# Supplementary Material on EcoTTA: Memory-Efficient Continual Test-time Adaptation via Self-distilled Regularization

Junha Song$^{1,2*}$, Jungsoo Lee$^{1}$, In So Kweon$^{2}$, Sungha Choi$^{1\dagger}$
$^{1}$Qualcomm AI Research$^{\ddagger}$, $^{2}$KAIST

## Appendix

In this supplementary material, we provide,

## A. Efficiency for TTA methods

**Memory efficiency.** Existing TTA works [23, 28, 29] update model parameters to adapt to the target domain. This process inevitably requires additional memory to store the activations. Fig. 6 describes Eq. (1) of the main paper in more detail. For instance, 1) the backward propagation from the layer $(c)$ to the layer $(b)$ can be accomplished without saving intermediate activations $f_i$ and $f_{i+1}$, since it only requires $\frac{\partial \mathcal{L}}{\partial f_{i+1}} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \mathcal{W}_{i+1}^T$ and $\frac{\partial \mathcal{L}}{\partial f_i} = \frac{\partial \mathcal{L}}{\partial f_{i+1}} \mathcal{W}_i^T = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \mathcal{W}_{i+1}^T \mathcal{W}_i^T$ operations. 2) During the forward propagation, the learnable layer $(a)$ has to store the intermediate activation $f_{i-1}$ to calculate the weight gradient $\frac{\partial \mathcal{L}}{\partial \mathcal{W}_{i-1}} = f_{i-1}^T \frac{\partial \mathcal{L}}{\partial f_i}$.

**Computational efficiency.** Wall-clock time and floating point operations (FLOPs) are standard measures of computational cost. We utilize wall-clock time to compare the computational cost of TTA methods since most libraries computing FLOPs only support inference, not training.

Unfortunately, wall-clock time of EATA [23] and our approach can not truly represent its computational efficiency since the current Pytorch version [24] does not support fine-grained implementation [1]. For example, EATA filters samples to improve its computational efficiency. However, its gradient computation is performed on the full mini-batch, so the wall-clock time for backpropagation in EATA is almost the same as that of TENT [28]. In our approach, our implementation follows Algorithm 1 to make each regularization loss $\mathcal{R}_{\theta_k}^k$ applied to parameters of k-th group of meta networks $\theta_k$ in Eq. (3). In order to circumvent such an issue, the authors of EATA report the theoretical

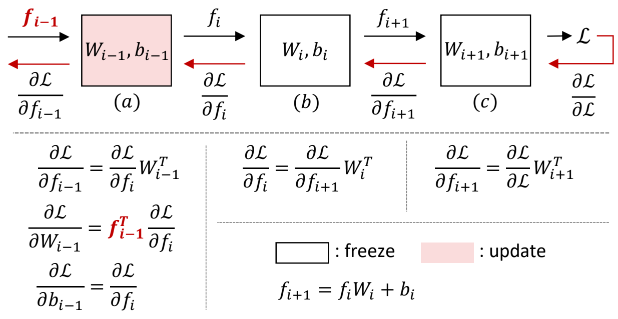

$f_{i-1}$
$W_{i-1}, b_{i-1}$
$(a)$
$f_i$
$W_i, b_i$
$(b)$
$f_{i+1}$
$W_{i+1}, b_{i+1}$
$(c)$
$\mathcal{L}$
$\frac{\partial \mathcal{L}}{\partial f_{i-1}}$
$\frac{\partial \mathcal{L}}{\partial f_i}$
$\frac{\partial \mathcal{L}}{\partial f_{i+1}}$
$\frac{\partial \mathcal{L}}{\partial \mathcal{L}}$
$\frac{\partial \mathcal{L}}{\partial f_{i-1}} = \frac{\partial \mathcal{L}}{\partial f_i} W_{i-1}^T$
$\frac{\partial \mathcal{L}}{\partial W_{i-1}} = f_{i-1}^T \frac{\partial \mathcal{L}}{\partial f_i}$
$\frac{\partial \mathcal{L}}{\partial b_{i-1}} = \frac{\partial \mathcal{L}}{\partial f_i}$
$\frac{\partial \mathcal{L}}{\partial f_i} = \frac{\partial \mathcal{L}}{\partial f_{i+1}} W_i^T$
$\frac{\partial \mathcal{L}}{\partial f_{i+1}} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} W_{i+1}^T$
: freeze
: update
$f_{i+1} = f_i W_i + b_i$

Figure 6. **Forward and backward propagation.** The black and red lines refer to forward and backward propagation, respectively. $f$ and $(a, b, c)$ are the activations and the linear layers, respectively.

| | Avg. err | Mem. (MB) | Theo. time | Wall time |
|---|---|---|---|---|
| WideResNet-40 [31] | | | | |
| Source | 69.7 | 11 | - | 40s |
| Con. TENT [28] | 38.3 | 188 | - | 2m 18s |
| CoTTA [29] | 38.1 | 409 | - | 22m 52s |
| EATA [23] | 37.1 | 188 | 2m 8s | 2m 22s |
| Ours (K=4) | 36.4 | **80** (80, 58%↓) | 2m 27s | 2m 49s |
| Ours (K=5) | **36.3** | 92 (77, 51%↓) | 2m 31s | 2m 52s |
| ResNet-50 [12] | | | | |
| Source | 73.8 | 91 | - | 1m 8s |
| Con. TENT [28] | 45.9 | 926 | - | 4m 2s |
| CoTTA [29] | 40.2 | 2064 | - | 38m 24s |
| EATA [23] | 39.9 | 926 | 3m 45s | 4m 15s |
| Ours (K=4) | 39.5 | **296** (86, 68%↓) | 4m 16s | 4m 41s |
| Ours (K=5) | **39.3** | 498 (76, 46%↓) | 4m 26s | 5m 14s |

Table 8. **Comparison of training time on CIFAR100-C.** We report both theoretical time (in short, theo. time) and wall-clock time, taking to adapt to all 15 corruption types. Theoretical time is calculated by assuming that the ML frameworks (*e.g.*, Pytorch [24]) provide fine-grained implementations [1]. Con. TENT refers to continual TENT.

*Work done during an internship at Qualcomm AI Research.
†Corresponding author. ‡Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

**Algorithm 1:** PyTorch-style pseudocode for EcoTTA.

```
# img_t: test image
# model: original and meta networks
#
# ent_min(): Entropy minimization loss
# Detach_parts(): Detach the graph connection
#                 between each partition of networks
# Attach_parts(): Attach the graph connection
#                 between each partition of networks

for img_t in test_loader:
    # 1. Forward
    output = model(img_t)
    # 2. Compute entropy loss
    loss_ent = ent_min(output)
    loss_ent.backward()

    # 3. Re-forward
    # (This process is not required
    # in fine-grained ML frameworks.)
    Detach_parts(model)
    _ = model(img_t)

    # 4. Compute regularization loss
    reg_loss = 0
    for k_th_meta in meta_networks:
        reg_loss += k_th_meta.get_l1_loss()
    reg_loss.backward()

    # 5. Update params of meta networks
    optim.step()
    optim.zero_grad()

    Attach_parts(model)
```

time, which assumes that PyTorch handles gradient backpropagation at an instance level. Similar to EATA, we also report both theoretical time and wall-clock time in Table 8. To compute the theoretical time of our approach, we simply subtract the time for re-forward (in Algorithm 1) from wall-clock time. We emphasize that this is mainly an engineering-based issue, and the optimized implementation can further improve computational efficiency. [23].

Using a single NVIDIA 2080Ti GPU, we measure the total time required to adapt to all 15 corruptions, including the time to load test data and perform TTA. The results in Table 8 show that our proposed method requires negligible overhead compared to CoTTA [29]. For example, CoTTA needs approximately 10 times more training time than Continual TENT [28] with WideResNet-40. Note that meta networks enable our approach to use 80% and 58% less memory than CoTTA and EATA, even with such minor extra operations.

## B. Discussion and further experiments

**Comparison on gradually changing setup.** In Table 1 and 2, we evaluate all methods on the continual TTA task, proposed in CoTTA [29] and EATA [23], where we continually adapt the deployed model to each corruption type sequentially. Additionally, we conduct experiments on the gradually changing setup. This gradual setup, proposed in CoTTA, represents the sequence by gradually changing severity for the 15 corruption types:

| Method | Con. TENT [28] | EATA [23] | CoTTA [29] | Ours (K=4) |
|---|---|---|---|---|
| Avg. err (%) | 38.5 | 31.8 | 32.5 | **31.4** |
| Mem. (MB) | 188 | 188 | 409 | **80** (58, 80%↓) |

Table 9. **Comparision on gradually changing setup.** To conduct experiments, we use WRN-40 backbone on CIFAR100-C. The values in parentheses refer to memory reduction rates compared to TENT/EATA and CoTTA, sequentially.

| Method | Con. TENT [28] | PETL [15] | PETL+SDR | Ours (K=4) |
|---|---|---|---|---|
| Avg. err (%) | 38.3 | 73.3 | 42.5 | **36.4** |
| Mem. (MB) | 188 | 141 | 141 | **80** |

Table 10. **Comparisons with methods for PETL.** We compare our method with methods [15] for parameter-efficient transfer learning (PETL) with WRN-40 on CIFAR100-C. PETL+SDR refers to PETL with our proposed self-distilled regularization.

| Round | Con. TENT | TS | DO | LS | KD | Ours (K=4) |
|---|---|---|---|---|---|---|
| 1 | 38.3 | 37.4 | 41.0 | 38.4 | 39.8 | **36.4** |
| 10 | 99.0 | 96.1 | 96.3 | 41.1 | 40.4 | **36.3** |

Table 11. **Comparisons with methods for continual learning.** We report an average error rate (%) of 15 corruptions using WRN-40 on CIFAR100-C. In the table, TS: Entropy minimization with temperature scaling [11], DO: Dropout [26], LS: Label smoothing with the pseudo label [21], and KD: Knowledge distillation [14].

$$\underbrace{\ldots 2\rightarrow 1}_{\text{t-1 and before}} \xrightarrow[type]{change} \underbrace{1\rightarrow 2\rightarrow 3\rightarrow 4\rightarrow 5\rightarrow 4\rightarrow 3\rightarrow 2\rightarrow 1}_{\text{corruption type t, gradually changing severity}} \xrightarrow[type]{change} \underbrace{1\rightarrow 2\ldots}_{\text{t+1 and on}},$$

The results in Table 9 indicate that our approach outperforms previous TTA methods [23,28,29] even with the gradually changing setup.

**Comparisons with methods for parameter efficient transfer learning.** While our framework may be similar to parameter-efficient transfer learning (PETL) [15, 16, 27] in that only partial parameters are updated during training time for PETL or test time for TTA, we utilized meta networks to minimize intermediate activations, which is crucial for memory-constrained edge devices. We conduct experiments by applying a PETL method [15] to the TTA setup. The adapter module is constructed by using 3x3 Conv and ReLU layers as the projection layer and the nonlinearity, respectively, and these modules are attached after each residual block of the backbone networks. The Table 10 shows that PETA+SDR needs a 177% increase in memory usage with a 6.1% drop in performance, compared to our method.

**Comparisons with methods for continual learning.** Typical continual learning (CL) and continual TTA assume supervised and unsupervised learning, respectively. However, since both are focused on alleviating catastrophic forgetting, we believe that CL methods can also be applied in continual TTA settings. The methods for addressing catastrophic forgetting can be divided into regularization- and replay-based methods. The former can be subdivided into weight regularization (*e.g.*, CoTTA [29] and EATA [23])

| Method | Mem. (MB) | Round 1 | Round 4 | Round 7 | Round 10 |
|---|---|---|---|---|---|
| Source | 280 | 37.2 | 37.2 | 37.2 | 37.2 |
| Con. TENT | 2721 | 54.6 | 49.6 | 37.4 | 29.9 |
| Con. TENT * | 2721 | 56.5 | 52.7 | 42.7 | 36.5 |
| CoTTA * | 6418 | **56.7** | 56.7 | 56.7 | 56.7 |
| Ours | **918** (66, 85%↓) | 55.2 | 55.4 | 55.4 | 55.4 |
| Ours * | **918** (66, 85%↓) | **56.7** | **56.8** | **56.9** | **56.9** |

Table 12. **Further experiments in semantic segmentation.** We represent the results based on mean intersection over union (mIoU). * means that the method utilizes the same cross-entropy consistency loss. The values in parentheses refer to memory reduction rates compared to TENT/EATA and CoTTA, sequentially.

| #Partitions | WRN-28 (12) | WRN-40 (18) | ResNet-50 (16) |
|---|---|---|---|
| **K=4** | 2,2,4,4 | 3,3,6,6 | 3,3,5,5 |
| **K=5** | 2,2,2,2,4 | 3,3,3,3,6 | 2,2,4,4,4 |

Table 13. **Details of # of blocks of each partition.** The list of numbers denotes the number of residual blocks for each part of the original networks, from the shallow to the deep parts sequentially. The values in parentheses are the total number of residual blocks.

and knowledge distillation [14], while the latter includes GEM [20] and dataset distillation [7]. Suppose dataset distillation is applied to the continual TTA setup; for example, we can periodically replay synthetic samples distilled from the source dataset to prevent the model from forgetting the source knowledge during TTA. Notably, our self-distilled regularization (SDR) is superior to conventional CL methods in terms of the efficiency of TTA in on-device settings. Specifically, unlike previous regularization- or replay-based methods, we do not require storing a copy of the original model or a replay-and-train process.

To further compare our SDR with existing regularization methods, we conduct experiments while keeping our architecture and adaptation loss but replacing SDR with other regularizations, as shown in Table 11. The results demonstrate that our SDR achieves superior performance compared to other regularizations. In addition, Knowledge distillation [14] alleviates the error accumulation effect in long-term adaptation (*e.g.*, round 10), while showing limited performance for adapting to the target domain.

**Superiority of our approach compared to existing TTA methods.** Our work focuses on proposing an efficient architecture for continual TTA, which has been overlooked in previous TTA studies [2, 5, 18, 19, 28, 29] by introducing meta networks and self-distilled regularization, rather than adaptation loss such as entropy minimization proposed in TENT [28] and EATA [23]. Thus, our method can be used with various adaptation losses. Moreover, even though our self-distilled regularization can be regarded as a teacher-student distillation from original networks to meta networks, it does not require a large activation size or the storage of an extra source model, unlike CoTTA [29].

In addition to the results in Table 6, we improve the segmentation experiments by comparing our approach with CoTTA [29]. As we aforementioned, our approach has scalability with diverse adaptation loss. Thus, as shown in Table 12, we additionally apply cross-entropy consistency loss* with multi-scaling input as proposed in CoTTA, where we use the multi-scale factors of [0.5, 1.0, 1.5, 2.0] and flip. Our method not only achieves comparable performance with 85% less memory than CoTTA, but shows consistent performance even for multiple rounds while continual TENT [28] suffers from the error accumulation effect.

| (K=4) | Kernel size= 1, padding=0 | | | Kernel size=3, padding=1 | | |
|---|---|---|---|---|---|---|
| Arch | Avg. err | params ↑ | Mem. | Avg. err | params ↑ | Mem. |
| WRN-28 | 17.2 | 0.8% | 396 | **16.9** | 9.5% | 404 |
| WRN-40 | 12.4 | 0.6% | 80 | **12.2** | 6.4% | 80 |
| ResNet-50 | 14.4 | 11.8% | 296 | **14.2** | 142.2% | 394 |

Table 14. **Kernel size in the conv layer.** We report the average error rate (%), the increase rate of the model parameters compared to the original model (%), and the total memory consumption (MB) including the model and activation sizes, based on the kernel size of the conv layer in meta networks.

| (K=4) | | | Transformations | | | |
|---|---|---|---|---|---|---|
| Dataset | Arch | EATA [23] | None | +Color | +Blur | +Gray |
| CIFAR10-C | WRN-40 | 13.0 | 12.5 | 12.3 | 12.3 | **12.2** |
| CIFAR10-C | WRN-28 | 18.6 | 17.8 | 17.4 | 17.2 | **16.9** |
| CIFAR100-C | WRN-40 | 37.1 | 36.9 | 36.7 | 36.6 | **36.4** |

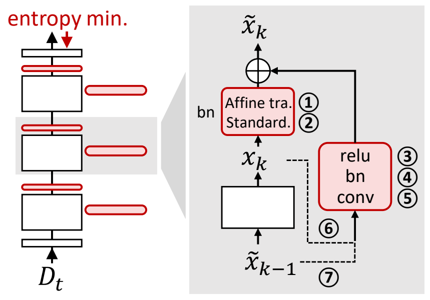
Table 15. **Ablation of the combination of transformations.** To warm up the meta networks, we use the following transformations in Pytorch: ColorJitter (Color), GaussianBlur (Blur), and RandomGrayscale (Gray). We report the average error rate (%).

## C. Further implementation details

**Partition of a pre-trained model.** As illustrated in Fig. 3, the given pre-trained model consists of three parts: classifier, encoder, and input conv, where the encoder denotes layer1 to 4 in the case of ResNet. Our method is applied to the encoder and we divide it into K parts. Table 13 describes the details of the number of residual blocks for each part of the encoder. Our method is designed to divide the shallow layers more (*i.e.*, densely) than the deep layers, improving the TTA performance as shown in Table 4c.

**Convolution layer in meta networks.** As the hyperparameters of the convolution layer[1], we set the bias to false and the stride to two if the corresponding part of the encoder includes the stride of two; otherwise, one. As shown in the gray area in Table 14, we conduct experiments by modifying the kernel size and padding for each architecture. To be more specific, we obtain better performances by setting the kernel size to three with WideResNet (with 10% additional number of model parameters). On the other hand, utilizing the kernel size of three with ResNet leads to significant in-

[1] https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html



(a) **Visualization of meta networks**

| (K=5) Variants | ① | ② | ③ | ④ | ⑤ | ⑥ | ⑦ | CIFAR10-C **WRN-28** | CIFAR10-C **WRN-40** | CIFAR100-C **WRN-40** |
|---|---|---|---|---|---|---|---|---|---|---|
| I |  | ✓ | ✓ | ✓ | ✓ |  | ✓ | 19.9 | 15.4 | 39.2 |
| II | ✓ |  | ✓ | ✓ | ✓ |  | ✓ | 18.6 | 13.4 | 38.0 |
| III |  |  | ✓ | ✓ | ✓ |  | ✓ | 18.7 | 13.7 | 38.2 |
| IV | ✓ | ✓ |  | ✓ | ✓ |  | ✓ | 18.6 | 12.4 | 36.7 |
| V | ✓ | ✓ | ✓ |  | ✓ |  | ✓ | 19.8 | 12.9 | 37.2 |
| VI | ✓ | ✓ |  |  | ✓ |  | ✓ | 32.3 | 14.5 | 51.8 |
| VII | ✓ | ✓ |  |  |  |  | ✓ | 20.7 | 14.9 | 40.1 |
| XIII | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |  | 18.1 | 12.6 | 37.2 |
| IX |  |  | ✓ | ✓ | ✓ | ✓ |  | 60.6 | 73.3 | 77.2 |
| **Ours** | ✓ | ✓ | ✓ | ✓ | ✓ |  | ✓ | **16.8** | **12.1** | **36.3** |

(b) **Comparison of average error rate (%) on continual TTA setup (K=5)**

Table 16. **Components of meta networks.** We conduct an ablation study on components of meta networks (*i.e.*, ① ∼ ⑦). Here, ① and ② refer to affine transformation and standardization in a BN layer after the original networks. ③∼⑤ and ⑥∼⑦, respectively, indicate modules in a convolution block and two kinds of inputs of it. In table (b), ✓ means applying the component to meta networks.

| (K=5) Dataset | Arch | EATA [23] | Ours $\mathcal{L}^1$ | $\mathcal{L}^2$ | $\mathcal{L}^3$ |
|---|---|---|---|---|---|
| CIFAR10-C | WRN-28 | 18.6 | 17.3 | <u>16.9</u> | **16.9** |
|  | WRN-40 | 13.0 | <u>12.2</u> | 12.3 | **12.1** |
|  | Resnet-50 | 14.2 | 15.0 | <u>14.3</u> | **14.1** |
| CIFAR100-C | WRN-40 | 37.1 | 36.5 | <u>36.4</u> | **36.3** |
|  | Resnet-50 | 39.9 | 40.7 | **38.8** | <u>39.4</u> |

Table 17. **Ablation study of main task loss.** We compare the average error rate (%) of three types of adaptation losses.

creases in parameters and memory sizes. Thus, we use one and three as the kernel size with ResNet and WideResNet, respectively.

**Warming up meta networks.** Before the model deployment, we warm up meta networks with the source data by applying the following transformations, which prevent the meta networks from being overfitted to the source domain.

Regardless of the pre-trained model's architecture and pre-training method, we use the same transformations to warm up meta networks. Even for WideResNet-40 pre-trained with AugMix [13], a strong data augmentation technique, the following simple transformations are enough to warm up the meta networks. In addition, we provide the ablation of the combination of transformations in Table 15.

```
from torchvision import transforms as T

TRANSFORMS = torch.nn.Sequential(
    RandomApply(T.ColorJitter(0.4,0.4,0.4,0.1), p=0.4)
    RandomApply(T.GaussianBlur((3,3), p=0.2)
    T.RandomGrayscale(P=0.1))
```

**Semantic segmentation.** For semantic segmentation experiments, we utilize ResNet-50-based DeepLabV3+ [3] from RobustNet repository[2] [4]. We warm up the meta networks on the train set of Cityscapes [6] with SGD optimizer with the learning rate of 5e-2 and the epoch of 5. Image transformations follow the implementation details of [4]. After model deployment, we perform TTA using SGD optimizer with the learning rate of 1e-5, the image size of 1600×800, the batch size of 2, and the importance of regularization $\lambda$ of 2. The main loss for adaptation is same as $\mathcal{L}^{ent}$ in Equ. (2).

[2]https://github.com/shachoi/RobustNet

## D. Additional ablations

**Main task loss for adaptation.** To adapt to the target domain effectively, selecting the main task loss for adaptation is a non-trivial problem. So, we conduct a comparative experiment on three types of adaptation loss: $\mathcal{L}^1$) entropy minimization [10], $\mathcal{L}^2$) entropy minimization with mean entropy maximization [17], and $\mathcal{L}^3$) filtering samples using entropy minimization [23]. With a mini-batch of $N$ test images, the three adaptation losses are formulated as follows:

$$\mathcal{L}^1 = \frac{1}{N}\sum_{i=1}^{N} H(\hat{y}_i), \tag{5}$$

$$\mathcal{L}^2 = \lambda_{m_1}\frac{1}{N}\sum_{i=1}^{N} H(\hat{y}_i) - \lambda_{m_2} H(\overline{y}), \tag{6}$$

$$\mathcal{L}^3 = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}_{\{H(\hat{y}_i)<H_0\}} \cdot H(\hat{y}_i), \tag{7}$$

where $\hat{y}_i$ is the logits output of $i$-th test data, $\overline{y} = \frac{1}{N}\sum_{i=1}^{N} p(\hat{y}_i)$, $H(y) = -\sum_C p(y)\log p(y)$, p$(\cdot)$ is the softmax function, C is the number of classes, and $\mathbb{I}_{\{\cdot\}}$ is an indicator function. $\lambda_{m_1}$ and $\lambda_{m_2}$ indicate the importance of each term in Eq. (6) which are set to 0.2 and 0.25, respectively, following SWR&NSP [5]. The entropy threshold $H_0$ is set to $0.4 \times \ln C$ following EATA [23].

The results are described in Table 17. Particularly, applying any of the three losses, our method achieves comparable performance to EATA. Among them, using $\mathcal{L}^3$ of Eq. (7) achieves the lowest error rate in most cases. Therefore, we apply $\mathcal{L}^3$ to our approach as mentioned in Section 3.1.

**Components of meta networks.** As shown in Table 16, we conduct an ablation study on each element of our proposed meta networks. We observe that the affine transformation is more critical than standardization in a BN layer after the original networks. Specifically, removing the standardization (variant II) causes less performance drop than removing the affine transformation (variant I). In addition, using

| (K=5) Dataset | Arch | Ours MSE loss (Eq. (8)) | Ours L1 loss (Eq. (4)) |
|---|---|---|---|
| CIFAR10-C | WRN-28 | 16.9 | 16.9 |
| | WRN-40 | 12.3 | **12.1** |
| | Resnet-50 | 14.1 | 14.1 |
| CIFAR100-C | WRN-40 | 36.6 | **36.3** |
| | Resnet-50 | 39.5 | **39.4** |

Table 18. **Ablation study of loss function of our regularization.** We present the average error (%) according to two types of loss functions for self-distilled regularization.

only a conv layer in conv block (variant VI) also cause performance degradation, so it is crucial to use the ReLU and BN layers together in the conv block.

**Loss function choice of our regularization.** As mentioned in Section 3.2, self-distilled regularization loss computes the mean absolute error (*i.e.*, L1 loss) of Eq. (4). This loss regularizes the output $\tilde{x}_k$ of each $k$-th group of the meta networks not to deviate from the output $x_k$ of each $k$-th part of frozen original networks. The mean squared error (*i.e.*, MSE loss) also can be used to get a similar effect which is defined as:

$$MSE = (\tilde{x}_k - x_k)^2. \tag{8}$$

We compare two kinds of loss functions for our regularization in Table 18. By observing a marginal performance difference, our method is robust to the loss function choice.

**Robustness to the importance of regularization $\lambda$.** We show that our method is robust to the regularization term $\lambda$. We conduct experiments using a wide range of $\lambda$ as shown in Figure 5b and the following table.

| **Round** $\backslash\lambda$ | 0 | 0.1 | 0.5 | 1 | 2 | 5 | 10 |
|---|---|---|---|---|---|---|---|
| 1 | 36.31 | 36.30 | **36.29** | 36.56 | 37.20 | 38.41 | 39.58 |
| 10 | 55.47 | 43.83 | 36.42 | 36.14 | 36.48 | 37.47 | 38.95 |

The experiments are performed with WideResNet-40 on CIFAR100-C. When $\lambda$ is changed from 0.5 to 1, the performance difference was only 0.27% in the first round. We also test $\lambda$ to be extremely large (*e.g.*, 5, 8, and 10). Since setting $\lambda$ to 10 may mean that we hardly adapt the meta networks to the target domain, the error rate (39.58%) with $\lambda$ of 10 was close to the one (41.1%) of BN Stats Adapt [22].

## E. Baseline details

### E.1. TTA works

We refer to the baselines for which the code was officially released: TENT[3], TTT++[4], CoTTA[5], EATA[6], and NOTE[7]. We did experiments on their code by adding the needed data loader or pre-trained model loader. In this section, implementation details of the baselines are provided.

**BN Stats Adapt [22]** is one of the non-training TTA approaches. It can be implemented by setting the model to the train mode[8] of Pytorch [24] during TTA.

**TTT++ [19]** was originally implemented as the offline adaptation, *i.e.*, multi-epoch training. So, we modified their setup to continual TTA. We further tuned the learning rate as 0.005 and 0.00025 for adapting to CIFAR10-C and CIFAR100-C, respectively.

**NOTE [8]** proposed the methods named IABN and PBRS with taking account of temporally correlated target data. However, our experiments were conducted with target data that was independent and identically distributed (i.i.d.). Hence, we adapted NOTE-i.i.d (*i.e.*, NOTE* in their git repository), which is a combination of TENT [28] and IABN without using PBRS. We fine-tuned the $\alpha$ of their main paper (*i.e.*, self.k in the code[9]) to 8 and the learning rate to 1e-5.

**Others** (*e.g.*, TENT [28], SWR&NSP [5], CoTTA [29], and EATA [23]). We utilized the best hyperparameters specified in their paper and code. In the case where the batch size of their works (*e.g.*, 200 and 256) differs from one for our experiments (*e.g.*, 64), we decreased the learning rate linearly based on the batch size [9].

**AdaptBN [25].** We set the hyperparameter $N$ of their main paper to 8. When AdaptBN is employed alongside TENT or our approach, we set the learning rate to 1e-5 or 5e-6 [18].

### E.2. On-device learning works

To unify the backbone network as ResNet-50 [12], we reproduced the following works by referencing their paper and published code: TinyTL[10], Rep-Net[11], and AuxAdapt. This section presents additional implementation details for reproducing the above three works.

**TinyTL [1].** We attach the LiteResidualModules[12] to layer1 to 4 in the case of ResNet-50[13]. As the hyperparameters of the LiteResidualModules, the hyperparameter *expand* is set to 4 while the other hyperparameters follow the default values.

**Rep-Net [30].** We divide the encoder of ResNet-50 into six parts, as each part of the encoder has 2,2,3,3,3,3 residual blocks (*e.g.*, BasicBlock or Bottleneck in Pytorch) from the shallow to the deep parts sequentially. Then, we connect the ProgramModules[14] to each corresponding part of the en-

[3]https://github.com/DequanWang/tent
[4]https://github.com/vita-epfl/ttt-plus-plus
[5]https://github.com/qinenergy/cotta
[6]https://github.com/mr-eggplant/EATA
[7]https://github.com/TaesikGong/NOTE

[8]pytorch.org/docs/stable/generated/torch.nn.Module.html#torch.nn.Module.train
[9]https://github.com/TaesikGong/NOTE/blob/main/utils/iabn.py
[10]https://github.com/mit-han-lab/tinyml/tree/master/tinytl
[11]https://github.com/ASU-ESIC-FAN-Lab/RepNet
[12]https://github.com/mit-han-lab/tinyml/blob/master/tinytl/tinytl/model/modules.py
[13]https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py
[14]github.com/ASU-ESIC-FAN-Lab/RepNet/blob/master/repnet/model/reprogram.py

| Time | | $t \longrightarrow$ | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Arch** | **Method** | **Gaus.** | **Shot** | **Impu.** | **Defo.** | **Glas.** | **Moti.** | **Zoom** | **Snow** | **Fros.** | **Fog** | **Brig.** | **Cont.** | **Elas.** | **Pixe.** | **Jpeg** | **Avg. err** | **Mem.** |
| WRN-40 (AugMix) | Source | 44.3 | 37.0 | 44.8 | 30.6 | 43.9 | 32.6 | 29.4 | 23.9 | 30.1 | 39.7 | 12.9 | 66.4 | 32.7 | 58.4 | 23.5 | 36.7 | 11 |
| | tBN [22] | 19.5 | 17.6 | 23.8 | 9.6 | 23.1 | 11.1 | 10.3 | 13.4 | 14.2 | 15.0 | 8.0 | 13.9 | 17.3 | 16.0 | 18.8 | 15.4 | 11 |
| | Single do. TENT [28] | 16.4 | 13.9 | 19.1 | 8.3 | 19.1 | 9.3 | 8.6 | 10.9 | 11.3 | 12.0 | 6.9 | 11.6 | 14.6 | 12.2 | 15.6 | 12.7 | 188 |
| | TENT continual [28] | 16.4 | 12.2 | 17.1 | 9.1 | 18.7 | 11.4 | 10.4 | 12.7 | 12.4 | 14.8 | 10.1 | 13.0 | 17.0 | 13.3 | 19.0 | 13.3 | 188 |
| | TTT++ [19] | 19.1 | 16.9 | 22.2 | 9.3 | 21.6 | 10.8 | 9.8 | 12.7 | 13.1 | 14.3 | 7.8 | 13.9 | 15.9 | 14.2 | 17.2 | 14.6 | 391 |
| | SWRNSP [5] | 15.9 | 13.3 | 18.2 | 8.4 | 18.5 | 9.5 | 8.6 | 11.0 | 10.2 | 11.7 | 7.0 | 8.1 | 14.6 | 11.3 | 15.1 | 12.1 | 400 |
| | NOTE [8] | 19.6 | 16.4 | 19.9 | 9.4 | 20.3 | 10.3 | 10.1 | 11.6 | 10.6 | 13.3 | 7.9 | 7.7 | 15.4 | 12.0 | 17.3 | 13.4 | 188 |
| | EATA [23] | 15.2 | 13.1 | 17.5 | 9.5 | 19.9 | 11.6 | 9.3 | 11.4 | 11.5 | 12.4 | 7.8 | 11.1 | 16.1 | 12.2 | 16.1 | 13.0 | 188 |
| | CoTTA [29] | 15.6 | 13.6 | 17.3 | 9.8 | 19.0 | 11.0 | 10.2 | 13.5 | 12.6 | 17.4 | 7.8 | 17.3 | 16.2 | 12.9 | 16.0 | 14.0 | 409 |
| | Ours (K=4) | 16.1 | 13.2 | 18.3 | 8.0 | 18.3 | 9.3 | 8.6 | 10.5 | 10.1 | 12.2 | 6.8 | 11.3 | 14.5 | 11.0 | 14.8 | 12.2 | 80 |
| | Ours (K=5) | 15.9 | 12.6 | 17.2 | 8.2 | 18.4 | 9.3 | 8.6 | 10.6 | 10.4 | 12.4 | 6.7 | 11.7 | 14.3 | 11.3 | 14.9 | 12.1 | 92 |
| WRN-28 | Source | 72.3 | 65.7 | 72.9 | 46.9 | 54.3 | 34.8 | 42.0 | 25.1 | 41.3 | 26.0 | 9.3 | 46.7 | 26.6 | 58.5 | 30.3 | 43.5 | 58 |
| | tBN [22] | 28.6 | 26.8 | 37.0 | 13.2 | 35.4 | 14.4 | 12.6 | 18.0 | 18.2 | 16.0 | 8.6 | 13.3 | 24.0 | 20.3 | 27.8 | 20.9 | 58 |
| | Single do. TENT [28] | 25.2 | 23.8 | 33.5 | 12.8 | 32.3 | 14.1 | 11.7 | 16.4 | 17.0 | 14.4 | 8.4 | 12.2 | 22.8 | 18.0 | 24.8 | 19.2 | 646 |
| | Continual TENT [28] | 25.2 | 20.8 | 29.8 | 14.4 | 31.5 | 15.4 | 14.2 | 18.8 | 17.5 | 17.3 | 10.9 | 14.9 | 23.6 | 20.2 | 25.6 | 20.0 | 646 |
| | TTT++ [19] | 27.9 | 25.8 | 35.8 | 13.0 | 34.3 | 14.2 | 12.2 | 17.4 | 17.6 | 15.5 | 8.6 | 13.1 | 23.1 | 19.6 | 26.6 | 20.3 | 1405 |
| | SWRNSP [5] | 24.6 | 20.5 | 29.3 | 12.4 | 31.1 | 13.0 | 11.3 | 15.3 | 14.7 | 11.7 | 7.8 | 9.3 | 21.5 | 15.6 | 20.3 | 17.2 | 1551 |
| | NOTE [8] | 30.4 | 26.7 | 34.6 | 13.6 | 36.3 | 13.7 | 13.9 | 17.2 | 15.8 | 15.2 | 9.1 | 7.5 | 24.1 | 18.4 | 25.9 | 20.2 | 646 |
| | EATA [23] | 23.8 | 18.8 | 27.3 | 13.9 | 29.7 | 16.0 | 13.3 | 18.0 | 16.9 | 15.7 | 10.5 | 12.2 | 22.9 | 17.1 | 23.0 | 18.6 | 646 |
| | CoTTA [29] | 24.6 | 21.6 | 26.5 | 12.1 | 28.0 | 13.0 | 10.9 | 15.3 | 14.6 | 13.6 | 8.1 | 12.2 | 20.0 | 14.9 | 19.5 | 17.0 | 1697 |
| | Ours (K=4) | 23.5 | 19.0 | 26.6 | 11.5 | 30.6 | 13.1 | 10.9 | 15.2 | 14.5 | 13.1 | 7.8 | 11.4 | 20.9 | 15.4 | 20.8 | 16.9 | 404 |
| | Ours (K=5) | 23.8 | 18.7 | 25.7 | 11.5 | 29.8 | 13.3 | 11.3 | 15.3 | 15.0 | 13.0 | 7.9 | 11.3 | 20.2 | 15.1 | 20.5 | 16.8 | 471 |
| Resnet-50 | Source | 65.6 | 60.7 | 74.4 | 28.9 | 79.9 | 46.0 | 25.7 | 35.0 | 49.4 | 54.7 | 13.0 | 83.2 | 41.2 | 46.7 | 27.7 | 48.8 | 91 |
| | tBN [22] | 18.0 | 17.2 | 29.3 | 10.7 | 27.2 | 15.5 | 8.9 | 16.7 | 14.6 | 21.0 | 9.3 | 12.7 | 20.9 | 12.4 | 14.8 | 16.6 | 91 |
| | Single do. TENT [28] | 16.6 | 15.7 | 25.7 | 10.0 | 24.8 | 13.8 | 8.3 | 14.9 | 13.8 | 17.6 | 8.7 | 10.0 | 19.1 | 11.5 | 13.8 | 15.0 | 925 |
| | TENT continual [28] | 16.6 | 14.4 | 22.9 | 10.4 | 22.6 | 13.4 | 10.3 | 15.8 | 14.6 | 18.0 | 10.5 | 11.7 | 18.4 | 13.1 | 15.3 | 15.2 | 925 |
| | TTT++ [19] | 18.2 | 16.9 | 28.7 | 10.5 | 26.5 | 14.5 | 8.9 | 16.5 | 14.5 | 20.9 | 9.0 | 9.0 | 20.4 | 12.3 | 14.7 | 16.1 | 1877 |
| | SWRNSP [5] | 17.3 | 16.1 | 26.1 | 10.6 | 25.6 | 14.1 | 8.7 | 15.6 | 13.6 | 18.6 | 8.8 | 10.0 | 19.3 | 12.0 | 14.2 | 15.4 | 1971 |
| | EATA [23] | 17.2 | 14.9 | 23.6 | 10.2 | 23.3 | 13.2 | 8.5 | 14.0 | 12.5 | 16.6 | 8.6 | 9.4 | 17.2 | 11.0 | 12.7 | 14.2 | 925 |
| | CoTTA [29] | 16.2 | 15.0 | 21.2 | 10.4 | 22.8 | 13.9 | 8.4 | 15.1 | 12.9 | 19.8 | 8.6 | 11.3 | 17.5 | 10.5 | 12.2 | 14.4 | 2066 |
| | Ours (K=4) | 16.5 | 14.5 | 24.3 | 9.7 | 23.7 | 13.3 | 8.8 | 14.7 | 12.9 | 17.0 | 9.1 | 9.4 | 17.6 | 11.4 | 13.1 | 14.4 | 296 |
| | Ours (K=5) | 16.6 | 14.4 | 23.6 | 9.8 | 23.4 | 12.7 | 8.6 | 14.5 | 12.6 | 16.6 | 8.7 | 9.0 | 17.0 | 11.3 | 12.6 | 14.1 | 498 |

Table 19. **Comparison of error rate (%) on CIFARC10-C with severity level 5.** We conduct experiments on continual TTA setup. Avg. err means the average error rate (%) of all 15 corruptions, and Mem. denotes total memory consumption, including model parameter sizes and activations. WRN refers to WideResNet. The implementation details of the baselines are described in Section E.1.

| Time | | $t \longrightarrow$ | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Arch** | **Method** | **Gaus.** | **Shot** | **Impu.** | **Defo.** | **Glas.** | **Moti.** | **Zoom** | **Snow** | **Fros.** | **Fog** | **Brig.** | **Cont.** | **Elas.** | **Pixe.** | **Jpeg** | **Avg. err** | **Mem.** |
| WRN-40 (AugMix) | Source | 80.1 | 77.0 | 76.4 | 59.9 | 77.6 | 64.2 | 59.3 | 64.8 | 71.3 | 78.3 | 48.1 | 83.4 | 65.8 | 80.4 | 59.2 | 69.7 | 11 |
| | tBN [22] | 45.9 | 45.6 | 48.2 | 33.6 | 47.9 | 34.5 | 34.1 | 40.3 | 40.4 | 47.1 | 31.7 | 39.7 | 42.7 | 39.2 | 45.6 | 41.1 | 11 |
| | Single do. TENT [28] | 41.2 | 40.6 | 42.2 | 30.9 | 43.4 | 31.8 | 30.6 | 35.3 | 36.2 | 40.1 | 28.5 | 35.5 | 39.1 | 33.9 | 41.7 | 36.7 | 188 |
| | continual TENT [28] | 41.2 | 38.2 | 41.0 | 32.9 | 43.9 | 34.9 | 33.2 | 37.7 | 37.2 | 41.5 | 33.2 | 37.2 | 41.1 | 35.9 | 45.1 | 38.3 | 188 |
| | TTT++ [19] | 46.0 | 45.4 | 48.2 | 33.5 | 47.7 | 34.4 | 33.8 | 39.9 | 40.2 | 47.1 | 31.8 | 39.7 | 42.5 | 38.9 | 45.5 | 41.0 | 391 |
| | SWRNSP [5] | 42.4 | 40.9 | 42.7 | 30.6 | 43.9 | 31.7 | 31.3 | 36.1 | 36.2 | 41.5 | 28.7 | 34.1 | 39.2 | 33.6 | 41.3 | 36.6 | 400 |
| | NOTE [8] | 50.9 | 47.4 | 49.0 | 37.3 | 49.6 | 37.3 | 37.0 | 41.3 | 39.9 | 47.0 | 35.2 | 34.7 | 45.2 | 40.9 | 49.9 | 42.8 | 188 |
| | EATA [23] | 41.6 | 39.9 | 41.2 | 31.7 | 44.0 | 32.4 | 31.9 | 36.2 | 36.8 | 39.7 | 29.1 | 34.4 | 39.9 | 34.2 | 42.2 | 37.1 | 188 |
| | CoTTA [29] | 43.5 | 41.7 | 43.7 | 32.2 | 43.7 | 32.8 | 32.2 | 38.5 | 37.6 | 45.9 | 29.0 | 38.1 | 39.2 | 33.8 | 39.4 | 38.1 | 409 |
| | Ours (K=4) | 42.7 | 39.6 | 42.4 | 31.4 | 42.9 | 31.9 | 30.8 | 35.1 | 34.8 | 40.7 | 28.1 | 35.0 | 37.5 | 32.1 | 40.5 | 36.4 | 80 |
| | Ours (K=5) | 41.8 | 39.0 | 41.9 | 31.2 | 42.7 | 32.5 | 31.0 | 35.0 | 35.0 | 39.9 | 28.8 | 34.5 | 37.5 | 32.8 | 40.5 | 36.3 | 92 |
| Resnet-50 | Source | 84.7 | 83.5 | 93.3 | 59.6 | 92.5 | 71.9 | 54.8 | 66.6 | 77.6 | 81.8 | 44.3 | 91.2 | 72.2 | 76.6 | 56.5 | 73.8 | 91 |
| | tBN [22] | 48.1 | 46.7 | 60.6 | 35.1 | 58.0 | 41.8 | 33.2 | 47.3 | 43.5 | 54.9 | 33.5 | 35.3 | 49.8 | 38.4 | 40.8 | 44.5 | 91 |
| | Single do. TENT [28] | 44.1 | 42.7 | 53.9 | 32.6 | 52.0 | 37.5 | 30.5 | 43.4 | 40.2 | 45.7 | 30.4 | 31.4 | 45.1 | 35.0 | 37.6 | 40.1 | 926 |
| | continual TENT [28] | 44.0 | 40.1 | 49.9 | 34.7 | 50.6 | 40.0 | 33.6 | 47.0 | 45.7 | 53.4 | 42.5 | 46.2 | 56.1 | 51.2 | 53.3 | 45.9 | 926 |
| | TTT++ [19] | 48.1 | 46.5 | 60.8 | 35.1 | 57.8 | 41.6 | 32.9 | 46.8 | 43.3 | 55.0 | 33.3 | 34.0 | 50.0 | 38.1 | 40.6 | 44.2 | 1876 |
| | SWRNSP [5] | 48.3 | 46.5 | 60.5 | 35.1 | 57.9 | 41.7 | 32.9 | 47.1 | 43.5 | 54.7 | 33.5 | 35.1 | 49.9 | 38.3 | 40.7 | 44.1 | 1970 |
| | EATA [23] | 44.8 | 41.9 | 52.6 | 33.0 | 51.1 | 37.8 | 30.3 | 43.0 | 40.1 | 45.1 | 30.1 | 31.8 | 45.2 | 35.2 | 37.4 | 39.9 | 926 |
| | CoTTA [29] | 43.6 | 42.8 | 50.4 | 34.2 | 51.6 | 39.2 | 31.4 | 43.4 | 39.6 | 47.4 | 31.3 | 32.2 | 43.4 | 35.8 | 36.7 | 40.2 | 2064 |
| | Ours (K=4) | 44.8 | 40.3 | 49.2 | 32.3 | 50.1 | 36.3 | 29.5 | 41.0 | 39.9 | 44.6 | 31.5 | 33.7 | 45.3 | 36.3 | 37.7 | 39.5 | 296 |
| | Ours (K=5) | 44.9 | 40.4 | 48.9 | 32.7 | 49.7 | 36.9 | 29.3 | 40.8 | 39.0 | 44.4 | 31.1 | 33.6 | 44.0 | 35.7 | 37.8 | 39.3 | 498 |

Table 20. **Comparison of error rate (%) on CIFARC100-C with severity level 5.** We conduct experiments on continual TTA setup. Avg. err means the average error rate (%) of all 15 corruptions, and Mem. denotes total memory consumption, including model parameter sizes and activations. WRN refers to WideResNet. The implementation details of the baselines are described in Section E.1.

coder. For the ProgramModule, we set the hyperparameter *expand* to 4 while the rest hyperparameters are used as their default values. We copy the input conv of ResNet-50 and make use of it as the input conv of Rep-Net.

**AuxAdapt [32].** We use ResNet-18 as the AuxNet. We create pseudo labels by fusing the logits output of ResNet-50 and ResNet-18, and optimize all parameters of ResNet-18 using the pseudo labels with cross-entropy loss.

**Warming up the additional modules.** Before model deployment, we pre-train the additional modules (*i.e.*, the LiteResidualModule of TinyTL [1], the ProgramModule of Rep-Net [30], and the AuxNet of AuxAdapt [32]) on the source data using the same strategy warming up the meta networks as mentioned in Section C.

## F. Results of all corruptions

We report the error rates (%) of all corruptions on continual TTA and memory consumption (MB) including the model parameters and activations in Table 19 and Table 20. These tables contain additional details to Table 1.

## References

[1] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tinytl: Reduce memory, not parameters for efficient on-device learning. In *NeurIPS*, 2020. 1, 5, 6

[2] Dian Chen, Dequan Wang, Trevor Darrell, and Sayna Ebrahimi. Contrastive test-time adaptation. In *CVPR*, 2022. 3

[3] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 4

[4] 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 *CVPR*, 2021. 4

[5] Sungha Choi, Seunghan Yang, Seokeon Choi, and Sungrack Yun. Improving test-time adaptation via shift-agnostic weight regularization and nearest source prototypes. In *ECCV*, 2022. 3, 4, 5, 6

[6] 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 *CVPR*, 2016. 4

[7] Zhiwei Deng and Olga Russakovsky. Remember the past: Distilling datasets into addressable memories for neural networks. *arXiv preprint arXiv:2206.02916*, 2022. 3

[8] Taesik Gong, Jongheon Jeong, Taewon Kim, Yewon Kim, Jinwoo Shin, and Sung-Ju Lee. Robust continual test-time adaptation: Instance-aware bn and prediction-balanced memory. In *NeurIPS*, 2023. 5, 6

[9] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 5

[10] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *NeurIPS*, 2004. 4

[11] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *ICML*, 2017. 2

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 5

[13] Dan Hendrycks, Norman Mu, Ekin D. Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. AugMix: A simple data processing method to improve robustness and uncertainty. In *ICLR*, 2020. 4

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NeurIPS*, 2014. 2, 3

[15] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *ICML*, 2019. 2

[16] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *ICLR*, 2022. 2

[17] Andreas Krause, Pietro Perona, and Ryan Gomes. Discriminative clustering by regularized information maximization. In *NeurIPS*, 2010. 4

[18] Hyesu Lim, Byeonggeun Kim, Jaegul Choo, and Sungha Choi. TTN: A domain-shift aware batch normalization in test-time adaptation. In *ICLR*, 2023. 3, 5

[19] Yuejiang Liu, Parth Kothari, Bastien van Delft, Baptiste Bellot-Gurlet, Taylor Mordan, and Alexandre Alahi. Ttt++: When does self-supervised test-time training fail or thrive? In *NeurIPS*, 2021. 3, 5, 6

[20] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *NeurIPS*, 2017. 3

[21] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? In *NeurIPS*, 2019. 2

[22] Zachary Nado, Shreyas Padhy, D Sculley, Alexander D'Amour, Balaji Lakshminarayanan, and Jasper Snoek. Evaluating prediction-time batch normalization for robustness under covariate shift. *arXiv preprint arXiv:2006.10963*, 2020. 5, 6

[23] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In *ICML*, 2022. 1, 2, 3, 4, 5, 6

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, and et al. Lin. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 1, 5

[25] Steffen Schneider, Evgenia Rusak, Luisa Eck, Oliver Bringmann, Wieland Brendel, and Matthias Bethge. Improving robustness against common corruptions by covariate shift adaptation. In *NeurIPS*, 2020. 5

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 2014. 2

[27] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. In *CVPR*, 2019. 2

[28] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *ICLR*, 2021. 1, 2, 3, 5, 6

[29] Qin Wang, Olga Fink, Luc Van Gool, and Dengxin Dai. Continual test-time domain adaptation. In *CVPR*, 2022. 1, 2, 3, 5, 6

[30] Li Yang, Adnan Siraj Rakin, and Deliang Fan. Rep-net: Efficient on-device learning via feature reprogramming. In *CVPR*, 2022. 5, 6

[31] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016. 1

[32] Yizhe Zhang, Shubhankar Borse, Hong Cai, and Fatih Porikli. Auxadapt: Stable and efficient test-time adaptation for temporally consistent video semantic segmentation. In *WACV*, 2022. 6