及ImageNet，并且结果显示他的表现很大幅度地超出基本的优化器。例如，我们降低了在ImageNet上排名第一的ResNet-50的错误率1.51%,在保证MobileNet-0.25的FLOPs的情况下降低了1.82%。我们在CIFAR上满足一半约束的重生资源，并且和之前最好的压缩方法进行对比，我们的效果在CIFAR-10上超出了0.87%,在CIFAR-100上超出了3.39%。

2. 相关工作

神经网络的效率 这被广泛地意识到因为在神经网络过度参数化来赢得　lter lottery tickets。这个效率问题被多种方法解决，包括权重量化，低秩近似，先验蒸馏以及网络修剪。最相关的方法是网络修剪，发现了子网络对结果影响更大。网络修建有几个研究方向，比如说权重修剪，结构修剪，等等。权重修剪注重于独立的权重，但是要求贡献的硬件和软件实施来达到压缩和加速，结构修剪表示从结构中去除通道和层，从而能够直接实现加速而不需要其他特殊处理。以下，我们通过对缩放因子强加惩罚项来鼓励渠道稀疏性。

不同于之前的那些方法，神经再生从一个新的角度探索了效率问题：我们致力于通过重新利用空闲计

算资源的方式来最大化利用率。作为彩票假设[14]的类比，神经再生就像获得无用的门票退款然后购买新门票一样。

交叉注意 在这个工作中,我们提出了去利用交叉注意力在不引入附加花费的前提下来提高网络的能力。这被在多分支网络增加二阶变换所驱动。我们提出使用交叉注意作为二阶变换条件而不是使用几何平均数来提高能力。在深度神经网络上注意力模型已经在很多图像和语言任务中被广泛使用，比如说物体检测，机器翻译，视觉问题回答，图像翻译，等等。不像之前的注意力机制，我们的方法是使用一组通道来为别的通道生成注意力，并且我们的注意力机制主要是为了提高能力。

架构搜索 我们被Eq.1制定的目标接近于通过在提前定义好的空间搜索架构$\mathcal{A}$来解决问题的神经架构搜索，所以他们需要训练大量的网络来寻找优化的结构。例如，NAS使用强化学习来寻找架构，通过使用更有条理的搜索空间来扩展它，并且通过逐步发现框架来提高搜索效率。但是计算的花费是很高的，比如使用2000块GPU好几天。还有好几种方法注重搜索问题。不同于架构搜索，神经再生不是通过训练成千上百的模型来搜索架构$\mathcal{A}$,虽然它确实稍微地改变了架构。相反地，我们的方法是只训练模型一次的优化方法。最相近的一个方法是MorphNet,我们都使用了线性扩展技术来探索资源分配。然而，它依然需要多个训练过程，并且它既不重生死神经元也不重新使用部分被训练的核。我们显示了和它直接的对比，并且大幅度地超过它的表现。

参数重新初始化 参数重新初始化是在优化器方面避免无效计算和提高性能的一个通用策略。例如，在k-means优化器阶段，空的簇会被自动地重新分配，并且大的簇会被激励分割成小的簇。我们的方法也会让人联想到这一点，因为它也会检测到不满意的组件并且重新初始化他们使得他们更加满足任务。

3. 神经再生

算法1展示了增加两个新模块的基本的神经再生框架：资源利用率监控（第六步）以及一个标准SGD优化器的死神经元再生（第七和第八步）。这个训练方案并不在这里显示，它将会在3.3小节被讨论。我们在一个预先定义的时间间隔定期设置神经再生标志用于监测利用率以及什么时候需要重生死神经元。在接下来的小节里面，我们会展示怎么实施每个组件。

3.1. 资源利用率监测

3.1.1 神经元的活泼

我们考虑一个每个卷积层都会跟着一个批归一化层的卷积神经网络。如果批归一化是不可行的，一个拥有可学习参数的仿射变换层也是允许的。对于每个拥有批归一化的卷积层，让$\mathcal{B} = \{u_1, ..., u_m\}$在卷积之后作为最小批的值。然后它的归一化输出$\{v_1, ..., v_m\}$是

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

每一个神经元（例如说通道）在卷积层中有着它自己的学习尺度参数γ，用来我们估计相应神经元的活泼性。正如我们的实验所表明的，如果通道的尺度参数γ小于$0.01 \times \gamma_{max}$,γ_{max}是在同一个批归一化处理后的卷积层的最大的γ，移除它对结果{以及损失$\mathcal{L}$影响很小。所以，这篇论文中所展示的所有实验中，一个神经元被认为是死的当他的尺度参数$\gamma < 0.01 \times \gamma_{max}$。使得$\mathcal{T}$作为架构$\mathcal{A}$中的所有尺度参数的集合。与[40]类似，我们在$\mathcal{T}$加入了$L1$惩罚参数为了激励神经元的稀疏性，例如，与给定一个损失函数$\mathcal{L}$不同,我们最小化下述的损失

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

，其中λ是一个超参数。

3.1.2 通过前向传播来计算$\nabla(\theta_(\mathcal{A}))$

这一小节，我们展示如何实时地基于神经元的活泼性来计算利用率$\nabla(\theta_(\mathcal{A}))$。我们通过类似于函数{的分离的前向传播来计算$\nabla(\theta_{\mathcal{A}})$。这个为了$\nabla(\theta_{\mathcal{A}})$前向传播的计算花费相对于{来说是微不足道的。我们首先重新定义函数f:

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

其中f_i是架构$\mathcal{A}$中的第i层。当计算$r(\theta_{\mathcal{A}})$时，与把从x计算的一个层的输出传递给下一层作为输入，而是每

Algorithm 1: 带有神经再生的SGD

Input : 学习率 ϵ, 利用率阈值 T_r, 初始化架构 $\mathcal{A}$ 和 $\theta_{\mathcal{A}}$, 以及资源约束 $\mathcal{C}$

while 停止标准不满足**:**
  采样最小批 $\{(x_1, y_1), ..., (x_m, y_m)\}$;
  计算梯度 $g \leftarrow \frac{1}{m}\nabla \sum_i \mathcal{L}(f(x_i; \mathcal{A}, \theta_{\mathcal{A}}), y_i)$;
  应用更新 $\theta_{\mathcal{A}} = \theta_{\mathcal{A}} - \epsilon \cdot g$;
  if 神经再生标志为真**:**
    计算资源利用率 $r(\theta_{\mathcal{A}})$;
    if $r(\theta_{\mathcal{A}}) < T_r$**:**
      再生死神经元并且在资源约束$\mathcal{C}$的条件下获得新的$\mathcal{A}$ 和 $\theta_{\mathcal{A}}$;
return架构 $\mathcal{A}$ 以及它的参数 $\theta_{\mathcal{A}}$;

一层f_i会输出一个二值化的掩膜用来说明它的神经元的活泼性。让M_i^{in}表示f_i层输入神经元的二值化掩膜。M_i^{in}表示他自己神经元的二值化掩膜。然后f_i有效的参数是$||M_i^{\text{in}}||_1 \cdot ||M_i^{\text{out}}||_1 \cdot K_w \cdot K_h$，如果$f_i$是有一组的卷积层并且没有偏差,，那么它的计算花费根据[23]可以通过$||M_i^{\text{in}}||_1 \cdot ||M_i^{\text{out}}||_1 \cdot K_w \cdot K_h \cdot O_w \cdot O_h$计算获得。这里$K_w$和$K_h$是核的尺寸，$O_w$和$O_h$是输出的尺寸。注意花费f是所有层$f_i$的花费之和；因此，我们可以在前向传播中传递有效的计算花费以及初始的花费。在这之后，我们就能够计算$U(\theta_{\mathcal{A}})$ 和 $c(\theta_{\mathcal{A}})$，于是就有$r(\theta_{\mathcal{A}})$。在计算$r(\theta_{\mathcal{A}})$的过程中，每一层都保留了一个它的及之前层的神经元的复制。一旦出现$r(\theta_{\mathcal{A}}) < T_r$这一信息就会在死神经元复活的步骤中用到。它也记录了输入神经元的尺度参数γ的值。这会在3.2小节讨论的神经重调整中用到。

3.1.3 适应性惩罚系数λ

利用率$r(\theta_{\mathcal{A}})$将取决于稀疏参数λ的值因为一个更大的λ更倾向于造成一个更加稀疏的网络。当$\lambda = 0$时，所有的神经元有可能都保持存活，因为我们有一个大致的阈值$0.01 \times \gamma_{max}$。结果是，算法1的第七步和第八步都不会得到执行，我们的优化器将表现得像一个标准的优化器。当λ更大，我们优化的真实的损失函数$\mathcal{L}_\lambda$将远远不想像原本的损失 $\mathcal{L}$。所以，表现会更满意。所以选择一个合适的λ对我们的问题来说是至关重要的，并且我们希望它是根据任务和架构自动的和优化的。

在神经再生中，λ的值动态取决于利用率$r(\theta_{\mathcal{A}})$的趋势。特别的，但神经再生的标志为真时，我们会对经过t次迭代的利用率$r(\theta_{\mathcal{A}})^t$进行记录。经过$\Delta t$次迭代后，我们会将目前的$r(\theta_{\mathcal{A}})^t$和之前的$r(\theta_{\mathcal{A}})^{t-\Delta t}$进行比较。如果$r(\theta_{\mathcal{A}})^t < r(\theta_{\mathcal{A}})^{t-\Delta t} - \Delta r$，我们会保持目前的$\lambda$;否则我们会在$\lambda$上加上$\Delta\lambda$。在这里，$\Delta t$, Δr 以及$\Delta\lambda$都是超参数。λ初始化为0，在第八步执行后，λ重新置为0。

以上述方式设置λ而不是在整个训练过程都是一个固定的值有益处的。首先，不同的任务和框架可能需要不同的λ。上述的方法可以将我们从基于不断试错中手动选择一个中释放出来。其次，进入步骤8的迭代次数是有界的。这是因为λ足够大之后，每个时间间隔Δt都会至少减少Δr的利用率。因此，达到T_r的迭代次数的界限是$(1 - T_r)/\Delta r + O(1)$。用一句话来说，这个策略自动地选择了$\lambda$的值，并且保证了在有限步的训练迭代次数下会达到条件$r(\theta_{\mathcal{A}}) < T_r$。

3.2. 死神经元重生

在检测到神经元的活泼性并且满足条件$r(\theta_{\mathcal{A}}) < T_r$，我们会执行算法1的第八步。在这一步，我们的目标是在满足资源约束$\mathcal{C}$的条件下重生死神经元以及重新把重生的神经元分配到最需要的地方。在重生死神经元中有三个主要步骤。我们将按照顺序展示他们。

资源重分配 第一步就是对被移除所有死神经元而保留的计算资源重新分配。移除操作将计算花费从$c(\mathcal{A})$ 降到了 $U(\theta_{\mathcal{A}})$；因此有着$c(\mathcal{A}) - U(\theta_{\mathcal{A}})$的可用于重新分配的资源。最主要的问题是将空闲的资源重新返回$\mathcal{A}$的哪里。让w_i 表示在f中输出通道的层的数量, 并且 w_i 会因为死神经元的移除而减少到 w_i'。让$\mathcal{A}'$ 表示在通过移除f_i层的w_i'输出通道死神经元之后的架构。然后，$c(\mathcal{A}') = U(\mathcal{A})$。为了把$\mathcal{A}'$的计算花费等级提升到$\mathcal{A}$的水平，我们的资源重新分配会根据一个共享的扩展率$\dashv$在所有的层$f_i$来线性扩展$w_i'' = \alpha \cdot w_i'$。带着一个通道数为$w_i''$来建立一个新的架构$\mathcal{A}''$。这里有一个假设是，如果有一个层有更高比例的活神经元，那么这个层需要更多的资源， 例如，更多的输出层；相反地，如果有一个层有更低的比例，那么意味着这一层的资源分配过多。这一假设被一个共享的线性扩张率α所建模。

这里使用到的资源重新分配和为了神经结构搜索

的MorphNet中的迭代挤压-扩展算法类似。但是区别也是明显的。神经再生模型包括了死神经元的重新初始化和重新分配，以及只训练网络一次的训练模式，但是MorphNet只对每一层的通道数感兴趣，而这将通过从头训练并且训练很多网络而被优化。

参数重新初始化 第二步是重新初始化被重新分配的神经元。让$\mathcal{S}_{\text{in}}$ 和 $\mathcal{R}_{\text{in}}$分别表示输入的S(幸存的)神经元以及R(重生的)神经元，$\mathcal{S}_{\text{out}}$ 和 $\mathcal{R}_{\text{out}}$分别表示输出的S和R神经元。然后参数W可以被分成四组：$W_{\mathcal{S}\to\mathcal{S}}$，$W_{\mathcal{S}\to\mathcal{R}}$,$W_{\mathcal{R}\to\mathcal{R}}$,$W_{\mathcal{R}\to\mathcal{S}}$,这些分别对应于从$\mathcal{S}_{\text{in}}$到$\mathcal{S}_{\text{out}}$,从$\mathcal{S}_{\text{in}}$到$\mathcal{R}_{\text{out}}$,从$\mathcal{R}_{\text{in}}$到$\mathcal{R}_{\text{out}}$并且从$\mathcal{R}_{\text{in}}$到$\mathcal{S}_{\text{out}}$的参数。在重新初始化的过程中，参数$W_{\mathcal{S}\to\mathcal{S}}$将会被保持因为他们从死神经元的测试中幸存。参数$W_{\mathcal{R}\to\mathcal{R}}$将会被随机地初始化并且它们的缩放参数$\gamma$将会根据他们的初始化等级重新存储。为了使得$S$神经元保持他们的映射功能，$W_{\mathcal{R}\to\mathcal{S}}$会被设置为0。我们也会把$W_{\mathcal{S}\to\mathcal{R}}$设置成0因为正如实验所显示的这个的初始化并不会影响表现。

神经重新缩放 回想一下，为了鼓励神经元的稀疏性，所有神经元都会受到相同的惩罚。这意味着不仅是死神经元有着少量的缩放值，一些S神经元也有相对于相应层的Γ_{max}很小的缩放值。正如4.2小节中实验所显示的，这对于基于梯度的训练是有害的。我们的解决方法是根据初始的等级重新缩放这些神经元，例如，$|\gamma_i'| = \max\{|\gamma_i|, \gamma_0\}\ \forall i$, 其中$\gamma_0$ 是γ的初始值。我们并不改变γ的符号。注意那种缩放是把所有的神经元都考虑进，包括有很大缩放值（$|\gamma| \geq |\gamma_0|$）的$\mathcal{S}$神经元，有很小的缩放值（$|\gamma| < |\gamma_0|$）的$\mathcal{S}$神经元,以及死的神经元（$|\gamma| \approx 0$）。在神经元重新缩放后，我们调整了参数用来恢复最初的映射。对于$\mathcal{S}$神经元,使得$s_i = \gamma_i'/\gamma_i$。为了使得$\mathcal{S}$神经元保持他们原本的映射功能，我们把用他们的参数通过s_i分开。实验证明这主导了表现的提升。

3.3. 训练具有混合类型的神经元

现在我们注重每一个单独的层。在神经元重生之后，每一层都有两种类型的输入神经元，$\mathcal{S}_{\text{in}}$ 和 $\mathcal{R}_{\text{in}}$, 以及两种类型的神经元, $\mathcal{S}_{\text{out}}$ 和 $\mathcal{R}_{\text{out}}$。为了简化，我们依然使用它们表示相应神经元的特征。然后根据卷积层的定义，我们有

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

其中$*$表示卷积操作。$W_{\mathcal{R}\to\mathcal{S}}$在重新初始化中被设置为0;所以，$\mathcal{S}_{\text{out}} = W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}}$，为了保持$\mathcal{S}_{\text{in}}$和$\mathcal{S}_{\text{out}}$之间的原始的映射。在这一小节，我们讨论如何训练$W$。

W的训练取决于网络有多需要重生神经元所带来的附加能力来适应数据。当$\mathcal{S}$神经元一点也不需要这种附加能力，通过by Eq. 5可能没有用因为单单是$\mathcal{S}$神经元就已经可以很好地满足数据的需求。结果是，在这种情况下为了利用附加能力改变训练方案是必要的。然而，当$\mathcal{S}$神经元单独无法满足数据，被$\mathcal{R}$所提供的附加能力都会修改训练。他们之前被发现是无用的不论是因为不合适的初始化还是因为低效的资源分配，但是现在被重新初始化和重新安排。我们在下面展示了详细讨论。

当 $\mathcal{S}$不需要$\mathcal{R}$ 这里，我们考虑网络能力大于需求的情况，并且单单是$\mathcal{S}$神经元就能很好地满足训练数据。一个例子就是在CIFAR [32]训练网络，在这个人物中大多数现代的结构都能获得99.0%的准确率。当类似于Eq. 5在结构中加入了$\mathcal{R}$神经元,由于被训练的$\mathcal{S}$神经元已经能很好地满足数据，损失的梯度反向传播将不鼓励现有映射$(\mathcal{S}_{\text{in}}, \mathcal{R}_{\text{in}}) \to (\mathcal{S}_{\text{out}}, \mathcal{R}_{\text{out}})$的大的改变。因此，像Eq. 5保持对计算机建模可能会导致$\mathcal{R}_{\text{in}}$神经元不久后死去并且$\mathcal{R}_{\text{out}}$产生冗余的的特征。

上述问题的原因是$\mathcal{R}$神经元和$\mathcal{S}$神经元之间的交叉连接，这会造成$\mathcal{R}$的短接。如果我们完全移除它们，*i.e.*,

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

然后$\mathcal{R}$神经元被强制学习新的，完全不同的特征。我们使用NR-CR来表示有着交叉连接被移除的神经再生。

当$\mathcal{S}$需要$\mathcal{R}$ 在这里，我们假设单单是$\mathcal{S}$神经元的能力并不能满足训练数据的需求。一个例子就是在ImageNet数据集 [52]上训练一个小网络。在这种情况下，需要保持交叉连接来提高能力。4.2小节的实验比较了一个简化的VGG网络在ImageNet上的表现，显示出有无交叉连接的神经再生都可以提高准确率，但是保持交叉连接提高地更多。

$\mathcal{S}$和$\mathcal{R}$之间的交叉连接 我们继续讨论，在这里我们假设$\mathcal{S}$需要$\mathcal{R}$的能力并且我们保持交叉连接。然后根

据Eq. 5，从$\mathcal{S}_{\text{in}}$和$\mathcal{R}_{\text{in}}$的输出都会加上$\mathcal{S}_{\text{out}}$，*i.e.*

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

由于这里的假设是模型的能力对于满足训练数据是低效的，如果我们不仅通过重生死神经元并且在既不多使用参数也不造成相比于卷积本身大量的计算增加就能够提高能力效果就会变好。因为$W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}}$被调整，所以我们着重于$W_{\mathcal{R}\to\mathcal{S}} * \mathcal{R}_{\text{in}}$。一个提高能力的方法就是使用二阶响应变换。最初的二阶响应变化是在残差学习中通过增加一个几何平均数定义的，例如：

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

对于我们的问题而言，尽管Eq. 7没有残差连接，在残差学习中 $W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}}$和$W_{\mathcal{R}\to\mathcal{S}} * \mathcal{R}_{\text{in}}$都被加上了；因此,我们可以给Eq. 7加一个类似的响应变换。不同于增加一个几何变换而可能导致训练的不稳定性 [56]，像所展示的那样我们提出了在Eq. 9使用交叉连接。

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

其中, $\sigma(\cdot)$表示Sigmoid函数。对称地，我们把交叉连接加到$\mathcal{R}_{\text{out}}$的输出中，例如：

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

我们使用 NR-CA 来表示有交叉连接的神经重生。

4. 实验

在这一节,我们将会展示实验结果来支持我们之前的讨论，并且展示不同架构下神经重生的提升。

4.1. 资源利用

图一我们展示了在ImageNet上当稀疏性方案被加进了损失时，训练ResNet-50和ResNet-101的资源利用率。在图中，我们根据训练轮数展示了参数利用率和有效的准确性。这样的模型当批处理的大小为256时需要90轮，当批处理大小是128时是100轮 [27]。在所有的实验中，系数参数λ被初始化为0，δt被设置为一轮，$\Delta r = 0.01$ 并且$\Delta\lambda = 5 \times 10^{-5}$。 T_r 不加其他说明的话一般被设置为0.5。

图. 1展示了携带着一下重要信息的两个典型的例子。 (1) 在像ImageNet这样大的数据集上训练不能避免

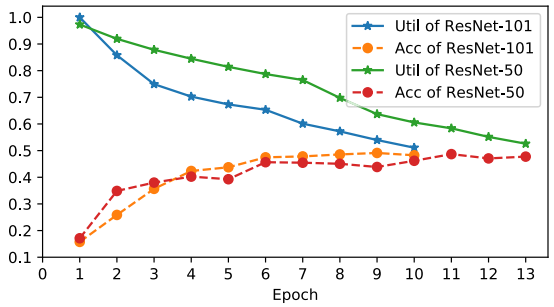

图 1. 从头在ImageNet上训练的ResNet-50和ResNet-101的参数利用率和有效率

计算资源的浪费；因此，我们也适用于大型训练。 (2) 在更大的模型中找死神经元比在比较小的模型中更容易。这和我们的直觉是一致的，因为更大的模型增加了能力和更多资源浪费的危险。 (3) 达到利用率的阈值0.5确实不需要太长时间，ResNet-101中10轮就足以节省一半的资源。

对于ImageNet的训练，我们仅对前三十个学习率为0.1的轮设置神经再生标志。由于$r(\theta_{\mathcal{A}})$经常需要10-20轮左右达到$T_r = 0.5$，所以会有大概1-2次会执行算法 1的第八步。为了简化实验，我们只在ImageNet上进行一次神经再生，并且在之后重置轮数为0。神经再生的训练时间因此会比一般的训练时间稍长，但是增加会少于20%并且实验证明这很值得。对于没有限制的训练时间，小节. 4.3展示出了有多次神经再生的CIFAR的表现。

4.1.1 在神经再生上的消融研究

为了给应用到训练神经网络的神经再生提供更好的理解，我们在表 1展示了一个消融研究，这演示了有不同变化的神经再生的结果。

网络是一个简化的VGG-19,被ImageNet中的低解析度图片所训练。用于训练和测试的图片大小是128x128。我们移除了最后的三个全连接层，并且用一个全局池化层和一个全连接层进行替代。最后的模型只有20.5M个参数。为了将来加速训练，我们把它的第一个卷积层替换成ResNett [22]中的一个卷积层。通过应用这些改变，我们可以利用4个Titan Xp GPU一天之内训练一个模型，出于消融研究的目的这是足够快的。

显而易见地，如此简单的一个模型没有足够的能

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

表 1. 有着$T_r = 0.25$的简化的VGG-19当保持参数数量时在ImageNet上的错误率。 BL: 基准 BL-CA: 有交叉连接的基准 NR-CR: 移除交叉连接的神经再生 NR-FS: 从头通过神经再生训练$\mathcal{A}$ NR: 带有交叉连接的神经再生 NR-BR: 带有神经所放的神经再生 NR-CA: 有着交叉注意的神经再生 NR-CA-BR: 带有交叉注意和神经缩放的神经再生

力满足ImageNet的需求。正如我们在小节. 3.3已经讨论过的，为了增加模型能力而保留交叉连接是很好的。正如这里所显示的，NR-CR比基础的提高了第一名的0.7%的准确率，但是比保留了检查连接的NR低了0.7%。我们之后显示交叉连接可以使得排名第一的错误率降低大约0.5%，并且神经缩放提高了准确率。在接下来的ImageNet上的实验中，我们将在所有的方法中使用NR-CA-BR。

4.2. 在**ImageNet**上的结果

表 2显示了在ImageNet数据集 [52]显示了表现的提升。ImageNet数据集是一个大的图像分类数据集，有着用于训练的1.28 百万的彩色图片，以及5，000的测试图片。表 2列出了在如此有挑战性的任务上达到很高准确率的一些现代架构。之前，很多的注意力都用在了为了不同的视觉任务而设计新颖的更适合的架构。我们的结果显示除了架构设计和搜索，现在的优化技术还有很多提升空间。我们的工作只注重于利用率问题，但是已经达到了很强的表现的提升。

在这里，我们简短地介绍为了便于重现的实验设置。如果模型能放进内存那么所有的模型都会使用批处理大小256的训练；否则，批处理大小设置为128。总共，当批大小为256时训练90轮，当批大小是128时训练100轮。学习率初始化为0.1,并且会在31^{st}, 61^{st}, and 91^{st} 轮时会除以10。

对于我们的任务而言，我们在这些现在最好效果的架构上做出了以下调整。对于VGG-16 [53]，我们在每一个卷积层加上一个批归一化层，并且移除最后三个全连接层。在这之后，我们加了两个有着4096通道的卷积层，这是为了和有这两个全连接层的最初的VGG-16输出相同数量的通道。在这两个卷积层之后，我们增加了一个全局池化层，以及为了图像分类加了一个把4096通道转换成1000通道的全连接层。最终的模型会有更少的参数（138M比37.7M）,但同时有着更少的错误率（27比24.26）。VGG-16的所有层保留了稀疏惩罚。对于ResNet [22]，除了加到主流后面的所有卷积层为了神经再生都在考虑之内。对于DenseNet [27]，由于GPU内存和速度的原因，我们只能运行有着121层的DenseNet。我们为了迎合我们的每一个卷积层直接跟一个批归一化层的假设把预激活的 [23]转换成后激活的 [22]。这一变化产生了和之前模型类似的准确率。

从我们的结果中有一个很快的发现就是有着更强能力的模型往往会从神经再生中获得更好的提升。这和我们 3.3小节的讨论以及 **??**小节的发现相一致。对于大规模的任务，模型的能力是重要的并且更大的模型往往意味着浪费更多的资源。所以，从大型模型中重生死神经元将会比在小的模型中做获得资源的更好的利用。在所有的模型中，DenseNet-121是最难发现死神经元的，所以也获得了最小的提升。这也许解释了在他们论文中所讨论的模型紧凑问题。更多地，有着NR的VGG-16在仅仅21.5M参数下达到了第一名仅仅23.71%的错误率，远比第二名在23.2M参数条件下达到最低36.66%的[39]要好。

接下来，我们在表 3显示了在MobileNet-0.5上和MobileNet-0.25上的实验。他们没有被包含在表 2中，因为他们的图像大小为128x128，并且学习率从0.1 [48]开始遵循余弦学习率。MobileNet是为有着较低计算资源的平台设计的。我们的NR的表现超过了之前的[16]方法并且表现出了很强的提升。

4.3. 在**CIFAR**上的结果

在CIFAR上的实验有两个部分。第一个部分是使用神经再生作为一种模型压缩方法，模型的大小被缩小一半后和之前的最好的模型进行对比。结果在表 4中展示。在第二个部分，我们把神经再生进行了多次并把表现展示在了表 5中。

模型压缩 表 4展示了在CIFAR-10/100数据集 [32]上显示了性能比较。CIFAR数据集是一个小的数据集，有着50,000训练图像和10,000测试图像。不像之前在ImageNet上的实验，其中，我们不重生死神经元来

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 [27]	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 [53]	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 [22]	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 [22]	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 [22]	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 [22]	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%

表 2. 训练带有和不带有神经再生的深度神经网络在ImageNet测试集上的错误率每个神经网络都有三个第一和第五的错误率集合，三个集合分别是基准,在资源约束下参数不变的神经再生，在资源约束下计算力不变的神经再生。最后一栏 相对收益 显示当保持参数数量或者计算力的情况下最好的第一错误率的相对收益。

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

表 3. 在ImageNet上MobileNet [26]的第一错误率。训练和测试的图像大小都是128x128。在所有的方法中计算力保持一致。BL: 在 [16]中所报告的基准表现, MN: MorphNet [16], BL*: 我们实验的基准表现, and NR: 神经再生。

使用所有的可用的计算资源；不同的是，我们把资源目标设置为$0.5 \times \mathcal{C}$，$\mathcal{C}$是初始的资源限制。在实践中，是通过设置$T_r = 0.25$以及重生模型来达到$0.5 \times \mathcal{C}$的水平。结果是，神经重生以只用一半的参数训练模型结束，从而可以和之前最好的神经修剪方法 [39]相提并论。

多次NR 表 5显示了不限制神经再生次数的VGG-19模型在CIFAR数据集上的性能。当神经重生的次数增加以后提升的趋势是明显的。相对收益分别是CIFAR-10的33.5%以及CIFAR-100的13.8%。

其中，我们详细介绍了实验的设置。对于VGG-19，我们因为之前的结构不是为CIFAR设计的而做出了以下的改变。首先我们删除了所有的全连接层，并且在卷积层之后加了一个全局平均池化层，并在之后跟了一个全连接层来输出最后的结果。接下来我们移除了最初的4个最大池化层并在第四和第十个卷积层之后加了2个全局池化层来实现降采样。这些变化改变了开始的CIFAR的架构，并且基线错误率变得更低，例如 在CIFAR-10上从6.66降到了5.44，在CIFAR-100上从28.05降到了23.11。我们把在ImageNet上所作的改变同样应用到了DenseNet上。对于有着瓶颈块的ResNet-164，类似于在ImageNet上的设置，我们只考虑神经再生的网络中不是主流的神经元。我们的方法是移除了所有交叉连接的NR-CR层。表 4显示了神经再生也可以被很好地用来训练小的模型。表 4显示了当训练很多次神经再生之后VGG-19的潜力。当保持着参数的数量的时候，神经再生能够大幅度地提高性能。

5. Conclusion

在这篇论文中，我们探索了通过提高计算资源利用率来改善深度神经网络的训练。这个问题被两个在深度网络上训练的两个发现所驱动， (1) 更多的计算资源往往带来更好的性能，并且 (2) 被标准优化器训练的模型的资源利用率可能是不充分的。因此，我们探究了最大化计算资源的问题。最后我们提出了一个叫神经再生的新颖的方法，它是通过重新分配和重新初始化神经元来重生死神经元。神经再生包括三个组件：资源利用率检测，死神经元再生以及有着混合类型神经元的神经网络训练方案。这些组件实时地检测神经元的活泼性，必要时再生死神经元，并且为有着混合类型神经元的网络提供不同的训练策略。我们在具有挑战性的CIFAR和ImageNet数据集上测试了神经再生，并且显示我们的方法在保持无论是参数数量还是计算量的同时提高了很多现在最好的网络的性能。更多的是，当我们想让架构有更小的计算花费时，神经再生能用于神经压缩，这也比之前最好的模型有更好的性能。总结的来说，神经再生是一个注重于资源利用率的优化技术，它通过增强利用率改善了深度神经网络的训练。

Architecture	Baseline		Network Slimming [39]		Neural Rejuvenation	
	C10 (Params)	C100 (Params)	C10 (Params)	C100 (Params)	C10 (Params)	C100 (Params)
VGG-19 [53]	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 [22]	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 [27]	3.64 (8.27M)	19.85 (8.37M)	3.75 (4.36M)	19.29 (4.65M)	3.40 (4.12M)	18.59 (4.31M)

表 4. 在CIFAR [32]上为了模型压缩的神经再生。在ImageNet的实验中, 当再生死神经元时计算资源会被保留。但是在这里，我们把神经再生的资源目标设置为初始使用的一半。从而，我们的神经再生变成了一种模型压缩方法，从而可以和最好的修剪方法 [39]相提并论。

# 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	—	—

表 5. 在保持参数数量的同时，VGG-19在CIFAR-10 (C10) and CIFAR-100 (C100)上多次神经再生后的错误率。

参考文献

[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. 1

[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. 7, 8

[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, 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.

[20] 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.

[21] 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.

[22] 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. 1, 6, 7, 8, 9

[23] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *ECCV*, 2016. 7

[24] 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.

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

[26] 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. 8

[27] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2017. 6, 7, 8, 9

[28] 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.

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

[30] 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.

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

[32] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009. 1, 5, 7, 9

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

[34] 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.

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

[36] 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.

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

[38] 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.

[39] 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. 1, 7, 8, 9

[40] 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.

[41] 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. 1

[42] 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.

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

[44] 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.

[45] 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.

[46] 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. 1

[47] 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.

[48] 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. 7

[49] 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.

[50] 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.

[51] 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.

[52] 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. 1, 5, 7

[53] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1, 7, 8, 9

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

[55] 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.

[56] 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. 6

[57] 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.

[58] 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.

[59] 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.

[60] 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.

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

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

[63] 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. 1

[64] 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.

[65] 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.

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

[67] 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.

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

[69] 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.