Supplementary Material on SURGEON: Memory-Adaptive Fully Test-Time Adaptation via Dynamic Activation Sparsity

Ke Ma[1,2] Jiaqi Tang[3] Bin Guo[1*] Fan Dang[4] Sicong Liu[1] Zhui Zhu[2]
Lei Wu[1] Cheng Fang[1] Ying-Cong Chen[3] Zhiwen Yu[1,5] Yunhao Liu[2*]
[1]Northwestern Polytechnical University [2]Tsinghua University
[3]The Hong Kong University of Science and Technology
[4]Beijing Jiaotong University [5]Harbin Engineering University

Project Page: https://github.com/kadmkbl/SURGEON

Abstract

In the supplementary material, we first provide additional experiments to intuitively illustrate the issue of adaptation memory cost. We then introduce the foundation of our method, activation sparsity, and examine its impact across different network architectures used in the main paper's experiments. Next, we analyze the factors contributing to **SURGEON***'s superior accuracy. Additionally, we provide details on the datasets and implementation of both* **SURGEON** *and the baselines. Furthermore, we present additional comparisons with backward-free methods. Finally, we discuss the limitations of our work and outline potential future directions.*

1. Excessive Memory Cost of Adaptation on IoT Terminals

To illustrate the memory cost issue of adaptation more intuitively, we implement experiments on three convolutional networks used in **SURGEON**. Specifically, we train WideResNet-28 [27] and ResNeXt-29 [26] on the CIFAR-10 and CIFAR-100 datasets [14], respectively, with a batch size of 64. Additionally, DeeplabV3+ [2] with ResNet-50 [7] is trained on Cityscapes [4], with a batch size of 2.

From Figure 1, we can observe the following: (i) The memory usage induced by adaptation easily exceeds the memory budgets of current mainstream IoT terminals, such as the Raspberry Pi 3B+[1] and Jetson XAVIER NX[2]. (ii) Among the components contributing to adaptation memory usage, *activations constitute the majority*, accounting for 81.0%, 97.1%, and 96.7% of the total memory usage, respectively. This finding supports the point we make in § 3.2 Optimization Objective of the main paper and serves as one of the motivations for our method. Our method directly and flexibly prunes activations at layer-specific dynamic ratios to achieve the optimal balance between minimizing memory cost and maximizing accuracy during FTTA.

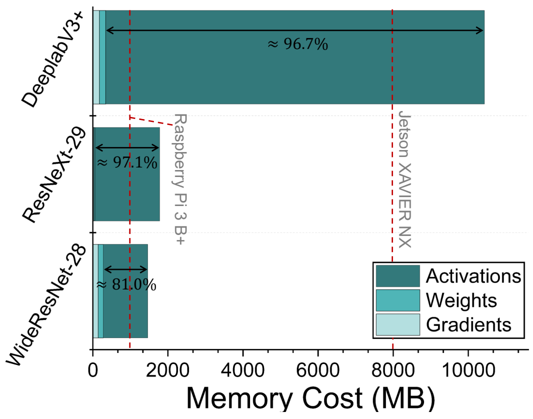


Figure 1. **Memory cost of different networks**. Memory usage in "Activations" is significantly higher than other components ("Weights" and "Gradients") during adaptation.

2. Activation Sparsity across Various Architectures

In this section, we will discuss the network architectures used in the experiments of the main paper and explain the impact of applying activation sparsity across these different types of architectures during adaptation.

In the experiments, we utilize different convolution-

*Corresponding authors. guob@nwpu.edu.cn, yunhao@tsinghua.edu.cn

[1]https://www.raspberrypi.com/products/

[2]https://www.nvidia.com/en-us/autonomous-machines/

based networks and one transformer-based network. To clarify how activation sparsity affects weight updates by altering weight gradient calculations during adaptation, we provide a detailed explanation of the weight gradient calculation process for the *trainable layers*. Let x denote the input features and z the output features.

2.1. Convolutional Layers

- Forward Propagation:

$$z_{m,n,k} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{c=1}^{C} W_{p,q,c,k} \cdot x_{m+p-1,n+q-1,c} + b_k, \tag{1}$$

where W denotes the convolutional filter, b represents the bias, P and Q are the height and width of the filter, C is the number of input channels, m and n are spatial indices, and k indicates the output channel index.

- Backward Propagation:

$$\begin{aligned} \delta_W^{p,q,c,k} &= \sum_{m=1}^{M} \sum_{n=1}^{N} \delta_z^{m,n,k} \cdot x_{m+p-1,n+q-1,c}, \\ \delta_b^k &= \sum_{m=1}^{M} \sum_{n=1}^{N} \delta_z^{m,n,k}, \end{aligned} \tag{2}$$

where $\delta_z^{m,n,k}$ is the gradient of the output z, $\delta_W^{p,q,c,k}$ represents the gradient of the convolutional filter's weights, and δ_b^k is the gradient of the convolutional filter's bias.

2.2. Fully Connected Layers

- Forward Propagation:

$$z_k = \sum_{j=1}^{J} W_{jk} x_j + b_k, \tag{3}$$

where W is the weight matrix, b is the bias vector, J is the input dimension, and k is the output element index.

- Backward Propagation:

$$\delta_W^{jk} = \delta_z^k \cdot x_j, \quad \delta_b^k = \delta_z^k, \tag{4}$$

where δ_z^k is the gradient of the output z, δ_W^{jk} represents the weight gradient, and δ_b^k represents the bias gradient.

2.3. Batch Normalization Layers

- Forward Propagation:

$$\begin{aligned} \mu = \frac{1}{N} \sum_{n=1}^{N} x_n, \quad \sigma^2 = \frac{1}{N} \sum_{n=1}^{N} (x_n - \mu)^2, \\ \hat{x}_n = \frac{x_n - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad z_n = \gamma \hat{x}_n + \beta, \end{aligned} \tag{5}$$

where γ and β are the learnable scale and shift parameters, N is the batch size, and ϵ is a small constant for numerical stability.

- Backward Propagation:

$$\delta_\gamma = \sum_{n=1}^{N} \delta_z^n \cdot \hat{x}_n, \quad \delta_\beta = \sum_{n=1}^{N} \delta_z^n, \tag{6}$$

where δ_z^n is the gradient of the output z, δ_γ denotes the gradient of the scale parameters, and δ_β denotes the gradient of the shift parameters.

2.4. Layer Normalization Layers

- Forward Propagation:

$$\begin{aligned} \mu = \frac{1}{D} \sum_{d=1}^{D} x_d, \quad \sigma^2 = \frac{1}{D} \sum_{d=1}^{D} (x_d - \mu)^2, \\ \hat{x}_d = \frac{x_d - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad z_d = \gamma \hat{x}_d + \beta, \end{aligned} \tag{7}$$

where γ and β are the learnable scale and shift parameters, D is the feature dimension, and ϵ is a small constant for numerical stability.

- Backward Propagation:

$$\delta_\gamma = \sum_{d=1}^{D} \delta_z^d \cdot \hat{x}_d, \quad \delta_\beta = \sum_{d=1}^{D} \delta_z^d \tag{8}$$

where δ_z^d is the gradient of the output z, δ_γ denotes the gradient of the scale parameters, and δ_β denotes the gradient of the shift parameters.

From the above equations, it is evident that pruning the input x of a given layer results in the corresponding weight gradient ($\delta_W^{p,q,c,k}$ of convolutional layers, δ_W^{jk} of fully connected layers, δ_γ of BN and LN layers) becoming more sparse. Therefore, applying different activation sparsity ratios to a layer during adaptation not only *flexibly optimizes its memory cost* but also *modulates the magnitude of weight updates, thereby adjusting the learning ability of that layer*.

3. Why Accuracy Gets Higher via Dynamic Activation Sparsity?

Here, we analyze why layer-specific dynamic activation sparsity leads to improved adaptation accuracy. Table 1 presents the TTA results on WideResNet-28 and CIFAR10-C using various updating strategies.

From the table, we can observe that (i) updating specific layers while freezing others during adaptation yields varying accuracy, and (ii) some subset updating strategies (e.g., updating "Conv1+Block1" or "Block2") even surpass the accuracy achieved by updating all layers ("All"). These results demonstrate that (i) *different layers contribute unequally to accuracy during adaptation*, and (ii) *limiting updates in less critical layers may reduce error accumulation, thereby enhancing TTA accuracy gains*.

Table 1. Mean online error (%) for TTA with different updating strategies. The experiments are implemented using WideResNet-28 and CIFAR10-C, with 1e-5 as the learning rate and 200 as the batch size.

Updating Layers	Conv1+ Block1	Block2	Block3	FC	All	Ours
Mean (%)	18.8	18.7	20.2	20.4	18.9	**18.1**

Motivated by the above, **SURGEON** allocates layer-specific activation sparsity ratios in a data-sensitive manner during adaptation. These sparsity ratios are determined based on layer importance metrics: Gradient Importance (G) and Layer Activation Memory (M), which respectively reflect the accuracy contributions and memory efficiency of different layers. By doing so, **SURGEON** automatically encourages adaptation in layers with higher accuracy contributions while suppressing adaptation in less critical layers. This ultimately mitigates error accumulation, allowing **SURGEON** to achieve higher accuracy.

4. Experimental Details

4.1. Details of Datasets

In the experiments, we employ two tasks: image classification and semantic segmentation. For the image classification task, deep models are trained on the CIFAR [14] and ImageNet [6], and test-time adaptation is implemented on the CIFAR-C and ImageNet-C [8]. For the semantic segmentation task, deep models are trained on the Cityscapes dataset [4] and then adapted at test time on the ACDC dataset [19].

- CIFAR10/100 [14] is an image classification dataset containing 10/100 classification categories and 50,000 training samples.
- ImageNet [6] is an image classification dataset containing 1000 classification categories and approximately 1.2 million training samples.
- CIFAR10/100-C and ImageNet-C [8] are generated by applying 15 types of different corruptions to the CIFAR/ImageNet dataset. It shares the same categories as CIFAR/ImageNet. In TTA, we select corruptions at the highest severity level (level 5), utilizing 10,000 unlabeled images from each corruption. For the test sequence setup, we follow CoTTA [23], where the test sequence from 15 corruptions is repeated once sequentially for CIFAR-C, and ten diverse corruption sequences are used for ImageNet-C. The input size is 32 × 32.
- Cityscapes [4] is a real-world semantic segmentation dataset consisting of images collected from daytime environments. It contains 19 categories and includes 2,975 training samples.
- ACDC [19] is a real-world semantic segmentation dataset comprising images collected from four different environments (i.e., rain, snow, fog, night). It shares the same categories as Cityscapes, and 400 unlabeled images from each environment are used for test-time adaptation. For the test sequence setup, we follow CoTTA, where the sequence is established as Fog $\rightarrow$ Night $\rightarrow$ Rain $\rightarrow$ Snow and repeated 10 times. The input size is 1920 × 1080 for DeeplabV3+, following the setting in RobustNet [3], and scaled to 960 × 540 for Segformer-B5, following the setting in CoTTA [23].

4.2. Details of Methods

For fair comparisons, the pre-trained weights for the network used in the experiments are all downloaded from public resources. WideResNet-28 [27], ResNeXt-29 [26], and ResNet-50 (AugMix) [7] are downloaded from RobustBench [5], DeeplabV3+ [2] is downloaded from RobustNet repository[3] [3], and Segformer-B5 [25] is downloaded from CoTTA repository[4] [23]. For each specific architecture, both our method and the baselines utilized the same pre-trained weights for initialization before TTA.

4.2.1. Details of SURGEON

The details of activation sparsity at the i-th layer can be seen in Figure 3. During test-time adaptation, at layer i: (a) In the **forward pass**, activations are pruned in an unstructured manner after calculating A_{i+1}, with elements set to zero based on the layer-wise sparsity ratio. Elements are selected according to their absolute values, with the smallest ones being set to zero first [11]. The sparse activations are then cached as: ① an index bitmap (1 bit per element via *BitArray package*), and ② a dense vector of non-zero values. At floating-point inference, the sparse activation cache size is $n/32 + n(1 - p_i)$, where n is the original activation size and p_i is the layer's sparsity ratio. (b) In the **backward pass**, the sparse activations are reconstructed initially and thus our method remains hardware-friendly as the sparse activations maintain the original format, size and the standard process for gradient calculation.

In the experiments, our method has two versions: one that updates all layers, **SURGEON**, and another that updates only the BN layers, **SURGEON** (BN).

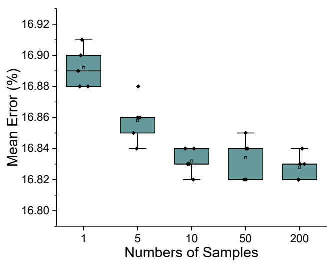
In the image classification experiments, the learning rates used for the reported results can be found in Table 2. For efficient evaluation of layer importance, we randomly select *10 samples* from a test batch during the calculation of importance metrics.

In the semantic segmentation experiments, the learning rates are set to 1e-5 for DeeplabV3+ and 2e-7 for Segformer-B5. For efficient evaluation of layer importance,

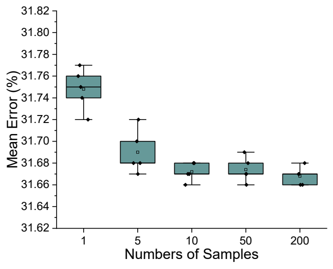
[3]https://github.com/shachoi/RobustNet
[4]https://github.com/qinenergy/cotta/issues/6

Table 2. The learning rates used for the reported results of SURGEON in the image classification task.

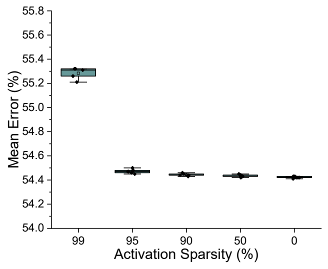
Architectures	Methods	Learning Rate	Architectures	Methods	Learning Rate	Architectures	Methods	Learning Rate
WideResNet-28	SURGEON	1.6e-5	ResNeXt-29	SURGEON	5e-6	ResNet-50	SURGEON	5e-6
	+ CSS	2e-5		+ CSS	1.2e-5		+ CSS	1e-5
	+ CSS & CR	2.2e-5		+ CSS & CR	1e-5		+ CSS & CR	5e-6
	SURGEON (BN)	8e-4		SURGEON (BN)	1e-4		SURGEON (BN)	1.5e-4
	+ CSS	8e-4		+ CSS	3e-4		+ CSS	5e-4
	+ CSS & CR	1e-3		+ CSS & CR	3e-4		+ CSS & CR	2e-4



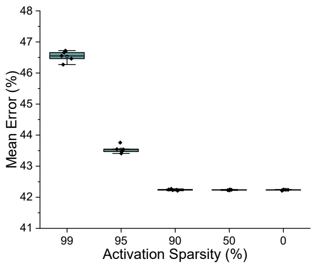
(a) WideResNet-28



(b) ResNeXt-29



(c) DeeplabV3+



(d) Segformer-B5

Figure 2. Different configurations for efficient layer importance evaluation in SURGEON and their impact on accuracy.

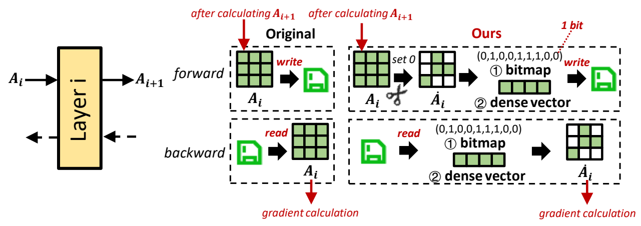


Figure 3. Details of activation sparsity at the i-th layer.

a global static pruning ratio of *90%* is applied to activations during the calculation of importance metrics.

The Impact of Efficient Layer Importance Calculation Configurations on Task Accuracy As illustrated in § 4.2 of the main paper, we employ two strategies for efficient layer importance evaluation in the additional forward-backward process. These strategies—randomly selecting samples and applying a high global static pruning ratio—ensure that the peak memory usage of the additional forward-backward process does not exceed that of the subsequent adaptation process. However, it remains unclear whether these configurations impact accuracy. To investigate this, we test different configurations for the additional forward-backward process and observe their effects on accuracy (i.e., Mean Online Error), as shown in Figure 2. For image classification tasks (Figure 2 (a),(b)), we employ the strategy of randomly selecting samples. The results show that varying the number of selected samples has a minimal impact on accuracy, with our configuration (*10 samples*), as reported in the main paper, demonstrating stable and highly similar accuracy compared to full evaluation (*200 samples*). For semantic segmentation tasks (Figure 2 (c),(d)), which have a small test batch size, we apply a high global static pruning ratio. The results indicate that our configuration (*90 %*) also yields stable and comparable accuracy, in contrast to full evaluation (*0 %*). Additionally, we can find that when the activation sparsity exceeds a certain threshold (e.g., *95 %* and *99 %*) in the forward-backward process, it significantly impacts model accuracy due to information loss in the gradients used for calculating layer importance. This finding aligns with the results reported in [11].

4.2.2. Details of Baselines

As outlined in the main paper, we compare our method, **SURGEON** , against seven baselines: **Source**, **BN-stat** [20], **Full Tuning**, **TENT** [22], **CoTTA** [23], **EcoTTA** [21], and **MECTA** [9]. In the following, we provide additional details on the TTA experiments for these baselines.

Source and BN-stat Both of these two baselines do not require backward propagation to update the model. **Source** refers to the original network without any adaptation. **BN-stat** updates the running statistics (i.e., mean μ and standard deviation σ) of BN layers during adaptation. We can set *model.train()*[5] instead of *model.eval()* to activate the update of BN statistics.

Full Tuning In the CIFAR-to-CIFAR-C experiments, we configure the learning rate to 1e-5 with the Adam optimizer for both WideResNet-28 and ResNeXt-29 during TTA. In the Cityscapes-to-ACDC experiments, we set the learning rate to 1e-5 with the SGD optimizer for DeeplabV3+.

[5] https://pytorch.org/docs/stable/generated/torch.nn.Module.html#torch.nn.Module.train

TENT In the CIFAR-to-CIFAR-C experiments, we set the TTA learning rates to 5e-4, 1e-4, and 2.5e-4 with the Adam optimizer for WideResNet-28, ResNeXt-29, and ResNet-50, respectively. In the Cityscapes-to-ACDC experiments, we set the learning rate to 1e-5 using the SGD optimizer for DeeplabV3+.

CoTTA We reproduce the results using the code from the official repository[6]. Following its settings, we use a restoration probability of $p = 0.01$ for all CoTTA experiments. In the CIFAR-to-CIFAR-C experiments, we set the learning rate to 1e-3 with the Adam optimizer for both WideResNet-28 and ResNeXt-29, and 1e-2 for ResNet-50, as specified in the official code. In the Cityscapes-to-ACDC experiments, we set the learning rate to 1e-4 with the Adam optimizer for DeeplabV3+. For Segformer-B5, the learning rate is set to 7.5e-6.

EcoTTA To the best of our knowledge, the authors have not yet released an official repository. We reproduce EcoTTA thanks to a community implementation[7].

In the CIFAR-to-CIFAR-C experiments, we set $K = 5$ in the plug-in meta networks for WideResNet-28 and ResNet-50, following the original settings. For ResNeXt-29, EcoTTA does not provide the configuration of meta networks. We employ partition strategies of $[3, 3, 3]$, $[2, 2, 2, 3]$, and $[1, 1, 2, 2, 3]$ to divide the encoder of ResNeXt-29 into 3, 4, and 5 partitions, respectively. In the experiments of the main paper, we report the results with the partition strategy $[3, 3, 3]$ ($K = 3$) as it shows the best performance among the three strategies.

In the experiments, we set the number of epochs to 10 and the learning rate to 5e-2 with the SGD optimizer for warming up the meta networks. During TTA, learning rates of 6e-2, 1e-2, 1e-2 with the SGD optimizer are used for WideResNet-28, ResNeXt-29, and ResNet-50, respectively. Note that the TTA learning rate for WideResNet-28 and ResNet-50 differs slightly from the original paper (5e-3), due to our use of a larger batch size during TTA.

Table 3 shows the EcoTTA experimental results for ResNeXt-29 with three different meta network architectures. Results for $K = 3$ are reported in the main paper.

Table 3. Mean online error (%) and cache size (MB) for EcoTTA implementation with different partition strategies in ResNeXt-29.

Partition Strategy	[3,3,3] (K=3)	[2,2,2,3] (K=4)	[1,1,2,2,3] (K=5)
Mean (%)	**33.3**	35.3	35.1
Cache (MB)	**1050**	1350	1950

[6]https://github.com/qinenergy/cotta

[7]https://github.com/Lily-Le/EcoTTA

Table 4. Mean online error (%), cache size (MB) and GFLOPs (per sample) for TTA using ResNet-50 (AugMix) on ImageNet-to-ImageNet-C. † refers to **SURGEON** (BN) + CSS, and ‡ refers to **SURGEON** (BN) + CSS & CR.

Methods	Backward-free			Backward-based		
	Source	LAME	FOA	EcoTTA	**SURGEON**†	**SURGEON**‡
Mean (%) ↓	74.4	72.3	62.8	<u>54.2</u>	54.6	**53.9**
Cache (MB) ↓	196	**196**	<u>199</u>	1503	907	1834
GFLOPs ↓	4.1	<u>4.2</u>	**4.1**	10.1	9.8	14.2

MECTA We reproduce the results of MECTA using the official code[8]. In the CIFAR-to-CIFAR-C experiments, MECTA is reported in two versions: one is the standalone implementation (MECTA), and the other is a combined implementation of MECTA and EATA (MECTA+EATA). The hyperparameters for EATA [16] remain consistent, which uses 2,000 training samples to estimate a Fisher matrix in the original training process.

For WideResNet-28, we set the TTA learning rate to 1e-2 with the SGD optimizer, a layer freezing threshold β_{th} of 0.0075, and a channel pruning ratio of 0.7. For ResNeXt-29, the TTA learning rates are set to 5e-4 and 1e-3 with the SGD optimizer for the MECTA version and the MECTA+EATA version, respectively. β_{th} is set to 0.0025, and the channel pruning ratio is 0.7, following the original settings. For ResNet-50, the TTA learning rate is set to 1e-3 with the SGD optimizer for both the MECTA version and the MECTA+EATA version. β_{th} is set to 0.00125, and the channel pruning ratio is 0.7, following the original settings. Note that the TTA learning rate for ResNeXt-29 (1e-4) and ResNet-50 (2.5e-4) differs slightly from the original paper due to our use of a larger batch size during TTA.

In the Cityscapes-to-ACDC experiments, we utilize only the standalone implementation of MECTA. For DeeplabV3+, we set the TTA learning rate to 5e-4 with the SGD optimizer, β_{th} to 0.0025, and the pruning ratio to 0.7. For Segformer-B5, the learning rate is set to 7.5e-6.

5. Further Comparisons with Backward-free Methods

Recently, test-time adaptation methods without backward propagation have garnered increasing attention [1, 17]. These methods introduce minimal memory and computational overhead, providing a substantial efficiency advantage over backward-based approaches [18, 22, 23], which typically incur significant costs. Therefore, we compare the performance of **SURGEON** with two other backward-free methods, LAME [1] and FOA [17]. The experiments are conducted using ResNet-50 (AugMix) on ImageNet-to-ImageNet-C. Results in Table 4 indicate that while backward-free baselines exhibit minimal additional over-

[8]https://github.com/SonyResearch/MECTA

head, they still cannot fully replace backward-based methods across various TTA settings due to the potential accuracy limitation.

6. Limitations

Surgeon is built upon sparse activation techniques, and its effectiveness on network architectures beyond those examined in this paper is yet to be explored (e.g., ReLU, sigmoid, and h-swish [10]). Moreover, sparse activations do not affect the calculation of bias gradients. Although our experiments demonstrate minimal bias impact, the evaluation of bias based on layer importance for its update process, particularly in other distribution shift scenarios, remains unexplored in this paper.

Furthermore, in **Surgeon** the calculation of pruning ratios necessitates an additional forward-backward process. Despite our efforts to mitigate the computation and memory cost of this process via designs like random sampling, it still introduces additional cost and latency.

7. Future Work

In our future work, we will explore the potential of **Surgeon** across a wider range of network architectures, such as activation function layers (e.g., ReLU, sigmoid), and its applicability to addressing diverse and complex tasks [13, 24]. Moreover, we will delve into combining **Surgeon** with other memory-efficient strategies, particularly system-level techniques [12, 15].

Additionally, we aim to investigate whether layer importance in the context of TTA is more directly correlated with issues related to error accumulation, such as the propensity for catastrophic forgetting with increasing update frequencies. Lastly, our future directions will also be assessing the importance of layers beyond gradients and establishing a better mapping from layer importance to layer-wise activation pruning ratios.

References

[1] Malik Boudiaf, Romain Mueller, Ismail Ben Ayed, and Luca Bertinetto. Parameter-free online test-time adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8344–8353, 2022. 5

[2] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018. 1, 3

[3] Sungha Choi, Sanghun Jung, Huiwon Yun, Joanne T Kim, Seungryong Kim, and Jaegul Choo. Robustnet: Improving domain generalization in urban-scene segmentation via instance selective whitening. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11580–11590, 2021. 3

[4] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016. 1, 3

[5] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. *NeurIPS 2021 Datasets and Benchmarks Track*, 2021. 3

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 3

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 3

[8] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *2019 International Conference on Learning Representations*, 2019. 3

[9] Junyuan Hong, Lingjuan Lyu, Jiayu Zhou, and Michael Spranger. Mecta: Memory-economic continual test-time model adaptation. In *2023 International Conference on Learning Representations*, 2023. 4

[10] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019. 6

[11] Ziyu Jiang, Xuxi Chen, Xueqin Huang, Xianzhi Du, Denny Zhou, and Zhangyang Wang. Back razor: Memory-efficient transfer learning by self-sparsified backpropagation. *Advances in Neural Information Processing Systems*, 35: 29248–29261, 2022. 3, 4

[12] Jaehoon Jung, Jinpyo Kim, and Jaejin Lee. Deepum: Tensor migration and prefetching in unified memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 207–221, 2023. 6

[13] Adilbek Karmanov, Dayan Guan, Shijian Lu, Abdulmotaleb El Saddik, and Eric Xing. Efficient test-time adaptation of vision-language models. *arXiv preprint arXiv:2403.18293*, 2024. 6

[14] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 1, 3

[15] Sicong Liu, Bin Guo, Cheng Fang, Ziqi Wang, Shiyan Luo, Zimu Zhou, and Zhiwen Yu. Enabling resource-efficient aiot system with cross-level optimization: A survey. *IEEE Communications Surveys & Tutorials*, 2023. 6

[16] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In *2022 International conference on machine learning*, pages 16888–16905. PMLR, 2022. 5

[17] Shuaicheng Niu, Chunyan Miao, Guohao Chen, Pengcheng Wu, and Peilin Zhao. Test-time model adaptation with only forward passes. In *Proceedings of the 41st International Conference on Machine Learning*, pages 38298–38315, 2024. 5

[18] Ori Press, Steffen Schneider, Matthias Kümmerer, and Matthias Bethge. Rdumb: A simple approach that questions our progress in continual test-time adaptation. *Advances in Neural Information Processing Systems*, 36:39915–39935, 2023. 5

[19] Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Acdc: The adverse conditions dataset with correspondences for semantic driving scene understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10765–10775, 2021. 3

[20] Steffen Schneider, Evgenia Rusak, Luisa Eck, Oliver Bringmann, Wieland Brendel, and Matthias Bethge. Improving robustness against common corruptions by covariate shift adaptation. *Advances in neural information processing systems*, 33:11539–11551, 2020. 4

[21] Junha Song, Jungsoo Lee, In So Kweon, and Sungha Choi. Ecotta: Memory-efficient continual test-time adaptation via self-distilled regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11920–11929, 2023. 4

[22] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *2021 International Conference on Learning Representations*, 2021. 4, 5

[23] Qin Wang, Olga Fink, Luc Van Gool, and Dengxin Dai. Continual test-time domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7201–7211, 2022. 3, 4, 5

[24] Zhiquan Wen, Shuaicheng Niu, Ge Li, Qingyao Wu, Mingkui Tan, and Qi Wu. Test-time model adaptation for visual question answering with debiased self-supervisions. *IEEE Transactions on Multimedia*, 2023. 6

[25] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in neural information processing systems*, 34: 12077–12090, 2021. 3

[26] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 1, 3

[27] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference 2016*, 2016. 1, 3