Imbal-OL: Online Machine Learning from Imbalanced Data Streams in Real-world IoT

Conference Paper · December 2021

DOI: 10.1109/BigData52589.2021.9671765

3 authors:

Bharath Sudharsan

29 PUBLICATIONS **196** CITATIONS

SEE PROFILE

John G Breslin

National University of Ireland, Galway

403 PUBLICATIONS **6,557** CITATIONS

SEE PROFILE

Muhammad Intizar Ali

Dublin City University

100 PUBLICATIONS **1,524** CITATIONS

SEE PROFILE

Imbal-OL: Online Machine Learning from Imbalanced Data Streams in Real-world IoT

Bharath Sudharsan*, John G. Breslin*, and Muhammad Intizar Ali§
*Confirm SFI Research Centre for Smart Manufacturing, Data Science Institute, NUI Galway, Ireland
{bharath.sudharsan, john.breslin}@insight-centre.org
§School of Electronic Engineering, Dublin City University, Ireland
ali.intizar@dcu.ie

***Abstract*—Typically a Neural Networks (NN) is trained on data centers using historic datasets, then a C source file (model as a char array) of the trained model is generated and flashed on IoT devices. This standard process impedes the flexibility of billions of deployed ML-powered devices as they cannot learn unseen/fresh data patterns (static intelligence) and are impossible to adapt to dynamic scenarios. Currently, to address this issue, Online Machine Learning (OL) algorithms are deployed on IoT devices that provide devices the ability to locally re-train themselves - continuously updating the last few NN layers using unseen data patterns encountered after deployment.**

In OL, catastrophic forgetting is common when NNs are trained using non-stationary data distribution. The majority of recent work in the OL domain embraces the implicit assumption that the distribution of local training data is balanced. But the fact is, the sensor data streams in real-world IoT are severely imbalanced and temporally correlated. This paper introduces Imbal-OL, a resource-friendly technique that can be used as an OL plugin to balance the size of classes in a range of data streams. When Imbal-OL processed stream is used for OL, the models can adapt faster to changes in the stream while parallelly preventing catastrophic forgetting. Experimental evaluation of Imbal-OL using CIFAR datasets over ResNet-18 demonstrates its ability to deal with imperfect data streams, as it manages to produce high-quality models even under challenging learning settings.

***Index Terms*—IoT Devices, TinyML, Online Learning, Imbalanced Data, Class Balancing.**

I. Introduction

The top-quality NN models that solve a range of complex tasks are typically trained at data centers by performing multiple passes over labeled historic datasets collected for decades. The knowledge gathered by ubiquitous devices in IoT is a vastly different story as, over the service span of devices, they perceive a data stream of unlabeled, temporally correlated observations and rarely revisit the same data multiple times [1]. Learning new knowledge needs to occur in the real world after deployment, rather than training at the data center then inference at the edge [2]. So, to enable on-the-fly knowledge gaining, the trend in IoT is moving towards Online Machine Learning (OL), where the last few network layers are continuously re-trained/updated at the edge level. In this context, non NN based TinyML algorithms also exist for on-device continual learning after deployment [3, 4]. In OL settings, NNs perform poorly when fed with streams containing non-stationary data distribution. Also, when learning to solve a sequence of distinct tasks, NNs learn from the data streams of the current task but forget previously learned ones [5] - this phenomenon is catastrophic forgetting.

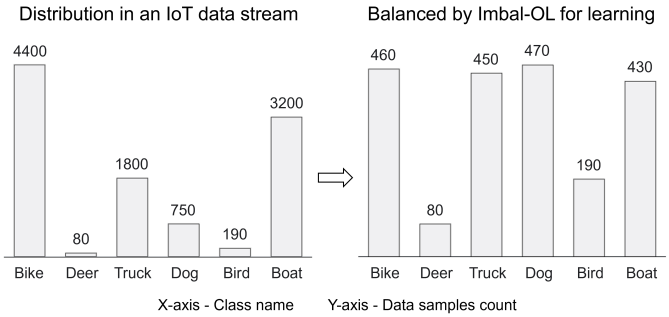

Fig. 1. Without discarding any data samples from the significantly underrepresented deer and bird classes, Imbal-OL balances the remaining classes.

Two distinct categories exist to overcome catastrophic forgetting. First is incremental approaches, where the learning algorithm is given all the data for the current task over which numerous passes are performed. The next is online approaches, where the learning algorithm is provided with tiny batches of data from the stream and cannot revisit batches from the current or previous tasks. This work focuses on the second category because the memory constraint nature of IoT edge devices restricts the accommodation of a large volume of data samples from the stream in its RAM, over which numerous passes can be performed. In incremental approaches, there is an option to store training data in external file systems, then perform passes. This option can lead to data compromises as devices can have poor configurations and open design when built with cost as the driving design tenet.

In OL methods, there exists a gap as the majority of studies [3, 4, 6] do not consider the data balance both in terms of sizes of the tasks to learn and the classes contained in each task. Moreover, the data streams used for training are assumed to be well-balanced. In contrast, the devices that aim to improve their intelligence during their service time need to learn from significantly imbalanced data streams with no boundaries and task identifiers/allocators. To perform OL in such non-ideal real-world settings, this paper presents Imbal-OL, an OL plugin that understands the supplied data stream and balances the class size before sending it for learning. Imbal-OL does not require prior knowledge about the incoming stream and its data

distribution. The Imbal-OL design highlight is its practicality as its implementation spans only a few lines in C++ code, enabling its deployment on billions of tiny devices in IoT. When practicing on device OL using Imbal-OL as a plugin, it aims to not impair the service life of devices by draining their battery nor causing memory SRAM overflows.

II. Imbal-OL Design

This section presents Imbal-OL design to enable practicing OL by using real-world data streams in IoT.

A. Design Setup

In OL, a lengthy data stream is fed to the on-device learning algorithm. This stream can be modeled as a sequence of distinct tasks, with each task containing data representing a set of classes. In IoT, these streams contain non-stationary data distribution among classes for each task - one of the bottlenecks during learning. Here, in a time step t, each data sample is represented by input x_t with its label y_t. Typically, in such a learning setup, to ease the learning process, task identifiers are considered to redirect data to the correct task both during learning and prediction. But in our setting, to improve generalization for a range of streams while having only a minimal prerequisite: we do not use boundaries or task identifiers to hint after learning a task; do not attempt to gather prior knowledge (length, no of tasks, class composition, etc.) of the stream to learn from; the incoming data cannot be revisited in future time steps.

B. Imbal-OL Core Technique

The Imbal-OL core technique is the primary contribution of this work. Imbal-OL understands the supplied IoT stream and decides which data samples need to be used for learning. Imbal-OL stores an iid data sample from each class to maintain balance among the samples constituting the various classes. Thus Imbal-OL preserves the distribution of each class, parallelly altering the distribution of the IoT stream to mitigate class imbalances. The device used for learning with m as its memory size is considered full when data from the stream fills the memory unit. In IoT streams, it is common for a particular class to be the largest by containing the majority of data samples, making the size of other classes look inferior. Also, there can exist two or more classes to be the largest when they roughly contain the same amount of samples. Imbal-OL sets a class to be full if it is the largest in the current or previous time step, and the full status is retained also in the future time steps. Imbal-OL restricts the growth of such full classes as it increases the imbalance.

Algorithm 1 presents the pseudocode of Imbal-OL. Imbal-OL operates in two steps. In the first step (Lines 9 to 10), till the memory gets filled, data from the stream keeps accumulating in the device memory without any sampling. After filling the memory, step two (Lines 12 to 24) starts where Imbal-OL starts checking the labels y_i to find if the data belongs to the largest classes. If so, the data gets discarded. Else it is stored in the non-largest class to which it belongs. Fig. 1 illustrates the Imbal-OL concept. As shown on the left side of the figure, the supplied stream can have high imbalances in the size of its classes. When Imbal-OL is used as an OL plugin, without getting influenced by the stream distribution, it stores all data samples of the two smallest classes (deer, bird) and balances the data representing other larger classes (bike, truck, dog, boat). The Imbal-OL processed stream that is OL ready is shown on the right side of the same figure. When such a balanced subset of the IoT stream is used, the on-device OL algorithm will consider that data from all shown classes to be equally difficult, making it equally important to learn.

Algorithm 1 Generalized OL design flow with Imbal-OL as a plugin to handle imbalanced data streams in IoT.

1: **input:**
2: f ▷ model or algorithm of choice
3: s, n_s ▷ batch size, steps per batch
4: $\{x_i, y_i\}_{i=1}^{n}$ ▷ data stream in real-world IoT
5: $\ell(\text{predictions}, \text{true labels})$ ▷ loss function of choice
6: **output:** updated model as a result of on-device OL using Imbal-OL processed data stream
7: **Imbal-OL plugin** ▷ process stream to make it OL ready
8: **for** $i = 1$ to n **do** ▷ n is stream length
9: **if** device memory **not** filled **then**
10: store x_i with y_i ▷ store data samples, labels
11: **else**
12: **if** $c \equiv y_i$ is **not** a fully filled class **then**
13: find all data samples of largest classes
14: randomly select data samples from them
15: overwrite selected data sample with x_i, y_i
16: **else**
17: $m_c \leftarrow$ data samples count of class $c \equiv y_i$
18: $n_{class} \leftarrow$ count of class $c \equiv y_i$ seen thus far
19: sample $u \sim$ Uniform $(0, 1)$
20: **if** $u \leq m_c / n_{class}$ **then**
21: random pick data sample of class $c \equiv y_i$
22: replace it with x_i, y_i
23: **else**
24: drop x_i, y_i ▷ data not used for OL
25: **end if**
26: **end if**
27: **end if**
28: **end for**
29: **online machine learning** ▷ OL after balancing classes
30: $X_a, y_a \leftarrow$ receive balanced batch of size s
31: $\alpha \leftarrow 1/n_{class}$
32: **for** n_s steps **do**
33: $\hat{y}_a = f(X_a)$ ▷ predict outputs
34: $\mathcal{L}_1 = \ell(\hat{y}_a, y_a)$ ▷ compute stream loss
35: $X_b, y_b \leftarrow$ sampled batch of size s from memory
36: $\hat{y}_b = f(X_b)$ ▷ predict outputs
37: $\mathcal{L}_2 = \ell(\hat{y}_b, y_b)$ ▷ compute replay loss
38: $\mathcal{L} = \alpha \times \mathcal{L}_1 + (1 - \alpha) \times \mathcal{L}_2$ ▷ compute joint loss
39: update f according to $\mathcal{L}$
40: **end for**

C. Imbal-OL for Action

This subsection presents a generalized OL design flow with Imbal-OL - sketched in the same Algorithm 1 in Lines 29 to 40. This process aims to apply to a range of IoT streams, TinyML algorithms, pre-trained models, and loss functions. To enable adapting to non-stationary streams while parallelly preventing catastrophic forgetting, the on-device model updates are guided using a two-element loss. The first-element $\mathcal{L}_1$ is calculated concerning a data batch X_a with labels y_a of size s from Imbal-OL processed stream in the current time-step. Then the second-element $\mathcal{L}_2$ is calculated concerning X_b with labels y_b with same size s. The loss functions employed here compute the cross-entropy between the on-device model predictions and true labels from the stream - users can explore using other types of loss functions. The final loss is the convex combination of $\mathcal{L}_1$ and $\mathcal{L}_2$ and given as: $\mathcal{L} = \alpha \times \mathcal{L}_1 + (1 - \alpha) \times \mathcal{L}_2$. Here, the relative importance between loss elements $\mathcal{L}_1$ and $\mathcal{L}_2$ is controlled by $\alpha \in [0, 1]$. Also, α can maintain the balance between quickly learning from the IoT stream and preserving the learned knowledge.

Prevailing schemes treat loss elements as equally important or decrease α with the completed tasks count - can lead to performance drops. In the Imbal-OL design setting, the task boundaries are not considered (see Subsection II-A). So, to manage without task information, we set $\alpha = 1/n_{class}$, where n_{class} is the classes counted in the IoT stream the device has seen so far. IoT devices have limited storage, so the batches in previous learning rounds cannot be revisited as they won't be stored. But using the currently stored batch from the stream, the model can be updated multiple times. When the OL method performs one update per time step, this results in underfitting. On the far side, when performing n_s multiple updates, the on-device model can adapt or learn faster from the non-stationary IoT stream, but at the cost of extra strain on the hardware.

D. Imbal-OL Characteristics

Imbal-OL is designed to show the following characteristics:

Underrepresented Classes. In the input stream containing n_{class} classes, if there exists a class containing less than m/n_{class} data samples, then Imbal-OL stores all data representing this class for learning. This remarkable character of Imbal-OL guarantees the inclusion of all data samples from severely underrepresented classes. This character can be observed in Fig. 1 with the input stream of $n_{class} = 6$ and $m = 2100$. Here, the classes deer and bird contain samples less than m/n_{class}, so Imbal-OL stores the entire data representing those classes.

Sampling. A subset of data samples from each class stored in device memory for learning is iid concerning the data samples from the stream. As it cannot be assumed that the data samples in each class are presented in an iid fashion in future time steps, this characteristic of Imbal-OL captures a representative sample for each class data temporarily stored in device memory for learning.

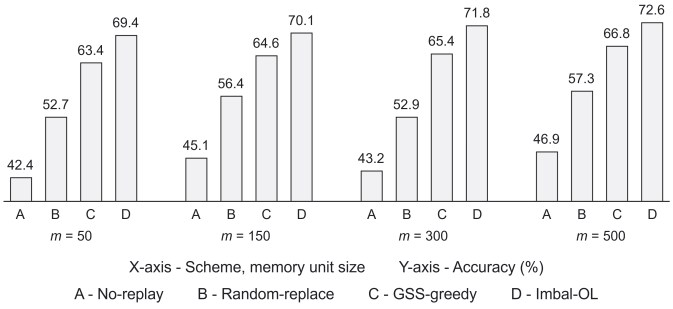

Fig. 2. Memory unit size vs accuracy of schemes over CIFAR-10.

Memory Utilization. There can exist cases where the available device storage cannot get fully used while simultaneously maintaining the balance among classes - this can be exemplified using Fig. 1. For $m = 2100$, and the smallest class in the stream contains only 80 data samples, balance can be achieved by keeping roughly 80 samples in each class. In this approach, only less than a quarter of the memory will get filled. To avoid such inefficient utilization of resources, Imbal-OL fills the entire memory, then balances the classes.

Weighted Replay. The most popular approach to mitigate the impact due to imbalance is to identify the minority classes over which oversampling is performed. Imbal-OL replays a data sample (shown again during learning) based on probability, which is inversely proportional to the size of classes from the stream. Thus, Imbal-OL makes data samples from minority classes have a higher probability for replay than the samples from larger classes. This is the weighted replay characteristic of Imbal-OL as it replays samples considering where they belong to rather than uniform replay where data samples from all classes have a similar probability for replay.

III. Imbal-OL Evaluation

This section evaluates ImbalOL. Initially, the evaluation setup is explained, followed by experiments and results.

A. Setup

Datasets and IoT Boards. The standard CIFAR-10 and CIFAR-100 datasets are selected for experiments using Imbal-OL. Each learning task is assigned with classes from the dataset in an incremental fashion. i.e., the first task is assigned with one class and subsequent tasks with more classes. This incremental approach creates distinct talks, inducing complexity during OL from the stream. Devices in IoT are heterogeneous, consisting of mid to high-end hardware specifications [7]. To replicate this scenario, the following IoT development boards are chosen: Google Coral Dev board, Intel Movidius NCS accelerated Raspberry Pi 4, and NVIDIA Jetson Nano. The results reported in this paper are the average values obtained when running experiments on these 3 boards.

Models. ResNet18 pre-trained on ImageNet is the network chosen to evaluate Imbal-OL using CIFAR datasets. Following are the hyperparameters: a learning rate of 0.02 is set as a result of grid search; the batch size s is set as 8 to reduce

TABLE I
ACCURACY (%) COMPARISON OF MODELS AFTER OL USING CIFAR-10.

Scheme	m = 100		m = 500	
	Uniform	Weighted	Uniform	Weighted
GSS-greedy	63.6	65.2	62.2	66.8
Imbal-OL	69.4	70.7	70.2	72.6

TABLE II
TIME, MEMORY CONSUMED BY SCHEMES TO MAKE STREAMS OL READY.

Scheme	Time (sec)		Memory (%)	
	CIFAR-10	CIFAR-100	CIFAR-10	CIFAR-100
Random-replace	4.17	4.9	3.7	3.4
GSS-greedy	24.3	23.6	31.8	32.6
Imbal-OL	8.18	9.06	4.6	5.1

strain and not memory overload the devices; steps per batch n_s is set as 2 as this can let the devices learn faster. The setup is implemented in TensorFLow since the TFLite version is well suited for edge GPUs and supports the used hardware accelerators.

Imbalances Simulation. For each class in the selected datasets, the retention factor is defined as the percentage of its data samples from the dataset that will be injected into the data stream for learning. A vector r is defined, that contains k retention factors $r = (r_1, r_2, \cdots, r_k)$. The retention factors are distributed to each class randomly, without repeating. r is started over for classes count larger than k - this is uncommon as IoT devices are not capable of large class counts. During evaluation, r is set as $(0.01, 0.03, 0.1, 0.3, 1)$. In this setup, the imbalance ratio between classes in the stream can range from 1:3 to 1:60. The selection of r decides the level of imbalances.

B. Experiments and Results

Learning on IoT boards is performed using Imbal-OL processed streams. For performance comparison, in this same setup, OL is performed also using two following schemes: No-replay - learning is performed by directly feeding the stream without any replay; Random-replace - samples data from the stream with a probability and stores it by randomly replacing data in the memory. The recent scheme named GSS-greedy [8] is included for comparison with Imbal-OL. The remainder of this subsection presents experiments and results in phases:

Memory Unit Size vs Performance. The initial phase of experiments aims to analyze the impact of memory unit size m on the performance of models learned on IoT boards using the CIFAR 10 dataset. m is varied from 50 to 500, and the results are given in Fig. 2. It can be observed that the models produced by training using Imbal-OL processed streams have: higher performance than the baselines; consistent performance across different m values; higher improvements for smaller m, making even devices with low memory to produce better quality models. Across all selected schemes including Imbal-OL, model accuracy increases with m. So, higher values of m can be explored when practicing OL on better resource IoT boards that fall under the edge GPU category.

Uniform vs Weighted Replay. This experimental phase aims to investigate the benefits of weighted-replay Imbal-OL characteristic (see subsection II-D). CIFAR-10 is used for learning, the accuracy metric for evaluation, and m is considered as 100 and 500. Results are given in Table I, from which observations can be made. The performance gap between Imbal-OL and GSS-greedy is higher under uniform replay. This is because, for GSS-greedy, the on-device learned model is highly influenced by the imbalances in the stream, causing the model to show top classification performance only for a few classes. So, it resulted in reduced accuracy when evaluating the learned model using the balanced test set. Using weighted replay has higher benefits than uniform replay because it partially masks the effect of imbalanced streams by oversampling the minority classes. Results for No-replay and Random-replace schemes are not presented due to their inferior performance (see Fig. 2) in comparison with GSS-greedy and Imbal-OL.

Time and Memory Consumption. Top hardware specifications cannot be expected from IoT devices. So, this experimental phase aims to investigate the computation efficiency of Imbal-OL using results given in Table II. Here the time consumption represents the elapsed time to process the stream for making it OL ready. Random-replace and Imbal-OL consume roughly the same time. But GSS-greedy consumes a considerably higher amount as it performs additional computation to calculate a similarity score for incoming data samples, making GSS-greedy non-suitable for processing streams on small CPUs and AIoT boards in IoT. The reported memory consumption is the space used by the employed processing scheme, subtracted by space occupied by the data samples stored in the memory unit m. The memory consumption of schemes was tracked till task completion, then averaged. Random-replace requires the least space, and GSS-greedy consumes the highest as it uses memory to store gradient vectors in 32-bit floating-point numbers. To speed up the storage process after sampling data from the stream, Imbal-OL consumes space to store the memory address of the classes - this results in using more memory than Random-replace.

C. Results Analysis

As presented above, Imbal-OL produces better accuracy models because of two reasons. First, the Imbal-OL processed stream is balanced, so all classes will get replayed roughly with the same frequency, thus reducing forgetting. When storing an un-processed stream, underrepresented classes will exist, which will not be replayed often during learning, leading to forgetting. This can be indirectly noticed in Table I - accuracy increases only slightly for Imbal-OL when switched from uniform to weighted replay because, Imbal-OL processes the stream, making classes balanced before temporarily storing till learning. Second, GSS-greedy and other schemes are biased by the imbalances, and it does not store an adequate number of data samples from minority classes. So, the model forgets such classes during evaluation. Since Imbal-OL stores all data samples from minority classes, forgetting is vastly reduced - observed in Table I, as models produced by training using Imbal-OL processed streams show higher accuracy.

IV. Related Work

The relevant research can be categorized into five groups as presented in this section. The first group covers *Regularization based Approaches* where one or more additional loss terms is applied during learning to remember previously gathered knowledge. Popular studies that follow this approach are learning without forgetting [5], synaptic intelligence [9], and elastic weight consolidation [10]. The second group is *Parameter Isolation based Approaches* where forgetting is reduced by allocating non-overlapping sets of model parameters for each learning task [11, 12]. Such methods require task identifiers during training and prediction time, increasing computational strain and overheads. *Replay based Approaches* form the third group. Here, to mitigate catastrophic forgetting, the previously observed instances from the data stream are replayed during future learning. This replay can be performed directly by storing a small subset of data from the stream [13], or indirectly using generative models [14]. *Incremental Approaches* forms the fourth group where the majority of work pertinent to OL proposes to use task-incremental settings to achieve higher inference performance while minimizing forgetting [15].

Other OL studies existing in IoT such as Train++ [4], TinyOL [3], TinyTL [6], Edge2Train [16], ML-MCU [2], have not explored learning using the challenging imbalanced sensor data streams. The final *Reservoir Sampling* group extracts an iid subset of data received from the stream and selects which data samples need to be locally stored for learning [17]. Approaches GSS-IQP and GSSGreedy [8] maximize the variance of the stored memories concerning the gradient direction of the generated model updates. Both approaches achieve higher accuracy than the classic reservoir sampling [18] when learning moderately imbalanced streams of the MNIST Digits dataset. The closely related studies [8, 18, 19] evaluate their schemes on high-performance standard GPU-based setup (e.g. on NVIDIA TITAN Xp). Orthogonal to such works, Imbal-OL is designed to be a resource-friendly scheme to boost the quality of model updates when practicing online machine learning on tiny ubiquitous devices in IoT.

V. Conclusion

This work investigated the catastrophic forgetting issue when practicing online machine learning on tiny devices using severely imbalanced data streams. Imbal-OL was proposed as an OL plugin to process real-world IoT streams before feeding it to the learner that updates the local on-device model. Also, a generalized design flow with two-element loss was presented to show the developers how Imbal-OL can be used in action during OL. Imbal-OL produced better quality models than the state-of-the-art as it shows unique characteristics when dealing with underrepresented classes, performing sampling, utilizing memory, and replaying data samples when learning.

Future work plans to extend Imbal-OL to be applicable in federated learning settings where OL needs to be performed using imbalanced and also incomplete data streams. Also, similar to the TinyML benchmark [20], we plan to use sophisticated datasets and conduct extensive Imbal-OL evaluation on the latest pocket-friendly FPGAs, SoCs and AIOT boards.

Acknowledgements

This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/16/RC/3918 (Confirm) and also by a research grant from SFI under Grant Number SFI/12/RC/2289_P2 (Insight), with both grants co-funded by the European Regional Development Fund.

References

[1] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," in *Elsevier Neural Networks*, 2019.

[2] B. Sudharsan, J. G. Breslin, and M. I. Ali, "Ml-mcu: A framework to train ml classifiers on mcu-based iot edge devices," in *IEEE Internet of Things Journal*, 2021.

[3] H. Ren, D. Anicic, and T. Runkler, "Tinyol: Tinyml with online-learning on microcontrollers," in *arXiv:2103.08295*, 2021.

[4] B. Sudharsan, P. Yadav, J. G. Breslin, and M. I. Ali, "Train++: An incremental ml model training algorithm to create self-learning iot devices," in *Proceedings of the 18th IEEE International Conference on Ubiquitous Intelligence and Computing (UIC)*, 2021.

[5] Z. Li and D. Hoiem, "Learning without forgetting," in *IEEE transactions on pattern analysis and machine intelligence*, 2017.

[6] H. Cai, C. Gan, L. Zhu, and S. Han, "Tinytl: Reduce memory, not parameters for efficient on-device learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2020.

[7] B. Sudharsan, P. Yadav, J. G. Breslin, and M. I. Ali, "An sram optimized approach for constant memory consumption and ultra-fast execution of ml classifiers on tinyml hardware," in *IEEE International Conference on Services Computing (SCC)*, 2021.

[8] R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio, "Gradient based sample selection for online continual learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[9] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *International Conference on Machine Learning*, 2017.

[10] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," in *Proceedings of the national academy of sciences*, 2017.

[11] J. Serra, D. Suris, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," in *International Conference on Machine Learning*, 2018.

[12] A. Mallya and S. Lazebnik, "Packnet: Adding multiple tasks to a single network by iterative pruning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018.

[13] D. Isele and A. Cosgun, "Selective experience replay for lifelong learning," in *AAAI Conference on Artificial Intelligence*, 2018.

[14] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *arXiv:1705.08690*, 2017.

[15] J. von Oswald, C. Henning, J. Sacramento, and B. F. Grewe, "Continual learning with hypernetworks," in *8th International Conference on Learning Representations (ICLR)*, 2020.

[16] B. Sudharsan, J. G. Breslin, and M. I. Ali, "Edge2train: A framework to train machine learning models (svms) on resource-constrained iot edge devices," in *Proceedings of the 10th International Conference on the Internet of Things (IoT'20)*, 2020.

[17] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. Torr, and M. Ranzato, "On tiny episodic memories in continual learning," in *arXiv:1902.10486*, 2019.

[18] J. S. Vitter, "Random sampling with a reservoir," in *ACM Transactions on Mathematical Software (TOMS)*, 1985.

[19] A. Chrysakis and M.-F. Moens, "Online continual learning from imbalanced data," in *International Conference on Machine Learning*, 2020.

[20] B. Sudharsan, S. Salerno, D.-D. Nguyen, M. Yahya, A. Wahid, P. Yadav, J. G. Breslin, and M. I. Ali, "Tinyml benchmark: Executing fully connected neural networks on commodity microcontrollers," in *IEEE 7th World Forum on Internet of Things (WF-IoT)*, 2021.