RANP: Resource Aware Neuron Pruning at Initialization for 3D CNNs

Zhiwei Xu[1,3]   Thalaiyasingam Ajanthan[1]   Vibhav Vineet[2]   Richard Hartley[1]

[1]Australian National University and Australian Centre for Robotic Vision

[2]Microsoft Research, Redmond, USA

[3]Data61, CSIRO, Australia

{zhiwei.xu,thalaiyasingam.ajanthan,richard.hartley}@anu.edu.au

vibhav.vineet@microsoft.com

Abstract

*Although 3D Convolutional Neural Networks (CNNs) are essential for most learning based applications involving dense 3D data, their applicability is limited due to excessive memory and computational requirements. Compressing such networks by pruning therefore becomes highly desirable. However, pruning 3D CNNs is largely unexplored possibly because of the complex nature of typical pruning algorithms that embeds pruning into an iterative optimization paradigm. In this work, we introduce a Resource Aware Neuron Pruning (RANP) algorithm that prunes 3D CNNs at initialization to high sparsity levels. Specifically, the core idea is to obtain an importance score for each neuron based on their sensitivity to the loss function. This neuron importance is then reweighted according to the neuron resource consumption related to FLOPs or memory. We demonstrate the effectiveness of our pruning method on 3D semantic segmentation with widely used 3D-UNets on ShapeNet and BraTS'18 as well as on video classification with MobileNetV2 and I3D on UCF101 dataset. In these experiments, our RANP leads to roughly **50%-95% reduction in FLOPs and 35%-80% reduction in memory** with negligible loss in accuracy compared to the unpruned networks. This significantly reduces the computational resources required to train 3D CNNs. The pruned network obtained by our algorithm can also be easily scaled up and transferred to another dataset for training.*

1. Introduction

3D image analysis is important in various real-world applications including scene understanding [1, 2], object recognition [3, 4], medical image analysis [5, 6, 7], and video action recognition [8, 9]. Typically, sparse 3D data can be represented using point clouds [10] whereas volumetric representation is required for dense 3D data which arises in domains such as medical imaging [11] and video segmentation and classification [1, 8, 9]. While efficient
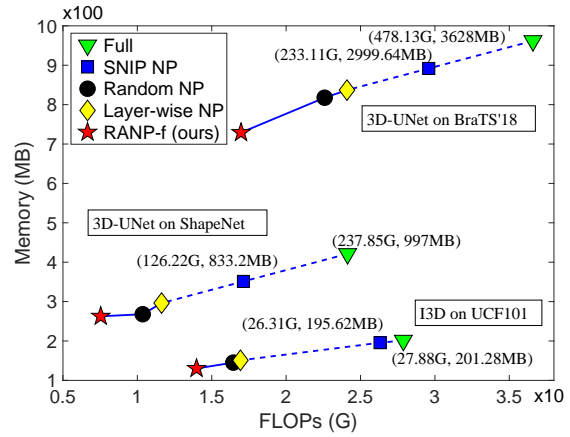

Figure 1: **Bottom-left is the best**. Comparison of neuron pruning methods. "Full" and "SNIP NP" are not drawn by scale but with their FLOPs (G) and memory (MB) values next to the markers. Our RANP-f performs best with large resource reductions while maintaining the accuracy. More details are in Table 2.

neural network architectures can be designed for sparse point cloud data [12, 13], conventional dense 3D Convolutional Neural Network (CNN) is required for volumetric data. Such 3D CNNs are computationally expensive with excessive memory requirements for large-scale 3D tasks. Therefore, it is highly desirable to reduce the memory and FLOPs required to train 3D CNNs while maintaining the accuracy. This will not only enable large-scale applications but also 3D CNN training on resource-limited devices.

Network pruning is a prominent approach to compress a neural network by reducing the number of parameters or the number of neurons in each layer [14, 15, 16, 17]. However, most of the network pruning methods aim at 2D CNNs while pruning 3D CNNs is largely unexplored. This is mainly because pruning is typically targeted at reducing the test-time resource requirements while computational requirements of training time are as large as (if not more

than) the unpruned network. Such pruning schemes are not suitable for 3D CNNs with dense volumetric data where training-time resource requirement is prohibitively large.

In this work, we introduce a Resource[1] Aware Neuron Pruning (RANP) that *prunes 3D CNNs at initialization*. Our method is inspired by, but superior to, SNIP [18] which prunes redundant parameters of a network at initialization and only tests with small scale 2D CNNs for image classification. With the same characteristics of effectively pruning at initialization without requiring large computational resources, RANP yields better-pruned networks compared to SNIP by removing neurons that largely contribute to the high resource requirement. In our experiments on video classification and more challenging 3D semantic segmentation, with minimal accuracy loss, RANP yields 50%-95% reduction in FLOPs and 35%-80% reduction in memory while only 5%-51% reduction in FLOPs and 1%-17% reduction in memory are achieved by SNIP NP.

The main idea of RANP is to prune based on a *neuron importance* criterion analogous to the connection sensitivity in SNIP. Note that, pruning based on such a simple criterion as SNIP has the risk of pruning the whole layer(s) at extreme sparsity levels especially on large networks [19]. Even though an orthogonal initialization that ensures layer-wise dynamical isometry is sufficient to mitigate this issue for parameter pruning on 2D CNNs [19], it is unclear if this could be directly applied to neuron pruning on 3D CNNs. To tackle this and improve pruning, we introduce a *resource aware reweighting scheme* that first balances the mean value of neuron importance in each layer and then reweights the neuron importance based on the resource consumption of each neuron. As evidenced by our experiments, such a reweighting scheme is crucial to obtain large reductions in memory and FLOPs while maintaining high accuracy.

We firstly evaluate our RANP on 3D semantic segmetation on a sparse point-cloud dataset, ShapeNet [10], and a dense medical image dataset, BraTS'18 [11, 20], with widely used 3D-UNets [5]. We also evaluate RANP on video classification using UCF101 with MobileNetV2 [21] and I3D [22]. Our RANP-f significantly outperforms other neuron pruning methods in *resource efficiency* by yielding large reductions in computational resources (**50%-95% FLOPs reduction and 35%-80% memory reduction**) with comparable accuracy to the unpruned network (Fig. 1).

Furthermore, we perform extensive experiments to demonstrate **1) scalability** of RANP by pruning with a small input spatial size and training with a large one, **2) transferability** by pruning using ShapeNet and training on BraTS'18 and vice versa, **3) lightweight** training on a single GPU, and **4) fast** training with increased batch size.

---

[1]We concretely define "resource" as FLoating Point Operations per second (FLOPs) and memory required by one forward pass.

2. Related Work

Previous works of network pruning mainly focus on 2D CNNs by parameter pruning [18, 17, 14, 15, 23] and neuron pruning [24, 16, 25, 26, 27, 28]. While a majority of the pruning methods use the traditional prune-retrain scheme with a combined loss function of pruning criteria [17, 16, 26], some pruning at initialization methods is able to reduce computational complexity in training [18, 29, 30, 31, 32]. While very few are for 3D CNNs [33, 34, 35], none of them prune networks at initialization, and thus, none of them effectively reduce the training-time computational and memory requirements of 3D CNNs.

2D CNN pruning. *Parameter pruning* merely sparsifies filters for a high learning capability with small models. Han *et al.* [17] adopted an iterative method of removing parameters with values below a threshold. Lee *et al.* [18] recently proposed a single-shot method with connection sensitivity by magnitudes of parameter mask gradients to retain top-κ parameters. These filter-sparse methods, however, do not directly yield large speedup and memory reductions.

By contrast, *neuron pruning*, also known as filter pruning or channel pruning, can effectively reduce computational resources. For instance, Li *et al.*[25] used l_1 normalization to remove unimportant filters with connecting features. He *et al.*[16] adopted a LASSO regression to prune network layers with reconstruction in the least square manner. Yu *et al.*[24] proposed a group-wise 2D-filter pruning from each 3D-filter by a learning-based method and a knowledge distillation. Structure learning based MorphNet [36] and SSL [37] aim at pruning activations with structure constraints or regularization. These approaches only reduce the test-time resource requirement while we focus on reducing those of large 3D CNNs at training time.

3D CNN pruning. To improve the efficiency on 3D CNNs, some works like SSC [12] and OctNet [38] use efficient data structures to reduce the memory requirement for sparse point-cloud data. However, these approaches are not useful for dense data, *e.g.*, MRI images and videos, and the resource requirement remains prohibitively large.

Hence, it is desirable to develop an efficient pruning for 3D CNNs that can handle dense 3D data which is common in real applications. Only very few works are relevant to 3D CNN pruning. Molchanov *et al.*[33] proposed a greedy criteria-based method to reduce resources via back-propagation with a small 3D CNN for hand gesture recognition. Zhang *et al.*[34] used a regularization-based pruning method by assigning regularization to weight groups with $4\times$ speedup in theory. Recently, Chen *et al.*[35] converted 3D filters into frequency domain to eliminate redundancy in an iterative optimization for convergence. Being a parameter pruning method, this does not lead to large FLOPs and memory reductions, *e.g.*, merely a $2\times$ speedup compared to our $28\times$ (ref. Sec. 5.3). In summary, these methods embed

pruning in the iterative network optimization and require extensive resources, which is inefficient for 3D CNNs.

Pruning at Initialization. While few works adopted pruning at initialization, some achieved impressive success. SNIP [18] is the first single-shot pruning method that presented a high possibility of pruning networks at initialization with minimal accuracy loss in training, followed by many recent works on single-shot pruning [29, 30, 31, 32]. But none are for 3D CNNs pruning.

In addition to being a parameter pruning approach, the benefits of SNIP was demonstrated only on small-scale datasets [18], such as MNIST and CIFAR-10. Therefore, it is unclear that whether these benefits could be transposed to 3D CNNs applied to large-scale datasets. Our experiments indicate that, while SNIP itself is not capable of yielding large resource reduction on 3D CNNs, our RANP can greatly reduce the computational resources without causing network infeasibility. Furthermore, we show that RANP enjoys strong transferability among datasets and enables fast and lightweight training of large 3D volumetric data segmentation on a single GPU.

3. Preliminaries

We first briefly describe the main idea of SNIP [18] which removes redundant parameters prior to training. Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{S}$ with input $\mathbf{x}_i$ and ground truth $\mathbf{y}_i$ and the sparsity level κ, the optimization problem associated with SNIP can be written as

$$\min_{\mathbf{c},\mathbf{w}} L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) = \min_{\mathbf{c},\mathbf{w}} \frac{1}{S} \sum_{i=1}^{S} \ell\left(\mathbf{c} \odot \mathbf{w}, (\mathbf{x}_i, \mathbf{y}_i)\right) , \quad (1)$$

$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m, \quad \mathbf{c} \in \{0, 1\}^m, \quad \|\mathbf{c}\|_0 \leq \kappa ,$$

where $\mathbf{w}$ is denoted a m-dimensional vector of parameters, $\mathbf{c}$ is the corresponding binary mask on the parameters, $\ell(\cdot)$ is the standard loss function (*e.g.*, cross-entropy loss), and $\|\cdot\|_0$ denotes l_0 norm. The mask $c_j \in \{0, 1\}$ for parameter w_j denotes that the parameter is retained in the compressed model if $c_j = 1$ and otherwise it is removed. In order to optimize the above problem, they first relax the binary constraint on the masks such that $\mathbf{c} \in [0, 1]^m$. Then an importance function for parameter w_j is calculated by the normalized magnitude of the loss gradient over mask c_j as

$$s_j = \frac{|g_j(\mathbf{w}; \mathcal{D})|}{\sum_{k=1}^{m} |g_k(\mathbf{w}; \mathcal{D})|}, \text{where } g_j(\mathbf{w}; \mathcal{D}) = \frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial c_j}\Big|_{\mathbf{c}=\mathbf{1}} . \quad (2)$$

Then, only top-κ parameters are retained based on the parameter importance, called connection sensitivity in [18], $\mathbf{s}$ defined above. Upon pruning, the retained parameters are trained in the standard way. It is interesting to note that, even though having the mask $\mathbf{c}$ is easier to explain the intuition, SNIP can be implemented without these additional variables by noting that $g_j(\mathbf{w}; \mathcal{D}) =$
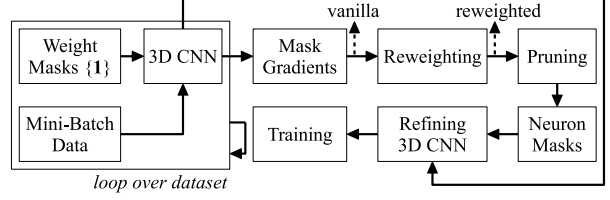

Figure 2: Flowchart of RANP algorithm. The refining generates a new yet slim network for resource-efficient training.

$(\partial L(\mathbf{w}; \mathcal{D})/\partial w_j) w_j$ [19]. This method has shown remarkable results in achieving $> 95\%$ sparsity on 2D image classification tasks with minimal loss of accuracy. Such a parameter pruning method is important, however, it cannot lead to sufficient computation and memory reductions to train a deep 3D CNN on current off-the-shelf graphics hardware. In particular, the sparse weight matrices cannot efficiently reduce memory or FLOPs, and they require specialized sparse matrix implementations for speedup. In contrast, neuron pruning directly translates into practical gains of reducing both memory and FLOPs. This is crucial in 3D CNNs due to their substantially higher resource requirement compared to 2D CNNs.

4. Resource Aware NP at Initialization

To explain the proposed RANP, we first extend the SNIP idea to neuron pruning at initialization. Then we discuss a resource aware reweighting strategy to further reduce the computational requirements of the pruned network. The flowchart of our RANP algorithm is shown in Fig. 2.

Before introducing our neuron importance, we first consider a fully-connected feed-forward neural network for the simplicity of notations. Consider weight matrices $\mathbf{W}^l \in \mathbb{R}^{N_l \times N_{l-1}}$, biases $\mathbf{b}^l \in \mathbb{R}^{N_l}$, pre-activations $\mathbf{h}^l \in \mathbb{R}^{N_l}$, and post-activations $\mathbf{x}^l \in \mathbb{R}^{N_l}$, for layer $l \in \mathcal{K} = \{1, \ldots, K\}$. Now the feed-forward dynamics is

$$\mathbf{x}^l = \phi\left(\mathbf{h}^l\right) , \quad \text{where } \mathbf{h}^l = \mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l , \quad (3)$$

where the activation function $\phi : \mathbb{R} \to \mathbb{R}$ has element-wise nonlinearity and the network input is denoted by $\mathbf{x}^0$. Now we introduce binary masks on neurons (*i.e.*, post-activations). The feed-forward dynamics is then modified to include this masking operation as

$$\mathbf{x}^l = \mathbf{c}^l \odot \phi\left(\mathbf{h}^l\right) , \quad \text{where } \mathbf{c}^l \in \{0, 1\}^{N_l} , \quad \forall l \in \mathcal{K} , \quad (4)$$

where neuron mask $c_u^l = 1$ indicates neuron x_u^l is retained and otherwise pruned. Here, neuron pruning can be written as the following optimization problem

$$\min_{\mathbf{w}} L(\mathbf{c}, \mathbf{w}; \mathcal{D}) = \min_{\mathbf{c},\mathbf{w}} \frac{1}{S} \sum_{i=1}^{S} \ell\left(\mathbf{c}, \mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)\right) , \quad (5)$$

$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m, \quad \mathbf{c} \in \{0, 1\}^n, \quad \|\mathbf{c}\|_0 \leq \kappa ,$$

where n is the total number of neurons and $\ell(\mathbf{c}, \cdot; \cdot)$ denotes a standard loss function of the feed-forward mapping with

neuron masks **c** defined in Eq. 4. This can be easily extended to convolutional and skip-concatenation operations.

As removing neurons could largely reduce memory and FLOPs compared to merely sparsifying parameters, the core of our approach is benefited by removing redundant neurons from the model. We use an influence function concept developed for parameters to establish neuron importance through the loss function, to locate redundant neurons.

4.1. Neuron Importance

Note that, neuron importance can be derived from the SNIP-based parameter importance discussed in Sec. 3. Another approach is to directly define neuron importance as the normalized magnitude of the neuron mask gradients analogous to parameter importance.

Neuron Importance with Parameter Mask Gradients. The first approach to calculate neuron importance depends on the magnitude of parameter mask gradients, denoted as Magnitude of Parameter Mask Gradients (MPMG). Thus, the importance of neuron x_u^l is

$$s_u^l = f\left(|g_{u1}^l|, \ldots, |g_{uN_{l-1}}^l| \right) , \qquad (6)$$

where $g_{uv}^l = \partial L\left(\mathbf{c} \odot \mathbf{w}; \mathcal{D}\right)/\partial c_{uv}^l$ with c_{uv}^l as the mask of parameter w_{uv}^l. Refer to Eq. 2. Here, $f(\cdot) : \mathbb{R}^{N_{l-1}} \to \mathbb{R}$ is a function mapping a set of values to a scalar. We choose $f(\cdot) = \text{sum}(\cdot)$ with alternatives, *i.e.*, mean and max functions, in Appendix D. Now, we set neuron masks as 1 for neurons with top-κ largest neuron importance.

Neuron Importance with Neuron Mask Gradients. Another approach is to directly compute mask gradients on neurons and treat their magnitudes as neuron importance, denoted as Magnitude of Neuron Mask Gradients (MNMG). The neuron importance of x_u^l is calculated by

$$s_u^l = \left| \left. \frac{\partial L\left(\mathbf{c}, \mathbf{w}; \mathcal{D}\right)}{\partial c_u^l} \right|_{\mathbf{c}=1} \right| . \qquad (7)$$

Noting that a non-linear activation function $\phi(\cdot)$ in CNN including but not limited to ReLU can satisfy $\phi(ch) = c\phi(h), \forall c \geq 0$. Given such a homogeneous function, the calculation of neuron importance with neuron masks can be derived from parameter mask gradients in the form of

$$\left. \frac{\partial L\left(\mathbf{c}, \mathbf{w}; \mathcal{D}\right)}{\partial c_u^l} \right|_{\mathbf{c}=1} = \sum_{v=1}^{N_{l-1}} \left. \frac{\partial L\left(\mathbf{c} \odot \mathbf{w}; \mathcal{D}\right)}{\partial c_{uv}^l} \right|_{\mathbf{c}=1} . \qquad (8)$$

Details of the influence of such an activation function on neuron importance are provided in Appendix B. These two approaches for neuron importance are in a similar form that while MPMG is by the sum of magnitudes, MNMG is by the magnitude of the sum of parameter mask gradients. It can be implemented directly from parameter gradients.

The neuron importance based on MPMG or MNMG approach can be used to remove redundant neurons. However,

(a) Vanilla NP Eq. 6   (b) Weighted NP Eq. 9
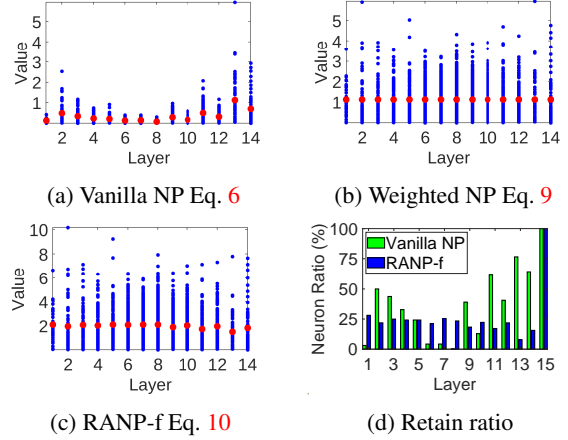
(c) RANP-f Eq. 10   (d) Retain ratio

Figure 3: ShapeNet: neuron importance of 3D-UNet becomes balanced and resource-aware from (a) to (c) at neuron sparsity 78.24%. Blue: neuron importance; red: mean values. More illustrations are in Appendix D.

they could lead to an imbalance of sparsity levels of each layer in 3D network architectures. As shown in Table 2, the computational resources required by vanilla neuron pruning are much higher than those by other sparsity enforcing methods, *e.g.*, random neuron pruning and layer-wise neuron pruning. We hypothesize that this is caused by the layer-wise imbalance of neuron importance which unilaterally emphasizes on some specific layer(s) and may lead to network infeasibility by pruning the whole layer(s). This behavior is also observed in [19], and orthogonal initialization is thus recommended to solve the problem for 2D CNN pruning, which however cannot result in balanced neuron importance in our case, see results in Appendix D.

In order to resolve this issue, we propose resource aware neuron pruning (RANP) with reweighted neuron importance, and the details are provided below.

4.2. Resource Aware Reweighting

To tackle the imbalanced neuron importance issue above, we first weight the neuron importance across layers. Weighting neuron importance of x_u^l can be expressed as

$$\tilde{s}_u^l = \frac{\max_{k=1}^K \bar{s}^k}{\bar{s}^l} s_u^l , \quad \text{where } \bar{s}^k = \frac{1}{N_k} \sum_{u=1}^{N_k} s_u^k, \ \forall k \in \mathcal{K} . \qquad (9)$$

Here, $\bar{s}^l$ is the mean neuron importance of layer l and $\tilde{s}_u^l$ is the updated neuron importance. This helps to achieve the same mean neuron importance in each layer, which largely avoids underestimating neuron importance of specific layer(s) to prevent from pruning the whole layer(s).

To further reduce the memory and FLOPs with minimal accuracy loss, we then reweight the neuron importance $\tilde{s}_u^l$ by available resource, *i.e.*, memory or FLOPs. This reweighting counts on the addition of weighted neuron importance and the effect of the computational resource, de-

noted as RANP-[m|f], where "m" is for memory and "f" is for FLOPs . We represent the importance of this available resource in layer l as τ_l, refer to Appendix C for details.

The reweighted neuron importance of neuron x_u^l by following weighted addition variant RANP-[m|f] is

$$\hat{s}_u^l = (1 + \lambda \, \mathrm{softmax}(-\tau_l)) \, \tilde{s}_u^l = \left(1 + \lambda \frac{e^{-\tau_l}}{\sum_{k=1}^{K} e^{-\tau_k}}\right) \tilde{s}_u^l ,$$

(10)

where coefficient $\lambda > 0$ helps to control the effect of resource on neuron importance. This effect represented by softmax constrains the values into a controllable range [0,1], making it easy to determine λ and function a high resource influence with a small resource occupation.

We demonstrate the effect of this reweighting strategy over vanilla pruning in Fig. 3. In more detail, vanilla neuron importance tends to have high values in the last few layers, making it highly possible to remove all neurons of such as the 7th and 8th layers. Weighting the importance in Fig. 3b makes the distribution of importance balanced with the same mean value in each layer. Furthermore, since some neurons have different numbers of input channels, each layer requires different FLOPs and memory. Considering the effect of computational resources on training, we embed them into neuron importance as weights.

In Fig. 3c, the last few layers require larger computational resources than the others, and thus their neurons share lower weights, see the tendency of mean values. Vividly, neuron ratio in Fig. 3d indicates a more balanced distribution by RANP-f than vanilla NP. For instance, very few neurons are retained in the 8th layer by vanilla NP, resulting in low accuracy and low maximum neuron sparsity. With reweighting by RANP-f, however, more neurons can be retained in this layer. Moreover, in Table 2, while weighted NP achieves high accuracy, its computational resource reductions are small. In contrast, RANP-f largely decreases the computational resources with a small accuracy loss.

Then, with reweighted neuron importance by Eq. 10 and $\ddot{s}_\kappa$ as the κth reweighted neuron importance in a descending order, the binary mask of neuron x_u^l can be obtained by

$$c_u^l = 1[\hat{s}_u^l - \ddot{s}_\kappa \geq 0] .$$

(11)

As mentioned in Sec. 2, our RANP is more effective in reducing memory and FLOPs than SNIP-based pruning which merely sparsifies parameters but needs high memory required by dense operations in training. RANP can easily remove neurons and all involved input channels at once, leading to huge reductions of input and output channels of the filter. Pseudocode is provided in Appendix A.

5. Experiments

We evaluated RANP on 3D-UNets for 3D semantic segmentation and MobileNetV2 and I3D for video classification. Experiments are on Nvidia Tesla P100-SXM2-16GB GPUs in PyTorch. More results are in Appendix D. Our code is available at *https://github.com/zwxu064/RANP.git*.

5.1. Experimental Setup

3D Datasets. For 3D semantic segmentation, we adopted the large-scale 3D sparse point-cloud dataset, ShapeNet [10], and dense biomedical MRI sequences, BraTS'18 [11, 20]. *ShapeNet* consists of 50 object part classes, 14007 training samples, and 2874 testing samples. We split it into 6955 training samples and 7052 validation samples as [12] to assign each point/voxel with a part class.

BraTS'18 includes 210 High Grade Glioma (HGG) and 75 Low Grade Glioma (LGG) cases. Each case has 4 MRI sequences, *i.e.*, T1, T1_CE, T2, and FLAIR. The task is to detect and segment brain scan images into 3 categories: Enhancing Tumor (ET), Tumor Core (TC), and Whole Tumor (WT). The spatial size is $240 \times 240 \times 155$ in each dimension. We adopted the splitting strategy of cross-validation in [39] with 228 cases for training and 57 cases for validation.

For video classification, we used video dataset, *UCF101* [40] with 101 action categories and 13320 videos. 2D spatial dimension from images and temporal dimension from frames are cast as dense 3D inputs. Among the 3 official train/test splits, we used split-1 which has 9537 videos for training and 3783 videos for validation.

3D CNNs. For 3D semantic segmentation on ShapeNet (sparse data) and BraTS'18 (dense data), we used the standard 15-layer 3D-UNet [5] including 4 encoders, each consists of two "3D convolution + 3D batch normalization + ReLU", a "3D max pooling", four decoders, and a confidence module by softmax. It has 14 convolution layers with 3^3 kernels and 1 layer with 1^3 kernel.

For video classification, we used the popular MobileNetV2 [21, 41] and I3D (with inception as backbone) [22] on UCF101. MobileNetV2 has a linear layer and 52 convolution layers while 18 of them are 3^3 kernels and the rest are 1^3. I3D has a linear layer and 57 convolution layers, 19 of which are 3^3 kernels, 1 is 7^3, and the rest are 1^3.

Hyper-parameters in learning. For *ShapeNet*, we set learning rate as 0.1 with an exponential decay rate $\gamma = 0.04$ by 100 epochs; batch size is 12 on 2 GPUs; spatial size for pruning and training is 64^3 while the spatial size for training is 128^3 in Sec. 5.7; optimizer is SGD-Nesterov [42] with weight decay 0.0001 and momentum 0.9.

For *BraTS'18*, learning rate is 0.001, decayed by 0.1 at 150th epoch with 200 epochs; optimizer is Adam[43] with weight decay 0.0001 and AMSGrad[44]; batch size is 2 on 2 GPUs; spatial size for pruning is 96^3 and 128^3 for training.

For *UCF101*, we adopted similar setup from [41] with learning rate 0.1, decayed by 0.1 at {40, 55, 60, 70}th epoch; optimizer by SGD with weight decay 0.001; batch size 8 on one GPU. Spatial size for pruning and training is 112^2 for MobileNetV2 and 224^2 for I3D; 16 frames are

Table 1: Vanilla NP by max neuron sparsity. "Metric" is mIoU for ShapeNet, ET for BraTS'18, top-1 for UCF101. "Param" and "Mem" are in MB. MPMG-sum is vanilla NP for large resource reductions and small metric loss.

Dataset (Model)	Manner	Sparsity	Param	GFLOPs	Mem	Metric
ShapeNet (3D-UNet)	Full[5]	0	62.26	237.85	997.00	83.79±0.21
	MNMG-sum	66.93	4.29	100.34	783.14	**83.65**±**0.02**
	MPMG-sum	78.24	2.54	**55.69**	**557.32**	83.26±0.14
BraTS'18 (3D-UNet)	Full[5]	0	15.57	478.13	3628.00	72.96±0.60
	MNMG-sum	81.32	0.35	**73.50**	**1933.20**	64.48±1.10
	MPMG-sum	78.17	0.55	104.50	1936.44	**71.94**±**1.68**
UCF101 (MobileNetV2)	Full[21]	0	9.47	0.58	157.47	47.08±0.72
	MNMG-sum	39.89	4.66	**0.43**	**120.01**	1.03±0.00[2]
	MPMG-sum	33.15	6.35	0.55	155.17	**46.32**±**0.79**
UCF101 (I3D)	Full[22]	0	47.27	27.88	201.28	51.58±1.86
	MNMG-sum	32.87	20.00	**16.03**	**125.17**	49.02±3.33
	MPMG-sum	25.32	29.93	25.76	192.42	**51.57**±**1.46**

used for the temporal size. Note that in [41] networks for UCF101 had higher performance since they were pretrained on Kinetics600, while we directly trained on UCF101. A feasible train-from-scratch reference could be [40].

For Eq. 10, we empirically set the coefficient λ as 11 for ShapeNet, 15 for BraTS'18, and 80 for UCF101. Glorot initialization [45] was used for weight initialization. Note that we used orthogonal initialization [46] to handle imbalanced layer-wise neuron importance distribution [19] but obtained lower maximum neuron sparsity.

In addition, loss function and metrics are in Appendix A.

5.2. Maximum Neuron Sparsity by Vanilla NP

We selected MPMG-sum and MNMG-sum for vanilla neuron importance for comparison. All neurons of the last convolutional layer are retained for the given classes.

In Table 1, MPMG-sum for ShapeNet achieves the largest neuron sparsity 78.24% by reducing 76.59% FLOPs, 95.92% parameters, and 44.10% memory with 0.53% accuracy loss. Meanwhile, for BraTS'18, MNMG-sum achieves the largest neuron sparsity 81.32% but has up to 8.48% accuracy loss. MPMG-sum, however, has the largest neuron sparsity 78.17% but smaller accuracy loss with decreased 78.14% FLOPs, 96.46% parameters, and 46.63% memory.

Hence, we selected MPMG-sum as vanilla NP considering the trade-off between the maximum neuron sparsity and the accuracy loss. This is applied to all methods related to weighted neuron pruning and RANP in our experiments. Results of mean and max are in Appendix D.

5.3. Evaluation of RANP on Pruning Capability

Random NP retains κ neurons with neuron indices randomly shuffled. Layer-wise NP retains neurons using the same retain rate as κ in each layer. For SNIP-based parameter pruning, the parameter masks are post-processed by removing redundant parameters and then making sparse filters

---

[3]Since 2 layers of the pruned MobileNetV2 by MNMG-sum have only 1 neuron due to the imbalanced layer-wise neuron importance distribution.

dense, which is denoted as SNIP NP. For a fair comparison with SNIP NP, we used the maximum parameter sparsity 98.98% for ShapeNet , 98.88% for BraTS'18, 86.26% for MobileNetV2, and 81.09% for I3D.

ShapeNet. Compared with random NP and layer-wise NP in Table 2, the maximum reduced resources by vanilla NP are much less due to the imbalanced layer-wise distribution of neuron importance. Weighted neuron importance by Eq. 9, however, further reduces 18.3% FLOPs and 29.6% memory with 0.14% accuracy loss.

Reweighting by RANP-f and RANP-m further reduces FLOPs and memory on the basis of weighted NP. Here, RANP-f can reduce 96.8% FLOPs, 95.3% parameters, and 73.7% memory over the unpruned networks. Furthermore, with a similar resource in Table 3, RANP achieves ~0.5% increase in accuracy. Note that a too-large λ can additionally reduce the resources but at the cost of accuracy.

BraTS'18. In Table 2, RANP-f achieves 96.5% FLOPs, 95.1% parameters, and 80% memory reductions. It further reduces 18.3% FLOPs and 33.3% memory over vanilla NP while increasing -1.21% ET, 5.11% TC, and 0.77% WT. With a similar resource in Table 3, RANP achieves higher accuracy than random NP and layer-wise NP.

Additionally, Chen *et al.*[35] achieved 2× speedup on BraTS'18 with 3D-UNet. In comparison, our RANP-f has roughly 28× speedup, which is theoretically evidenced by the reduced FLOPs from 478.13G to 16.97G in Table 2.

UCF101. In Table 2, for MobileNetV2, RANP-f reduces 55.2% FLOPs, 49% parameters, and 44.1% memory with around 1% accuracy loss. Meanwhile, for I3D, it reduces 49.9% FLOPs, 43.5% parameters, and 35.3% memory with around 2% accuracy increase. The RANP-based methods can reduce much more resources than other methods.

5.4. Resources and Accuracy with Neuron Sparsity

Here, we further studied the tendencies of resources and accuracy with an increasing neuron sparsity level from 0 to the maximum one with network feasibility.

Resource Reductions. In Figs. 4a-4d, RANP, marked with (w), achieves much larger FLOPs and memory reductions than vanilla NP, marked with (w/o), due to the balanced distribution of neuron importance by reweighting.

Specifically, for *ShapeNet*, RANP prunes up to 98.57% neurons while only up to 78.24% by vanilla NP in Fig. 4a. For *BraTS'18*, RANP can prune up to 96.24% neurons while only up to 78.17% neurons can be pruned by vanilla NP in Fig. 4b. For *UCF101*, RANP can prune up to 80.83% neurons compared to 33.15% on MobileNetV2 in Fig. 4c, and 85.3% neurons compared to 25.32% on I3D in Fig. 4d.

Accuracy with Pruning Sparsity. For *ShapeNet* in Fig. 4e, the 23-layer 3D-UNet achieves a higher mIoU than the 15-layer one. Extremely, when pruned with the maximum neuron sparsity 97.99%, it can achieve 78.10% mIoU. With the maximum neuron sparsity 98.57%, however, the

Table 2: Evaluation of neuron pruning capability. All models are **trained from scratch** for 100 epochs on ShapeNet and UCF101, 200 on BraTS'18. Metrics are calculated by the last 5 epochs. "sparsity" is max parameter sparsity for SNIP NP and max neuron sparsity for others. Among the neuron pruning methods, we marked bold **the best** and underlined the second best. "↓" denotes reduction in %. Overall, our RANP-f performs best with large reductions of **main resource consumption** (GFLOPs and memory) with negligible accuracy loss.

Dataset	Model	Manner	Sparsity(%)	Param(MB)	GFLOPs	Memory(MB)	Metrics(%)		
							mIoU		
ShapeNet [10]	3D-UNet	Full[5]	0	62.26	237.85	997.00	83.79±0.21		
		SNIP[18] NP	98.98	5.31 (91.5↓)	126.22 (46.9↓)	833.20 (16.4↓)	**83.70±0.20**		
		Random NP		3.05 (95.1↓)	10.36 (95.6↓)	267.95 (73.1↓)	82.90±0.19		
		Layer-wise NP		2.99 (95.2↓)	11.63 (95.1↓)	296.22 (70.3↓)	83.25±0.14		
		Vanilla NP (ours)	78.24	2.54 (95.9↓)	55.69 (76.6↓)	557.32 (44.1↓)	83.26±0.14		
		Weighted NP		2.97 (95.2↓)	12.06 (94.9↓)	301.56 (69.8↓)	83.12±0.09		
		RANP-m		3.39 (94.6↓)	**6.68 (97.2↓)**	**214.95 (78.4↓)**	82.35±0.24		
		RANP-f		2.94 (95.3↓)	7.54 (96.8↓)	262.66 (73.7↓)	83.07±0.22		
							ET	TC	WT
BraTS'18 [11, 20]	3D-UNet	Full[5]	0	15.57	478.13	3628.00	72.96±0.60	73.51±1.54	86.79±0.35
		SNIP[18] NP	98.88	1.09 (93.0↓)	233.11 (51.2↓)	2999.64 (17.3↓)	**73.33±1.89**	71.98±2.15	**86.44±0.39**
		Random NP		0.75 (95.2↓)	22.59 (95.3↓)	817.59 (77.5↓)	67.27±0.99	71.62±1.20	74.16±1.33
		Layer-wise NP		0.75 (95.2↓)	24.09 (95.0↓)	836.88 (77.0↓)	69.74±1.33	71.49±1.62	86.38±0.39
		Vanilla NP (ours)	78.17	0.55 (96.5↓)	104.50 (78.1↓)	1936.44 (46.6↓)	71.94±1.68	69.39±2.29	84.68±0.78
		Weighted NP		0.79 (95.0↓)	22.40 (95.3↓)	860.64 (76.3↓)	71.50±0.63	**75.05±1.19**	84.05±0.65
		RANP-m		0.87 (94.4↓)	**13.47 (97.2↓)**	**506.97 (86.0↓)**	66.70±2.94	62.99±2.38	82.90±0.41
		RANP-f		0.76 (95.1↓)	16.97 (96.5↓)	729.11 (80.0↓)	70.73±0.66	74.50±1.05	85.45±1.06
							Top-1	Top-5	
UCF101 [40]	MobileNetV2	Full[21]	0	9.47	0.58	157.47	47.08±0.72	76.68±0.50	
		SNIP[18] NP	86.26	3.67 (61.3↓)	0.54 (6.9↓)	155.35 (1.3↓)	45.78±0.04	75.08±0.17	
		Random NP		4.58 (51.6↓)	0.34 (41.4↓)	106.68 (32.3↓)	44.74±0.36	74.69±0.58	
		Layer-wise NP		4.56 (51.8↓)	0.33 (43.1↓)	106.92 (32.1↓)	44.90±0.36	75.54±0.34	
		Vanilla NP (ours)	33.15	6.35 (32.9↓)	0.55 (5.2↓)	155.17 (1.5↓)	**46.32±0.79**	75.42±0.60	
		Weighted NP		4.82 (49.1↓)	0.30 (48.3↓)	100.33 (36.3↓)	46.19±0.51	75.72±0.30	
		RANP-m		4.87 (48.6↓)	0.27 (53.4↓)	**84.51 (46.3↓)**	45.11±0.41	75.53±0.37	
		RANP-f		4.83 (49.0↓)	**0.26 (55.2↓)**	88.01 (44.1↓)	45.87±0.41	**75.75±0.30**	
	I3D	Full[22]	0	47.27	27.88	201.28	51.58±1.86	77.35±0.63	
		SNIP[18] NP	81.09	30.06 (36.4↓)	26.31 (5.6↓)	195.62 (2.8↓)	52.38±3.55	78.32±3.24	
		Random NP		26.36 (44.2↓)	16.45 (41.0↓)	145.07 (27.9↓)	52.42±2.52	79.05±2.06	
		Layer-wise NP		26.67 (43.6↓)	16.93 (39.3↓)	150.95 (25.0↓)	52.77±1.99	78.41±1.07	
		Vanilla NP (ours)	25.32	29.93 (36.7↓)	25.76 (7.6↓)	192.42 (4.4↓)	51.57±1.46	78.07±1.34	
		Weighted NP		26.57 (43.8↓)	15.56 (44.2↓)	142.57 (29.2↓)	54.09±0.82	79.26±0.61	
		RANP-m		26.75 (43.4↓)	14.08 (49.5↓)	130.44 (35.2↓)	52.11±3.05	77.54±2.64	
		RANP-f		26.69 (43.5↓)	**13.98 (49.9↓)**	**130.22 (35.3↓)**	**54.27±2.88**	**79.27±2.13**	

Table 3: In addition to Table 2, with similar GFLOPs or memory on 3D-UNets, our RANP-f achieves the highest accuracy.

Manner	ShapeNet					BraTS'18						
	Sparsity	Param	GFLOPs	Mem	mIoU	Sparsity	Param	GFLOPs	Mem	ET	TC	WT
Random NP	81.01	2.27	∼7.54	253.12	82.66±0.23	81.08	0.56	∼16.97	685.77	61.09±1.87	68.94±2.44	78.89±2.47
Layer-wise NP	82.82	1.84	∼7.54	255.67	82.82±0.26	83.50	0.46	∼16.97	700.64	70.50±0.63	74.27±0.95	83.63±0.92
Random NP	78.83	2.87	9.57	∼262.66	82.86±0.45	80.90	0.57	17.95	∼729.11	68.45±1.11	70.67±1.21	75.02±0.79
Layer-wise NP	82.81	1.94	8.14	∼262.66	82.52±0.13	82.45	0.51	17.31	∼729.11	70.45±1.03	69.27±1.95	82.42±0.68
RANP-f(ours)	78.24	2.94	7.54	262.66	**83.07±0.22**	78.17	0.76	16.97	729.11	**70.73±0.66**	**74.50±1.05**	**85.45±1.06**

15-layer 3D-UNet achieves only 61.42%.

For *BraTS'18* in Fig. 4f, the 23-layer 3D-UNet does not always outperform the 15-layer one and has a larger fluctuation which could be caused by the limited training samples. Nevertheless, even in the extreme case, the 23-layer 3D-UNet has small accuracy loss. Clearly, RANP makes it feasible to use deeper 3D-UNets without the memory issue.

For *UCF101* in Figs. 4g-4h, RANP-f achieves <3% accuracy loss at 70% neuron sparsity, indicating its effectiveness of greatly reducing resources with small accuracy loss.

5.5. Transferability with Interactive Model

In this experiment, we trained on ShapeNet with a transferred 3D-UNet by RANP on BraTS'18 with 80% neuron

Table 4: Transfer learning by 23-layer 3D-UNets interactively pruned and trained between ShapeNet and BraTS'18. Accuracy loss from RANP-f to T-RANP-f is negligible. "T": transferred.

Manner	ShapeNet	BraTS'18		
	mIoU(%)	ET(%)	TC(%)	WT(%)
Full[5]	**84.27±0.21**	**74.04±1.45**	**75.11±2.43**	84.49±0.74
RANP-f(ours)	83.86±0.15	71.13±1.43	72.40±1.48	83.32±0.62
T-RANP-f(ours)	83.25±0.17	72.74±0.69	73.25±1.69	**85.22±0.57**

sparsity. Interactively, with the same neuron sparsity, a transferred 3D-UNet by RANP on ShapeNet was applied to train on BraTS'18. Results in Table 4 demonstrate that training with transferred models crossing different datasets can largely maintain high or higher accuracy.
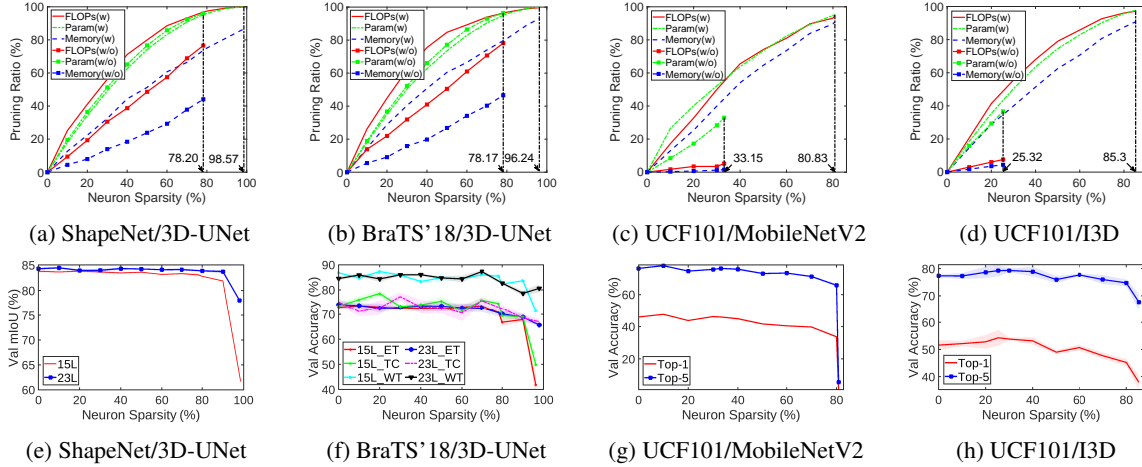
Figure 4: With minimal accuracy loss, more resources are reduced with (w) reweighting by RANP-f than without (w/o) by vanilla NP. (a)-(d) are resources reductions (w) and (w/o) reweighting; (e)-(h) are accuracy and sparsity. Best view in color.

Table 5: ShapeNet: a deeper 23-layer 3D-UNet is achievable on a single GPU with 80% neuron pruning.

Manner	Layer	Batch	GPU(s)	Sparsity(%)	mIoU(%)
Full	15	12	2	0	83.79±0.21
Full	23	12	2	0	84.27±0.21
RANP-f(ours)	23	12	**1**	80	**84.34±0.21**

5.6. Lightweight Training on a Single GPU

RANP with high neuron sparsity makes it feasible to train with large data size on a single GPU due to the largely reduced resources. We trained on ShapeNet with the same batch size 12 and spatial size 64^3 in Sec. 5.1 using a 23-layer 3D-UNet with 80% neuron sparsity on a single GPU. With this setup, RANP-f reduces $\sim 35\times$ GFLOPs (from 259.59 to 7.39) and $\sim 3.9\times$ memory (from 1005.96MB to 255.57MB), making it feasible to train on a single GPU instead of 2 GPUs. It achieves a higher mIoU, 84.34±0.21%, than the 15-layer and 23-layer full 3D-UNets in Table 5.

The accuracy increase is due to the enlarged batch size on each GPU. With limited memory, however, training a 23-layer full 3D-UNet on a single GPU is infeasible.

5.7. Fast Training with Increased Batch Size

Here, we used the largest spatial size 128^3 of one sample on a single GPU and then extended it to RANP with increased batch size from 1 to 4 to fully fill GPU capacity. The initial learning rate was reduced from 0.1 to 0.01 due to the batch size decreased from 12 in Table 5. This is to avoid an immediate increase in training loss right after 1st epoch due to the unsuitably large learning space.

In Fig. 5a, RANP-f enables increased batch size 4 and achieves a faster loss convergence than the full network. In Fig. 5c, the full network executed 6 epochs while RANP-f reached 26 epochs. Vividly shown by training time in Figs. 5b and 5d, RANP-f has much lower loss and higher accuracy than the full one. This greatly indicates the practical advantage of RANP on fastening training convergence.
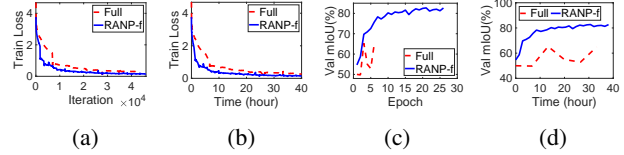

Figure 5: ShapeNet: a faster convergence on a single GPU with 23-layer 3D-UNet and increased batch size due to the largely reduced resources by RANP-f. Batch size is 1 for "Full" and 4 for "RANP-f". Experiments run for 40 hours.

6. Conclusion

In this paper, we propose an effective resource aware neuron pruning method, RANP, for 3D CNNs. RANP prunes a network at initialization by greatly reducing resources with negligible loss of accuracy. Its resource aware reweighting scheme balances the neuron importance distribution in each layer and enhances the pruning capability of removing a high ratio, say 80% on 3D-UNet, of neurons with minimal accuracy loss. This advantage enables training deep 3D CNNs with a large batch size to improve accuracy and achieving lightweight training on one GPU.

Our experiments on 3D semantic segmentation using ShapeNet and BraTS'18 and video classification using UCF101 demonstrate the effectiveness of RANP by pruning 70%-80% neurons with minimal loss of accuracy. Moreover, the transferred models pruned on a dataset and trained on another one are succeeded in maintaining high accuracy, indicating the high transferability of RANP. Meanwhile, the largely reduced computational resources enable lightweight and fast training on one GPU with increased batch size.

Acknowledgement

Appendix

We first provide the pseudocode of our RANP algorithm, then discuss our selection of MPMG-sum as vanilla NP, and justify our reweighting scheme against orthogonal initialization with more ablation experiments.

A. Pseudocode of RANP Procedures

In Alg. 1, we provide the pseudocode of the pruning procedures of RANP. In Alg. 2, we used a simple half-space method to automatically search for the max neuron sparsity with network feasibility. Note that this searching cannot guarantee a small accuracy loss but merely to decide the maximum pruning capability. The relation between pruning capability and accuracy was studied in the experimental section in the main paper and Table 6.

Loss Function and Metrics. Due to the page limitation, we provide loss functions and metrics used in our experiments. Standard cross-entropy function was used as the loss function for ShapeNet and UCF101. For BraTS'18, the weighted function in [39] is

$$L = L_{ce} + \alpha L_{dice} = L_{ce} + \alpha \frac{1}{C} \sum_{i=1}^{C} \frac{2|\mathbf{P}_i \cap \mathbf{G}_i|}{|\mathbf{P}| + |\mathbf{G}|} , \quad (12)$$

where $\alpha = 0.25$ is an empiric weight for dice loss, $\mathbf{P}$ is prediction, $\mathbf{G}$ is ground truth, and C is the number of classes. Meanwhile, ShapeNet accuracy was measured by mean IoU over each part of object category [47] while IoU by $|\mathbf{P} \cap \mathbf{G}|/|\mathbf{P} \cup \mathbf{G}|$ was adopted for BraTS'18. For UCF101 classification, top-1 and top-5 recall rates were used.
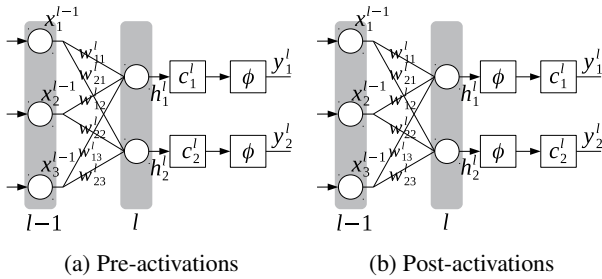
B. Impacts of the Activation Function

(a) Pre-activations   (b) Post-activations

Figure 6: Pre-activations and post-activations, where $\mathbf{x}$ are layer inputs, $\mathbf{w}$ are weights, $\mathbf{c}$ are neuron masks, $\phi(\cdot)$ is an activation function, $\mathbf{h}$ are hidden values, and $\mathbf{y}$ are outputs.

In the following, we first establish the relation between MPMG and MNMG for calculating neuron importance given a homogeneous activation function $\phi(\cdot)$ that includes but not limited to ReLU used in the 3D CNNs. Then we analyze the impact of such an activation function on the calculation of neuron importance by derivating the mask gra-

dients on post-activations and pre-activations illustrated in Figs. 6b and 6a respectively.

Proposition 1 *For a network activation function $\phi(w)$: $\mathbb{R} \to \mathbb{R}$ being a homogeneous function of degree 1 satisfying $\phi(cw) = c\phi(w), \forall c \geq 0$, the neuron mask gradient equals the sum of parameter mask gradients of this neuron.*

Proof: Given a neuron mask c_1 before the activation function $\phi(\cdot)$ in Fig. 6a and the output of the 1st neuron as y_1^l, we have

$$\begin{aligned} y_1^l &= \phi(c_1^l \odot h_1^l) \\ &= \phi \left(c_1^l \odot \left(x_1^{l-1} w_{11}^l + x_2^{l-1} w_{12}^l + x_3^{l-1} w_{13}^l \right) \right) \quad (13) \\ &= \phi \left(c_1^l x_1^{l-1} w_{11}^l + c_1^l x_2^{l-1} w_{12}^l + c_1^l x_3^{l-1} w_{13}^l \right) . \end{aligned}$$

The gradient of loss L over the neuron mask c_1^l is

$$\begin{aligned} \frac{\partial L}{\partial c_1^l} &= \frac{\partial L}{\partial y_1^l} \frac{\partial y_1^l}{\partial c_1^l} \\ &= \frac{\partial L}{\partial y_1^l} \left(x_1^{l-1} w_{11}^l + x_2^{l-1} w_{12}^l + x_3^{l-1} w_{13}^l \right) . \end{aligned} \quad (14)$$

Meanwhile, if setting masks on weights of this neuron directly, we can obtain

$$y_1^l = \phi(c_{11}^l x_1^{l-1} w_{11}^l + c_{12}^l x_2^{l-1} w_{12}^l + c_{13}^l x_3^{l-1} w_{13}^l) , \quad (15)$$

then the gradient of weight mask, *e.g.*, c_{11}^l, from loss is

$$\frac{\partial L}{c_{11}^l} = \frac{\partial L}{\partial y_1^l} \frac{\partial y_1^l}{\partial c_{11}^l} = \frac{\partial L}{\partial y_1^l} x_1^{l-1} w_{11}^l . \quad (16)$$

Similarly,

$$\begin{aligned} &\frac{\partial L}{\partial c_{11}^l} + \frac{\partial L}{\partial c_{12}^l} + \frac{\partial L}{\partial c_{13}^l} \\ &= \frac{\partial L}{\partial y_1^l} \left(x_1^{l-1} w_{11}^l + x_2^{l-1} w_{12}^l + x_3^{l-1} w_{13}^l \right) . \end{aligned} \quad (17)$$

Clearly, Eq. 14 equals Eq. 17. Hence, the neuron mask gradients can be calculated by parameter mask gradients. To this end, the proof is done.

Furthermore, given such a homogeneous activation function in Prop. 1, the importance of a post-activation equals the importance of its pre-activation. In more detail, for post-activations in Fig. 6b, output y_1^l is

$$\begin{aligned} y_1^l &= c_1^l \odot \phi(h_1^l) \\ &= c_1^l \odot \phi \left(x_1^{l-1} w_{11}^l + x_2^{l-1} w_{12}^l + x_3^{l-1} w_{13}^l \right) . \end{aligned} \quad (18)$$

Algorithm 1: Pruning Procedures of RANP-[f|m].

Input: Dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{S}$ with B samples per batch, neuron sparsity κ, resource importance $\{\tau_l\}$, coefficient $\lambda > 0$, and parameter masks $\mathbf{c} = \{c_{uv}^l\}$, where layer $l \in \mathcal{K} = \{1, ..., K\}$, and neuron $u \in \mathcal{N}_l = \{1, ..., N_l\}$.

Output: Binary neuron masks $\hat{\mathbf{c}} = \{\hat{c}_u^l\}$.

1 **for** batch $t \in \{1, ..., \lfloor S/B \rfloor\}$ **do**

2     $\mathcal{D}^t \leftarrow \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=(t-1)B+1}^{tB}$ ▷mini-batch

3     $g_{uv}^l \leftarrow \partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}^t)/\partial c_{uv}^l$ ▷parameter mask gradient, Eq. 2

4     $g_{uv}^l \leftarrow |g_{uv}^l|$, for MPMG   ▷parameter mask importance, Eq. 6

5     $\nabla c_{uv}^l \overset{+}{\leftarrow} g_{uv}^l, \forall u \in \mathcal{N}_l, \forall v \in \mathcal{N}_{l-1}$ ▷gradient accumulation

6 $\nabla c_{uv}^l \leftarrow \nabla c_{uv}^l / \lfloor S/B \rfloor, \forall u \in \mathcal{N}_l, \forall v \in \mathcal{N}_{l-1}, \forall l \in \mathcal{K}$ ▷average on mini-batch

7 $s_u^l \leftarrow |\sum_{v=1}^{N_{l-1}} \nabla c_{uv}^l|, \forall u \in \mathcal{N}_l, \forall l \in \mathcal{K}$ ▷vanilla neuron importance, Eq. 8

8 $\bar{s}^l \leftarrow \sum_{u \in \mathcal{N}_l} s_u^l / N_l, \forall l \in \mathcal{K}$ ▷mean neuron importance, Eq. 9

9 $\tilde{s}_u^l \leftarrow (\max_{j \in \mathcal{K}} \bar{s}^j / \bar{s}^l) s_u^l, \forall u \in \mathcal{N}_l, \forall l \in \mathcal{K}$ ▷weighting, Eq. 9

10 $\hat{s}_u^l \leftarrow (1 + \lambda e^{-\tau_l} / \sum_{j \in \mathcal{K}} e^{-\tau_j}) \tilde{s}_u^l, \forall u \in \mathcal{N}_l, \forall l \in \mathcal{K}$ ▷reweighting, Eq. 10

11 $\{\ddot{s}_u\} \leftarrow \text{SortDescending}(\{\hat{s}_u^l\}), \forall u \in \mathcal{N}_l, \forall l \in \mathcal{K}$ ▷sorting in descending

12 $\hat{c}_u^l \leftarrow 1[\hat{s}_u^l - \ddot{s}_\kappa \geq 0], \forall u \in \mathcal{N}_l, \forall l \in \mathcal{K}$ ▷binary neuron mask, Eq. 11

Algorithm 2: Auto-Search for Max Neuron Sparsity .

Input: Dataset $\mathcal{D}$, layerwise resource usage $\boldsymbol{\tau}$ w.r.t. FLOPs or memory, coefficient $\lambda > 0$, lower and upper sparsity κ_{min} and κ_{max}, threshold $\delta = 1e-4$. "feasible network" means not all neurons are removed in each layer.

Output: Max neuron sparsity κ^*.

1 Initilize $\kappa_{min} \leftarrow 0, \kappa_{max} \leftarrow 1$

2 **while** $(\kappa_{max} - \kappa_{min} > \delta)$ **do**

3     $\kappa = 0.5(\kappa_{min} + \kappa_{max})$

4     $y = \text{NeuronPruning}(\mathcal{D}, \boldsymbol{\tau}, \lambda, \kappa)$ ▷Alg. 1

5   **if** $y == 0$(feasible network) **then**

6        $\kappa_{min} \leftarrow \kappa$

7   **else**

8        $\kappa_{max} \leftarrow \kappa$

9 $\kappa^* = \kappa$

Since the activation function satisfies $c\phi(w) = \phi(cw)$,

$$y_1^l = \phi(c_1^l x_1^{l-1} w_{11}^l + c_1^l x_2^{l-1} w_{12}^l + c_1^l x_3^{l-1} w_{13}^l). \quad (19)$$

The neuron importance determined by neuron mask c_1^l is

$$\begin{aligned}
\frac{\partial L}{\partial c_1^l} &= \frac{\partial L}{\partial y_1^l} \frac{\partial y_1^l}{\partial c_1^l} \\
&= \frac{\partial L}{\partial y_1^l} \left(x_1^{l-1} w_{11}^l + x_2^{l-1} w_{12}^l + x_3^{l-1} w_{13}^l \right).
\end{aligned} \quad (20)$$

Clearly, Eq. 20 equals Eq. 14. Now, the importance of pre-activations and post-activations is the same given such a homogeneous activation function.

C. Resource Aware Reweighting Scheme

As described in Sec. 4.2 in the main paper, the reweighting of RANP is conducted by first balancing the layer-wise distribution of neuron importance and then adopting resource importance τ_l for layer $l \in \mathcal{K}$ to further reduce resources. Since FLOPs and memory are the main resources of 3D CNNs, τ_l is defined by FLOPs or memory as follows.

Generally, given input dimension of the lth layer $(x_{\text{in}}, x_h, x_w, x_d)$[3], neuron dimension $(f_{\text{out}}, f_{\text{in}}, f_h, f_w, f_d)$, and output dimension $(y_{\text{in}}, y_h, y_w, y_d)$ with $x_{\text{in}} = f_{\text{in}}$ and $f_{\text{out}} = y_{\text{in}}$, the resource importance in terms of FLOPs or memory is defined by

$$\begin{aligned}
\text{FLOPs:} \quad \tau_l &= [(f_h f_w f_d + f_h f_w f_d - 1) f_{\text{in}} \\
&\quad + f_{\text{in}} - 1 + 1|_{\text{bias}}] y_{\text{in}} y_h y_w y_d \\
&= (2 f_h f_w f_d f_{\text{in}} - 1 + 1|_{\text{bias}}) y_{\text{in}} y_h y_w y_d, \quad (21a)
\end{aligned}$$

$$\text{Memory:} \quad \tau_l = y_{\text{in}} y_h y_w y_d, \quad (21b)$$

where $(f_h f_w f_d)$ is the number of operations of multiplications of filter[4] and layer input, $(f_h f_w f_d - 1)$ is for additions of values from the multiplications, (f_{in}) is for multiplications over all f_{in} filters, $(f_{\text{in}} - 1)$ is for additions of values from all these multiplications, $(1|_{\text{bias}})$ is for an addition when the neuron has a bias, and $(y_{\text{in}} y_h y_w y_d)$ is for all elements of the layer output.

D. More Ablation Study

In this section, we add more experimental results for the analysis of selecting MPMG-sum as vanilla NP, Glorot initialization for network initialization compared with orthog-

---

[3]The dimension order follows that of PyTorch.

[4]Here, we refer a 3D filter with dimension (f_h, f_w, f_d).

onal initialization [19] to handle the imbalanced layer-wise distribution of neuron importance, and visualization of neuron distribution by RANP for BraTS'18 in addition to that for ShapeNet in the main paper.

Figures in this sections are for 3D-UNets on ShapeNet and BraTS'18 because 3D-UNets used in our experiments typically clarify the neuron imbalance and memory issues and are clear for illustration with a limited number of layers, *i.e.*, 15 layers, while MobileNetV2 and I3D have more than 55 layers but many are not typical 3D convolutional layers with 3^3 kernel size filters.

D.1. MPMG-sum as Vanilla Neuron Pruning

In Sec. 5.2 in the main paper, we select MPMG-sum as vanilla neuron pruning for the trade-off between computational resources and accuracy. To give a comprehensive study of this selection, we demonstrate detailed results of mean, max, and sum operations of MPMG and MNMG in Table 6. Note that we relax the sum operation in Eq. 8 in the main paper to mean, max, and sum.

In Table 6, we aim at obtaining the maximum neuron sparsity due to the target of reducing the computational resources at an extreme sparsity level with minimal accuracy loss. Vividly, for *ShapeNet*, MPMG-sum achieves the largest maximum neuron sparsity 78.24% among all with only ∼0.53% accuracy loss. Differently, for *BraTS'18*, MNMG-sum has the largest maximum neuron sparsity 81.32%; however, the accuracy loss can reach up to ∼8.48%. In contrast, while MPMG-sum has the second-largest maximum neuron sparsity 78.17%, the accuracy loss is much smaller than MNMG-sum. For *UCF101*, it is surprising that many manners have low accuracy. As we analyse the reason in the footnote in Table 6, with the extreme neuron sparsity, some layers of the pruned networks have only 1 neuron retained, losing sufficient features for learning, and thus, leading to low accuracy.

Hence, considering the comprehensive performance of reducing resources and maintaining the accuracy, MPMG-sum is selected as vanilla NP. Note that any neuron sparsity greater than the maximum neuron sparsity will make the pruned network infeasible by pruning the whole layer(s).

---

[3]For MobileNetV2 pruned by MPMG-mean, MPMG-max, MNMG-max, and MNMG-sum, the accuracy is very low because 1) the neuron sparsity here is the extreme (largest) value, a larger one will make network infeasible by removing whole layer(s) and 2) the distribution of neuron importance is rather imbalanced possibly caused by the high mixture of 1^3 kernels and 3^3 in MobileNetV2.

In the pruned networks, we observe that, for MPMG-mean, MPMG-max, and MNMG-max, the last convolutional layer has only 1 neuron retained; for MNMG-sum, 2 convolutional layers have only 1 neuron retained. Note that, this imbalance issue can be greatly alleviated by the reweighting of our RANP, while we select MPMG-sum as vanilla NP merely according to the results in Table 6.

D.2. Initialization for Neuron Imbalance

The imbalanced layer-wise distribution of neuron importance hinders pruning at a high sparsity level due to the pruning of the whole layer(s). For 2D classification tasks in [19], orthogonal initialization is used to effectively solve this problem for balancing the importance of parameters; but it does not improve our neuron pruning results in 3D tasks and even leads to a poor pruning capability with a lower maximum neuron sparsity than Glorot initialization [45]. This is briefly mentioned in Sec. 4.1 in the main paper. Here, we compare the resource reducing capability using Glorot initialization and orthogonal initialization.

Resource reductions. In Table 7, vanilla neuron pruning (*i.e.*, MPMG-sum) with Glorot initialization, *i.e.*, vanilla-xn, achieves smaller FLOPs and memory consumption than those with orthogonal initialization, *i.e.*, vanilla-ort, except FLOPs with 3D-UNet on ShapeNet and I3D on UCF101. This exception of I3D on UCF101 is possibly caused by the high ratio of 1^3 kernel size filters in I3D, *i.e.*, 37 out of 57 convolutional layers, because those 1^3 kernel size filters can be regarded as 2D filters on which orthogonal initialization can effectively deal with [19]. While this ratio is also high in MobileNetV2, *i.e.*, 34 out of 52 convolutional layers, it is unnecessary to have the same problem as I3D since it is also affected by the number of neurons in each layer. Note that since 3D-UNets used are all with 3^3 kernel size filters, the orthogonal initialization for 3D-UNet in most cases is inferior to Glorot initialization according to our experiments. Meanwhile, in Table 7, this gap between vanilla-ort and vanilla-xn is very small on MobileNetV2 and I3D.

Nevertheless, with RANP-f and Glorot initialization, *i.e.*, RANP-f-xn, more FLOPs and memory can be reduced than using orthogonal initialization, *i.e.*, RANP-f-ort.

Balance of Neuron Importance Distribution. More importantly, with reweighting by RANP in Fig. 7, the values of neuron importance are more balanced and stable than those of vanilla neuron importance. This can largely avoid network infeasibility without pruning the whole layer(s).

Now, we analyse the neuron distribution from the observation of neuron importance values and network structures. *Fig. 8 illustrates a detailed comparison between orthogonal and Glorot initialization by each two subfigures in column of Fig. 7.* In Figs. 8a-8c, vanilla neuron importance by Glorot initialization is more stable and compact than that by orthogonal initialization. After applying the reweighting scheme of RANP-f, the importance tends to be in a similar tendency, shown in Figs. 8b-8d. Consequently, in Figs. 8e-8f, neuron ratios are more balanced after the reweighting than without reweighting, especially the 8th layer. Thus, we choose Glorot initialization as network initialization. Note that we adopt the same neuron sparsity for these two initialization experiments in Table 7 and Fig. 8.

Table 6: More results of vanilla NP in addition to Table 1 in the main paper. **Main resource consumption** (GFLOPs and memory) are considered but not parameters whose resource consumption is much smaller than memory. Among the neuron pruning methods, we marked bold **the best** and underlined <u>the second best</u>. Overall, we selected MPMG-sum as vanilla NP and the corresponding neuron sparsity for large resource reductions with small accuracy loss.

Dataset	Model	Manner	Sparsity(%)	Param(MB)	GFLOPs	Memory(MB)	Metrics(%)		
							mIoU		
ShapeNet	3D-UNet	Full[5]	0	62.26	237.85	997.00	83.79±0.21		
		MPMG-mean	68.10	5.08	110.14	819.97	83.33±0.18		
		MPMG-max	70.24	4.54	107.38	809.88	**83.79±0.10**		
		MPMG-sum	78.24	2.54	**55.69**	**557.32**	83.26±0.14		
		MNMG-mean	63.03	4.23	112.95	834.98	83.46±0.13		
		MNMG-max	73.93	3.67	103.57	796.44	83.51±0.08		
		MNMG-sum	66.93	4.29	<u>100.34</u>	<u>783.14</u>	<u>83.65±0.02</u>		
							ET	TC	WT
BraTS'18	3D-UNet	Full[5]	0	15.57	478.13	3628.00	72.96±0.60	73.51±1.54	86.79±0.35
		MPMG-mean	65.64	1.48	226.86	3038.27	<u>73.51±0.82</u>	<u>73.28±1.14</u>	<u>87.15±0.43</u>
		MPMG-max	75.78	0.83	189.43	2812.53	**73.67±0.98**	72.73±1.70	86.44±0.71
		MPMG-sum	78.17	0.55	<u>104.50</u>	<u>1936.44</u>	71.94±1.68	69.39±2.29	84.68±0.78
		MNMG-mean	63.85	1.08	176.76	2790.64	73.35±0.70	**73.38±0.94**	**87.21±0.38**
		MNMG-max	80.05	0.59	169.99	2676.05	72.52±1.91	72.40±1.74	84.63±0.60
		MNMG-sum	81.32	0.35	**73.50**	**1933.20**	64.48±1.10	68.47±1.59	80.71±1.07
							Top-1	Top-5	
UCF101	MobileNetV2	Full[21]	0	9.47	0.58	157.47	47.08±0.72	76.68±0.50	
		MPMG-mean	26.31	4.39	0.55	156.00	2.98±0.14 [5]	14.04±0.14	
		MPMG-max	29.48	3.96	0.54	155.38	3.49±0.12	13.64±0.10	
		MPMG-sum	33.15	6.35	0.55	155.17	**46.32±0.79**	**75.42±0.60**	
		MNMG-mean	38.91	2.79	<u>0.50</u>	<u>147.69</u>	<u>29.13±0.92</u>	<u>62.93±1.37</u>	
		MNMG-max	50.33	2.59	0.53	153.45	2.84±0.06	13.40±0.23	
		MNMG-sum	39.89	4.66	**0.43**	**120.01**	1.03±0.00	5.76±0.00	
	I3D	Full[22]	0	47.27	27.88	201.28	51.58±1.86	77.35±0.63	
		MPMG-mean	16.47	31.57	26.50	196.51	<u>51.88±2.00</u>	77.98±1.46	
		MPMG-max	19.83	30.06	26.31	195.62	**52.44±1.25**	**78.08±1.27**	
		MPMG-sum	25.32	29.93	25.76	192.42	51.57±1.46	<u>78.07±1.34</u>	
		MNMG-mean	35.36	16.69	**15.37**	**124.85**	49.26±0.96	75.70±1.49	
		MNMG-max	40.27	17.86	23.73	184.77	44.90±1.19	74.43±1.26	
		MNMG-sum	32.87	20.00	<u>16.03</u>	<u>125.17</u>	46.90±1.26	74.02±1.25	

D.3. Visualization of Balanced Neuron Distribution by RANP

In Fig. 9, neuron importance by MPMG-sum is more balanced than by MNMG-sum, which avoids pruned networks by MPMG-sum to be infeasible, that is at least 1 neuron will be retained in each layer.

In addition to the distribution of retained neuron ratios in Fig. 2 in the main paper for ShapeNet, which is also shown in the first row of Fig. 10, the last row of Fig. 10 is for BraTS'18. Moreover, Fig. 11 illustrates the distribution of neurons retained in each layer by vanilla neuron pruning (*i.e.*, vanilla NP) and RANP-f compare to the full network.

Clearly, upon pruning, neurons in each layer are largely reduced except the last layer where all neurons are retained for the number of segmentation classes. In Fig. 11, vanilla NP has very few neurons in, *e.g.*, the 8th layer, resulting in low accuracy or network infeasibility. By contrast, the neuron distribution by RANP-f is more balanced to improve the pruning capability.

References

[1] H. Zhang, K. Jiang, Y. Zhang, Q. Li, C. Xia, and X. Chen, "Discriminative feature learning for video semantic segmentation," *International Conference on Virtual Reality and Visualization*, 2014.

[2] C. Zhang, W. Luo, and R. Urtasun, "Efficient convolutions for real-time semantic segmentation of 3D point cloud," *3DV*, 2018.

[3] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *TPAMI*, 2013.

[4] R. Hou, C. Chen, R. Sukthankar, and M. Shah, "An efficient 3D CNN for action/object segmentation in video," *BMVC*, 2019.

[5] O. Cicek, A. Abdulkadir, S. Lienkamp, T. Brox, and O. Ronneberger, "3D U-Net: Learning dense volumetric segmentation from sparse annotation," *MICCAI*, 2016.

[6] F. Zanjani, D. Moin, B. Verheij, F. Claessen, T. Cherici, T. Tan, and P. With, "Deep learning approach to semantic segmentation in 3D point cloud intra-oral scans of teeth," *Proceedings of Machine Learning Research*, 2019.

(a) ShapeNet, Vanilla NP-ort (b) ShapeNet, Vanilla NP-xn (c) ShapeNet, RANP-f-ort (d) ShapeNet, RANP-f-xn

(e) BraTS'18, Vanilla NP-ort (f) BraTS'18, Vanilla NP-xn (g) BraTS'18, RANP-f-ort (h) BraTS'18, RANP-f-xn
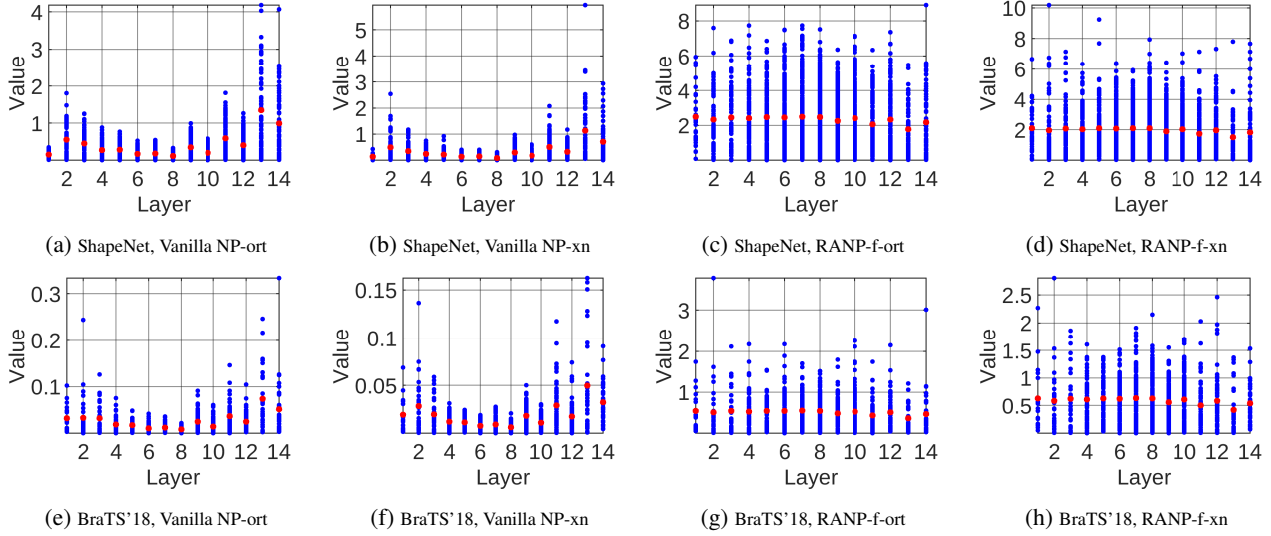
Figure 7: Neuron importance of 15-layer 3D-UNet by MPMG-sum with orthogonal and Glorot initialization. Blue: neuron values; red: mean values. By vanilla NP, orthogonal initialization does not result in a balanced neuron importance distribution compared to Glorot initialization whereas by our RANP-f, the values are more balanced and resource aware on FLOPs, enabling pruning at the extreme sparsity.

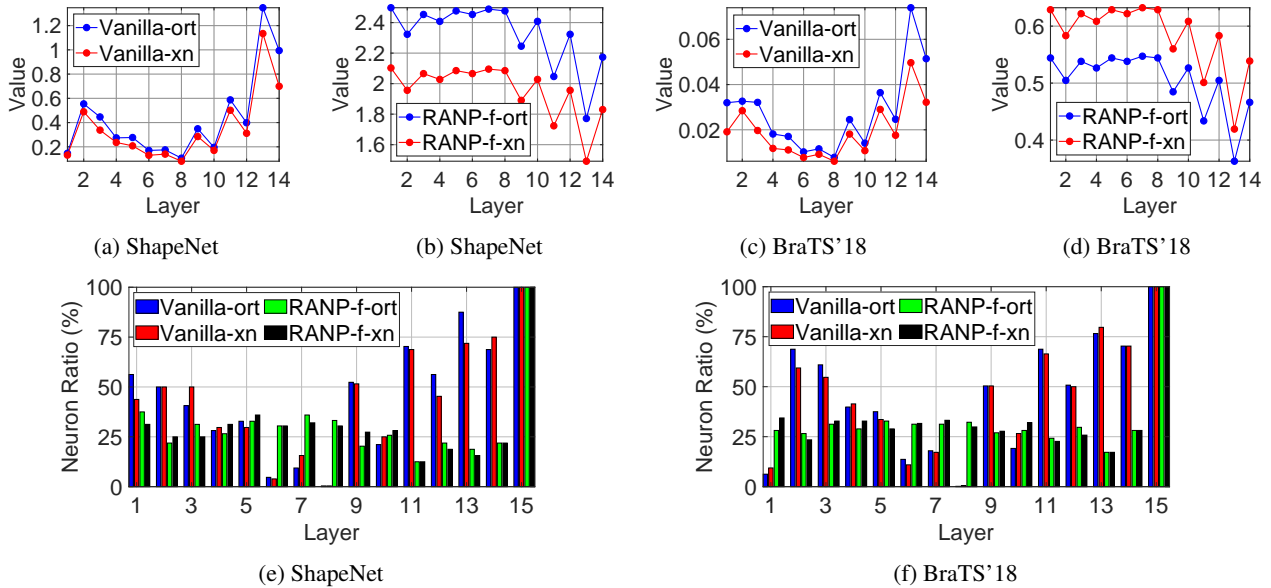(a) ShapeNet (b) ShapeNet (c) BraTS'18 (d) BraTS'18

(e) ShapeNet (f) BraTS'18

Figure 8: Comparison of neuron distribution with orthogonal and Glorot initialization before and after reweighting. (a)-(d) are neuron importance values. (e)-(f) are neuron retained ratios. Vanilla versions (both orthogonal and Glorot initializations) prune all the neuron in layer 8, leading to network infeasibility while our RANP-f versions have a balanced distribution of retained neurons.

[7] J. Kleesiek, G. Urban, A. Hubert, D. Schwarz, K. Hein, M. Bendszus, and A. Biller, "Deep MRI brain extraction: A 3D convolutional neural network for skull stripping," *NeuroImage*, 2016.

[8] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," *CVPR*, 2014.

[9] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *NeurIPS*, 2014.

[10] L. Yi, V. Kim, D. Ceylan, I. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, and L. Guibas, "A scalable active framework for region annotation in 3D shape collections," *SIGGRAPH Asia*, 2016.

(a) ShapeNet, MNMG-sum (b) ShapeNet, MPMG-sum (c) BraTS'18, MNMG-sum (d) BraTS'18, MPMG-sum
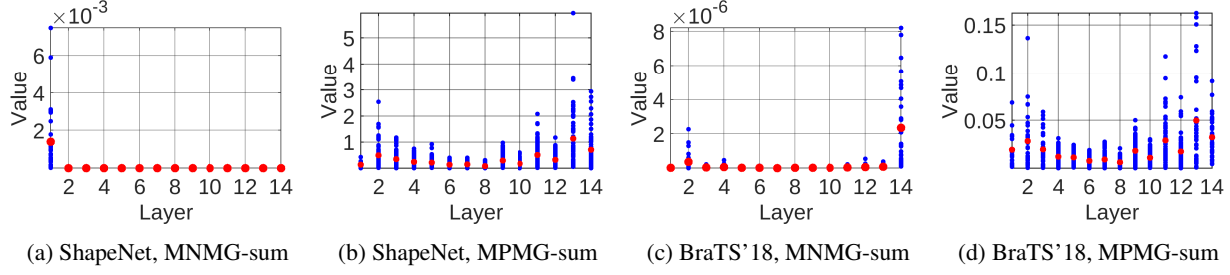
Figure 9: MNMG-sum and MPMG-sum on ShapeNet and BraTS'18 with max neuron sparsity in Table 6. Blue: neuron values; red: mean values. Clearly, neuron importance distribution by MPMG-sum is more balanced than by MNMG-sum.

(a) ShapeNet, Vanilla NP Eq. 6 (b) ShapeNet, Weighted NP Eq. 9 (c) ShapeNet, RANP-f Eq. 10 (d) ShapeNet, Retain ratio

(e) BraTS'18, Vanilla NP Eq. 6 (f) BraTS'18, Weighted NP Eq. 9 (g) BraTS'18, RANP-f Eq. 10 (h) BraTS'18, Retain ratio
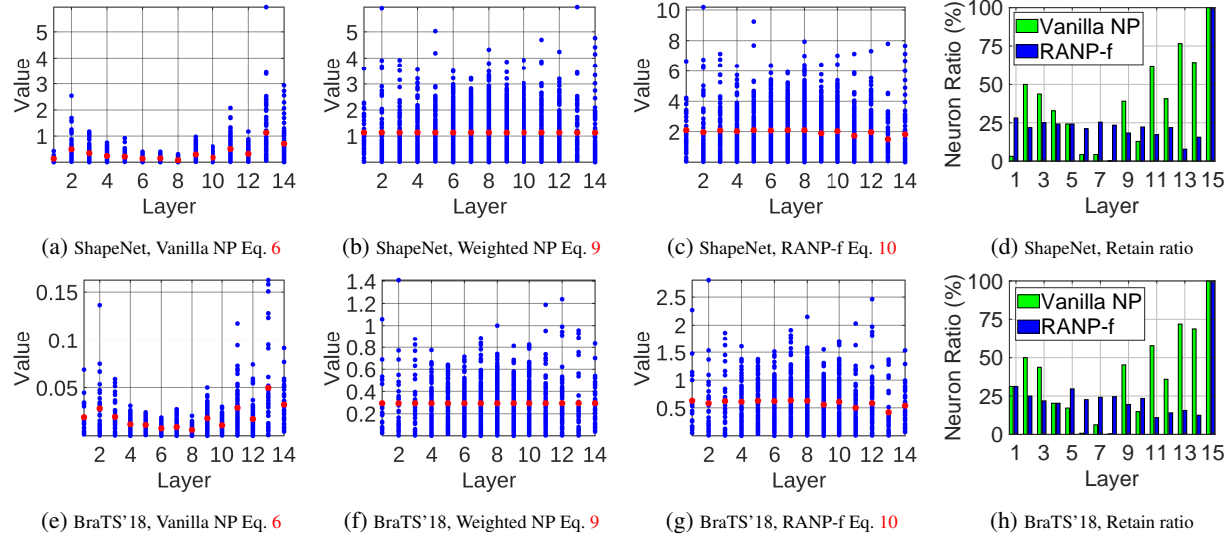
Figure 10: Balanced neuron importance distribution by MPMG-sum on ShapeNet and BraTS'18. Neuron sparsity is 78.24% on ShapeNet and 78.17% on BraTS'18. Blue: neuron values; red: mean values.
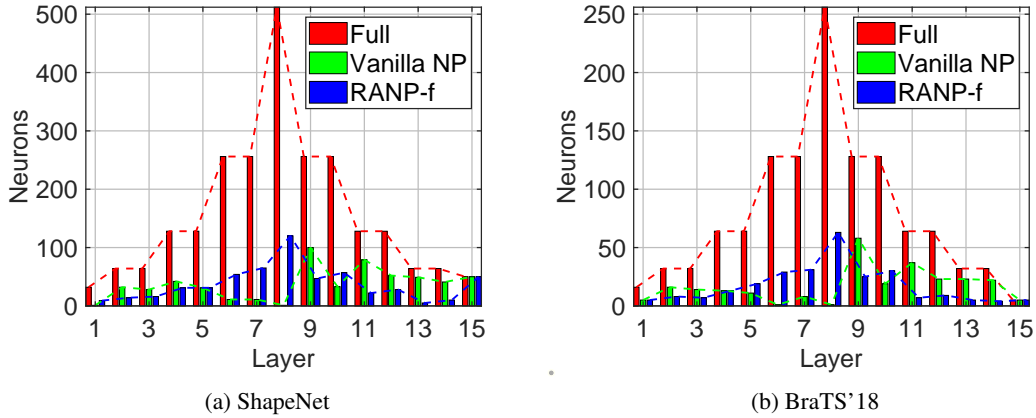

(a) ShapeNet (b) BraTS'18

Figure 11: Layer-wise neuron distribution of 3D-UNets.

[11] B. Menze, A. Jakab, and S. B. *et al*, "The multimodal brain tumor image segmentation benchmark (brats)," *IEEE Transactions on Medical Imaging*, 2015.

[12] B. Graham, M. Engelcke, and L. Maaten, "3D semantic segmentation with submanifold sparse convolutional networks," *CVPR*, 2018.

[13] C. Qi, H. Su, K. Mo, and L. Guibas, "Pointnet: Deep learning on point sets for 3D classification and segmentation," *CVPR*,

Table 7: Impact of parameter initialization on neuron pruning. "ort": orthogonal initialization; "xn": Glorot initialization; "f": FLOPs. "Sparsity" is the least max neuron sparsity among all manners to ensure the network feasibility. RANP-f with Glorot initialization achieves the least FLOPs and memory consumption.

Dataset(Model)	Manner	Sparsity(%)	Param(MB)	GFLOPs	Mem(MB)
ShapeNet (3D-UNet)	Full[5]	0	62.26	237.85	997.00
	Vanilla-ort	70.53	4.40	72.65	630.00
	Vanilla-xn	70.53	4.56	73.22	618.35
	RANP-f-ort	70.53	5.40	21.73	366.29
	RANP-f-xn	70.53	5.52	**15.06**	**328.66**
BraTS'18 (3D-UNet)	Full[5]	0	15.57	478.13	3628.00
	Vanilla-ort[19]	72.20	0.95	159.91	2240.33
	Vanilla-xn	72.20	0.92	130.28	2109.19
	RANP-f-ort	72.20	1.24	33.28	967.56
	RANP-f-xn	72.20	1.29	**23.31**	**850.56**
UCF101 (MobileNetV2)	Full[21]	0	9.47	0.58	157.47
	Vanilla-ort[19]	30.21	6.80	0.56	155.71
	Vanilla-xn	30.21	6.77	0.55	155.48
	RANP-f-ort	30.21	5.12	0.32	105.88
	RANP-f-xn	30.21	5.19	**0.28**	**94.50**
UCF101 (I3D)	Full[22]	0	47.27	27.88	201.28
	Vanilla-ort[19]	24.24	30.56	25.83	192.70
	Vanilla-xn	24.24	30.64	25.83	192.88
	RANP-f-ort	24.24	27.39	15.94	144.10
	RANP-f-xn	24.24	27.38	**14.63**	**133.80**

2017.

[14] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," *NeurIPS*, 2016.

[15] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," *NeurIPS*, 2017.

[16] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," *ICCV*, 2017.

[17] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *NeurIPS*, 2015.

[18] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," *ICLR*, 2019.

[19] N. Lee, T. Ajanthan, S. Gould, and P. Torr, "A signal propagation perspective for pruning neural networks at initialization," *ICLR*, 2020.

[20] S. Bakas, H. Akbari, A. Sotiras, M. Bilello, M. Rozycki, J. Kirby, J. Freymann, K. Farahani, and C. Davatzikos, "Advancing the cancer genome atlas glioma MRI collections with expert segmentation labels and radiomic features," *Nature Scientific Data*, 2017.

[21] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *CVPR*, 2018.

[22] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the Kinetics dataset," *CVPR*, 2017.

[23] C. Chen, F. Tung, N. Vedula, and G. Mori, "Constraint-aware deep neural network compression," *ECCV*, 2018.

[24] N. Yu, S. Qiu, X. Hu, and J. Li, "Accelerating convolutional neural networks by group-wise 2D-filter pruning," *IJCNN*, 2017.

[25] H. Li, A. Kadav, I. D. H. Samet, and H. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[26] R. Yu, A. Li, C. Chen, J. Lai, V. Morariu, X. Han, M. Gao, C. Lin, and L. Davis, "NISP: Pruning networks using neuron importance score propagation," *CVPR*, 2018.

[27] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," *ECCV*, 2018.

[28] Y. He, J. Lin, Z. Liu, H. Wang, L. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," *ECCV*, 2018.

[29] M. Zhang and B. Stadie, "One-shot pruning of recurrent neural networks by jacobian spectrum evaluation," *arXiv:1912.00120*, 2019.

[30] C. Li, Z. Wang, X. Wang, and H. Qi, "Single-shot channel pruning based on alternating direction method of multipliers," *arXiv:1902.06382*, 2019.

[31] J. Yu and T. Huang, "Autoslim: Towards one-shot architecture search for channel numbers," *arXiv:1903.11728*, 2019.

[32] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," *ICLR*, 2020.

[33] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *ICLR*, 2017.

[34] Y. Zhang, H. Wang, Y. Luo, L. Yu, H. Hu, H. Shan, and T. Quek, "Three dimensional convolutional neural network pruning with regularization-based method," *ICIP*, 2019.

[35] H. Chen, Y. Wang, H. Shu, Y. Tang, C. Xu, B. Shi, C. Xu, Q. Tian, and C. Xu, "Frequency domain compact 3D convolutional neural networks," *CVPR*, 2020.

[36] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," *CVPR*, 2018.

[37] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *NeurIPS*, 2016.

[38] G. Riegler, A. Ulusoy, and A. Geiger, "OctNet: Learning deep 3D representations at high resolutions," *CVPR*, 2017.

[39] P. Kao, T. Ngo, A. Zhang, J. Chen, and B. Manjunath, "Brain tumor segmentation and tractographic feature extraction from structural MR images for overall survival prediction," *Workshop on MICCAI*, 2018.

[40] K. Soomro, A. Zamir, and M. Shah, "UCF101: A dataset of 101 human action classes from videos in the wild," *CRCV-Techinal Report*, 2012.

[41] O. Kopuklu, N. Kose, A. Gunduz, and G. Rigoll, "Resource efficient 3D convolutional neural networks," *ICCVW*, 2019.

[42] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," *ICML*, 2013.

[43] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ICLR*, 2015.

[44] S. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *ICLR*, 2018.

[45] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

[46] A. Saxe, J. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *ICLR*, 2014.

[47] L. Yi, L. Shao, and M. Savva, "Large-scale 3D shape reconstruction and segmentation from shapenet core55," *arXiv preprint arXiv:1710.06104*, 2017.