DARDA: Domain-Aware Real-Time Dynamic Neural Network Adaptation *

Shahriar Rifat
Northeastern University
United States
rifat.s@northeastern.edu

Jonathan Ashdown
Air Force Research Laboratory
United States
jonathan.ashdown@us.af.mil

Francesco Restuccia
Northeastern University
United States
f.restuccia@northeastern.edu

Abstract

Test Time Adaptation (TTA) has emerged as a practical solution to mitigate the performance degradation of Deep Neural Networks (DNNs) in the presence of corruption/ noise affecting inputs. Existing approaches in TTA continuously adapt the DNN, leading to excessive resource consumption and performance degradation due to accumulation of error stemming from lack of supervision. In this work, we propose Domain-Aware Real-Time Dynamic Adaptation (DARDA) to address such issues. Our key approach is to proactively learn latent representations of some corruption types, each one associated with a sub-network state tailored to correctly classify inputs affected by that corruption. After deployment, DARDA adapts the DNN to previously unseen corruptions in an unsupervised fashion *by (i) estimating the latent representation of the ongoing corruption; (ii) selecting the sub-network whose associated corruption is the closest in the latent space to the ongoing corruption; and (iii) adapting DNN state, so that its representation matches the ongoing corruption. This way, DARDA is more resource-efficient and can swiftly adapt to new distributions caused by different corruptions without requiring a large variety of input data. Through experiments with two popular mobile edge devices – Raspberry Pi and NVIDIA Jetson Nano – we show that DARDA reduces energy consumption and average cache memory footprint respectively by* $1.74\times$ *and* $2.64\times$ *with respect to the state of the art, while increasing the performance by* 10.4%, 5.7% *and* 4.4% *on CIFAR-10, CIFAR-100 and TinyImagenet.*

1. Introduction

Traditional mobile edge computing scenarios assume that the inputs of DNNs are received uncorrupted. However, in many real-life scenarios, sudden and unexpected corruptions (e.g., snowy or foggy conditions) can cause a drastic change in data distribution, consequently causing performance loss [1, 15]. For example, a semantic segmentation DNN trained with data collected in normal weather conditions has been shown to exhibit a performance loss of more than 30% when tested in snowy conditions [14], while an image classification DNN can experience a similar decrease in the case of reduced lighting conditions [5].

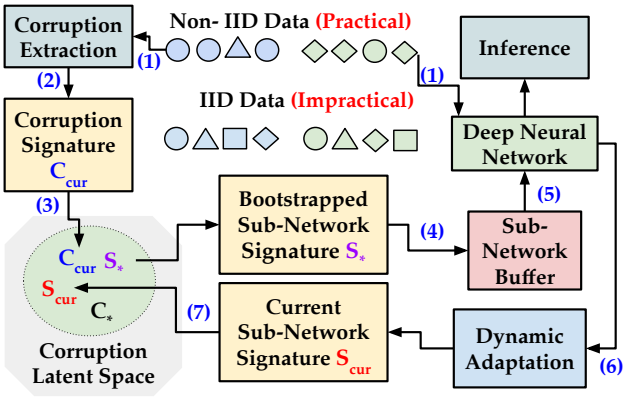


Figure 1. Overview of the proposed DARDA framework.

Test-time Adaptation (TTA) tackles this issue by adapting the DNN with unlabeled test data in an online manner, thus handling distributional shifts in real time. Existing TTA methods lose performance when encountering continuously changing distributions with highly correlated input samples [17, 19]. This assumption is true in many real-world scenarios. For example, an unmanned autonomous vehicle (UAV) monitoring an outdoor environment will likely encounter similar classes as video feeds are very likely to be highly correlated when considering limited time spans. Continuous adaptation of an edge deployed DNN to such a challenging yet practical scenario causes many adaptation methods to fail. Moreover, existing methods lack awareness of when the domain shift happens, thus they continuously fine-tune the DNN even if there is no shift in data distribution. However, in a real-life deployment scenario, certain data distribution might persist for a certain period of time (e.g., a bright sunny day). This imposes unnecessary burden on energy consumption and cache memory – without yielding performance improvements.

*Approved for Public Release; Distribution Unlimited: AFRL-2024-5704.

To address the critical issues defined above, we propose a new framework named *Domain-Aware Real-Time Dynamic Adaptation* (DARDA), shown in Fig. 1. We now provide a step-by-step walk-through of DARDA. First, the data stream can shift due to some corruption which is also correlated in label space (distribution of labels are not uniform) (step 1). The corruption process is then detected and extracted (step 2), resulting in a latent representation (step 3), which we call *signature* for brevity. Then, the corruption signature C_{cor} is assigned to the closest *corruption centroid* C_*. Each centroid is learned during training and represents a known corruption type. Moreover, it is associated with a "bootstrapped" sub-network of the main DNN that is specifically tailored to the specific corruption. The sub-network signature S_* is then used to retrieve the actual sub-network structure and weight (step 4), which is then immediately plugged into the DNN (step 5). Next, the DNN is updated to match the type of ongoing corruption (*not seen during the subnetwork and corruption signature training phase*) (step 6) by "moving" the current subnetwork signature S_{cur} closer to the actual ongoing corruption C_{cur} (step 7).

Summary of Novel Contributions

• We propose Domain-Aware Rapid Dynamic Adaptation (DARDA) to seamlessly and effectively adapt in real-time state-of-the-art DNN to unseen corruptions (Section Sec. 4). The key innovation of DARDA is a brand-new approach to learn a latent space putting together the corruption process and the state of the DNN, which is done through a *corruption extractor* (Section Sec. 4.1), a corruption encoder (Section Sec. 4.2) and a *sub-network encoder* (Section Sec. 4.3), which together make DARDA able to seamlessly adapt the DNN with *unlabeled* inputs.

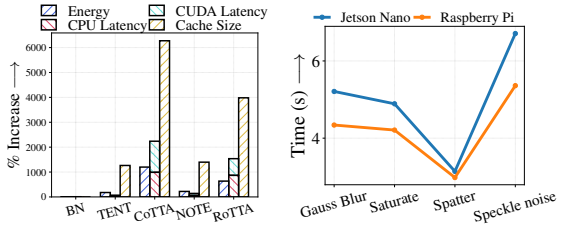
• We prototype DARDA and evaluate its performance against five state of the art approaches, namely BN [10], TENT [17], CoTTA [19], NOTE [2], RoTTA [21] on the ResNet-56 DNN trained with the CIFAR-10 and CIFAR-100 datasets augmented with same known corruption types that are considered to be known prior to deployment. We show that DARDA improves the performance by 10.4% and 5.7% on CIFAR-10 and CIFAR-100 respectively, while performing 29% less forward computation and 77% less backward passes during the adaptation process and with only 16.57% of additional memory.

• We implement DARDA on Jetson-Nano and Raspberry Pi 5, commonly used to exhibit efficiency of edge-deployed DNNs [4, 20]. Experiments show that DARDA handle distribution shifts while being $1.74\times$ more energy efficient than the best-performing state of the art TTA algorithm. For adaptation and inference task DARDA takes $7.3\times$ less time per sample while it occupies $2.64\times$ less cache memory.

2. Related Work and Existing Issues

Some existing works [2, 16] on TTA have provided empirical evidence of performance improvement by only re-estimating the normalization statistics of Batch Normalization (BN) layers from test data. The absence of supervision is typically covered by two unsupervised forms of losses. Firstly, a line of work [3, 12, 18] minimizes the entropy of the predictions over a batch of data to prevent the collapse of a trivial solution. Invariance regularization-based TTA algorithms perform some data augmentation (e.g. rotation [19], adversarial perturbation [11]) on test data during inference. The inconsistency of the prediction of DNN on different augmented test data is leveraged as an unsupervised loss function to update the learnable parameters during inference. The proposed DARDA framework uses cross-modal learning to acquire a shared representation space between the corruption space and the DNN space [13, 23]. However, to our knowledge, none of the existing research addresses cross-modal learning between the corruption process and the state of the DNN model. Next, we discuss into some practical limitations of TTA in edge vision application.

Excessive Resource Consumption. To improve performance, existing TTA approaches typically involve continuous adaptation even with uncorrupted input samples, thus imposing a heavy burden on edge resources. Ideally, adaptation should be only performed upon changes in the corruption process, thereby conserving constrained resources such as energy and processing power at the edge. Despite the potential benefits, current methods have yet to explore this direction. Fig. 2a shows a significant increase in resources between inference-only and existing TTA approaches in Jetson Nano. Specifically, the energy consumption increases by up to $11.9\times$, along with a $6.8\times$ increase in CPU latency and a $3.1\times$ increase in GPU latency.



(a) Increase in resource consumption with respect to inference-only execution

(b) Adaptation latency for Jetson Nano and Raspberry Pi for different corruption types

Figure 2. Current issues of Test-time Adaptation.

Higher Average Cache Usage. Mobile edge devices have limited memory size. As such, it is compelling to ensure TTA uses minimal memory footprint. Specifically, during inference, various blocks of a DNN are executed sequentially, and the cache memory usage at any given time is bounded by the data size of the activation map of the block

being computed. However, as the TTA dynamically updates the DNN, local gradients of learnable parameters with respect to intermediate activation are stored, a quantity that scales with both the DNN size and the number of learnable parameters. From Fig. 2a, we observe that the average cache usage increases by up to 21.7× compared to a DNN deployed for inference only.

Dependence on Sample Diversity. To be effective in real-world scenarios, the dynamic adaptation of a DNN needs to happen in a rapid fashion. However, Figure Fig. 2b shows that existing state of the art work [21] takes significant time to restore the performance of the original DNN when the noise condition keeps changing. The reason behind such behavior lies in the inherent dependency on sample diversity of [21]. In other words, if diverse samples are not observed with a new corruption process, the work [21] cannot achieve its optimum performance gains.

3. Problem Statement

We define d_s as the number of available learning domains, each characterizing a different imperfection and/or corruption type. We further define the related set of d_s datasets where x_i^d and y_i^d indicates i^{th} data sample and label from domain d respectively, as

$$\mathcal{D}^d = \left\{\left(x_i^d, y_i^d\right)\right\}_{i=1}^{n_d}, \text{ with } 0 \leq d < d_s. \tag{1}$$

Each dataset $\mathcal{D}^d$ is composed of n_{d_s} independent and identically distributed (IID) samples characterized by some probability distribution $P^d(X,Y)$ where X and Y are random variables of input and output respectively. We assume a DNN has been trained on an uncorrupted dataset and d_s sub-networks $f(.;\theta_d)$ are created so that (i) their architecture includes the batch normalization layer and the dense layers of the DNN; (ii) their weights θ_d are obtained by fine-tuning each sub-network to each specific domain. We assume continuous and correlated (thus, non-IID) data flow to the DNN in real time, coming from d_u unknown domain datasets $\mathcal{D}^u$, with $0 \leq u < d_u$. By unknown we mean $P^u(X,Y) \neq P^d(X,Y)$, for all $(0 \leq d < d_s\,, 0 \leq u < d_u)$. We define a *domain latent space* $\mathcal{O} \subset \mathbb{R}^o$, where o is the dimension of the latent space. Our goal is to (i) sense when the data flow has changed domain from the current domain d to the unknown domain u; (ii) infer domain t that is closest to u in the latent space; (ii) select the related fine-tuned sub-network $f(\cdot;\theta_t)$ to quickly recover performance, and (iii) adapt $f(\cdot;\theta_t)$ so as to find optimal $f^*(.;\theta^*)$ such that:

$$\theta_t^* = \arg\min_{\theta_t} \frac{1}{n_u} \sum_{i=1}^{n_u} \mathcal{L}\left\{f\left(x_i^u;\theta_t\right); y_i^u\right\} \tag{2}$$

where n_u is a given number of samples in the unknown domain. Such samples are assumed to be available sequentially and the distribution of labels is different from the current domain's, i.e., $P^u(Y) \neq P^d(Y)$. Notice that ground-truth labels y_i^u are usually not available in real-world settings and are only used for performance evaluation.

4. Description of DARDA Framework

The main components of DARDA are a corruption extractor (Section Sec. 4.1), a corruption encoder (Section Sec. 4.2) and a model encoder (Section Sec. 4.3), a new corruption-aware memory bank (Section Sec. 4.4), new batch normalization scheme (Section Sec. 4.5) and a new real-time adaptation module (Section Sec. 4.6).

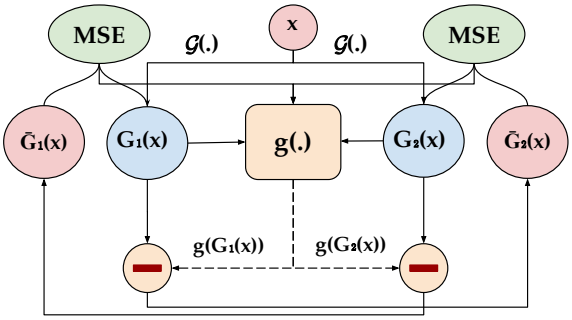


Figure 3. Proposed Corruption Extractor.

4.1. DARDA Corruption Extractor

Fig. 3 shows our proposed corruption extraction approach. Our key intuition is that features related to corruption and semantic features for inference are tightly intertwined. Since decoupling these features is difficult without corresponding clean samples, we design a process to decouple corruption features without corresponding clean sample. Specifically, we learn the corruption features by mapping corrupted data to a different corrupted version of the same data [6, 8]. For a given corrupted data x we downsample it through two convolution kernels with static filters $G_1(.) = [[0, 0.5], [0.5, 0]]$ and $G_2(.) = [[0.5, 0], [0, 0.5]]$ to generate two downsampled version of the corrupted data. From the first downsampled corrupted data $\mathbf{G}_1(x)$, we try to create an exact copy of the other downsampled data $\mathbf{G}_2(x)$ by subtracting some residual information learned by passing $\mathbf{G}_1(x)$ through the corruption extractor $g(\cdot)$. We denote this predicted copy as $\tilde{\mathbf{G}}_2(x)$. Similarly, we compute $\tilde{\mathbf{G}}_1(x)$ from $\mathbf{G}_2(x)$. The mapping functions are as follows:

$$\tilde{\mathbf{G}_2}(x) = \mathbf{G}_1(x) - g(\mathbf{G}_1(x)) \tag{3}$$

$$\tilde{\mathbf{G}_1}(x) = \mathbf{G}_2(x) - g(\mathbf{G}_2(x)) \tag{4}$$

In Fig. 3, the extracted residual is denoted by dotted lines. The parameters of $g(\cdot)$ can be optimized by minimizing the following loss function, which is the loss mean squared error (MSE) indicated in Fig. 3.

$$\mathcal{L}_N = \sum_{c=1}^{d_s} \sum_{i=1}^{n_{d_s}} \frac{1}{2} \Big(\Big\| \tilde{\mathbf{G}}_2(x_{c,i}) - \mathbf{G}_2(x_{c,i}) \Big\|_2^2 + \Big\| \tilde{\mathbf{G}}_1(x_{c,i}) - \mathbf{G}_1(x_{c,i}) \Big\|_2^2 \Big) \tag{5}$$

The intuition behind our approach is that the pixel values of the uncorrupted data in close proximity are usually highly correlated. Therefore, two downsampled versions of the data would be almost the same since they were generated by averaging values in close proximity. The corruption process breaks this correlation. Thus, our extractor $g(\cdot)$ learns to extract a representation of the corruption to generate uncorrupted data. We design this loss function in Eq. (5) to create an opposite dynamics that enable us to extract information about the corruption process. We have theoretically proven through Proposition 1 that, for additive noise our proposed approach indeed learns to extract information about the corruption. In Tab. 1, it is empirically verified that the proposed corruption extractor is useful for different kinds of corruption in general.

4.2. DARDA Corruption Encoder

We use the corruption-related features to detect a corruption shift in real time. Specifically, we use a corruption encoder $h(\cdot)$ to encode corruption information of the input data from the known corruption types into a projection in the corruption latent space. While generating the latent space, we ensure that samples from the same corruption distribution are grouped together in that latent space and samples from different corruption distribution are located distant from each other. For the N samples $\{C^i, D^i\}_{i=1}^N$ in a training data batch, we define $\mathcal{C}$ as the set of each corruption projection C^i in the latent space. We also define as D^i as the corruption label for projection C^i, and as $\mathcal{D}$ the corresponding set. We define the following supervised contrastive loss function for a batch of training data:

$$\mathcal{L}_D(\mathcal{C}, \mathcal{D}) = \sum_{i=1}^{2 \cdot N} \mathcal{L}_D^i(\mathcal{C}, \mathcal{D}) \tag{6}$$

where $\mathcal{L}_D^i(\mathcal{C}, \mathcal{D})$ is defined as

$$\mathcal{L}_D^i(\mathcal{C}, \mathcal{D}) = \frac{-1}{2 \cdot n_{d_s} - 1} \sum_{j=1}^{2 \cdot N} \mathbf{1}_{(i \neq j) \& (D^i \neq D^j)} \times \log \frac{exp(C^i . C^j / \tau)}{\sum_{k=1}^{2 \cdot N} \mathbf{1}_{(k \neq i)} exp(C^i . C^k / \tau)} \tag{7}$$

where N represents the total number of samples in the batch and τ is a scaling parameter. While training the corruption encoder, we generate a soft augmentation (random rotations and flips) from each data sample to have more samples from each noise classes. This way, our training batch size becomes $2 \cdot N$. For each sample i in our training batch, we calculate its contrastive loss using Eq. Eq. (7). Here, the numerator enforces cosine similarity between similar corruption types and the denominator penalizes high similarity between projections which are from different corruption class. Thus, similar corruption projections are positioned closer and dissimilar ones are positioned far apart. We jointly train the corruption extractor $g(.)$ and corruption encoder $h(.)$ by minimizing the loss function:

$$\mathcal{L} = \mathcal{L}_D + \lambda_e \cdot \mathcal{L}_N \tag{8}$$

where λ_e is a constant which does not impact performance yet makes the convergence of $g(.)$ and $h(.)$ faster. The training process is described in Algorithm 2.

4.3. DARDA Sub-Network Encoder

To guide the adaptation of the sub-network, we need to obtain a "fingerprint" of the current sub-network, whose state space is by definition continuous and infinite. We address this issue by creating a set of unique fingerprints $F^1 \cdots F^{d_s}$ of each sub-network by feeding a fixed Gaussian noise into the DNN and consider its output response vector as the fingerprint of the sub-network. Our intuition is that since a DNN works as a non-linear function approximator, it will produce a different output for the same input with different parameters. We generate a signature S^d of each sub-network from each fingerprint F^d as $S^d = \mathcal{S}(F^d; \psi)$ where $\mathcal{S} : F \rightarrow \mathbb{R}^o$ which is a shallow neural network parameterized by ψ that maps the fingerprint to the corruption latent space. The $\mathcal{S}$ encoder is trained so as to minimize the following loss function, which maximizes the cosine similarity between the latent space projections:

$$\mathcal{L}_{CM} = \sum_{i=1}^{d_s} \sum_{j=1}^{d_s} \mathbf{1}_{i=j} \left\{ exp(- S^i . C^j) \right\} \tag{9}$$

Here, $\mathbf{1}(.)$ is the indicator function. We use a regularization term in addition to $\mathcal{L}_{CM}$ to distribute encoder $\mathcal{S}$'s projection in regions from where sub-network's projections would produce well-performing sub networks. The measured cosine similarity between a sub-network signature S^i and a corruption signature C_j is converted into probability distribution π_{ij} using:

$$\pi_{ij} = \sigma \left(\frac{S^i \cdot C^j}{\sum_{k=1}^{d_s} S^k \cdot C^k} \right), i, j \in [0, d_s] \tag{10}$$

where σ is the softmax function. If a_{ij} indicates accuracy of sub-network i in corruption domain j we can generate a probability distribution α_{ij} such as

$$\alpha_{ij} = \sigma \left(\frac{\log 1/(1 - a_{ij})}{\sum_{k=1}^{d_s} \log 1/(1 - a_{ik})} \right), i, j \in [0, d_s] \tag{11}$$

We can calibrate the sub-network encoder to to generate projections that have affinity with other corruption domains where it can perform well by minimizing the KL divergence between π_{ij} and α_{ij}. The regularization term $\mathcal{L}_r$ and the loss $\mathcal{L}_m$ that the sub-network encoder is trained on are

$$\mathcal{L}_r = \sum_{i=1}^{d_s}\sum_{j=1}^{d_s} \log \frac{\pi_{ij}}{\alpha_{ij}} \tag{12}$$

$$\mathcal{L}_m = \mathcal{L}_{CM} \ + \ \lambda_r \, \mathcal{L}_r, \tag{13}$$

where $0 < \lambda_r < 1$ is a regularization parameter.

4.4. Corruption-Aware Memory Bank

In practical scenarios, the distribution of labels differs from the actual label distribution. Importantly, while during training, the DNN is given input data with IID labels, in real-world scenarios sequential data is highly correlated while other classes are very scarce at a particular time. Adaptation to this unreliable label distribution leads to substantial performance loss in traditional approaches, as shown in Section Sec. 5. To address this problem, we need to have a stable snapshot of the ongoing corruption at inference time. Thus, we create and maintain a memory bank $\mathcal{M}$ with $\mathcal{N}$ slots to store samples. We construct the memory bank in a label-balanced manner. Recalling that Y is the set of labels, for each class $y \in Y$, we store $\frac{\mathcal{N}}{|Y|}$ number of incoming test samples. As we do not have labeled data, the labels are inferred from the prediction $\hat{y}$ of the model. However, sampling based on prediction of continuously adapted model leads to error accumulation [21].

To solve error accumulation, existing methods [19, 21] resort to inference with multiple DNNs by feeding different augmented views of test samples to them. However, this requires additional computation and memory for multiple inference. Although sensing the corruption and bootstrapping with proper sub-network signature leads to reliable memory bank construction, we store the samples that are only representative of the ongoing corruption. For each incoming test samples we predict its class label, and store it in the memory bank if we have room for that particular class and if it is highly representative of the ongoing corruption. The process of memory bank construction is described in Algorithm 1.

4.5. Corruption-Aware Batch Normalization

Due to sudden corruption, the normalization statistics $(\bar{\mu}, \bar{\sigma}^2)$ estimated on uncorrupted training data become unreliable. Although using a specific sub-network with normalization statistics (μ_s, σ_s^2) would be feasible, we can use the samples stored in the memory bank to further refine our estimation of the ongoing statistics (μ_t, σ_t^2). To this end, we use BN [7]. Let $A^l \, \epsilon \, \mathbb{R}^{B \times Ch^l \times N^l}$ be a batch of activation tensors of the l^{th} convolutional layer, where B corresponds to the batch size, Ch^l denotes the number of channels in l^{th} layer and N_l is the dimension of activations in each channel. A BN layer first calculates $\mu_{Ch} = \frac{1}{|B||N^l|}\sum_{b \epsilon B,\ n \epsilon N^l}(A^l)$ and $\sigma_{Ch} = \frac{1}{|B|}\sum_{b \epsilon B}(A^l - \mu_{Ch})^2$ and subtracts μ_{Ch} from all input activations in the channel. Subsequently, BN divides the centered activation by the standard deviation σ_{Ch}. Normalization is applied:

$$BN\left(A^l_{b,Ch,N^l}\right) \leftarrow \gamma \times \frac{A^l_{b,Ch,N^l} - \mu_{Ch}}{\sqrt{\sigma^2_{Ch} + \epsilon}} + \beta \quad \forall\, b, Ch, N^l \tag{14}$$

Here, γ and β are the affine scaling and shifting parameters followed by normalization, while $(\epsilon > 0)$ is a small constant added for numerical stability. The normalized and affine transformed outputs are passed to the next $(l+1)^{th}$ layer, while the normalized output is kept to the l^{th} layer. BN also keeps track of the estimate of running mean and variance to use during the inference phase as a global estimate of normalization statistics, and γ and β are optimized with the other DNN parameters through back propagation.

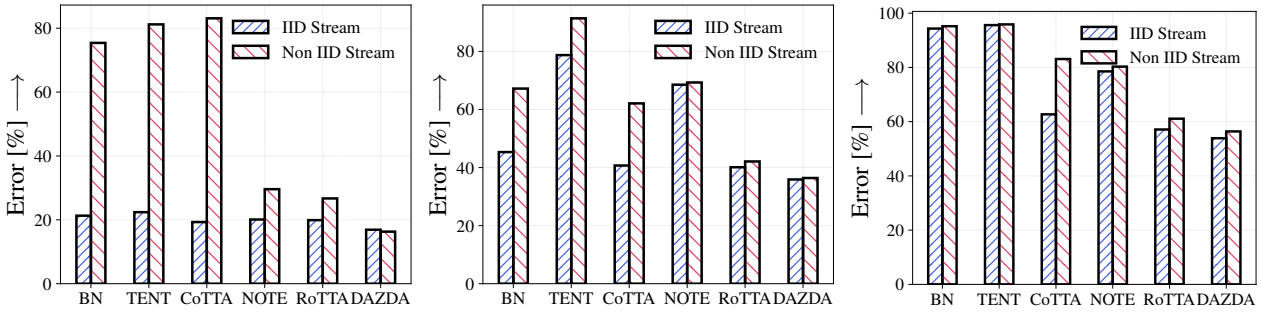
Whenever the corruption changes, the projection from the corruption encoder matches with the closest corruption centroid in the latent space. At the same time, samples affected by the new corruption are being stored in the memory bank. As we constrict the memory bank to have certain amount of samples from a particular class, due to our design of non IID real world data stream, initially there will be samples from previous corruption distribution also on the memory bank. Fig. Fig. S4 shows that the projections for even the unknown corruption get clustered nearby in the latent space. When the samples of the memory bank become representative of the current corruption, they should have low variance among their cosine similarity with current closest corruption centroid C_{cur}. Therefore, when the change in corruption is detected and the variance of cosine similarity from the centroid becomes lower than φ_{thresh}, current DNN normalization statistics are updated as:

$$\begin{aligned} \mu_s &= (1-m)\cdot\mu_s \ + \ m \cdot \hat{\mu}^t \\ \sigma_s^2 &= (1-m)\cdot\sigma_s^2 \ + \ m \cdot \hat{\sigma}_t^2 \end{aligned} \tag{15}$$

where m is the momentum and $(\hat{\mu}_t, \hat{\sigma}_t^2)$ are the current normalization statistics of different layers of the DNN, which we obtain by making one forward pass using the samples in the memory bank.

4.6. Corruption-Aware Real-Time DNN Adaptation

Adapting the parameters of the current DNN in an unsupervised manner usually needs careful selection of hyperparameters. To avoid this issue, only the sub-network is adapted. As explained earlier, we make one forward pass with the samples in the memory bank and the fixed Gaussian noise to calculate the normalization statistics of the current ongoing corruption and current sub-network fingerprint. The mean of the corruption embedding $\bar{C}$ =



(a) CIFAR-10 (b) CIFAR-100 (c) Tiny Imagenet

Figure 4. Performance Comparison in Different Popular Corruption Benchmark Datasets (Unseen Corruptions).

$\frac{1}{\mathcal{N}}\sum_{i=1}^{\mathcal{N}} C^i$ of the samples in the memory bank and the sub-network projection S is also calculated. Specifically, the tuneable parameters of the sub-network (shift β and scale γ parameters of the BN layer and final fully connected layers) are updated using gradient descent to minimize the following unsupervised loss:

$$\mathcal{L}_u = exp(-S \cdot \bar{C}) \tag{16}$$

Notice that we do not assume any access to labeled data, and do not use pseudo-labels (i.e., DNN prediction) as labels. Conversely, we minimize the loss between corruption embedding and model state in the latent space which is not disrupted by the distribution of labels in the batch.

5. Experimental Results

Datasets. We use the tool described in [9] to synthetically generate realistic corruptions for CIFAR-10 and CIFAR-100 datasets. Both CIFAR-10 and CIFAR-100 have 50,000 training images and 10,000 testing images. In line with prior work, we use the following 15 different corruptions of different categories which are "Noise" (*Gaussian, shot, impulse*), "Blur" (*defocus, glass, motion, zoom*), "Weather" (*snow, frost, fog, bright*) and "Digital" (*contrast, elastic, pixelate, JPEG*) to train our corruption encoder. For fair comparison, we evaluate the performance on 4 corruptions (*Gaussian blur, saturate, spatter, speckle noise*) which are unseen during training phase. In line with [2, 17, 19, 22], we consider the corruptions in their highest severity.

5.1. Comparison with State-of-the-Art Benchmarks

Fig. 4a and Fig. 4b show the performance DARDA as compared to other state of the art approaches. We show results with ideal IID assumption and a more realistic non-IID assumption (i.e., samples are correlated).

As we can observe, DARDA performs consistently better in both datasets and across both setups, with DARDA improving the performance by 10.4% on CIFAR-10 test corruptions and 5.7% on CIFAR-100 compared to the 2nd best performing baseline RoTTA. Among the other considered baselines, TENT, CoTTA and BN achieves poor performance for non-IID samples. Since DARDA starts from a bootstrapped sub-network in changed corruption domain and update only with samples from our memory bank and corruption latent space, DARDA performs consistently well in both cases. Since NOTE is equipped to handle correlation among online data batches, there is no significant performance drop for non-IID assumption. However, NOTE resets the DNN after evaluation on each corruption type which is unrealistic as it does not have the capability to know when current corruption domain is changing. Thus, due to error accumulation for continuous adaptation, the performance is fairly poor. DARDA does not accumulate errors from the previous corruption domain, as in every new corruption domain we start from a new DNN state.

5.2. Sensitivity to Correlated Samples

The performance of DARDA does not depend on the label space to bootstrap from an appropriate sub-network. To prove this point, we plot the average performance in the first five batches of incoming data by varying the value of the correlation parameter δ for the CIFAR-100 dataset using sequences similar to those used for Fig. 4b. From Fig. 5 it emerges that the correlation does not have a sensible effect on DARDA. Furthermore, as the value of δ decreases, the performance of BN and CoTTA drastically decreases, and for the best method RoTTA, performance starts to plummet when encountering severe correlation. Indeed, with high correlation among samples, there is not enough sample diversity to have a stable update of the DNN. However, DARDA can reliably sense corruption drift even with a single sample and bootstrap with the most similar sub-network stored in the buffer. Thus, even for extremely correlated samples, we have a well-performing sub-network from the early instances of data batches after incurring in corruption. This indicates the efficacy and reliability of DARDA in critical mobile edge computing scenarios.

5.3. Effect of Batch Size and Dirichlet Parameter

We can observe from Fig. 6 that the batch size and Dirichlet parameter do not have a significant effect on performance of DARDA. This is because the corruption signature can be extracted even with a single sample. Moreover, by updating the normalization statistics of the BN layer us-

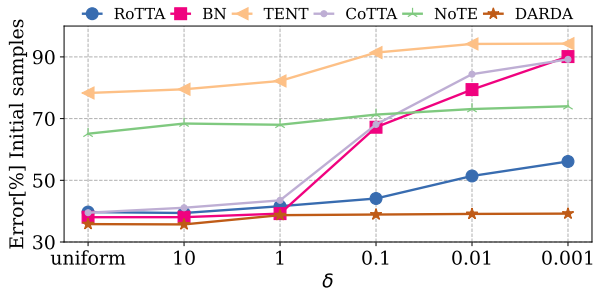


Figure 5. Performance across first five data batches for continuously incurred corruption on CIFAR-100.

ing a memory bank that enforces diversity among samples, we ensure that samples are representative of the ongoing corruption. Among other approaches, a higher value of batch size leads to high performance gain for TENT, BN and CoTTA. Especially for CoTTA, with batch size greater than 128, the performance reaches up to RoTTA. Also, the overall effect of Dirichlet parameter δ is less prominent than the initial batches.

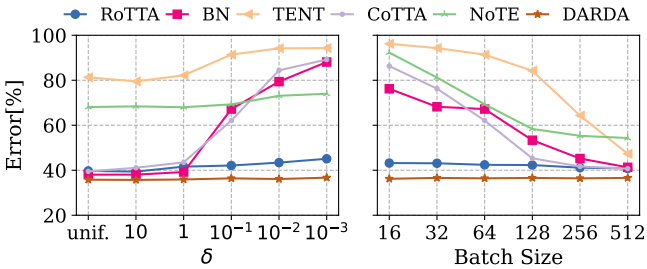
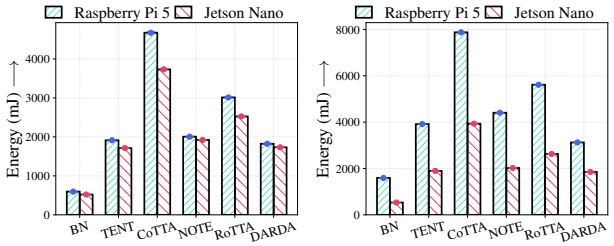


Figure 6. Performance vs correlation coefficient and batch size.



(a) CIFAR-100 (b) Tiny Imagenet

Figure 7. Energy consumption on common edge devices for a batch of 64 data samples.

5.4. Evaluation of Catastrophic Forgetting

The adaptation of DNN should not result in performance degradation on uncorrupted inputs, since in most real-life scenarios uncorrupted data is most common. However, during TTA, the DNN might get specialized in certain corruptions and fail to deliver performance. Fig. 8 shows the performance showing 5 uncorrupted data batches from CIFAR-10 after two sequences, which are (on the left subfigure) saturate → Gaussian blur → spatter → speckle and (on the right subfigure) Gaussian blur → saturate → spatter → speckle. Fig. 8 concludes that conversely from DARDA, state-of-the-art approaches incur in catastrophic forgetting. The best-performing baseline RoTTA has its accuracy degraded by up to 22% when uncorrupted data is fed after speckle noise. The classification error is particularly high when uncorrupted data is fed after a more severe corruption type (e.g, Gaussian blur) than the milder one (e.g, spatter).

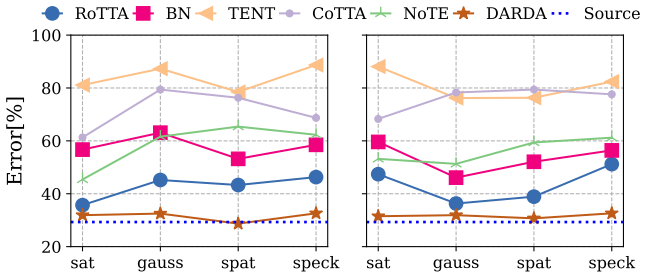


Figure 8. Performance when 5 uncorrupted data batches are fed after every adaptation, for CIFAR-100 and $\delta = 0.01$.

5.5. On-Device Dynamic Adaptation Efficiency

In this section, we evaluate the on-device performance of DARDA in diverse edge platforms. We select commonly available Raspberry-Pi-5 and Nvidia Jetson-Nano, since they are representative of resource-constrained devices widely applied for mobile vision applications.

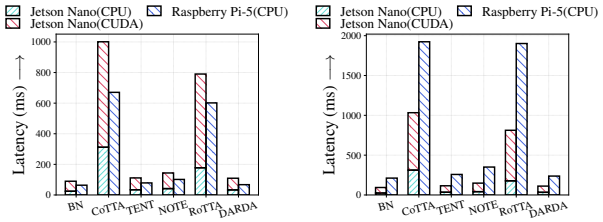
DARDA adapts using one backward pass to update the sub-network only when a corruption is perceived by the corruption extractor. The corruption extractor and corruption encoder are also active for each data sample. Tab. 1 reports the average number of multiply and accumulate (MAC) operations for the forward pass of each method. For the backward pass, the average number of samples for different corruptions with which the backward pass was called is reported. To calculate DARDA forward pass MAC, all the operations involved in the corruption extractor, corruption encoder and sub-network encoder are summed with the operation performed by different sub-networks. To support continuous adaptation without error accumulation, CoTTA and RoTTA continuously perform two forward passes with the original data sample and another augmented sample respectively. Thus, their number of operations is two times more than BN, TENT and CoTTA. Although BN, TENT and NOTE have less forward computation than DARDA, the unrealistic assumption of IID data stream and episodic adaptation – notice that the DNN state is continuously reset after adaptation – which makes them not applicable in real-time mobile edge applications. From Fig. 9 it is also evident that DARDA incurs lower CPU and GPU latency compared to closest performing benchmarks.

From Tab. 1, we can be observe that there is an excessive amount of cache usage during adaptation, except for BN, which can slow down or even block some other tasks we are interested in using the same device. For example, the closest performing baseline RoTTA in continual adapta-

Method	Cache(Mb)	Average no. samples Forward MACS	Average no. Samples Backward Pass
BN	67	1.28×10^{12}	0
TENT	914	1.28×10^{12}	10,000
CoTTA	4271	2.56×10^{12}	10,000
NOTE	1003	1.28×10^{12}	10,000
RoTTA	2735	2.56×10^{12}	10,000
DARDA	312	1.82×10^{12}	2294

Table 1. Computation and Average Cache Usage of DARDA on CIFAR-100.

tion settings needs $8.78x$ cache than DARDA. Although BN needs less cache usage than DARDA to operate as it does not need to store gradient for backward pass, it performs poorly, as shown in Sec. 5.1. From Fig. 7, it is observed that DARDA uses $2.9\times$ less energy than the closest performing benchmark in terms of performance. To calculate the energy consumption value, we use the setup in Fig. S3.



(a) CIFAR-100 (b) Tiny Imagenet

Figure 9. Adaptation Latency on common edge devices for a batch of 64 data samples.

5.6. Impact of Submodules of DARDA

To investigate the contribution of different components toward performance gain, we replace different parts of DARDA using different alternative options. We report the related performance in Tab. 2. We consider (i) DARDA without Corruption Extractor, where corrupted data is directly projected into latent space; (ii) DARDA without context-aware BN & Adaptation, directly using the DNN constructed from current sub-network signature for prediction; (iii) DARDA with context-aware BN & without fine tuning, we update the normalization values of the BN layers using samples from the memory bank but do not update the tunable parameters; (iv) DARDA with context-aware BN and Entropy Minimization, where we update the tunable parameters by minimizing entropy of predictions.

From Tab. 2 it can be observed that the corruption extractor is crucial for the performance as erroneous corruption projection would select the wrong sub-network. Creating corruption projections using only corrupted data leads to performance drop of 20.4% and 16.6% on CIFAR-10 and CIFAR-100 unseen corruption respectively. Context-aware BN with adaptation is also important as the accuracy reduces by 6.4% and 5.1% respectively for test corruptions on CIFAR-10 and CIFAR-100, respectively. It is an interesting observation that for CIFAR-10, updating the tunable

Variants of DARDA	Error [%] (CIFAR-10)	Error [%] (CIFAR-100)
w/o Corruption Extractor	36.7	56.9
w/o Context Aware BN & Adaptation	22.7	41.5
with Context Aware BN w/o Adaptation	23.8	42.9
with Context Aware BN & Entropy Minimization	23.9	42.7
Ours	**16.3**	**36.4**

Table 2. Effect of Individual Components of DARDA.

parameters by minimizing entropy of predictions degrades the performance by 0.1% rather that improving. However, our loss $\mathcal{L}_n$ in Eq. (16) results up to 1.2% performance gain which proves that a learned cross-modal latent space can guide DNN adaptation.

6. Conclusion

In this work, we have proposed Domain-Aware Real-TimeDynamic Neural Network Adaptation (DARDA). DARDA adapts the DNN to *previously unseen* corruptions in an *unsupervised fashion* by (i) estimating the latent representation of the ongoing corruption; (ii) selecting the sub-network whose associated corruption is the closest in the latent space to the ongoing corruption; and (iii) adapting DNN state, so that its representation matches the ongoing corruption. This way, DARDA is more resource-efficient and can swiftly adapt to new distributions without requiring a large variety of input data. Through experiments with two popular mobile edge devices – Raspberry Pi and NVIDIA Jetson Nano – we show that DARDA reduces energy consumption and average cache memory footprint respectively by $1.74\times$ and $2.64\times$ with respect to the state of the art, while increasing the performance by 10.4%, 5.7% and 4.4% on CIFAR-10, CIFAR-100 and TinyImagenet.

Acknowledgment of Support and Disclaimer

This work has been funded in part by the National Science Foundation under grants CNS-2134973, ECCS-2229472, CNS-2312875 and ECCS-2329013, by the Air Force Office of Scientific Research under contract number FA9550-23-1-0261, by the Office of Naval Research under award number N00014-23-1-2221, and by the Air Force Research Laboratory via *Open Technology and Agility for Innovation (OTAFI)* under transaction number FA8750-21-9-9000 between SOSSEC, Inc. and the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of U.S. Air Force, U.S. Navy or the U.S. Government.

References

[1] Amani Al-Shawabka, Francesco Restuccia, Salvatore D'Oro, Tong Jian, Bruno Costa Rendon, Nasim Soltani, Jennifer Dy, Kaushik Chowdhury, Stratis Ioannidis, and Tommaso Melodia. Exposing the Fingerprint: Dissecting the Impact of the Wireless Channel on Radio Fingerprinting. *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2020. 1

[2] Taesik Gong, Jongheon Jeong, Taewon Kim, Yewon Kim, Jinwoo Shin, and Sung-Ju Lee. Note: Robust continual test-time adaptation against temporal correlation. *Advances in Neural Information Processing Systems*, 35:27253–27266, 2022. 2, 6

[3] Sachin Goyal, Mingjie Sun, Aditi Raghunathan, and J Zico Kolter. Test time adaptation via conjugate pseudo-labels. *Advances in Neural Information Processing Systems*, 35:6204–6218, 2022. 2

[4] Rui Han, Qinglong Zhang, Chi Harold Liu, Guoren Wang, Jian Tang, and Lydia Y Chen. Legodnn: block-grained scaling of deep neural networks for mobile vision. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 406–419, 2021. 2

[5] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019. 1

[6] Tao Huang, Songjiang Li, Xu Jia, Huchuan Lu, and Jianzhuang Liu. Neighbor2neighbor: Self-supervised denoising from single noisy images. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14781–14790, 2021. 3

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015. 5

[8] Youssef Mansour and Reinhard Heckel. Zero-shot noise2noise: Efficient image denoising without any data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14018–14027, 2023. 3

[9] Claudio Michaelis, Benjamin Mitzkus, Robert Geirhos, Evgenia Rusak, Oliver Bringmann, Alexander S Ecker, Matthias Bethge, and Wieland Brendel. Benchmarking robustness in object detection: Autonomous driving when winter is coming. *arXiv preprint arXiv:1907.07484*, 2019. 6

[10] Zachary Nado, Shreyas Padhy, D Sculley, Alexander D'Amour, Balaji Lakshminarayanan, and Jasper Snoek. Evaluating prediction-time batch normalization for robustness under covariate shift. *ICML 2020 Workshop on Uncertainty and Robustness in Deep Learning*, 2020. 2

[11] A Tuan Nguyen, Thanh Nguyen-Tang, Ser-Nam Lim, and Philip HS Torr. Tipi: Test time adaptation with transformation invariance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24162–24171, 2023. 2

[12] 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. 2

[13] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 2

[14] 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. 1

[15] Moamar Sayed-Mouchaweh and Edwin Lughofer. *Learning in Non-Stationary Environments: Methods and Applications*. Springer Science & Business Media, 2012. 1

[16] 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. 2

[17] 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, 2, 6

[18] 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. 2

[19] 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, 2, 5, 6

[20] Hao Wen, Yuanchun Li, Zunshuai Zhang, Shiqi Jiang, Xiaozhou Ye, Ye Ouyang, Yaqin Zhang, and Yunxin Liu. Adaptivenet: Post-deployment neural architecture adaptation for diverse edge environments. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, pages 1–17, 2023. 2

[21] Longhui Yuan, Binhui Xie, and Shuang Li. Robust test-time adaptation in dynamic scenarios. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15922–15932, 2023. 2, 3, 5

[22] Daqing Zhang, Dan Wu, Kai Niu, Xuanzhi Wang, Fusang Zhang, Jian Yao, Dajie Jiang, and Fei Qin. Practical Issues and Challenges in CSI-based Integrated Sensing and Communication. *arXiv preprint arXiv:2204.03535*, 2022. 6

[23] Tong Zhu, Leida Li, Jufeng Yang, Sicheng Zhao, Hantao Liu, and Jiansheng Qian. Multimodal sentiment analysis with image-text interaction network. *IEEE Transactions on Multimedia*, 2022. 2