Unsupervised Test-Time Adaptation of Deep Neural Networks at the Edge: A Case Study

Kshitij Bhardwaj, James Diffenderfer, Bhavya Kailkhura, and Maya Gokhale
Lawrence Livermore National Laboratory, Livermore, CA
{bhardwaj2, diffenderfer2, kailkhura1, gokhale2}@llnl.gov

***Abstract*—Deep learning is being increasingly used in mobile and edge autonomous systems. The prediction accuracy of deep neural networks (DNNs), however, can degrade after deployment due to encountering data samples whose distributions are different than the training samples. To continue to robustly predict, DNNs must be able to adapt themselves post-deployment. Such adaptation at the edge is challenging as new labeled data may not be available, and it has to be performed on a resource-constrained device. This paper performs a case study to evaluate the cost of test-time fully unsupervised adaptation strategies on a real-world edge platform: Nvidia Jetson Xavier NX. In particular, we adapt pretrained state-of-the-art robust DNNs (trained using data augmentation) to improve the accuracy on image classification data that contains various image corruptions. During this prediction-time on-device adaptation, the model parameters of a DNN are updated using a single backpropagation pass while optimizing entropy loss. The effects of following three simple model updates are compared in terms of accuracy, adaptation time and energy: updating only convolutional (Conv-Tune); only fully-connected (FC-Tune); and only batch-norm parameters (BN-Tune). Our study shows that BN-Tune and Conv-Tune are more effective than FC-Tune in terms of improving accuracy for corrupted images data (average of 6.6%, 4.97%, and 4.02%, respectively over no adaptation). However, FC-Tune leads to significantly faster and more energy efficient solution with a small loss in accuracy. Even when using FC-Tune, the extra overheads of on-device fine-tuning are significant to meet tight real-time deadlines (209ms). This study motivates the need for designing hardware-aware robust algorithms for efficient on-device adaptation at the autonomous edge.**

***Index Terms*—Robust deep learning, on-device neural network adaptation, unsupervised adaptation, edge devices.**

I. Introduction

Deep learning has been increasingly utilized on edge devices for a variety of autonomous applications [1]. These applications vary from smart phones using deep neural networks (DNNs) for image classification and speech recognition to unmanned aerial vehicles using DNNs for path planning and object detection. While DNNs have proven to be highly effective for these applications, the networks are still prone to degradation of prediction accuracy post-deployment during the real world operation.

There can be several reasons why the accuracy of the DNNs may suffer after deployment on the autonomous edge devices. The DNNs may encounter input samples that are different from the ones that were used for training (called *dataset shifts* [2].) The new data may also be noisy due to sensor or environmental (such as weather) noises. While adversarial training techniques have been used to build DNNs robust to distribution shifts and noise [3], [4], these approaches may not be able to completely handle all kinds of real-time data changes. Therefore, neural network adaptation strategies have been introduced to improve prediction accuracy of DNNs [5]–[7].

A challenge associated with post-deployment neural network adaptation is the lack of labels for the new incoming and possibly domain-shifted data. For example, the devices operating in remote places without human intervention: human action recognition on military drones without labeled data [8] or performing laser-induced breakdown spectroscopy on Mars [9]. Additionally, the cost of labelling the new data may be too high and not feasible in real time [10].

For robust post-deployment DNN operation, the DNNs must be adapted based on the newly seen, possibly domain shifted, data. This adaptation is challenging as: (i) new labeled data may not be available; (ii) the adaptation must be on device as the device may not have connection to the cloud (e.g., in military zones) or may want to avoid sending data to the cloud to eliminate the extra transmission latency or for security purposes; (iii) in order to meet the tight deadlines, the on-device adaptation should be fast; and (iv) as these devices are typically resource-constrained and battery operated, the adaptation approach should be memory- and energy-efficient.

Contributions. In this paper, we perform a measurement case study to evaluate the effectiveness of unsupervised adaptation algorithms on a real-world edge device. In particular the study considers: (i) three robust DNNs, pretrained using data augmentation techniques for CIFAR-10 image classification data: ResNeXt, Wide-ResNet, and ResNet-18; (ii) three test-time unsupervised adaptation approaches that modify different DNN parameters using a single training (backpropagation) pass during prediction time: re-tuning parameters of only convolutional layers (Conv-Tune), re-tuning only fully-connected layer parameters (FC-Tune); and re-tuning only batch-norm parameters (BN-Tune). We also consider no adaptation as the baseline; (iii) Nvidia Jetson Xavier NX platform (with embedded Volta GPU) that runs the DNN models and the adaptation approaches; and (iv) three cost metrics: prediction accuracy (on the CIFAR-10-C dataset with image corruptions), inference and any adaptation time, and energy consumption. Based on this comprehensive evaluation, we find the overall optimal robust DNN and adaptation algorithm that equally optimize the three metrics.

The following are the main findings of our study: **(i)**

978-3-9819263-6-1/DATE22/©2022 EDAA

adaptation is required to improve the prediction accuracy, even though the models are trained offline using robust training approaches. BN-Tune shows the lowest prediction error for CIFAR-10-C, followed by Conv-Tune and FC-Tune (average improvements of 6.6%, 4.97%, and 4.02%, respectively over no adaptation); **(ii)** tuning the fully-connected layers is a more effective adaptation technique in terms of performance and energy, compared to BN-Tune and Conv-Tune as the number of FC parameters in a DNN model are considerably smaller than the other two parameters, leading to more efficient adaptation; **(iii)** FC-Tune using Wide-ResNet and ResNet-18 outperforms ResNeXt in terms of adaptation latency and energy as the former include much lower number of FC parameters than the latter. ResNeXt also results in out of memory issues due to it's complex and memory-intensive model; **(iv)** the use of the embedded GPU of Xavier NX shows considerable improvements in latency and energy over CPU; and **(v)** overall, FC-Tune with Wide-ResNet and running on NX GPU is the best choice in terms of equally optimizing the three cost metrics. It achieves on average 63.07% lower latency and 64.9% lower energy compared to the other two adaptation algorithms for Wide-ResNet (with an average 1.9% loss of accuracy). While this configuration is able to reduce the prediction error by 3.04% compared to no adaptation with the same network, it's performance and energy overheads are significant (209ms, 1.90J) for real-time operation on a resource-constrained device. Our study motivates the need for designing hardware-aware robust DNN models and adaptation algorithms that execute efficiently on resource-constrained edge devices.

II. Background: Improving DNN robustness during offline training

This section provides an overview of the offline training techniques used to improve robustness of DNNs. There are two widely used methods: 1) data augmentation, and 2) adversarial training.

A. Data augmentation

The data augmentation approach trains a neural network using both the "clean" original samples (e.g., images in CIFAR-10) and with additional noisy versions of these original samples [11]. This technique improves the generalization accuracy of the DNNs. Some of the common examples of data augmentation include: *Cutout,* where a portion of an image is deleted [12], and *Cutmix* which replaces a part of an image with another image [13]. Recently, *Augmix* data augmentation method is shown to achieve greater robustness [11]. It applies a series of different augmentations (e.g., rotate, posterize, etc.) to an image, followed by mixing the modified images into a new image. Such samples are not only different from the original but are also realistic and capture various common noises. Recently, *FourierMix* [14] extended Augmix to support spectral augmentations as well. Augmix is used in this study due to its strong performance.

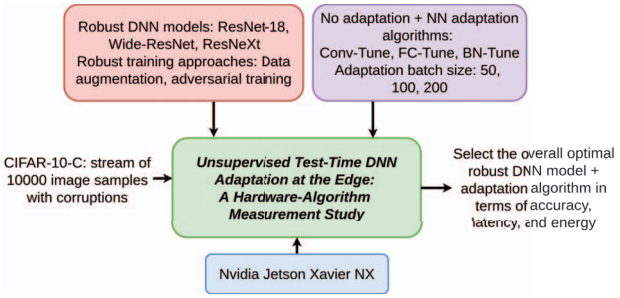


Fig. 1: Unsupervised test-time adaptation measurement study

B. Adversarial training

Adversarial training is used to make a model robust to adversarial examples. It requires solving a min-max optimization problem, where: (i) adversarial image samples are generated from the clean samples by finding imperceptible perturbations to the clean images such that the prediction loss of a DNN is maximized, and (ii) the prediction loss for these adversarial samples is minimized. In this study, we use a state-of-the-art adversarial training approach where imperceptibility is defined using learned perceptual image patch similarity (LPIPS) distance [4]. LPIPS distance is based on the activations of a DNN evaluated on two different images and is more suitable to common image corruptions. However, this problem is challenging to solve since LPIPS distance is defined by a neural network, and the projection onto the LPIPS-ball—as required when using Projected Gradient Descent (PGD) to solve the adversarial training does not admit a closed-form expression. To overcome this challenge, the authors in [4] developed an efficient approach using a single-iteration adversarial attack similar to fast gradient method (FGM) referred to as relaxed LPIPS adversarial training (RLAT).

III. DNN Adaptation at the Edge: A Case Study

As shown in Figure 1, we evaluate the effectiveness and cost of adapting robust DNNs on edge devices. Three robust DNNs are considered that are trained using data augmentation and adversarial training methods for CIFAR-10 image classification data. These DNNs are then deployed on Nvidia Jetson Xavier NX device. The input to the DNNs is unlabeled CIFAR-10-C data, with different types of image corruptions, which is used by the adaptation approaches to re-tune the various parameters (convolutional, fully-connected, batch-norm) of the DNNs. We measure the non-adapted and adapted prediction accuracy of the DNNs, as well as the performance and energy cost of this unsupervised on-device adaptation. We perform detailed analysis of the results, followed by selecting the best robust DNN model and adaptation approach that equally optimize the three cost metrics. The rest of the section describes the various components of this study.

A. Robust DNN models

We consider three variations of ResNet style architectures that have been demonstrated to yield robust models: ResNet, Wide-ResNet, and ResNeXt. These models are among the

top models on the robustbench leaderboard [15]. Robustbench systematically tracks the progress made in adversarial robust machine learning and keeps a record of the DNN models that perform the best in terms of prediction accuracy for corrupted/noisy datasets (however, it does not include adaptation approaches).

Each of these model architectures is composed primarily of groups of residual blocks followed by pooling and a fully-connected layer. Additionally, there is an initial convolution layer used to increase the number of channels before the first residual block. Hence, these architectures are typically described by specifying the number of groups and the number of blocks within each group. For models trained on CIFAR-10, the ResNet and Wide-ResNet architectures use residual blocks, where w is the network width multiplier. In the ResNet architecture, each block has a width multiplier of $w = 1$. The residual blocks in the ResNeXt architecture are designed to split inputs across k different paths, where k is called the network cardinality. A batch-norm layer follows each convolution layer in both block structures. The convolutional layers within each block denote filter size and number of output channels (C). Transformations are performed in skip-connection layers (dashed) within blocks if block input and output dimensions do not match. We now outline the specific models considered as part of the DSE.

1) ResNet-18: ResNet-18 has 4 groups with 2 residual blocks within each group. To improve robustness, ResNet-18 is trained using both Augmix data augmentation technique for CIFAR-10 dataset (Section II-A, similar to [11]) as well as using LPIPS-based adversarial training (Section II-B, similar to [4]). This trained ResNet-18 has 0.56 Giga multiply-accumulate (GMAC) operations and 11.17M total parameters, out of which there are 11159232 convolutional, 5130 fully-connected, and 7808 batch-norm parameters. Overall it takes 86MB memory.

2) Wide-ResNet: We leverage Wide-ResNet-40-2 which has 3 groups of 6 residual blocks and a width of $w = 2$. Wide-ResNet-40-2 is trained using Augmix data augmentation technique for CIFAR-10 (Section II-A, similar to [11]). This model has 0.33 GMAC operations, 2.24M total parameters (2236848 convolutional, 1290 fully-connected, and 5408 batch-norm parameters), and takes 9MB memory.

3) ResNeXt: We also use ResNeXt29_32x4d which has 3 groups of 3 residual blocks, a cardinality of $k = 4$, and a base width of 32. ResNeXt is also trained using Augmix data augmentation for CIFAR-10 (Section II-A, similar to [11]). This model has 1.08 GMAC operations, 6.81M total parameters (6772416 convolutional, 10250 fully-connected, and 25216 batch-norm parameters), and takes 26MB memory.

B. CIFAR-10-C data

The CIFAR-10-C dataset is used for testing the above three models that were pretrained using CIFAR-10 dataset. CIFAR-10-C includes 15 types of corruptions, for example, Gaussian noise, snow, fog, brightness, pixelate noise, etc. It also consists of 5 different severity levels for each corruption type (5 being the most severe and 1 being the least). In our study, all 15 corruptions are used with level 5 only.

The test data comprises streaming 10000 unlabeled CIFAR-10-C image samples for each corruption type. To perform real-time online adaptation during test time, batches of recently seen image samples are used by the adaptation algorithms. Three different batch sizes are used that have a direct impact on the adaptation effectiveness, performance, energy, as well as memory consumption during adaptation: 50, 100, and 200.

C. Adaptation approaches

In this study, we use a no adaptation and three DNN adaptation algorithms on the CIFAR-10-C test data: *(i) no adaptation,* where the three pre-trained robust DNNs are used with no modifications; *(ii) Conv-Tune adaptation:* re-tunes all the convolutional parameters of the models by running a single backpropagation pass during adaptation using a collected batch of recently seen unlabeled image samples. During this backpropagation, the convolutional parameters are optimized for entropy loss function using the Adam optimizer. Shannon Entropy for a prediction y can be computed without the need for labeled data, defined as: $H(y) = -\sum_c p(y_c) \log p(y_c)$ for probability of y for class c [5]; *(iii) FC-Tune adaptation:* re-tunes only the fully-connected layer parameters using the same entropy-based single backpropagation pass during on-device adaptation; and *(iv) BN-Tune adaptation:* re-tunes only the batch-norm parameters similarly.

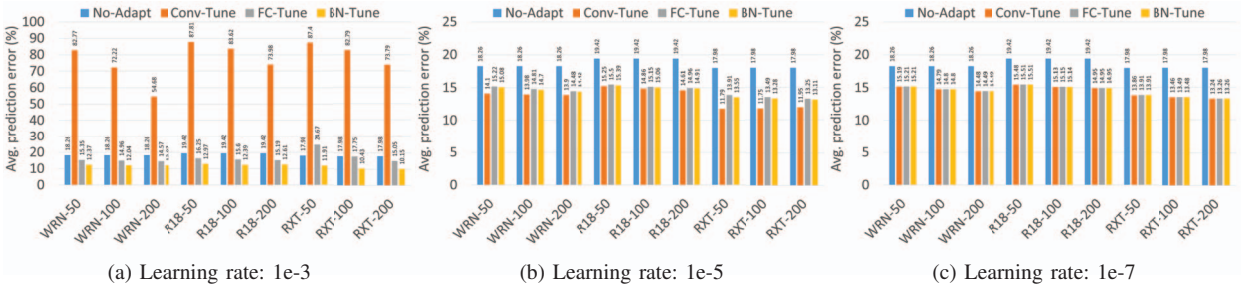
All algorithms are implemented using Pytorch [16]. During test time with CIFAR-10-C, the no adaptation algorithm keeps the model in *eval()* mode of Pytorch (i.e., inference only), while the other three adaptation approaches require the model to be in *train()* mode (performing re-training and inference). The *forward* passes for Conv-Tune, FC-Tune, and BN-Tune are modified to adapt the DNN: at each online adaptation point, these algorithms first perform inference followed by updating (using a single backpropagation pass) the target parameters based on the collected recently seen batch of data. Since the focus of this paper is on adaptation (or retraining) on edge devices, edge-optimized frameworks for fast inference, such as Tensorflow-Lite [17] and TensorRT [18], are not applicable and therefore not used.

D. Nvidia Jetson Xavier NX device

NX uses a 6-core Nvidia Carmel ARM SoC (128 KB L1 instruction cache + 64 KB L1 data cache, and 6MB L2 cache and 4MB L3 cache) that can run up to 1.9GHz speed. It also integrates a 384-core Volta GPU (maximum frequency of 1.1GHz) and 8GB LPDDR4 memory. NX uses Jetpack 4.4 software development kit with Linux4Tegra operating system (similar to Ubuntu 18.04) and comes with CUDA 10.2 and GPU NN acceleration libraries (cuDNN 8.0). Pytorch-1.9.0 is installed using Nvidia's pre-built installer for Jetson devices.

E. Objective functions

This study performs a multi-objective evaluation. Three objective functions are considered: overall prediction accuracy



(a) Learning rate: 1e-3 (b) Learning rate: 1e-5 (c) Learning rate: 1e-7

Fig. 2: While Conv-Tune and FC-Tune perform better when using learning rate of 1e-5, BN-Tune shows the lowest average error for learning rate of 1e-3. No-Adapt is not affected by learning rate.

for the entire CIFAR-10-C test stream, average forward time per batch (inference + any adaptation), and average energy consumption for inference and any adaptation per batch. Accuracy and performance are computed using Pytorch. To compute energy, we measure power per batch using the Kuman wall outlet power meter which connects to the device. The optimal robust DNN model and adaptation algorithm are selected that equally optimize the three cost metrics. In particular, we combine the three metrics using $0.33*time+0.33*energy+0.33*pred\_error$ as the single objective function to minimize. The three objectives are normalized before combining them.

IV. Results

This section first presents the prediction accuracy results for the different robust DNNs and adaptation approaches, followed by their detailed performance, energy, and multi-objective cost tradeoff analysis for the Xavier NX device.

A. Prediction accuracy

Figure 2 (a-c) shows the prediction error for the four algorithms with a varying learning rate: no adaptation (No-Adapt), Conv-Tune, FC-Tune, and BN-Tune, for a stream of 10000 unlabeled CIFAR-10-C image samples. The error is averaged across the 15 noise types (10000 samples per noise). For each noise, three test-time batch sizes are used: 50, 100, and 200. The adaptation is online, similar to [5], where inference and adaptation are performed during forward pass for the entire test stream batch by batch (the reported prediction errors are averaged across the batches for the stream). Accuracy for the three robust DNNs are shown: ResNeXt with AugMix [11] (RXT), Wide-ResNet with AugMix [11] (WRN), and ResNet-18 with both AugMix [11] and adversarial training [4] (R18). For each network, three learning rates are considered for backpropagation-based adaptation while using the Adam optimizer: 1e-3, 1e-5, and 1e-7.

For both Conv-Tune and FC-Tune, the learning rate of 1e-5 leads to better adaptation accuracy compared to the other two learning rates. For BN-Tune, learning rate of 1e-3 is the most effective. No-Adapt is not affected by learning rate as there is no re-tuning (or re-learning) of the DNN parameters. Compared to No-Adapt, FC-Tune (learning rate: 1e-5) shows 4.02% lower prediction error on average. Conv-Tune (learning rate: 1e-5) achieves 4.97% lower error on average. BN-Tune (learning rate: 1e-3) performs the best with 6.67% lower average error than No-Adapt. These results show that offline robust training techniques, such as AugMix and adversarial training, are not enough, and online adaptation is required for effective prediction in the wild. Additionally, for all three adaptation approaches, while prediction error shows considerable improvement when using 100 as batch size compared to 50, using a batch size of 200 shows almost similar error compared to 100. Moreover, bigger batch sizes (such as 200) for online adaptation may not be reasonable for memory-constrained devices. Overall, we find that RXT-200 achieves the lowest prediction errors for the different approaches, however, as we see in the next section, it is not an efficient model for on-device adaptation.

B. Performance, energy, and multi-objective analysis

This section presents the forward time and energy results of running the various robust DNNs and adaptation approaches on Xavier NX, followed by accuracy-performance-energy tradeoff analysis. The goal of this study is to evaluate if convolutional, fully-connected, and batch-norm based adaptation can be accelerated using embedded GPUs. NX includes 6-core Nvidia Carmel Arm cluster, a 384-core Volta GPU, and 8GB memory. For the CPU results, Pytorch is run multi-threaded (with intra-operation parallelism) on the Arm cores, and for GPU, Pytorch CUDA is used.

Performance analysis. Figure 3 shows the performance of No-Adapt, FC-Tune, BN-Tune, and Conv-Tune for all 9 cases using the CPU and the GPU. Interestingly, RXT-200 with BN-Tune and Conv-Tune runs out of memory when executed on GPU. To perform backpropagation, Pytorch creates a dynamic computational graph during forward pass that includes the nodes that are enabled for gradient computation (conv, BN, etc.). This graph created for RXT-AM-200 and BN-Tune takes up around 5.1GB and for Conv-Tune 5.4GB (when profiled on CPU). But the use of GPU leads to higher use of memory during execution because of spawning of several memory-intensive CUDA threads [19] which causes the out of memory issues. Even for RXT-100 with Conv-Tune, memory becomes a bottleneck for GPU, causing slow down. Beyond these exceptions, GPU yields significant speedup over CPU (on average): for No-Adapt 90.5%, for FC-Tune 68.5%, for BN-Tune 79.21%, and for Conv-Tune 66.98%. The average

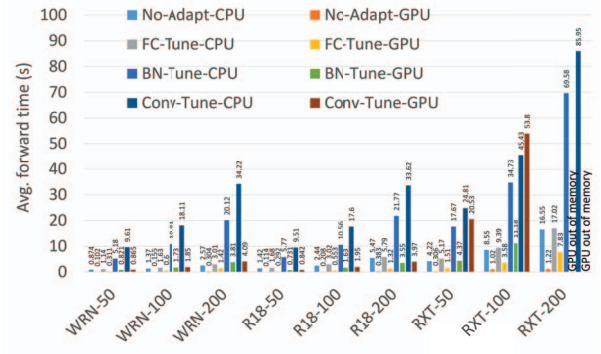


Fig. 3: Xavier NX forward times (inference + any adaptation)

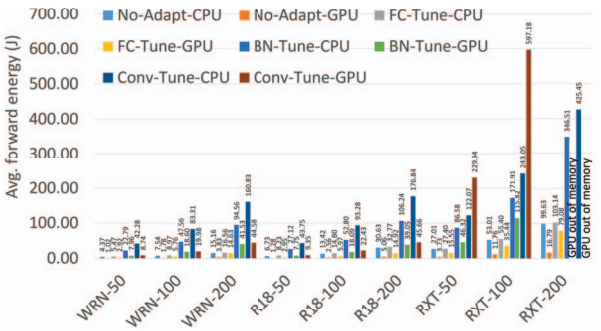


Fig. 4: Xavier NX forward energy (inference + any adaptation)

overhead of FC-Tune-GPU over No-Adapt-GPU is 1.51s. This overhead is much higher for BN-Tune-GPU (3.15s) and Conv-Tune-GPU (10.66s). RXT incurs the most overheads as it involves many more number of MAC operations than the other two models (1.08 GMACs vs. 0.56/0.33 GMACs). Also, since the number of BN and Conv parameters of the three models are much greater than the FC parameters, the adaptation overhead of these algorithms is more than FC-Tune. Overall, the lowest adaptation overhead over No-Adapt-GPU is shown by R18-50 using FC-Tune-GPU (178ms), closely followed by WRN-50 with FC-Tune-GPU (209ms).

Energy analysis. Figure 4 shows the energy consumption results during the forward pass (inference + any adaptation) of the four adaptation approaches and the robust DNNs, for both CPU and GPU. Compared to CPU, the use of GPU shows 80.03%, 41.65%, 57.69%, and 27.79% lower energy for No-Adapt, FC-Tune, BN-Tune, and Conv-Tune, respectively. The GPU's significantly faster execution time recoups the added power costs, leading to overall more energy efficiency, except for RXT-100/200 for Conv-Tune where GPU was not able to achieve speedups. In addition, compared to No-Adapt-GPU, the following extra overheads were incurred by the various adaptation algorithms on average (running on GPU): FC-Tune: 14.38J, BN-Tune: 33.01J, Conv-Tune: 118.35J. R18-50 with FC-Tune shows the lowest energy overhead (1.67J), closely followed by WRN-50 (1.90J).

Performance-energy-accuracy trade-offs. Figure 5 shows the overall results for the three cost metrics (excluding those

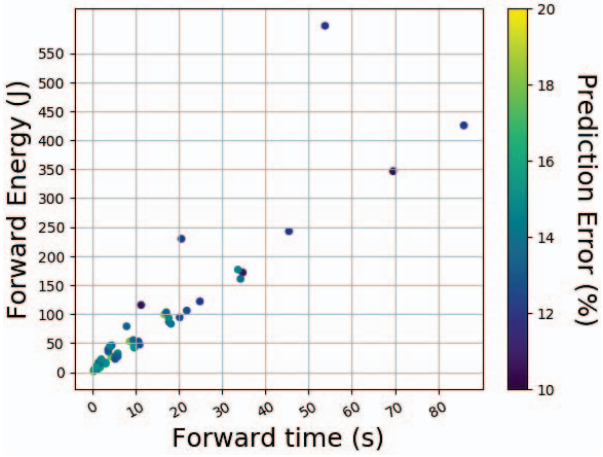


Fig. 5: Multi-objective results: plotting all configurations

Conv-Tune-GPU and BN-Tune-GPU points that run out of memory). For efficient on-device adaptation, in addition to accuracy, performance and energy are also critical. Among the points shown in this figure, if we consider strict constraints of {1s forward time, 5J forward energy, and 18% prediction error}, we find three interesting points: (i) No-Adapt-GPU, RXT-50 with 0.306s forward time, 3.73J forward energy, and 17.98% error; (ii) FC-Tune-GPU, WRN-50 with 0.311s forward time, 2.92J forward energy; and 15.22% error; and (iii) FC-Tune-GPU, R18-50 with 0.292s forward time, 2.94J forward energy, and 15.5% error. Among these points, (ii) with FC-Tune-GPU and WRN-50 is an overall effective solution for a resource-constrained edge device as it equally optimizes the three costs (i.e. minimizes our target objective function: $0.33 * time + 0.33 * energy + 0.33 * pred\_error$). Comparing WRN-50 and FC-Tune-GPU with BN-Tune-GPU and Conv-Tune-GPU for the same network and batch size shows on average 63.07% lower forward time and 64.9% lower forward energy, respectively with an average accuracy loss of 1.9% accuracy.

While FC-Tune with WRN-50 is able to reduce the prediction error by 3.04% over No-Adapt for the same network and batch size, it's performance and energy overheads over No-Adapt (both using GPU) are substantial for real-time operation and meeting tight deadlines: 209ms and 1.90J, respectively. Additionally, note that even though BN-Tune with RXT-200 showed the best accuracy, it is too expensive to run on Xavier NX due to the complicated structure of RXT and a large number of BN parameters (25216). Therefore, there is a need for designing hardware-aware robust DNN models and unsupervised adaptation techniques for efficient and effective on-device adaptation at the resource-constrained edge.

V. Hardware-aware unsupervised adaptation: An Example

In this section, we present an example of a hardware-aware unsupervised adaptation algorithm that is suitable for low-power edge devices. While adaptation techniques studied

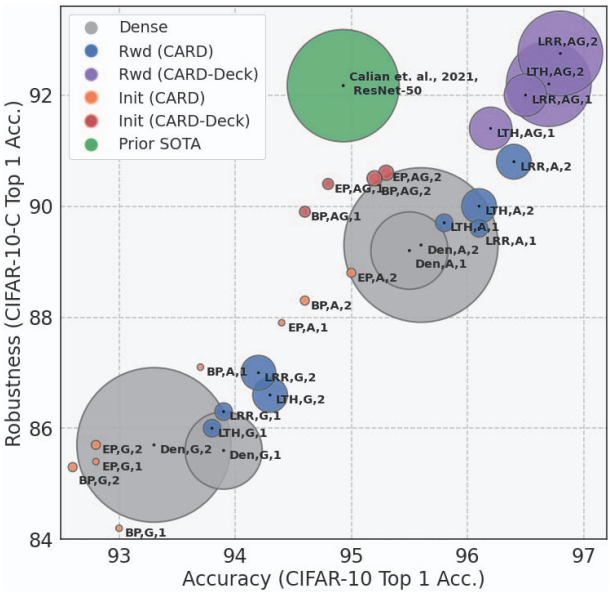


Fig. 6: CARDs and CARD-Deck accuracy for CIFAR-10 and CIFAR-10-C, and memory usage (circle area). Annotation provides {Compression Method, Data Augmentation, ResNet-18 Width} with A for AugMix, G for Gaussian, and AG for both (used by CARD-Decks).

in this paper rely on on-device retraining (via backpropagation), our group also recently introduced an inference-only adaptation method after first demonstrating that some model compression techniques, such as pruning and binarization, can produce compact, accurate, and robust deep neural networks (CARDs) [7]. By training CARDs offline with data augmentation methods for improving robustness (e.g., AugMix), domain-adaptive ensembles consisting of CARDs, called CARD-Decks, adapt to data at inference time using a spectral similarity metric to select a subset of CARDs in the CARD-Deck for which the training data was most similar to the test data at inference. Accuracy, robustness, and memory usage of CARDs and CARD-Decks and dense models (utilizing the same architecture) are illustrated in Figure 6. CARDs and CARD-Deck ensembles show comparable accuracy and robustness compared to dense models while significantly reducing the memory usage. Future work will deploy CARD-Decks on edge devices to measure and compare their adaptation time and energy with the other retraining approaches.

VI. Conclusion

This paper performed a study of test-time unsupervised DNN adaptation techniques to quantify their latency and energy on Nvidia's Xavier NX. The study found the optimal adaptation approach (between on-device re-tuning of Conv, FC, and BN parameters) and the type of robust DNN (among ResNeXt, Wide-ResNet, and ResNet-18) in terms of different cost metrics. Overall, we found Wide-ResNet with FC-tune, running on Xavier NX's GPU, to be highly effective at equally optimizing accuracy, performance, and energy costs. However, the adaptation overhead is still expensive for meeting real-time deadlines. Hardware-aware design of robust algorithms is required to achieve efficient on-device DNN adaptation.

Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

References

[1] Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proc. IEEE*, 107(8):1655–1674, 2019.
[2] Saikiran Bulusu, Bhavya Kailkhura, Bo Li, Pramod K Varshney, and Dawn Song. Anomalous example detection in deep learning: A survey. *IEEE Access*, 8:132330–132347, 2020.
[3] Kaidi Xu, Chenan Wang, Hao Cheng, Bhavya Kailkhura, Xue Lin, and Ryan Goldhahn. Mixture of robust experts (more): A robust denoising method towards multiple perturbations. *arXiv preprint arXiv:2104.10586*, 2021.
[4] Klim Kireev, Maksym Andriushchenko, and Nicolas Flammarion. On the effectiveness of adversarial training against common corruptions. *arXiv preprint arXiv:2103.02325*, 2021.
[5] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *ICLR Spotlight*, 2021.
[6] Steffen Schneider, Evgenia Rusak, Luisa Eck, Oliver Bringmann, Wieland Brendel, and Matthias Bethge. Improving robustness against common corruptions by covariate shift adaptation. *NeurIPS*, 33, 2020.
[7] James Diffenderfer, Brian R Bartoldson, Shreya Chaganti, Jize Zhang, and Bhavya Kailkhura. A winning hand: Compressing deep networks can improve out-of-distribution robustness. *NeurIPS*, 2021.
[8] Jinwoo Choi, Gaurav Sharma, Manmohan Chandraker, and Jia-Bin Huang. Unsupervised and semi-supervised domain adaptation for action recognition from drones. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1717–1726, 2020.
[9] Kshitij Bhardwaj and Maya Gokhale. Semi-supervised on-device neural network adaptation for remote and portable laser-induced breakdown spectroscopy. *On-device Intelligence Workshop (MLSys Conference)*, 2021.
[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.
[11] Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. *ICLR*, 2020.
[12] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.
[13] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019.
[14] Jiachen Sun, Akshay Mehra, Bhavya Kailkhura, Pin-Yu Chen, Dan Hendrycks, Jihun Hamm, and Z Morley Mao. Certified adversarial defenses meet out-of-distribution corruptions: Benchmarking robustness and simple baselines. *arXiv preprint arXiv:2112.00659*, 2021.
[15] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. *arXiv preprint arXiv:2010.09670*, 2020.
[16] Pytorch. Pytorch. http://www.pytorch.org/.
[17] TensorFlow. TensorFlow Lite. http://www.tensorflow.org/lite.
[18] Nvidia. Nvidia TensorRT. http://developer.nvidia.com/tensorrt.
[19] Jake Choi, Heon Young Yeom, and Yoonhee Kim. Implementing cuda unified memory in the pytorch framework. In *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pages 20–25. IEEE, 2021.