Supplementary Materials for REALM: Robust Entropy Adaptive Loss Minimization for Improved Single-Sample Test-Time Adaptation

Skyler Seto, Barry-John Theobald, Federico Danieli, Navdeep Jaitly, Dan Busbridge
Apple
{sseto,barryjohn_theobald, f_danieli, njaitly, dbusbridge}@apple.com

A. REALM Details

In this section, we detail the derivation showing that REALM is an SPL objective, and provide pseudocode for our implementation of REALM in Algorithm 1.

A.1. REALM as an SPL Objective

As outlined in Sec. 4, the EATA procedure can be re-cast as a SPL method with the explicit regularizer

$$g(w;\lambda) = -\lambda\|w\|_1. \tag{1}$$

Similarly, minimizing the loss function

$$\rho(x;\alpha,\lambda) = \frac{|\alpha-2|}{\alpha}\cdot C\cdot\left[\left(\frac{(x/\lambda)}{|\alpha-2|}+1\right)^{\alpha/2}-1\right], \tag{2}$$

used in our REALM framework can be reinterpreted as solving a regularized optimization problem similar to

$$\begin{aligned} w^*,\theta^* &= \arg\min_{w,\theta}\mathbb{E}\left[w(x)\mathcal{L}(\theta;x)+g(w;\lambda)\right], \\ &= \arg\min_{w,\theta}\frac{1}{N}\sum_{i=1}^{N}\left[w(x_i)\mathcal{L}(\theta;x_i)+g(w(x_i);\lambda)\right], \end{aligned} \tag{3}$$

with a specific regularizer $g(w;\alpha,\lambda)$. In this section, we aim to derive an explicit formula for $g(w;\alpha,\lambda)$.

For our analysis, we closely follow the derivation in [2]. The goal is to equivalently cast the robust minimization problem REALM:

$$\theta^*,\alpha^*,\lambda^* = \min_{\theta,\alpha,\lambda} S_{div}(x)\rho\left(\mathcal{L}(\theta;x);\alpha,\lambda\right), \tag{4}$$

which we simplify as

$$\min_{\theta}\rho\left(\mathcal{L}(\theta);\alpha,\lambda\right), \tag{5}$$

as a regularized minimization problem, in the form

$$\min_{\theta,w\in[0,1]}\left[w\frac{\mathcal{L}(\theta)}{\lambda}+g(w;\alpha)\right]. \tag{6}$$

Note that in Eq. (5) we are ignoring the term S_{div}, which is treated as a constant; we are also not considering the optimization with respect to α and λ, since this can be treated separately. Moreover, with a slight abuse of notation, let us re-define more compactly $t = \mathcal{L}(\theta)/\lambda$ as the argument of ρ, and not report the dependency on α explicitly, that is

$$\rho(t) \coloneqq \frac{|\alpha-2|}{\alpha}\left(\left(\frac{2t}{|\alpha-2|}+1\right)^{\frac{\alpha}{2}}-1\right). \tag{7}$$

For the equivalence between Eq. (5) and Eq. (6) to hold, we must ensure that minimizing either term over $\mathcal{L}(\theta)$ results in the same solution. To this end, we impose the equivalence of both

$$\rho\left(t\right) = \min_{w}\left[wt+g(w)\right], \tag{8}$$

as well as their derivatives with respect to $\mathcal{L}(\theta)$,

$$\frac{\rho'\left(t\right)}{\lambda} = \frac{w}{\lambda} \quad\Longrightarrow\quad \rho'(t) = w. \tag{9}$$

Differentiating Eq. (8) with respect to w and substituting Eq. (9), at the optimal point we get

$$t + g'\left(\rho'(t)\right) = 0. \tag{10}$$

Multiplying by $\rho''(t)$ and integrating by parts, we recover

$$\begin{aligned} &g'\left(\rho'(t)\right)\rho''(t) = -t\rho''\left(t\right) \\ \Longleftrightarrow &\left(g\left(\rho'(t)\right)\right)' = -\left(t\rho'(t)\right)' + \rho'(t) \\ \Longleftrightarrow &g\left(\rho'(t)\right) = -t\rho'(t) + \rho(t) \\ \Longleftrightarrow &g\left(w\right) = -w\left(\rho'\right)^{-1}\left(w\right) + \rho\left(\left(\rho'\right)^{-1}\left(w\right)\right), \end{aligned} \tag{11}$$

where again we substituted Eq. (9) to express the regularizing term $g(w)$ as a function of w. Notice that indeed the inverse of $\rho'(t)$ is well-defined for our choice of $\rho(t)$: we have in fact, after some simplifications,

$$\begin{aligned} &\rho'(t) = \left(\frac{2t}{|\alpha-2|}+1\right)^{\frac{\alpha-2}{2}} \\ \Longrightarrow &\left(\rho'\right)^{-1}\left(w\right) = \frac{|\alpha-2|}{2}\left(w^{\frac{2}{\alpha-2}}-1\right) \end{aligned} \tag{12}$$

Substituting this into Eq. (11) allows us to write $g(w)$ more explicitly:

$$g(w;\alpha) = \frac{|\alpha-2|}{\alpha}\left(w^{\frac{\alpha}{\alpha-2}}\left(1-\frac{\alpha}{2}\right)+\frac{\alpha}{2}w-1\right). \quad (13)$$

Now that we have a candidate form for $g(w;\alpha,\lambda)$, we need to verify that this indeed identifies a valid regularizer. First, $w=\phi'(t)$ must be an actual minimum for Eq. (6): in other words, we must have

$$\frac{\partial^2}{\partial w^2}\left(w\frac{\mathcal{L}(\theta)}{\lambda}+g(w;\alpha)\right)>0 \quad \Longrightarrow \quad g''(w;\alpha)>0. \quad (14)$$

This condition can be equivalently rewritten by taking the derivative of Eq. (10) with respect to t,

$$(t+g'(\rho'(t)))'=0 \quad \Longrightarrow \quad g''(\rho'(t))=-\frac{1}{\rho''(t)}, \quad (15)$$

which shows that, for $g(w,;\alpha)$ to be convex, it suffices to ask for $\rho(t)$ to be concave. This can be promptly verified:

$$\rho''(t)=\frac{\alpha-2}{|\alpha-2|}\left(\frac{2t}{|\alpha-2|}+1\right)^{\frac{\alpha}{2}-2}<0 \quad \text{for } \alpha<2. \quad (16)$$

Incidentally, this also implies $\rho''(t)\neq 0$, which we made use of in our derivation; moreover, in light of this, $\rho'(t)$ is monotone decreasing. We also need $w=\rho'(t)$ to span the whole domain of $w\in[0,1]$, namely

$$\lim_{t\to 0}\rho'(t)=1, \quad \text{and} \quad \lim_{t\to\infty}\rho'(t)=0. \quad (17)$$

This too can be verified from its definition in Eq. (12), and holds for $\alpha<2$. Finally, notice that Eqs. (16) and (17) above correspond to the conditions in [10, Definition 1], further confirming that the REALM robust loss function falls within the SPL framework.

A note on the adaptation of ρ for entropy minimization The original definition of the robust loss function appearing in [1] reads:

$$\rho^0(\mathcal{L};\alpha,\lambda) \coloneqq \frac{|\alpha-2|}{\alpha}\left(\left(\frac{(\mathcal{L}/\lambda)^2}{|\alpha-2|}+1\right)^{\alpha/2}-1\right). \quad (18)$$

Notice that, compared against our definition in Eq. (2), the argument of this function $(\mathcal{L}/\lambda)$ appears squared: in fact, Eq. (18) was originally intended for a squared-error type of loss function. Indeed, starting from the original definition of ρ^0 and following a similar derivation as the one outlined in Appendix A.1, one can show that the corresponding regularized minimization problem is given by

$$\min_{\theta,w\in[0,1]}\left[\frac{1}{2}w\left(\frac{\mathcal{L}(\theta)}{\lambda}\right)^2+g(w;\alpha)\right], \quad (19)$$

rather than Eq. (6): that is, the objective of the regularized problem is alsosquared. Since our target loss function is entropy (rather than squared entropy), we adapted ρ accordingly in Eq. (2), so to ensure consistency between the objectives of the robust and regularized minimization problems.

A.2. Pseudocode for REALM

Pseudocode for REALM is given in Algorithm 1. REALM is relatively easy to implement requiring only additional computation of the robust loss function to scale the entropy, and gradient updates for both α, and λ. This is the objective used in REALM as ut satisfies the desired properties, and results in a scaled entropy objective as desired.

A.3. Sources of Entropy Collapse

Methods such as Tent and MEMO lead to model collapse resulting in all samples having the same class prediction due to online entropy optimization with single sample batch sizes. While a primary cause of this collapse is noisy samples, which we aim to handle in this work, there may be other causes including high learning rates [3], or mixed distribution shifts [14]. These sources of collapse are outside the scope of our work.

Algorithm 1 REALM: Robust Entropy Adaptive Loss Minimization

Input: $\mathcal{D}_{\text{Test}}$, $f(\cdot\,;\theta_0)$, α_0, λ_0
for $t = 0$ **to** T **do**
  Compute predictions $\hat{y}_t = f(x_t;\theta)$
  Compute robust loss $\mathcal{L}^* = \rho(\mathcal{L}(\hat{y}_t))$
  Compute weight $S_{\text{div}} = \mathbb{1}\left\{\cos\left(f(x_t;\theta), m_{t-1}\right) \; < d\right\}$
  Scale the robust loss $\mathcal{L}^* \leftarrow S_{\text{div}}\mathcal{L}^*$
  Update $\theta_{t+1} \leftarrow \theta_t - \eta\nabla_\theta\mathcal{L}^*$
  Update $\alpha_{t+1}, \lambda_{t+1} \leftarrow \alpha_t, \lambda_t - \eta\nabla_{\alpha,\lambda}\mathcal{L}^*$
end for

B. Experimental Details for REALM

In this section, we provide additional details for models, datasets, and hyperparameters for all methods.

B.1. Additional Model Details

ResNet-26 GN - We use the ResNet-26 network from [16]. The model is trained on the CIFAR-10 train set.
ResNet-50 GN - We use the ResNet-50 GN architecture with pretrained weights from the `timm` library available under the name `resnet50_gn`. The model is trained on the ImageNet train set.
Vit - We use the Vit base architecture with pretrained weights from the `timm` library available as `vit_base_patch16_224`. The model is trained on the ImageNet train set.
Swin - We use the Swin tiny architecture with pretrained weights from `timm` available as `swin_tiny_patch4_window7_224`. The model is trained on the ImageNet train set.
ConvNext - We use the ConvNext tiny architecture with pretrained weights from the `timm` library as `convnext_tiny`. The model is trained on ImageNet train set.

Both the ResNet and Vit models are evaluated for comparison to [14]. The Swin and ConvNext architectures are SOTA models used to demonstrate that REALM performs well across architectures. We use the tiny versions to maintain similar parameter counts to the ResNet-50. We note that variability in performance is likely not a result of the number of model parameters, rather architectural differences such as attention layers.

B.2. Additional Dataset Details

CIFAR-10-C - CIFAR-10-C is a collection of 15 different corruptions types from four categories (noise, blur, weather, and digital) applied to the CIFAR-10 test set across five different severity levels. We evaluate all approaches on the highest severity. Each corruption has a total of 10,000 samples matching the CIFAR-10 test set [6].
ImageNet-C - ImageNet-C is a collection of 15 different corruptions types from four categories (noise, blur, weather, and digital) applied to the ImageNet validation set across five different severity levels. We evaluate all approaches on the highest severity. Each corruption has a total of 50,000 samples matching the original validation set [6].
ImageNet-R - ImageNet-R contains renditions of a subset of the classes in ImageNet including paintings, sculptures, embroidery, cartoons, origami, and toys. A total of 30,000 samples across 200 of the ImageNet classes are contained in ImageNet-R. For TTA on ImageNet-R we subset the network outputs before adapting [5].
ImageNet-A - ImageNet-A contains samples collected from Flickr and iNaturalist according to a subset of 200 of the classes in ImageNet. All samples collected are incorrectly classified by a ResNet model, and the probability of the correct class is lower than 15%. A total of 7,500 adversarially filtered samples are used for adaptation. For TTA on ImageNet-A we subset the network outputs before adapting [8].

B.3. Additional Hyerparameter Details

We outline hyperparameters for all methods. For all models, we adapt only the normalization layer parameters following [17].
TENT - For CIFAR-10 we use SGD with no momentum, and a batch size of one. We set the learning rate to 0.005. For ImageNet, we use SGD with momentum of 0.9, a learning rate of 0.00025, and batch size of one. The learning rate is scaled to account for small batch size as $\text{lr} = (\text{lr}/32)$ following [14]. The initial learning rate for the Vit model is set to 0.001 and is scaled similarly.
EATA - In addition to the hyperparameters used in TENT, we set $\lambda = 0.4 \times \log(c)$ where c is the number of classes in the dataset. For CIFAR-10, the threshold for S_{div} is set to 0.4. For ImageNet, the threshold is set to 0.05.
SAR - In addition to the hyperparameters for EATA and

TENT, we scale the learning rate as $\text{lr} = (\text{lr}/16)$ except for the Vit model, and we freeze the last block of the network. For CIFAR-10, we use a much smaller threshold at $\lambda = 0.1$ as the loss is much smaller than on ImageNet, and we found that adapting on samples with $\mathcal{L} \in [0.1, 0.4 \times \log(10)]$ resulted in unstable adaptation.

SFT - We follow the same hyperparameters as those for TENT, except we adapt only the first conv layer of the network. We did not scale the learning rate as the method creates a batch of data via augmentations. We set the number of augmentations to 64 following the batch size used in EATA [13]. For MEMO, we use an identical implementation, only we do not freeze any part of the network.

REALM - We follow the same hyperparameters as SAR for fair comparison. We set the initial $\alpha = 0.15$, $\lambda = 0.1$ for CIFAR-10, and $\lambda = 0.4 \times \log(1000)$ for ImageNet. We did not tune these values for fair comparison to SAR. The value of α is chosen as it is close to 0 and mimics the behavior of the reliable sample criteria. We set the learning rate for α and λ to a factor of 2 of the model parameter learning rates for CIFAR-10 and the same for ImageNet. Note that in Appendix D we find that setting the learning rate to the same as that for the model parameters, and dropping the learning rate improves performance. However, extra tuning of the hyperparameters outside using the same learning rate as for the model parameters may give an unfair advantage to REALM over competing methods. For fair comparison across methods in the main text we did not tune these.

C. Additional Commentary on REALM in Single Sample TTA

C.1. Results on Forgetting

Our primary focus in the main paper is to demonstrate that REALM improves online adaptation performance generalizing better to the target distribution. However, an unwanted consequence of adaptation is forgetting the original distribution resulting in decreased performance on the original in-distribution data. We investigate whether REALM results in additional forgetting over a method such as EATA [13], which reduces forgetting explicitly via the anti-forgetting regularizer in [9]. To investigate, we adapt the model on the ImageNet-C gaussian noise corruption at severity 5, then re-evaluate model performance on the original validation set. Results are summarized in Tab. 1 indicating that adaptation with REALM results in little to no drop in performance on the clean validation data.

We did additional experiments using the anti-forgetting regularizer in [9]. Using the same hyperparameters as in [13], we found that we were able to precisely maintain performance on in-distribution data, however dropped performance on the gaussian noise corruption indicating that there is a tradeoff in choosing the correct hyperparameter to control the regularizer. For other datasets outside the ones reported in this paper, it is possible that adding the additional regularizer is necessary to preserve performance on in-distribution data, however, for our experiments, we found that performance was already comparable on clean data without the need for the additional regularizer.

Method	Gaussian Noise	Clean
No Adapt	18.0	**80.0**
EATA	24.9	79.8
REALM	**26.1**	79.6

Table 1. Accuracy for ResNet-50 GN evaluated after adaptation on Gaussian noise corruptions and clean validation set. Results are averaged over 3 runs.

C.2. Wall-Clock Runtime Comparison

We implement REALM, SAR, and EATA using the Pytorch library [15], and report the wall-clock time adaptation takes for a single sample on average[1]. For the ResNet-50 with group normalization layers on ImageNet, all methods with no gradient update take between 5-10ms with standard inference taking 5ms, REALM taking 7ms, and EATA taking 10ms. When the model backwards on the single instance, REALM finishes the fastest at around 45ms, EATA takes 60ms, and SAR takes 80ms. Further note that in the case of batch updates, although fewer samples may be needed for the backward pass, a standard implementation cannot take advantage of this, and will result in similar wall-clock time per batch as found in [13]. Thus, without specialized hardware, in practical experiments, we've found REALM's runtime to be comparable to EATA and faster than SAR.

C.3. Additional Commentary on Training Robustness

Prior studies investigate methods for improving robustness to distribution shifts during training. A common practice is to train with augmentations aimed at smoothing the training data distribution, and improving robustness to covariate shifts. Some examples include Augmix [7], AutoAugment [4], Adversarial augment [20], and Mixup [18]. This approach for attaining robustness happens at training time, and our TTA approach is agnostic to this. Further, improving robustness at training time is difficult as one needs to carefully construct an augmentation policy which covers any expected distribution shifts. In contrast, applying such augmentations at test-time as is done in prior works [11, 12, 19] does not yield better performance overall, and makes the adaptation procedure much slower.

[1]We use the torch profiler and time the `model_inference` component according to the example in https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html

C.4. Additional Commentary on Batch Size of One

Our focus in this work is on the setting where online TTA is performed with a batch size of one. Recall our example where an individual is taking picture(s) on their phone on a rainy day. A phone app providing object recognition capabilities would be expected to immediately classify on the new sample. We reiterate that this is an important setting, as it may be infeasible to wait for enough samples to batch, and one may not see enough samples to form a batch. Further, in the batch size of one setting, batch normalization can fail to adapt parameters and running statistics on a single sample from the new distribution, and methods that rely on sample skipping can skip a large portion of the test samples. In this work, we do not explore larger batch size setting as we believe that online inference is less practical when waiting for a batch of data, especially when data aggregation is not feasible due to privacy or storage concerns, and optimizing over a subset of the batch has little practical advantage in terms of efficiency as long as the original batch fits in memory without specialized software, or hardware.

D. Ablation Studies

REALM reduces the impact of outliers during online adaptation by applying the robust loss ρ to penalize the gradient update of high entropy samples. While our approach is theoretically grounded in the self-paced learning literature, its implementation is rather straightforward. In this section, we study the impact of several parameter choices, and implementation details of our objective. In particular, we examine the impact of S_{div}, different terms in the loss function, reduced gradient updates on all model blocks, and different initial values for the loss. For all ablations, we report results for the gaussian noise corruption at severity 5, however we expect results to be consistent across all corruptions as removing components, and changing the objective will harm performance and nullify theoretical justification. In some cases, when tuning hyperparamters unique to REALM, we show that improved performance can be achieved. In particular, tuning the learning rate for α and λ and updating the last block of the network are both shown to increase corruption robustness. In the main text, however, we do not tune these extra hyperparameters to keep as fair comparison as possible between REALM and competing methods. Nonetheless, if one is able to tune these extra parameters, higher performance can be achieved.

D.1. On the Importance of S_{div}

Recall that for the EATA adaptation procedure there are two weights: (1) S_{ent}, which controls adaptation on outliers based on their entropy, and (2) S_{div}, which controls adaptation redundancy based on whether the predictions are similar. We connect S_{ent} to optimization of the self-paced learning objective, and treat S_{div} as independent as no gradient is applied through this weight. Results are summarized in Tab. 3.

We find that the S_{div} term is crucial for adaptation performance of REALM. We hypothesize this is because we are no longer skipping samples based on reliability. As such, we see many additional updates which have similar predictions. This means S_{div} will down-weight many of the samples we may not have used to update the model prediction EMA in other approaches.

We further show that even with the greater impact S_{div} has on learning in REALM, we still find that REALM updates over more samples than EATA. On ImageNet-C with gaussian noise corruption at severity 5, we find that REALM updates on twice as many samples as EATA in Fig. 1. We believe this leads to REALM achieving higher accuracy.

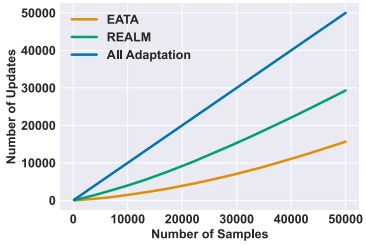


Figure 1. Number of updates for ResNet-50 GN evaluated on gaussian noise corruptions illustrating that REALM updates on twice as many samples as EATA.

D.2. Optimizing Scaled Squared Entropy

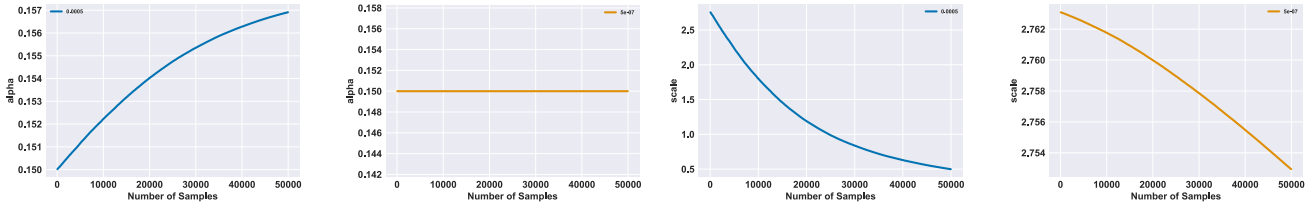
Our choice of ρ is modified from the general robust loss function defined in [1]. However, that loss function makes an implicit assumptions that the loss itself is a scaled squared norm: $\left(\frac{x}{\lambda}\right)^2$. Traditionally robust loss functions are used in robust regression settings where the scaled squared norm is suitable choice of loss. However, in other settings such as here, minimizing the squared entropy is unconventional and an unmotivated objective. Nonetheless, we show in Tab. 4 that REALM still produces the same results on the gaussian noise corruption at severity 5 as this optimization has a similar form to that of implicit SPL.

D.3. Removing Loss Scaling

The loss function reported in [1] scales the gradient by a function of the loss scale λ. For a given update, this lack of re-scaling entails that the the gradient will also be scaled by λ. We modified the loss function to account for this scaled gradient and in Tab. 5 find that this term is crucial as it increases performance on the gaussian noise corruption by 4%.

Method	Noise			Blur				Weather				Digital				Avg.
	Gauss.	Shot	Impul.	Defoc.	Glass	Motion	Zoom	Snow	Frost	Fog	Brit.	Contr.	Elastic	Pixel	JPEG	
ResNet50 (GN)	18.0	19.8	17.9	**19.8**	11.4	21.4	24.9	40.4	**47.3**	33.6	69.3	36.3	18.6	28.4	52.3	30.6
+ REALM ($2.5e-4$)	26.9	29.9	28.0	18.4	18.2	29.6	31.1	45.6	43.6	45.5	71.2	44.4	28.9	49.7	55.5	37.8
+ REALM ($5e-4$)	26.1	28.9	27.2	18.3	17.7	29.1	30.4	45.0	43.2	44.8	71.0	43.9	27.9	49.1	55.3	37.2
+ REALM ($5e-7$)	**27.9**	**31.1**	**29.2**	18.6	**18.9**	**30.4**	**32.1**	**46.4**	**44.1**	**46.3**	**71.5**	**45.1**	**30.1**	**50.4**	**55.8**	**38.5**

Table 2. Accuracy across all corruptions in ImageNet-C comparing REALM with different learning rates for α and λ. Results are averaged over 3 runs.



(a) α - lr $= 5e-4$ (b) α - lr $= 5e-7$ (c) λ - lr $= 5e-4$ (d) λ - lr $= 5e-7$

Figure 2. α and λ during adaptation for the robust loss function. Figures are over a single run to illustrate parameter changes.

Method	Gaussian Noise
REALM	**26.1**
- S_{div}	9.2

Table 3. Accuracy for ResNet-50 GN evaluated on gaussian noise corruptions clarifying the importance of the S_{div} term. Results are averaged over 3 runs.

Method	Gaussian Noise
REALM	26.1
REALM^2	26.1

Table 4. Accuracy for ResNet-50 GN evaluated on gaussian noise corruptions comparing our modified robust loss function with the version used for regression problems. Results are averaged over 3 runs.

Method	Gaussian Noise
REALM	**26.1**
- Scale Factor	22.0

Table 5. Accuracy for ResNet-50 GN evaluated on Gaussian noise corruptions comparing REALM with and without the scale factor for scaling the gradient update. Results are averaged over 3 runs.

D.4. Frozen Top Block

In prior work, it is shown that deep layers in the network are more sensitive and important to preserve for the original model as they retain semantic information, whereas the shallow layers of the model are important for covariate shifts. Thus, in our main results, following SAR, we do not update the last block of the ResNet architecture with REALM (`layer4`). We, however test whether we need to freeze blocks of the network in REALM. Tab. 6 shows it is unnecessary, and performance increases by $\sim 1\%$ on the gaussian noise corruption. Still, we believe it may be important to limit model updates on higher severity corruptions as this can negatively impact adaptation, and potentially increase model forgetting on the in-distribution data.

Method	Gaussian Noise
REALM	26.1
- Frozen Block	**27.2**

Table 6. Accuracy for ResNet-50 GN evaluated on Gaussian noise corruptions comparing REALM with freezing the last block vs. all block norm layer updates. Results are averaged over 3 runs.

D.5. Sensitivity to Learning Rate

REALM is an extension of other entropy minimization methods that rely on sample skipping by reformulating as a SPL objective. When recasting our objective, we have a few additional hyperparameters, most notably the learning rate on α and λ which controls how much these parameters update. To keep comparisons between REALM and SAR consistent, we did not tune this value in the main text and set this value at $\text{lr}_{\alpha,\lambda} = 2 \times \text{lr}_{\theta}$. Alternatively setting this directly to lr_{θ} yields improved performance as well. In either case, we do not consider tuning $\text{lr}_{\alpha,\lambda}$ in the main text as we did not want to give REALM any unfair advantage. However, we find empirically in Tab. 2 that better performance using a smaller $\text{lr}_{\alpha,\lambda}$ can be achieved, however we were not able to tune for this value without a proper validation set.

D.6. Visualizing α and λ

Finally, we plot α and λ for the robust loss during training in Fig. 2. We find that α increases or stays constant

during training, while λ decreases. Both trends follow from the loss and adaptation behavior, since as we adapt during inference, the loss on samples from the new distribution will gradually decrease on average, meaning α should increase for reduced penalization, and λ should decrease for more gradient update.

References

[1] Jonathan T Barron. A general and adaptive robust loss function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4331–4339, 2019. 2, 5

[2] Michael Black and Anand Rangarajan. On the unification line processes, outlier rejection, and robust statistics with applications in early vision. *International Journal of Computer Vision*, 19:57–91, 07 1996. 1

[3] Sungha Choi, Seunghan Yang, Seokeon Choi, and Sungrack Yun. Improving test-time adaptation via shift-agnostic weight regularization and nearest source prototypes. In *European Conference on Computer Vision*, pages 440–458. Springer, 2022. 2

[4] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 113–123, 2019. 4

[5] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8340–8349, 2021. 3

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

[7] Dan Hendrycks, Norman Mu, Ekin Dogus Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. In *International Conference on Learning Representations*, 2020. 4

[8] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15262–15271, June 2021. 3

[9] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017. 4

[10] M Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. *Advances in neural information processing systems*, 23, 2010. 2

[11] Yoonho Lee, Annie S Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. Surgical fine-tuning improves adaptation to distribution shifts. *arXiv preprint arXiv:2210.11466*, 2022. 4

[12] 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? *Advances in Neural Information Processing Systems*, 34:21808–21820, 2021. 4

[13] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In *International conference on machine learning*, pages 16888–16905. PMLR, 2022. 4

[14] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Zhiquan Wen, Yaofo Chen, Peilin Zhao, and Mingkui Tan. Towards stable test-time adaptation in dynamic wild world. *arXiv preprint arXiv:2302.12400*, 2023. 2, 3

[15] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 4

[16] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*, pages 9229–9248. PMLR, 2020. 3

[17] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021. 3

[18] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. 4

[19] Marvin Zhang, Sergey Levine, and Chelsea Finn. Memo: Test time robustness via adaptation and augmentation. *Advances in Neural Information Processing Systems*, 35:38629–38642, 2022. 4

[20] Xinyu Zhang, Qiang Wang, Jian Zhang, and Zhao Zhong. Adversarial autoaugment. *arXiv preprint arXiv:1912.11188*, 2019. 4