Supplementary Material

S1. Different Domain Generalization Setup

The problem of adapting a Deep Neural Network (DNN) to tackle real life data corruption at the edge can be formulated by different kind of Domain Generalization (DG) settings based on the nature of data stream and learning paradigm. Fig. S1 summarizes the four main DG approaches in literature, namely Fine Tuning (FT), Unsupervised Domain Adaptation (UDA), Source-Free Domain Adaptation (SFDA) and Test Time Adaptation (TTA).

Fine Tuning (FT) adapts a DNN by making it match labeled test data [1, 5]. FT approaches includes Few-Shots Learning (FSL), among others [7]. The downside of FT is that it requires labeled test data and is performed offline, thus they are hardly applicable in a mobile edge computing context. Moreover, it does not take into account samples from the previous domain, so it incurs in catastrophic forgetting [2].

Unsupervised Domain Adaptation (UDA). This approach addresses the issues of FT considering samples from the previous domain and thereby eliminating the need of labels from the new domain [10]. However, as FT, UDA assumes that we can simultaneously access (unlabeled) samples from the current domain and from the prior domain, which is not always the case. In stark opposition, our goal is achieve real-time adaptation of a DNN in dynamic and uncertain scenarios.

Source-Free Domain Adaptation (SFDA). Conversely from UDA, in SFDA the DNN adaptation is performed using unlabeled data from the target domain only [3, 4]. While SFDA approaches take into account numerous losses for several epochs during optimization, the key downside is that it can hardly be applied in real-time learning settings.

Test Time Adaptation (TTA). A practical approach to address distributional shifts in real-time settings is TTA, which utilizes only unlabeled test data (online) to adapt the DNN. While existing TTA approaches improve performance, they are sensitive to the diversity of samples in incoming distributions. For example, even in cases of minor changes in brightness, approaches adapting the normalization layer based on entropy minimization such as [6, 8] can experience a significant decrease in accuracy, dropping to less than 19% of accuracy on CIFAR-10. Moreover, existing

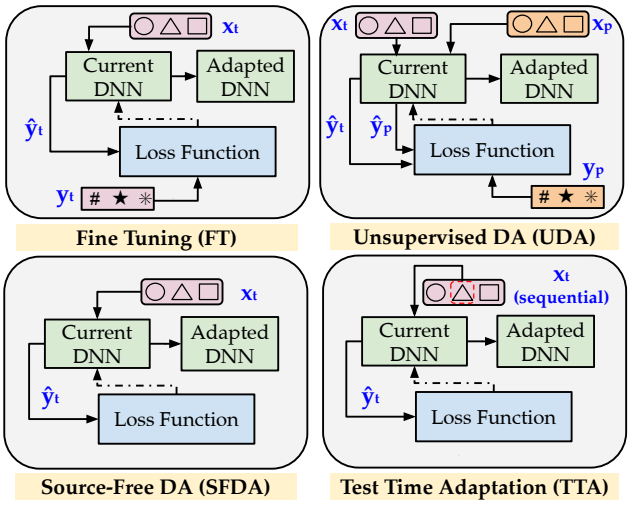


Figure S1. Domain Generalization (DG) approaches. (x_t, y_t) indicate the current test sample and its true label, while (x_p, y_p) indicate source sample and its true label. $\hat{y}_t$ indicates the corresponding predictions by the current DNN. We point out that DARDA is a TTA approach.

methods are not aware of domain changes, so they continuously update the DNN even in the presence of no domain change. Notice that decoupling the corruption from the features relevant for classification is extremely challenging. Such continuous adaptation leads to the risk of catastrophic forgetting. While state of the art work [9] uses stochastic restoration of parameters to the initial domain to tackle the issue, it needs 78.37% more storage for ResNet56 architecture than our proposed approach.

S2. Distribution and Label Shift

Different corruptions lead mainly to a distribution shift in the input data, which is also widely known as a covariate shift. Distribution shift happens when the distribution of input data changes while the distribution of true labels remain unchanged. In a real-life adaptation of DNNs at inference time, we usually have a batch of samples to work with that have both distribution and label shift (due to correlation in labels in certain scenarios) at the same time. Fig. S2 illustrates this scenario with an example. This setting is chal-

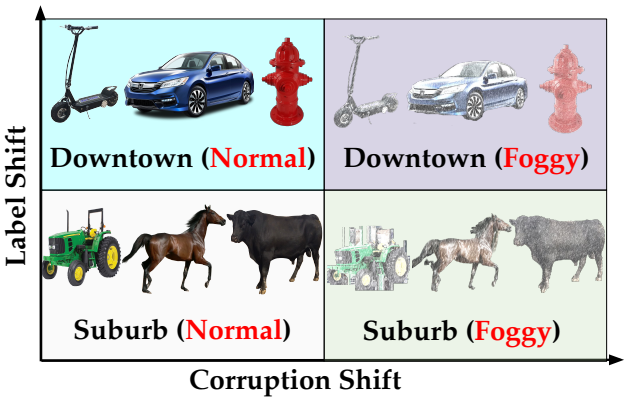


Figure S2. Example of Label Distribution & Corruption Shift.

lenging for most existing TTA algorithms, but is more practical.

S3. Hyperparameters and Implementation Details.

We implement DARDA with the PyTorch framework. For generating non independent and identically distributed (IID) real time data flow, we adopted Dirichlet distribution (with parameter δ) to create a non-IID data flow. Furthermore, to simulate a domain change due to corruption, we feed samples from different test corruption types sequentially one after another following Dirichlet distribution with control parameter δ, when all samples from the current test corruption are exhausted. With lower values of the Dirichlet parameter δ, there is less diversity among online data batches, thus less correlation. The noise extractor is designed in a lightweight manner with three convolution layer with Leaky-ReLU non-linearity stacked sequentially. For the noise encoder, we use a sequential model consisting of two convolution units each consisting of a single convolution layer with ReLU non-linearity and a MaxPooling layer. The sequential model is followed by two dense layers. The sub-network encoder consists of two dense layers with ReLU non-linearity. For the reported results, we choose the output dimension of the sub-network encoder and the corruption encoder, i.e. the dimension of latent space to be 128 and the batch size is kept at 64, while the size of the memory bank is kept the same as batch size.

For CIFAR-100 corrupted dataset, we cannot insert samples from all classes in the memory bank. The parameter value $\delta = 0.1$ is considered across all test scenarios unless otherwise specified. However, we have found empirically that presence of representative samples from the majority of the classes is sufficient. We use Adam optimizer with learning rate 1×10^3 to perform the adaptation. To generate the sub-network signature, we first train a ResNet-56 backbone on the uncorrupted training dataset (CIFAR-10 and CIFAR-100) and fine-tune the sub-networks for 20 epochs using data from 15 train corruption domain to create the 15 sub-networks and their related signatures. For the hyper parameters, we assume a fixed set of values throughout the experiments, which are $\lambda_r = 0.2$, $\lambda_e = 10$, $\varphi_{thresh} = 0.005$ and momentum value $m = 0.5$. As we make one step upgrade of the current Batch Normalization (BN) layer statistics by making sure that the samples of the memory banks are reliable, we chose a rather aggressive momentum value to weigh the normalization of current test samples highly.

S4. Power Measurement Setup

As both Raspberry Pi and Jetson Nano do not have a system integrated in them to measure power at a certain instance, we use the setup of Fig. S3 to calculate the energy consumption of different adaptation methods. The ina219 IC can provide accurate power consumption for a device at a certain time. We initially take some samples of power measurement to estimate the idle power usage of the device. Then for the whole adaptation period of each algorithm, we take samples of power drawn by the device every $10ms$. Multiplying with the sampling time and averaging over batches of data we get a very good estimate of the energy consumption of different adaptation algorithms.

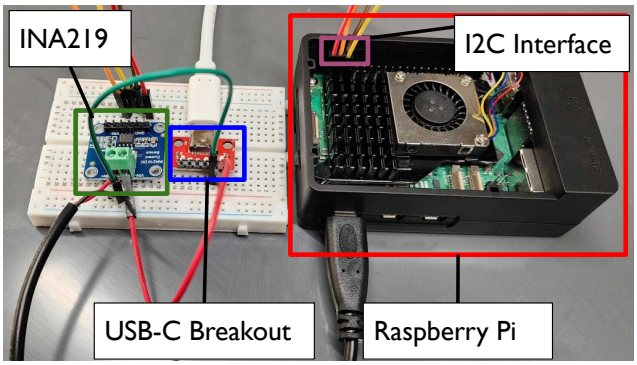


Figure S3. Power Measurement Setup.

S5. Performance of Corruption Extractor

As discussed previously, in a corrupted data sample the corruption related features are intertwined with label information which impedes extraction of contextual information from data.

To verify whether the corruption extractor is effective in extracting corruption information from data, we trained our corruption extractor and encoder using data available from 15 different corruptions and evaluate how it performs for unknown corruption and different severity. Fig. S4 shows the t-distributed stochastic neighbor embedding (t-SNE) of the projections in the latent space of data from different corruption domains and different levels of severity.

Algorithm 1: Memory Bank Construction

Input: A test sample x^t; and associated corruption embedding C^t; current corruption projection C^{curr}
Define: memory bank $\mathcal{M}$; total capacity $\mathcal{N}$; class distribution $n[y]$, where $y \in Y$; total occupancy $\mathcal{O}c$
Calculate $\hat{y} = \arg\max_y f_\theta^{'}(y \mid x^t)$
if $n[\hat{y}] < \lceil \frac{\mathcal{N}}{|Y|} \rceil \; and \; \mathcal{O}c <\mid \mathcal{N} \mid$ **then**
  Add (x^t, C^t) to $\mathcal{M}$
else
  Calculate cosine similarity f_{sim} of corruption projection among instances in $\{n[\hat{y}] \cup x^t\}$ and $\mathcal{C}^{curr}$
  Find instance $(\tilde{x}, \tilde{C})$ in $n[\hat{y}]$ with the lowest similarity $\arg\min_{x \in n[\hat{y}]} f_{sim}(x, C^{curr})$ to current signature
end if
if $f_{sim}(\tilde{x}) > f_{sim}(x^t)$ **then**
  Discard (x^t, C^t)
else
  Remove instance $(\tilde{x}, \tilde{C})$ from $\mathcal{M}$
  Add (x^t, C^t) to $\mathcal{M}$
end if

In Fig. S4a and Fig. S4c, the corruption projections are made directly using the input data. **Here, we can observe that directly encoding the corrupted data does not produce clusters with tight boundaries.** Interestingly, Fig. S4b and Fig. S4d point out that projecting the corruption signature extracted from the corrupted data does produce significantly better clusters. Moreover, although samples from the "spatter" and "saturate" have an overlapping boundary in latent space, from Tab. 1 **we can see that although they are visually dissimilar, a subnetwork that performs well for the "spatter" also works well for "saturate"**. This means that DARDA is effective not only in categorizing and mapping corruptions, but also in designing appropriate subnetworks. Moreover, Fig. S4d shows that the corruption extractor produces an even better cluster with higher severity data from unknown corruption domain even if it was trained on a lower corruption severity. This proves the intuition that the corruption extractor extracts corruption information rather than simply over-fitting to the joint distribution of data and corruption.

S6. Memory Bank Construction Process

The memory bank construction process is described in detail in Algorithm 1.

S7. More Details of Corruption Encoder

The processes involved both in the training and inference phase of the proposed Corruption Encoder is illustrated through Algorithm 2.

Algorithm 2: Corruption Encoder

Input: dataset $\mathcal{D}^d$; epochs E; batch size N; constant τ and λ_e; embed dimension o; pair downsampler $\mathcal{G}(.)$; transformation set $\mathcal{T}$; structure of $g(.)$ and $h(.)$
Output: projection vector into corruption latent space
{training}
for $epoch = 1$ **to** E **do**
  sample $\{x_i, D^i\}_{i=1}^N$
  sample two augmentations $\mathcal{T}^a$, $\mathcal{T}^b \sim \mathcal{T}$
  generate two down sampled data $\mathbf{G}_1(x_i)$, $\mathbf{G}_2(x_i) = \mathcal{G}(x_i)$
  calculate $\tilde{\mathbf{G}}_1(x_i)$, $\tilde{\mathbf{G}}_2(x_i)$ using Eq. (3)
  calculate $\mathcal{L}_n$ using Eq. (5)
  generate x_i^a, x_i^b = $\mathcal{T}^a(x_i)$,$\mathcal{T}^b(x_i)$
  generate $\mathbf{G}_1(x_i^a)$, $\mathbf{G}_2(x_i^a) = \mathcal{G}(x_i^a)$ and $\mathbf{G}_1(x_i^b)$, $\mathbf{G}_2(x_i^b) = \mathcal{G}(x_i^b)$
  extract x_i's corruption x_{res_i}, by concatenating x_{res_i} = $g((\mathbf{G}_1(x_i)) \;||\; g(\mathbf{G}_2(x_i))$
  calculate the latent space projections C^i by $C^i = h(x_{res_i})$
  calculate $\mathcal{L}_D$ using Eq. (6)
  calculate overall loss using Eq. (8)
  update $g(.)$ and $h(.)$ to minimize $\mathcal{L}$
end for{test}
for x **in** $\mathcal{D}^u$ **do**
  generate $\mathbf{G}_1(x)$, $\mathbf{G}_2(x) = \mathcal{G}(x)$
  calculate and concatenate corruption features x_{res} = $g(\mathbf{G}_1(x))||g(\mathbf{G}_2(x))$
  calculate projection into latent space by $C = h(x_{res})$
end for

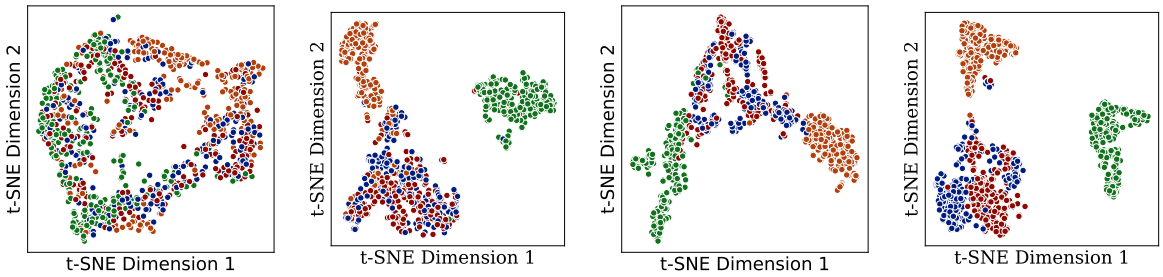
S7.1. Theoretical Analysis of the Corruption Encoder

We offer insights into the information learned by the corruption encoding process through a theoretical analysis of the corruption encoder for additive noise.

Proposition 1. *Let, two noisy observation from original sample x be $y_1 = G_1(x)$ and $y_2 = G_2(x)$. Assuming zero mean and independent additive noise; $y_1 = x + e_1$ and $y_1 = x + e_1$; where noise is denoted by e_i. If e_i can be approximated by $g(.) : e_i = g_\phi(y_i)$ by minimizing MSE loss between clean observation x_i and noisy observation y_i, minimizing the loss between two noisy observation approximate the same thing.*

True Corruption Class	Spatter	Gaussian Blur	Speckle Noise	Saturate
Data Sample				
JPEG Compression	84%	64.3%	78.1%	84.8%
Glass Blur	72.5%	79.4%	56%	74%
Shot Noise	78%	39.4%	83.8%	75.8%
Brightness	83.7%	52.8%	66.7%	85.6%

Table 1. Performance (accuracy on CIFAR-10) comparison of the sub-network signature closest to the unknown corruption signatures in the latent space. The left most column indicates the corresponding sub-network signatures. The unknown corruption domain and its closest sub-network signatures from latent space are: Spatter → JPEG Compression, Gaussian Blur → Glass Blur, Speckle Noise → Shot Noise, Saturate → Brightness.



(a) Projection of Corrupted Data (Low Severity) (b) Projection of Extracted Corruption (Low Severity) (c) Projection of Corrupted Data (High Severity) (d) Projection of Extracted Corruption (High Severity)

Figure S4. t-distributed stochastic neighbor embedding (t-SNE) of different samples in latent space for CIFAR-10. Here the green, orange, blue, and red colors indicate Gaussian blur, speckle noise, and saturate; respectively.

Proof.

$$
\begin{aligned}
g(y_1, x; \phi) &= \underset{\phi}{\operatorname{argmin}}\, \mathbb{E}\left[\|y_1 - g_\phi(y_1) - x\|_2^2\right] \\
&= \underset{\phi}{\operatorname{argmin}}\, \mathbb{E}\left[\|g_\phi(y_1)\|_2^2 - 2y_1^T g_\phi(y_1) + 2x^T g_\phi(y_1)\right] \\
g(y_1, y_2; \phi) &= \underset{\phi}{\operatorname{argmin}}\, \mathbb{E}\left[\|y_1 - g_\phi(y_1) - y_2\|_2^2\right] \\
&= \underset{\phi}{\operatorname{argmin}}\, \mathbb{E}\left[\|y_1 - g_\phi(y_1) - x - e_2\|_2^2\right] \\
&= \underset{\phi}{\operatorname{argmin}}\, \mathbb{E}[\|g_\phi(y_1)\|_2^2 - 2y_1^T g_\phi(y_1) + 2x^T g_\phi(y_1) \\
&\quad + 2e_2^T g_\phi(y_1)] \\
&= \underset{\phi}{\operatorname{argmin}}\, \mathbb{E}\left[\|g_\phi(y_1)\|_2^2 - 2y_1^T g_\phi(y_1) + 2x^T g_\phi(y_1)\right] \\
&= g(y_1, x; \phi)
\end{aligned}
$$

Assuming zero mean $\mathbf{E}(e_i) = 0$ and independent noise the second to last equality is satisfied □

References

[1] Yunhe Gao, Xingjian Shi, Yi Zhu, Hao Wang, Zhiqiang Tang, Xiong Zhou, Mu Li, and Dimitris N Metaxas. Visual Prompt Tuning for Test-Time Domain Adaptation. *arXiv preprint arXiv:2210.04831*, 2022. 1

[2] Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind Your Neural Network to Prevent Catastrophic Forgetting. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 466–483. Springer, 2020. 1

[3] Vinod K Kurmi, Venkatesh K Subramanian, and Vinay P Namboodiri. Domain impression: A source data free domain adaptation method. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 615–625, 2021. 1

[4] Rui Li, Qianfen Jiao, Wenming Cao, Hau-San Wong, and Si Wu. Model adaptation: Unsupervised domain adaptation without source data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9641–9650, 2020. 1

[5] Zhiqiu Lin, Samuel Yu, Zhiyi Kuang, Deepak Pathak, and Deva Ramanan. Multimodality Helps Unimodality: Cross-Modal Few-Shot Learning with Multimodal Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19325–19337, 2023. 1

[6] 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. 1

[7] Feng Tian, Yue Yu, Xu Yuan, Bin Lyu, and Guan Gui. Predicted Decoupling for Coexistence Between WiFi and LTE in Unlicensed Band. *IEEE Transactions on Vehicular Technology*, 69(4):4130–4141, 2020. 1

[8] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *arXiv preprint arXiv:2006.10726*, 2020. 1

[9] 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. 1

[10] Garrett Wilson and Diane J Cook. A survey of unsupervised deep domain adaptation. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(5):1–46, 2020. 1