Concept Drift Mitigation on Resource-Constrained IoT Devices via Self-Learning

Amin Kargar
Tyndall National Institute
University College Cork
Walsh Scholar
Cork, Ireland
amin.kargar@tyndall.ie

Dimitrios Zorbas
School of Engineering & Digital Sciences, Nazarbayev University
Astana, Kazakhstan
dimitrios.zorbas@nu.edu.kz

Michael Gaffney
Horticulture Development Department
Teagasc Ashtown Food Research Centre
Dublin, Ireland
michael.gaffney@teagasc.ie

Brendan O'Flynn
Tyndall National Institute
University College Cork
Cork, Ireland
brendan.oflynn@tyndall.ie

Salvatore Tedesco
Tyndall National Institute
University College Cork
Cork, Ireland
salvatore.tedesco@tyndall.ie

2025 IEEE Sensors Applications Symposium (SAS) | 979-8-3315-1193-7/25/$31.00 ©2025 IEEE | DOI: 10.1109/SAS65169.2025.11105109

***Abstract*— Real-world deployments of Internet of Things (IoT) sensing systems equipped with artificial intelligence (AI) models generally experience a reduction in accuracy over time due to concept drift and data shift. To overcome this issue, it is suggested to periodically retrain the embedded AI model using new incoming data. However, this requires powerful processing hardware and labelled data, which are not generally available on IoT edge-based devices deployed in the real-world. In this study, we propose a method that benefits from a Deep Learning (DL) model for feature extraction and K-Means clustering for classification. The K-Means clustering technique can use the new incoming unlabeled data to update the model, thus helping the system to adapt to any changes that might occur in the new data. The proposed method is evaluated on two image datasets: the first is a public dataset with artificially added concept drift, and the second is a real-world dataset that suffers from concept drift issues. This is a lightweight model with only 610 KB of size and 608 KB of peak memory, which needs less than 0.6 s to perform and lower than 0.7 J to analyse each sample and update the model. Therefore, the method could easily be stored and executed on resource-constrained microcontroller-based (MCU-based) devices to deal with concept drift.**

Keywords—Continual Learning, Edge AI, On-Device Learning, Microcontroller, TinyML, Concept Drift

I. INTRODUCTION

The combination of IoT architectures based on resource-constrained low-power devices and AI models allows for the efficient monitoring of real-world scenarios across a wide range of domains, including agriculture, environment, industry and manufacturing, as well as healthcare. DL models are nowadays widely used to process data captured continuously by these devices [1][2]. An issue that often arises in real-world applications is related to the fact that the characteristics of the data collected by the system can change over time, a problem known as concept drift. This can occur because of seasonality, environmental changes, or even IoT components and sensor ageing [3][4]. These changes negatively affect the DL models and degrade the performance of real-world IoT-based services. Thus, it is necessary to apply some mitigation techniques [5].

Incremental learning (IL), also known as continual learning, is a technique that suggests re-training a DL model on batches of newly captured data. This technique requires sending local data to a cloud or a server, since model training is generally a computationally intensive task [6]. Besides, new data requires to be labelled accurately by experts, which is also a time-consuming and tedious task. Afterwards, the updated model is sent back to the IoT devices to, once again, accurately perform the prediction task. Despite its advantages in terms of accuracy improvements, this procedure is inefficient for real-world applications due to the limited communication infrastructure, bandwidth, computational resources, and battery capacity of IoT devices. Moreover, it raises privacy concerns regarding sharing local data with third-party platforms. Hence, on-device training is proposed to address offline training issues by re-training the model on the IoT device itself. Doing that on resource-constrained devices is a challenging task, which is the aim of this paper.

In this work, we propose an on-device self-improving method based on K-Means clustering, a memory and computation efficient method, added on top of a tiny DL model. This allows edge IoT devices to effectively adapt to the dynamic environment they have been deployed in, without transferring data to third-party platforms or requiring any user to label the new incoming data. In this regard, the main contributions of this work are as follows:

- A lightweight on-device self-improving mechanism is proposed to deal with concept drifts or data shifts which occur in real-world scenarios using newly acquired unlabelled data. A tiny DL model is adopted as a model for feature extraction, followed by K-Means clustering used for classification, where only the K-Means centroids are being updated on new unlabelled data iteratively.
- The overall model is tiny with only approximately 610 KB of size and peak memory usage of about 608 KB, which makes it suitable for MCU-based IoT devices.
- The proposed method performance is evaluated on two datasets: the "cat and dog" dataset [7] with concept drift simulated to take into account potential camera sensor aging and changing capturing conditions, and the HALY.ID dataset [8] captured in real-world conditions which suffers from concept drift.

The rest of this paper is organised as follows. Section II reviews several related studies in the domains of IL for resource-constrained devices. In Section III, the proposed method for enabling IL on MCU-based IoT devices is explained in detail. The proposed method evaluation in terms of accuracy and hardware metrics on two different datasets is

described in Section IV. Findings and potential factors that affected the proposed method performance are discussed in Section V. Finally, we draw the conclusions in Section VI.

II. Related Work

IL is a post-deployment strategy for dealing with concept drift occurring in real-world situations, which has been widely studied recently. Yang et al. [3] proposed online sequential extreme learning machines (OS-ELMs) to deal with and detect the concept drift that can occur in nonstationary environments by analysing variations resulting from new incoming data. The authors verified and evaluated the model using synthetic case studies, a real-world dataset, and a real problem. The results showed that their method could better detect the concept drift compared to state-of-the-art methods, thus it allows and helps IL to increase the system accuracy by retraining the model. Ashfahani et al. [9] proposed an autonomous deep clustering network (ADCN), which is an unsupervised method. The proposed ADCN is a self-evolved design, and along with its self-clustering mechanism, it updates and adapts the model with new data. They used several synthetic and real-world datasets to evaluate their methods and artificially added concept drift to datasets. However, such approaches are generally too complex for resource-constrained IoT devices.

There are several studies that specifically focused on lightweight models for resource-constrained IoT devices to deal with concept drift. Liu et al. [10] proposed an on-device self-training method, which is a self-supervised training approach for machine learning models to deal with post-deployment concept drift. The authors also evaluated the proposed method on a real-world gas sensor drift dataset, and also on image datasets in which the concept drift was artificially added. Moreover, the proposed method was implemented on a Raspberry Pi 4B which showed a maximum speedup of about 32 times in comparison with other methods. More recently, researchers have been focused on MCUs as processors since they are a good fit for IoT devices based on their low-power and low-cost features. In this regard, Disabato et al. [11] developed tiny machine learning for concept drift (TML-CD) in which a DL model was used for feature extraction, and then a k-nearest neighbours (K-NNs) model was used for classification. The authors suggested a hybrid adaptation component to overcome the concept drift issue. The proposed method was implemented on MCUs and evaluated on speech command and image classification datasets. Similarly to most studies, the authors artificially generated concept drift and applied it to the datasets. Ren et al. [12] introduced a tiny machine learning model with an online learning (TinyOL) solution allowing on-device IL for MCU-based devices. The model was deployed on an Arduino Nano board equipped with a three-axis accelerometer to monitor a fan condition. Although the results demonstrated the effectiveness of the method, the study did not explore more complex tasks with higher input dimensions, such as image classification.

To deal with the resource limitations of MCU-based IoT devices, several studies have worked on federated learning in which the model training or fine-tuning was distributed among multiple IoT devices. Despite its advantages, federated learning is still not an efficient method because of the resource limitations of IoT-based services in some real-world scenarios, especially in terms of energy and communication resources [13]. Therefore, in this work, we proposed a method to enable IL on MCU-based IoT devices to adapt itself to concept drift issues. The proposed model includes a Feature Extractor (FE) and K-Means for clustering in which the FE remains unchanged on the MCU while the K-Means centroids are updated on new incoming samples to adapt to any potential shift that might occur on new data.

III. Proposed Method

The proposed method diagram is depicted in Figure 1. The method consists of three phases, the first two are carried out offline on a powerful device, either in the cloud or on a server, and the third phase is performed on resource-constrained IoT edge devices. In Figure 1, the individual sub-elements of these phases are also numbered, and these are described in the following sections. In the offline phase, the model is trained using labelled data and deployed on the IoT device. Then, in the on-device phase, in addition to the prediction, the model updates itself based on the newly captured data over time, which helps the model to adapt itself to data shifts.

A. Offline Step

1) Training Feature Extractor (FE) (Phase 1): In this phase, a DL model was selected and trained on the available labelled data in the training set to extract relevant features by using a transfer learning-based DL model. Since the model should be able to run on resource-constrained IoT devices, we selected a pre-trained MobileNetv2 [14] with an alpha value of 0.35 and an input size of 160×160 as a main feature extractor (❶). This is a small but powerful model trained on the ImageNet dataset and is a suitable model to be used as a FE in this context. The extracted feature map dimensions were then reduced using a global average pooling (GAP) layer and were then connected to a fully connected (FC) layer with two neurons (❷). Finally, the activation function of softmax was used for classification (❸). During the training, all layers of the pre-trained MobileNetv2 were frozen to prevent training and only the last layers were unfrozen to be updated and fine-tuned to better extract features from images of the used dataset. This is a general procedure that is widely used in such applications, but because of the concept drift that occurs in real-world scenarios, the model accuracy decreases over time. To overcome this accuracy reduction, we added the next phases to improve the model accuracy.

2) K-Means fitting (Phase 2): In this phase, we used the fine-tuned model from Phase 1 as a feature extractor (❹) to extract the features from the training set, then we applied and fitted the K-Means method to the extracted features for clustering and prediction (❺). This model was then exported

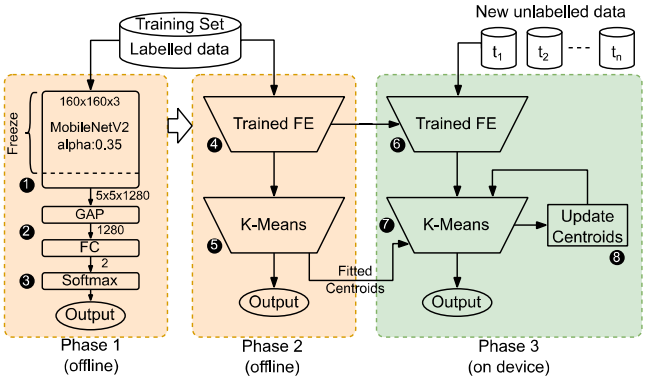

Figure 1: The proposed method diagram.

to the IoT device to perform predictions on newly captured data. The exported model consists of the trained FE, used for feature extraction, and the fitted centroids of the K-Means, which are then used for clustering. It is worth noting that K-Means is a simple and memory- and computationally-efficient method that is suitable for resource-constrained IoT devices.

B. On-Device (Phase 3):

In this phase, IoT devices capture new data over time and perform the prediction task using the provided model from the offline phase, which includes the trained FE (❻), responsible for feature extraction from the input, and the K-Means to perform clustering (❼). To decrease the accuracy loss due to concept drift, K-Means needed to be updated on new data reflecting the data shifts (❽).

Generally, K-Means is updated on batches of data, but this is not efficient or even possible to implement on resource-constrained devices. Therefore, we suggested updating the K-Means centroids on each individual collected sample. Therefore, for each new sample, the algorithm calculates the distance (e.g., Euclidean distance) between the sample point and each centroid and assigns the sample to the nearest cluster. Afterwards, the K-Means centroid to which the sample was assigned is updated using the following equations. Eqs. 1-2, which are based on the learning rate decay concept, along with Eq. 3, collectively guide K-Means smooth learning and prevent drastic shifts in centroid updates, mitigating the possible impact of outliers in the incoming data.

$$c_i = (1 - \eta) \times c_{i-1} + \eta \times x_i \quad (1)$$

$$\eta = \frac{lr}{n_s + 1} \quad (2)$$

$$c_n = c_n - M(c_n - c_{n-1}) \quad (3)$$

where:

- lr is an initial learning rate, which is a constant value,
- c_i is the K-Means centroids at iteration i,
- x_i is the features at iteration i (new data),
- n_s is the number of samples per class seen so far,
- M is a constant value that limits the centroids to prevent drastic shifts.

C. Datasets

Two datasets were used in this study to investigate the performance of the proposed method:

1) Cats and Dogs Dataset [7]: This dataset consists of around 25,000 images of cats and dogs. In this study, in order to simulate real-world scenarios where the captured image quality decreases over time, we split the dataset into 20 subsets and then applied different levels of noises to each set (i.e., by applying gaussian noise, simulated blurriness through resizing and colour shifts), the first subset includes original images without any noises while the last subset consists of images with the highest level of noises. Therefore, we artificially generated a dataset with concept drift to mimic real-world scenarios. The first subset was used for the DL model training and calculation of the K-Means centroids as labelled data, and the rest of the sets were used as an unlabelled data stream. Examples of the dataset are shown in Figure 2, including images in which the noise level increased from the top left corner to the bottom right corner.

Figure 2: Several samples from the Cat and Dog dataset with artificially added concept drift. The level of concept drift increases from the top-left corner to the bottom-right corner.

2) HALY.ID Dataset [8]: This is a binary classification dataset consisting of 3,899 images, out of which 1,777 images belong to a target insect - Halyomorpha Halys (HH) - and the rest are other insects named Non-HH (2,122 samples). Images were captured two times a day from a pear orchard infested with the HH in 2023 using an IoT-based device described in detail in our previous work [15]. The dataset is split into 8 subsets based on the capturing dates and device setup. Visual inspection of images from these subsets clearly shows that concept drift has occurred over time, as is also evident in Figure 3.

IV. Results

A. Experiment Setup

In this section, we evaluate the performance of the proposed method on two different datasets. For this purpose, we investigated two different FEs created from the trained model in Phase 1:

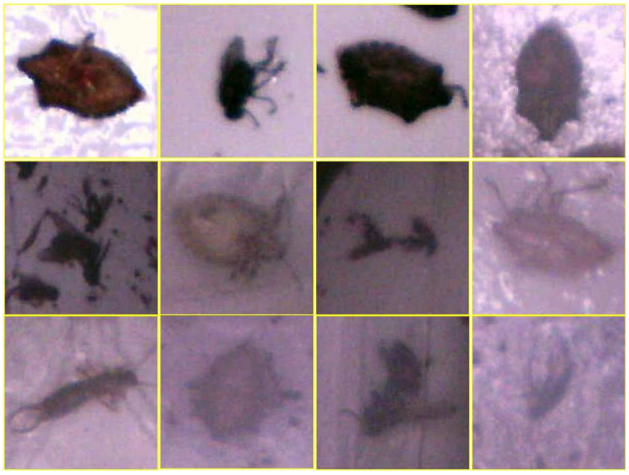

Figure 3: Samples from HALY.ID dataset which suffers from concept drift. The level of concept drift increases from the top-left corner to the bottom-right corner.

- ***FE-GAP***: In this FE, the output features were taken after the GAP layer and layers after the GAP were removed from the FE (see Figure 1). With this FE the model extracts 1280 features.
- ***FE-FC***: In this FE, the FC layer was selected as the output layer and the layers after it were removed (see Figure 1), thus the number of extracted features would be two.

To measure the power consumption and inference time of the proposed method in the device mode (Phase 3), an MCU-based device named OpenMV board equipped with an STM32H7 as the main processor and a camera was selected.

To evaluate the performance of the proposed model, we compared its accuracy in three different modes against the baseline. These three modes along with the baseline are as follows:

- **KM (W-U)**: The proposed K-Means-based method with updating centroids.
- **KM (W/O-U)**: The proposed method without updating centroids (e.g., removing step 8).
- **KM (SK)**: It is the same as the KM (W-U) except that for the updating part, the partial-fitting method provided by the Scikit library [16] was used; this can be used on more powerful hardware such as Raspberry Pi boards that can load and run the Scikit library.
- **DL**: The baseline is the classical DL model trained for classification on the train set which is in fact the model trained in Phase 1.

Moreover, to train the DL model in Phase 1, we set the learning rate to a small value of 0.0001 since the last layer of the pre-trained MobileNetv2 was unfrozen to be fine-tuned on the new dataset. Additionally, the constant values in Eqs. 1-3 were adjusted by trial-and-error at $lr = 0.1$ and $M = 0.01$.

B. Results

In this subsection, we evaluate the performance of the proposed method on the two datasets in terms of accuracy, and hardware and implementation metrics.

1) Cats and Dogs Dataset:

Figure 4 shows the extracted features from the training set samples using both trained FE-GAP and FE-FC where the two classes are shown with different colours. Principal Component Analysis (PCA) [17] projection was adopted to reduce the FE-GAP dimension from 1280 to 2 for visualisation purposes. The first row of the figure illustrates that K-Means can be used for clustering on this dataset since samples belonging to the same class are close to each other, while they are far apart from samples belonging to another class. Figure 5 shows the performance of different modes of the proposed method by using FE-GAP and FE-FC compared to each other, as explained in Section IV-A; this is the average of ten iterations with different initial conditions, such as random weight initialization and data shuffles during training and fitting for all methods. As evident from the figures, the proposed method had higher accuracy compared to the baseline. Moreover, the baseline showed a higher reduction rate in accuracy over time, meaning that the proposed method can better adapt to the concept drift that occurs over time. Comparing FE-FC and FE-GAP reveals that the proposed method in all three modes had the same performance by using FE-FC while by using FE-GAP the proposed KM (SK) had better performance. Additionally, a five percent higher accuracy was obtained over all samples by the proposed method compared to the baseline.

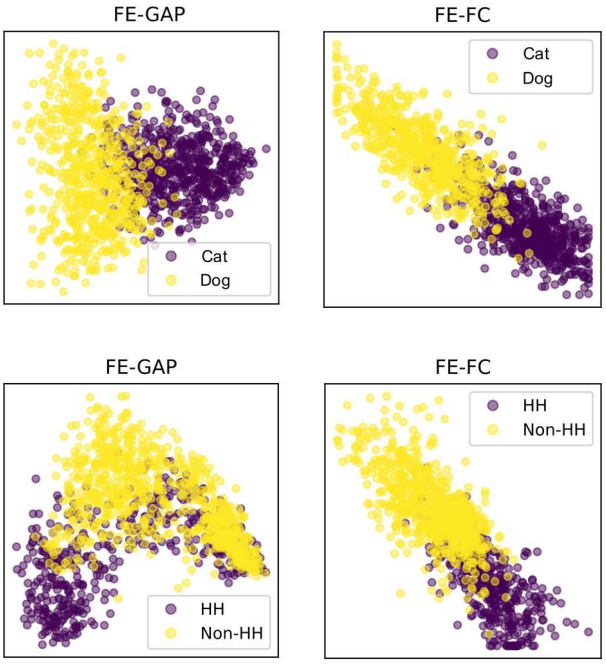

Figure 4: PCA-based visualization of the extracted features in 2D space. The first row belongs to the Cat and Dog dataset and the second row belongs to the HALY.ID dataset.

2) HALY.ID Dataset:

The extracted features of the HALY.ID Dataset, as described in Section IV.B, are also illustrated in Figure 4 for both FE-GAP and FE-FC. Figure 6 shows the accuracy of different modes of the proposed method compared to the baseline. As opposed to the Cat and Dog dataset, using FE-

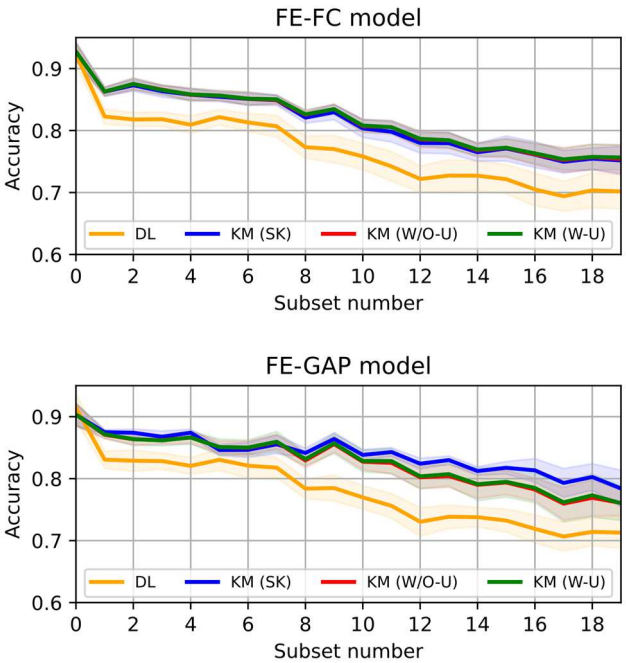

Figure 5: Accuracy performance of the proposed method by using FE-GAP and FE-FC compared to the baseline on the Cat and Dog dataset over time. DL: Baseline deep learning model trained in Phase 1; KM (SK): K-Means with Scikit-learn partial-fitting; KM (W/O-U): K-Means without centroid updates; KM (W-U): K-Means with centroid updates.

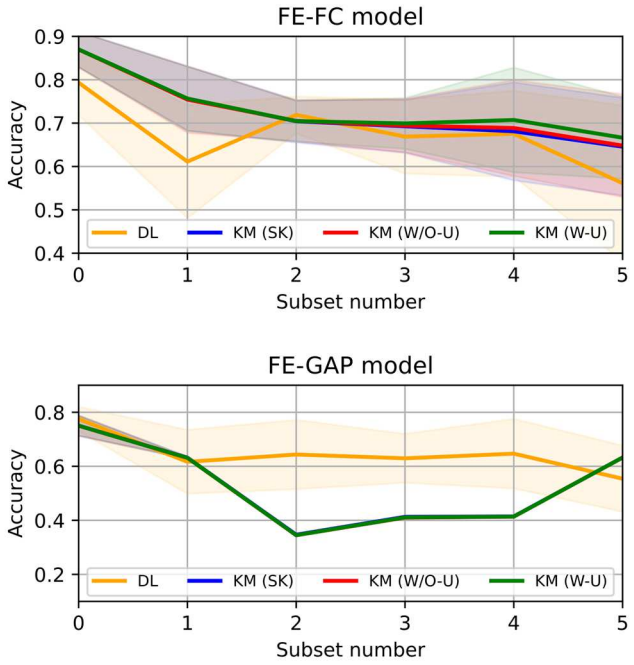

Figure 6: Accuracy performance of the proposed method by using FE-GAP and FE-FC compared to the baseline on HALY.ID dataset over time. DL: Baseline deep learning model trained in Phase 1; KM (SK): K-Means with Scikit-learn partial-fitting; KM (W/O-U): K-Means without centroid updates; KM (W-U): K-Means with centroid updates.

GAP did not perform well, while the FE-FC worked better with a six percent increase in the accuracy over all samples in comparison with the baseline. This is expected because, based on the second row of Figure 4, there is a significant overlap between the two classes for FE-GAP, resulting in K-Means not being able to perform clustering accurately. Moreover, similarly to the previous dataset, the different modes showed equivalent performance.

To compare the performance of continuous learning with the baseline DL model without learning, Table I reports the mean accuracy of the proposed methods using continuous learning versus the DL model without learning. The table shows that the proposed methods increased classification accuracy by at least five percent, except for the GAP-based model on the HALY.ID dataset, as discussed above.

3) Hardware Metrics and Performance on MCU

In this work, we used quantization to map DL model weights and parameters from float32 to int8 to reduce the model size and memory usage. The DL part was implemented using TensorFlow and then exported as a tflite file, which is a compatible format to be loaded onto the OpenMV board using a MicroPython script. Table II reports the performance of the proposed method in terms of hardware metrics on the mentioned MCU-based board. The model size and peak memory usage of the DL part used in the proposed method were approximately 610 KB and 608 KB, respectively, which are in the range of most modern MCUs available on the market. Therefore, the proposed method could be stored and run on MCU-based IoT devices. In addition, the number of parameters for the DL model and FE-FC was about 412 thousand, while it was 410 thousand for FE-GAP since the classifier part was removed in this model. Regarding execution time and energy consumption, the baseline method needed 442 ms and 499 mJ to analyse each image, while these factors increased to 565 ms and 612 mJ for K-Means with FE-

TABLE I. MEAN ACCURACY OF THE PROPOSED METHODS USING CONTINUOUS LEARNING COMPARED TO THE BASELINE DL MODEL WITHOUT LEARNING OVER THE ENTIRE DATASET. DL: BASELINE DEEP LEARNING MODEL TRAINED IN PHASE 1; KM (SK): K-MEANS WITH SCIKIT-LEARN PARTIAL-FITTING; KM (W/O-U): K-MEANS WITHOUT CENTROID UPDATES; KM (W-U): K-MEANS WITH CENTROID UPDATES.

	Cat and Dog		**HALY.ID**	
Method	**FE-GAP**	**FE-FC**	**FE-GAP**	**FE-FC**
DL	0.78	0.77	0.64	0.67
KM (W-U)	0.83	0.82	0.53	0.73
KM (W/O-U)	0.83	0.82	0.53	0.73
KM (SK)	0.84	0.82	0.53	0.73

TABLE II. PROPOSED METHOD HARDWARE AND IMPLEMENTATION METRICS ON MCU-BASED BOARD.

Method	**Inf. time (ms)**	**Energy (mJ)**	**Peak mem usage (KB)**	**Model size (KB)**	**Num params**
Full DL (baseline)	442	499	608	610	412,770
K-Means FE-FC	565	612	608	610	412,770
K-Means FE-GAP	641	688	608	607	410,208

FC and 641 ms and 688 mJ for K-Means with FE-GAP, respectively. This is expected since the proposed method needs to perform K-Means prediction and update on the features extracted by the FE model, which increases the execution time and consequently the energy consumption.

Despite the similar structure of the proposed method in both FE-GAP and FE-FC, the method with FE-GAP had higher values of execution time and energy consumption. The reason is that FE-GAP requires the K-Means to be applied to 1,280 features, while FE-FC requires the K-Means to be applied to a vector with only two features.

V. DISCUSSION

Based on the results, it is clear that the simple K-Means without updating, KM (W/O-U), had higher accuracy compared to the baseline DL model. This reveals that replacing the classifier with a simple K-Means clustering method can better deal with shifts in data. K-Means tends to achieve higher accuracy since it assigns the nearest cluster to the new data based on the extracted features. Therefore, it is more flexible to unseen data than the DL classifier, which generally suffers from poor generalization or overfitting, especially when the training set is small or lacks diversity.

Moreover, comparing the performance of the proposed method on both datasets highlighted the real difference between real-world scenarios and the artificially created ones. This is due to several factors: firstly, our device was deployed in an actual orchard and captured images two times a day. Images were captured during the night using an LED light to minimize the lighting conditions. However, as orchards are harsh environments, several factors affected the camera and consequently the captured images, from dust, dirt, and rain on the camera lens to high temperature fluctuations during the day that affect the camera sensor. These factors led to a high concept drift rate on the dataset that affected the performance of the proposed method. Besides, the target object was an insect, thus, their frequency varies over time (i.e., their

population varies during the growing season), which makes the dataset and its subsets imbalanced, while this is not generally considered in synthetically created datasets where different types of noises are added to public datasets. Thus, actual concept drift-affected datasets should be used in similar analyses rather than synthetic data. A potential solution to overcome such problems could be an adaptable learning rate mechanism in which the model analyses the input features and based on the level of shift adjust the learning rate. Although we controlled the learning rate, especially based on the number of samples for each class (n_s in Eq. 2), it could be more effective to adjust the learning rate based on shift level by automatically adjusting the constant values of the suggested learning rate (Eq. 1-3), which could be a future work of this study.

Moreover, in this study, we used an MCU-based OpenMV board; however, more powerful options equipped with dedicated machine learning (ML) hardware accelerators can be explored in future work to achieve better performance. These are expected to reduce inference time and power consumption, which are critical factors for such applications.

VI. Conclusion

In this work, we proposed a lightweight method to ameliorate concept drift occurring in real-world applications for resource-constrained MCU-based devices. The proposed method includes two main parts: 1) A Feature Extractor, which is a lightweight DL model used to extract important features from the input data, and 2) a K-Means method, which is responsible for clustering and updating itself based on new data to deal with any potential data shift which might happen over time. This is a small model in size and peak memory usage, at only 610 KB and 608 KB, respectively, which makes it a suitable option for MCU-based devices.

Acknowledgement

This project is co-funded by the European Regional Development Fund (ERDF) under Ireland's European Structural and Investment Funds Programmes 2014–2020. This work was carried out as part of the Haly.ID project -- 2020EN508 funded by Ireland's Department of Agriculture, Food and the Marine under Grant: 2020 Trans National ERA-NET. The first author is supported by a Walsh Scholarship funded by Teagasc, The Irish Food and Agriculture Authority. Aspects of this work have been supported by Research Ireland under Grant 12/RC/2289-P2-INSIGHT2, 13/RC/2077-CONNECT and 21/RC/10303_P2-VISTAMILK2 which are co-funded by the European Regional Development Fund (ER DF) under Ireland's European Structural and Investment Funds Programme 2014-2020. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

[1] M. Khani and M. Alizadeh, “Continuous Learning for Lightweight Machine Learning Inference at the Edge,” 2023.

[2] V. Rajapakse, I. Karunanayake, and N. Ahmed, “Intelligence at the Extreme Edge: A Survey on Reformable TinyML,” *ACM Comput Surv*, vol. 55, no. 13, Dec. 2023, doi: 10.1145/3583683.

[3] Z. Yang, S. Al-Dahidi, P. Baraldi, E. Zio, and L. Montelatici, “A Novel Concept Drift Detection Method for Incremental Learning in Nonstationary Environments,” *IEEE Trans Neural Netw Learn Syst*, vol. 31, no. 1, pp. 309–320, Jan. 2020, doi: 10.1109/TNNLS.2019.2900956.

[4] F. Bayram, B. S. Ahmed, and A. Kassler, “From concept drift to model degradation: An overview on performance-aware drift detectors,” *Knowl Based Syst*, vol. 245, p. 108632, Jun. 2022, doi: 10.1016/J.KNOSYS.2022.108632.

[5] C. H. Lu and G. Y. Fan, “Environment-Aware Dense Video Captioning for IoT-Enabled Edge Cameras,” *IEEE Internet Things J*, vol. 9, no. 6, pp. 4554–4564, Mar. 2022, doi: 10.1109/JIOT.2021.3104289.

[6] S. Zhu, T. Voigt, J. Ko, and F. Rahimian, “On-device Training: A First Overview on Existing Systems,” *ACM Trans Sens Netw*, vol. 0, no. 6, p. 39, Dec. 2022, doi: 10.1145/3696003.

[7] “Kaggle Cats and Dogs Dataset from Official Microsoft Download Center.” Accessed: Feb. 25, 2025. [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=54765

[8] A. Kargar, D. Zorbas, S. Tedesco, M. Gaffney, and B. O’Flynn, “Image Dataset of Brown Marmorated Stink Bug (BMSB) or Halyomorpha Halys (HH),” 2024, *Zenodo*. doi: 10.5281/zenodo.7887045.

[9] A. Ashfahani and M. Pratama, “Unsupervised Continual Learning in Streaming Environments,” *IEEE Trans Neural Netw Learn Syst*, vol. 34, no. 12, pp. 9992–10003, Dec. 2023, doi: 10.1109/TNNLS.2022.3163362.

[10] J. Liu, X. Yu, and T. Rosing, “Self-Train: Self-Supervised On-Device Training for Post-Deployment Adaptation,” *Proceedings - 2022 IEEE International Conference on Smart Internet of Things, SmartIoT 2022*, pp. 161–168, 2022, doi: 10.1109/SMARTIOT55134.2022.00034.

[11] S. Disabato and M. Roveri, “Tiny Machine Learning for Concept Drift,” *IEEE Trans Neural Netw Learn Syst*, vol. 35, no. 6, pp. 8470–8481, Jun. 2024, doi: 10.1109/TNNLS.2022.3229897.

[12] H. Ren, D. Anicic, and T. A. Runkler, “TinyOL: TinyML with Online-Learning on Microcontrollers,” *Proceedings of the International Joint Conference on Neural Networks*, vol. 2021-July, Jul. 2021, doi: 10.1109/IJCNN52387.2021.9533927.

[13] L. U. Khan, W. Saad, Z. Han, E. Hossain, and C. S. Hong, “Federated learning for internet of things: Recent advances, taxonomy, and open challenges,” *IEEE Communications Surveys and Tutorials*, vol. 23, no. 3, pp. 1759–1799, Jul. 2021, doi: 10.1109/COMST.2021.3090430.

[14] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, Dec. 2018, doi: 10.1109/CVPR.2018.00474.

[15] A. Kargar, D. Zorbas, S. Tedesco, M. Gaffney, and B. O’Flynn, “Detecting Halyomorpha halys using a low-power edge-based monitoring system,” *Comput Electron Agric*, vol. 221, p. 108935, Jun. 2024, doi: 10.1016/J.COMPAG.2024.108935.

[16] F. Pedregosa FABIANPEDREGOSA *et al.*, “Scikit-learn: Machine Learning in Python,” *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Nov. 2011, doi: 10.5555/1953048.2078195.

[17] I. T. Jollife and J. Cadima, “Principal component analysis: a review and recent developments,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, Apr. 2016, doi: 10.1098/RSTA.2015.0202.