POLITECNICO DI TORINO
Repository ISTITUZIONALE

Divertor Tokamak Test facility project: status of design and implementation

Original
Divertor Tokamak Test facility project: status of design and implementation / Romanelli, F., Abate, D., Acampora, E., Agguiaro, D., Agnello, R., Agostinetti, P., Agostini, M., Aimetta, A., Albanese, R., Alberti, G., Albino, M., Alessi, E., Almaviva, S., Alonzo, M., Ambrosino, R., Andreoli, P., Angelone, M., Angelucci, M., Angioni, C., Angrisani Armenio, A., et al.. - In: NUCLEAR FUSION. - ISSN 0029-5515. - 64:11(2024). [10.1088/1741-4326/ad5740]

Availability:
This version is available at: 11583/2992980 since: 2024-10-01T15:08:57Z

Publisher:
Institute of Physics

Published
DOI:10.1088/1741-4326/ad5740

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

06 October 2026

MAUPITI: On-Device Prototype-Based Learning on a Smart Infrared Sensor

Beatrice Alessandra Motetti, Tanguy Dugas du Villard, Matteo Risso, Alessio Burrello, Francesco Daghero, Enrico Macii, Massimo Poncino, Marco Castellano, Alfio Basile, Daniele Jahier Pagliari

arXiv:2608.07192v1 [cs.LG] 7 Aug 2026

***Abstract*—Low-resolution infrared (IR) array sensors represent an interesting solution for privacy-preserving human sensing in embedded systems. In this letter, we describe a smart multi-pixel IR sensor integrating a 16×16 thermal MOSFET (TMOS) array and a RISC-V microcontroller extended with low-precision SIMD instructions, capable of *on-device learning* and *continual adaptation* for pose and gesture recognition tasks under tight memory and power constraints (<32 kB on-chip memory, ≈1.5 mW). To avoid the memory overheads of backpropagation and replay buffers, we adopt a prototype-based Nearest Class Mean (NCM) classifier in which a simple Convolutional Neural Network (CNN) encoder is trained and quantized offline, while class prototypes are stored and updated on the device in streaming mode. With experiments on two datasets, we show that this approach yields accuracy on par with a conventional classifier, with negligible latency overheads in both the classification and the prototype update (<0.29% considering both phases), effectively enabling online adaptation of the perception framework.**

***Index Terms*—TinyML, on-device learning, infrared arrays, smart sensors, RISC-V.**

I. Introduction and Related Works

Low-resolution infrared (IR) arrays are increasingly adopted in embedded systems for occupancy monitoring, people counting, and human-computer interaction, as they combine low power consumption, low cost, and strong privacy guarantees by sensing only coarse thermal patterns instead of high-resolution RGB images [1], [2].

Prior work has focused on exploring machine-learning-based algorithms for IR-based perception [3], [4]. However, the focus of these early approaches was not on the direct deployment of the system on an embedded device, requiring expensive data transmissions to servers for inference [5]. More recently, other works implemented IR perception algorithms able to run directly on embedded devices, such as a Raspberry PI [6], [7] or an STM32F microcontroller (MCU) [1], [8]. In our previous work [9], we proposed a full-stack HW-SW optimization flow for tiny Convolutional Neural Networks (CNNs) on IR arrays, focusing on privacy-preserving people counting. By combining hardware-aware differentiable neural architecture search and mixed-precision quantization we produced models that could run on MAUPITI, a smart IR sensor integrating a 16×16 TMOS array and a *Ibex* RISC-V core. By executing optimized CNNs directly on the sensor, we demonstrated significant energy benefits w.r.t. MCU-based deployments [9]. All these previous works, however, focused solely on *local inference*, while training was kept offline on a more powerful machine.

B.A. Motetti, T. Dugas du Villard, M. Risso, A. Burrello, F. Daghero, E. Macii, M. Poncino, and D. Jahier Pagliari are with Politecnico di Torino, Turin, Italy. Corresponding author email: beatrice.motetti@polito.it.

M. Castellano, and A. Basile are with STMicroelectronics, Cornaredo, Italy.

This publication is part of the project PNRR-NGEU which has received funding from the MUR – DM 118/2023.

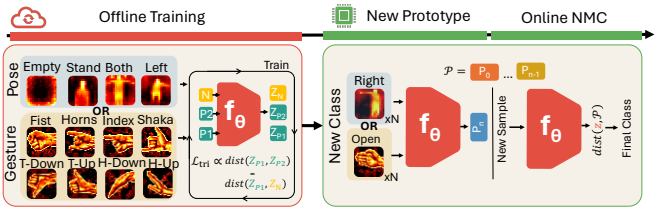

Fig. 1. Overview of our proposed approach. An offline training is performed with triplet loss prior to deployment (left). New classes can easily be added online (right), with minimal overhead and without retraining the encoder.

In many smart-sensing deployments, fixed models are not sufficient. Sensor placement, background temperature, and patterns to recognize vary across locations and over time. Sending data to the cloud to re-train the model requires constant connectivity and may undermine energy efficiency. Instead, *on-device learning* and *continual adaptation* are desirable, but remain challenging under the tight memory and compute budgets of sensor-class embedded hardware.

This letter extends MAUPITI [9] from an inference-only smart sensor to a platform capable of *on-device prototype-based learning*, as shown in Fig. 1. Namely:

- We integrate a Nearest Class Mean (NCM) classifier [10], [11] on top of a CNN encoder running on MAUPITI. The encoder is trained offline using metric learning and quantization-aware training, while prototypes can be updated on the device without backpropagation.
- We present an embedded implementation that fits within MAUPITI's 16 kB instruction and 16 kB data memories, enabling online prototype updates with negligible latency and memory overhead.
- We demonstrate the approach on two new applications for MAUPITI, i.e., pose and gesture recognition, achieving comparable accuracy with respect to a traditional CNN classifier trained with backpropagation, while enabling continual learning and few-shot adaptation on-board.

Our code and datasets are publicly available at `https://github.com/eml-eda/maupiti-odl`.

II. Methodology

A. Nearest Class Mean Classifiers

Let $f_\theta(\cdot)$ denote the CNN encoder with parameters θ, which we derive in this work by removing the classifier head from

©2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

the CNN templates of [1], [9]. Given an input IR frame $\mathbf{x} \in \mathbb{R}^{16\times16}$, the encoder outputs an embedding $\mathbf{z} = f_\theta(\mathbf{x}) \in \mathbb{R}^p$. where p is the latent representation dimension (and prototype size). In an NMC classifier [11], for each class c, a prototype vector $\mathbf{m}_c \in \mathbb{R}^p$ is computed as the mean of the embeddings of all samples of that class ($\mathbf{x} \in \mathcal{D}_c$) encountered during training:

$$\mathbf{m}_c = \frac{1}{|\mathcal{D}_c|} \sum_{i \in \mathcal{D}_c} \mathbf{z}_i \tag{1}$$

At inference time, the distance between a new sample $\mathbf{x}$ and all prototypes is computed, and the class of the closest one in the embedding space corresponds to the NCM's prediction $\hat{c}$:

$$\hat{c} = \arg\min_c d(\mathbf{z}, \mathbf{m}_c), \tag{2}$$

where $d(\cdot,\cdot)$ is a distance metric (e.g., Euclidean or Manhattan). On-device training can be performed by incrementally updating prototypes. In particular, when a new labeled sample $(\mathbf{x}, c)$ is available, we derive $\mathbf{z} = f_\theta(\mathbf{x})$ and update the prototype of class c accordingly:

$$N_c \leftarrow N_c + 1, \quad \mathbf{m}_c \leftarrow \mathbf{m}_c + \frac{1}{N_c}\big(\mathbf{z} - \mathbf{m}_c\big). \tag{3}$$

In a Continual Learning scenario, it is also straightforward to add a new class, as it is sufficient to allocate a new prototype slot and compute the mean latent representation over a set of new labeled samples, as shown in Fig. 1.

The prototype update has $O(p)$ time and memory complexity and does not require storing old samples in *replay buffers*, nor gradients, making it suitable for sensors with a few tens of KiB of memory. Since no backpropagation is performed on-device, training latency is determined by embedding computation and prototype updates, with the latter adding only a few hundred operations on top of the forward pass of the CNN for realistic values of p. For more details on NCM models, we refer readers to [11].

B. Training Flow

Our offline training (Fig. 1, left) comprises two stages:

1) Metric learning trains the encoder to generate an embedding space in which samples from the same class are clustered together, while samples of different classes are pushed away. We evaluated triplet loss with margins of 20, 50, and 100, as well as prototypical loss [10], using both Manhattan and Euclidean distances, and selected the Euclidean-distance triplet loss with a margin of 100 as it yielded the best results.

2) Quantization-aware training (QAT) to make the network compatible with the integer-only MAUPITI hardware. We quantize using the PLiNIO library [12], employing INT8 affine quantization for weights and activations with PACT activation clipping [13].

We also experimented with adding an initial pretraining stage in which the CNN encoder was trained in a supervised way, using a cross-entropy loss with a classification head that was later removed, but we experimentally found the performance of the NCM classifier not to improve. Thus, the results in the paper are obtained by performing metric learning and QAT only. After training, quantized models are compiled from PyTorch to C code using optimized kernel routines for MAUPITI, using a custom translator.

We explore the CNN encoder architecture through Bayesian optimization with Optuna [14] varying the number of filters of the encoder's convolutional layers, along with the kernel size and the pooling size. The optimization is conducted with a dual objective, using Optuna's built-in multi-objective optimization capabilities: (i) maximizing the validation accuracy of the model on all classes on metric learning and (ii) minimizing the number of encoder parameters to reduce its memory occupation and fit MAUPITI's tight constraints.

C. Embedded Deployment on MAUPITI

MAUPITI is a smart infrared sensor SoC implemented in 130 nm CMOS technology and clocked at 20 MHz. It integrates two main blocks [9]:

- A 16×16 TMOS thermal array with 8 parallel analog front-ends, each reading one row, enabling acquisition of a full frame in two steps. The array draws approximately 0.62 mW at 2.4 V and reaches 10 FPS.
- A digital processing block with a customized 32-bit RISC-V Ibex core, 16 kB instruction memory, 16 kB data memory, boot ROM, calibration registers, an instruction tracer, one-time programmable memory, and I2C/SPI3 communication interfaces. It also comprises an SDOTP unit that supports SIMD dot products between two 32-bit registers, interpreted as 4×8-bit or 8×4-bit vectors, plus a 32-bit accumulator. The digital block consumes about 0.9 mW under nominal conditions.

In this work, we store prototypes on MAUPITI as INT32 vectors. For C classes, prototype dimension p, the total prototype memory footprint is:

$$M_{\text{proto}} = C \cdot p \cdot 4 \text{ bytes} \tag{4}$$

Deployed models use custom hand-written kernels and a lightweight runtime. Convolutions and dense layers use INT8 inputs and weights. As our goal is to demonstrate the feasibility of NCM learning on a typical embedded sensor node, we do not use the SDOTP unit and instead perform scalar MAC operations with the main core, which are more representative of the typical hardware available on these devices. Intermediate activations are stored in a pair of alternating buffers to minimize memory occupation while enabling concurrent inference and prototype updates. The NCM head reuses the same buffers to compute distances between the current embedding and prototypes. Moreover, we replace the division in Eq. 3 with a shift by accumulating $\mathbf{z}$ values until N_c is a power of two before subtracting and normalizing.

Online training is integrated into the same embedded C code that manages sensor acquisition and classification. Training is triggered by a control signal (e.g., a button press or a command from a host) specifying whether to create a new class or update an existing one.

III. Experimental Evaluation

We perform most of our experiments on a simple **pose recognition** dataset, containing IR images of a human in front

of the sensor in five different poses (*empty scene*, *standing*, *both arms raised*, *right arm raised*, and *left arm raised*). To assess the generality of our approach, we additionally evaluate it on a more complex, 9-class **hand gesture recognition** dataset, including images of the following gestures, performed with the hand in front of the sensor, pointing sideways: *fist*, *horns*, *index*, *shaka*, *thumb-down/up wrist-down/up*, *open hand*. Both datasets are recorded in multiple indoor environments (offices, labs), with varied backgrounds and environmental temperatures. We split data session-wise as in [9] to better match a real-world scenario in which the sensor is trained in a different environment w.r.t. the one in which it is deployed.

Unless stated otherwise, we consider as encoder a tiny CNN that follows the blueprint architecture of [9], and includes three 2D-convolutional layers with 3x3 filters and 8 channels, followed by batch normalization and ReLU activations. Max pooling with 2x2 stride is inserted after the 2nd and 3rd convolution. Lastly, a single linear layer projects the flattened output of the last convolution to the prototype space of size p.

In the following experiments we compare with three different baselines, considering a conventional CNN classifier with the same backbone, namely: i) full-network retraining, similarly to [15], ii) retraining only the last layer, as in previous works [16], [17], and iii) retraining only the parameters associated to the added output neuron(s) [18]. For each baseline we consider two variants, i.e., with and without replay.

A. Pose recognition performance

As a first experiment, we evaluate a NCM classifier in an offline training setting (all classes available in the initial training set), varying the prototype size p. We obtain test accuracies in the range (91.72%-92.76%) for $p \in \{8, 16, 32, 64\}$, with a drop $\approx 2-3\%$ relative to the 94.81% obtained by the same CNN backbone attached to a softmax classifier head with 5 output neurons, thus confirming that replacing softmax with NCM does not significantly degrade accuracy for this task.

Next, we consider a continual learning setup, where either the last (1C) or the last two classes (2C) are not present during the initial training of the encoder and prototypes. These classes are introduced later on-device using a limited number of labeled samples (shots). For this experiment we fix $p = 64$, and we compare our approach with the baselines. When considering the variant with replay, we build the replay buffer by storing a random subset of samples from all other classes equal to the number of shots.

Fig. 2 shows the accuracy results versus the number of shots for NCM and the considered baselines after the metric learning stage. We use power of two values for the number of shots because these allow replacing divisions with shifts as explained in Sec. II. First, we note that NCM achieves very good accuracy even with a single sample to build the new class prototype, although the variability of results is reduced with 4+ shots. With 4 shots, the 1C average accuracy is 88% (vs 94% of offline training), which degrades to 68% for 2C, as expected due to the increased difficulty of the online learning task without any encoder adaptation, as novel classes must be accommodated within a fixed latent space. In an additional

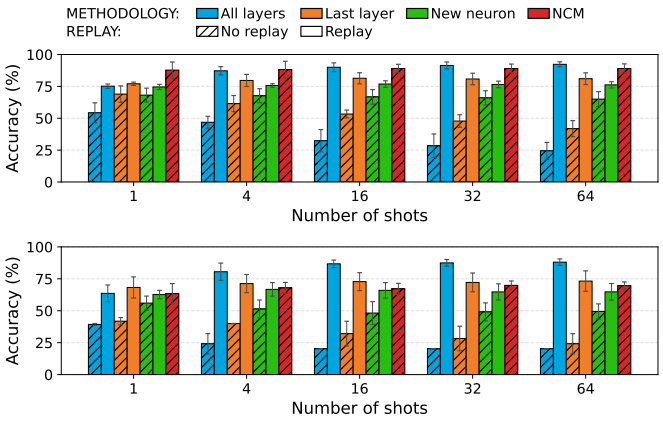

Fig. 2. 5-class accuracy of the NCM classifier and the considered baselines, trained online on one (top) or two (bottom) previously unseen classes. The error bars report the standard deviation over 5 seeds.

TABLE I
BACKBONE ARCHITECTURE EXPLORATION RESULTS AFTER QAT

Architecture	N. of parameters	1C Acc. (%)	2C Acc. (%)
Baseline	3,408 (1.00×)	88.16 ± 6.52	67.88 ± 4.29
Optimized (L)	6,392 (1.88×)	91.26 ± 3.26	68.93 ± 1.70
Optimized (M)	1,323 (0.39×)	85.28 ± 4.40	68.94 ± 4.19
Optimized (S)	755 (0.22×)	69.89 ± 4.32	61.11 ± 2.32

scalability evaluation, NCM outperforms the best non-replay baseline by ∼4 percentage points in a 3C scenario. Second, NCM significantly *outperforms all other methods that do not use replay examples*, which suffer in particular in multiple-shots training due to catastrophic forgetting [19], achieving accuracies in the range (87.77%, 89.04%) for 1C and (63.40%, 69.66%) for 2C. This is despite NCM not incurring the large memory and latency/energy overheads of backpropagation, and being fully compatible with integer quantization, which is problematic for methods that require small weight updates. In the 1C case, NCM remains competitive even against replay methods, *including full network fine-tuning*, albeit not requiring any extra input storage.

Next, we apply the Bayesian Optimization procedure described in Sec. II-B on the metric learning stage. We obtain a Pareto front of CNNs, and report in Tab. I the 4-shots test accuracy after the QAT phase in the 1C and 2C cases for three relevant results, together with the corresponding model sizes. The Optimized (L) architecture outperforms the baseline (+3.10 and +1.05 percentage points in 1C and 2C, respectively), at the cost of a 1.88× larger architecture. The Optimized (M) architecture substantially reduces the number of parameters (1,323 vs 3,408), at the price of a slightly lower accuracy than the baseline in the 1C setting, while remaining comparable in the 2C case. The Optimized (S) showcases significantly lower accuracy, yet competitive considering it consists in only 755 parameters.

B. Latency, memory, and training overhead

Having demonstrated the effectiveness of CNN+NCM models for IR data, we now analyze the cost of deploying them on MAUPITI. We deploy a full-INT8 version of the Optimized

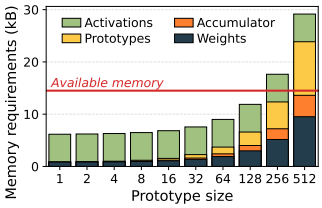

Fig. 3. Memory requirements of the NCM classifier

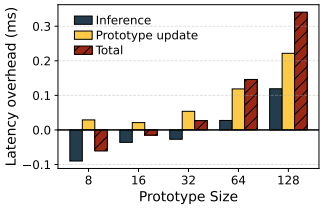

Fig. 4. Latency overhead with respect to a traditional classifier

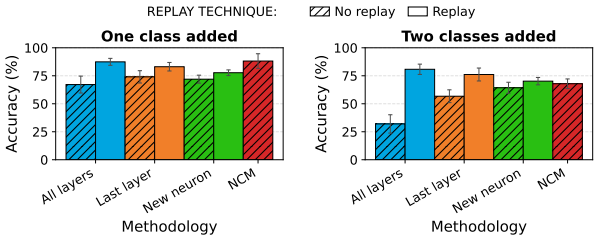

Fig. 5. 9-class accuracy of the NCM classifier and the considered baselines for hand gesture recognition ($p = 64$, 4 shots).

(M) model from Tab. I, i.e., the one achieving the best accuracy versus size trade-off. When quantizing the models, the test accuracy drop with respect to floating point versions is equal to 0.47 percentage points for 1C and 1.70 for 2C.

Fig. 3 shows the memory occupation breakdown of our method as a function of p. Notably, increasing p affects the weights occupation due to the parameters of the last projection layer. As shown, with $p \leq 64$, our model easily fits within the 16 kB data memory of MAUPITI, with prototypes contributing only a few hundred bytes (maximum $64 \times 5 \times 4 = 1.28$kB where 5 is the number of classes and 4 the bytes per element). For comparison, in a 4-shot setting, an input replay buffer would require $4 \times 16 \times 16 \times 5 = 5$kB ($4\times$ more), i.e., almost half of the total memory.

We also measure the *inference latency* by executing multiple forward passes and NCM classifications on the real sensor. For $p \leq 64$ the total classification latency (sensor readout + CNN inference + NCM head) is ≈ 50 ms, comfortably below the 100 ms requirement imposed by the sensor frame rate. The overhead with respect to a traditional classifier (same backbone, 5-neurons softmax head) is < 0.03 ms (0.06% of the total for $p = 64$) when considering only the inference phase, as shown in Fig. 4. Notably, inference is slightly *faster* for $p < 64$. Closest prototype extraction and the softmax head both require $O(p \cdot C)$ operations, so this difference is caused entirely by non-idealities (e.g. loop overheads). Different distance metrics (Euclidean, Manhattan, angular) yield negligible latency differences. Lastly, the extra latency overhead due to *prototype updates*, i.e., on-device training, is also < 0.12 ms, as expected since each update requires only $O(N_c \cdot p)$ operations per class, making online training effectively real-time. Considering both training and inference, the total overhead sums up to 0.15 ms (0.29%).

C. Generalization to other datasets

To illustrate the generality of our approach, we apply the same CNN+NCM pipeline to the hand gesture recognition task. The encoder architecture and quantization settings are kept identical to those used for pose recognition. We consider $p = 64$ and 4-shot training. The results of our method and the baselines after the metric learning training stage are reported in Fig. 5. Despite the lower overall accuracy, given by the higher number of classes and difficulty of the task, the overall trends are confirmed. The NCM achieves similar or better performance than the other non-replay methods, and for the 1C case, remains competitive even with replay, including full-backbone fine-tuning. Notably, these results show that the sensor is able to autonomously learn 1(2) new gestures, while maintaining $> 88.16\%(67.88\%)$ accuracy on all 9 classes.

IV. Conclusion

In this letter we presented an on-device learning framework based on the MAUPITI smart infrared sensor. We showed how a CNN encoder coupled with a Nearest Class Mean classifier enables efficient and real-time pose and gesture recognition. Particularly, continual learning can be easily applied on-device, without the expensive costs of backpropagation over a quantized model, while matching the performance of a traditional classifier.

References

[1] C. Xie *et al.*, "Efficient deep learning models for privacy-preserving people counting on low-resolution infrared arrays," *IEEE Internet of Things Journal*, vol. 10, no. 15, pp. 13 895–13 907, 2023.

[2] M. Bouazizi *et al.*, "Low-resolution infrared array sensor for counting and localizing people indoors: When low end technology meets cutting edge deep learning techniques," *Information*, vol. 13, no. 3, 2022.

[3] G. Diaz *et al.*, "Cross-domain human activity recognition using low-resolution infrared sensors," *Sensors*, vol. 24, no. 19, 2024.

[4] C. Yin *et al.*, "Human activity recognition with low-resolution infrared array sensor using semi-supervised cross-domain neural networks for indoor environment," *IEEE Internet of Things Journal*, vol. 10, no. 13, pp. 11 761–11 772, 2023.

[5] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[6] K. A. Muthukumar *et al.*, "A novel hybrid deep learning model for activity detection using wide-angle low-resolution infrared array sensor," *IEEE Access*, vol. 9, pp. 82 563–82 576, 2021.

[7] M. Kraft *et al.*, "Low-cost thermal camera-based counting occupancy meter facilitating energy saving in smart buildings," *Energies*, vol. 14, no. 15, 2021.

[8] A. Metwaly *et al.*, "Edge computing with embedded ai: Thermal image analysis for occupancy estimation in intelligent buildings," in *INTESA 2019*.

[9] M. Risso *et al.*, "Hw-sw optimization of dnns for privacy-preserving people counting on low-resolution infrared arrays," in *DATE 2024*.

[10] J. Snell *et al.*, "Prototypical networks for few-shot learning," in *NeurIPS*, 2017, p. 4080–4090.

[11] T. Mensink *et al.*, "Distance-based image classification: Generalizing to new classes at near-zero cost," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2624–2637, 2013.

[12] D. Jahier Pagliari *et al.*, "Plinio: A user-friendly library of gradient-based methods for complexity-aware dnn optimization," in *FDL*, 2023.

[13] J. Choi *et al.*, "Pact: Parameterized clipping activation for quantized neural networks," 2018.

[14] T. Akiba *et al.*, "Optuna: A next-generation hyperparameter optimization framework," in *SIGKDD*, 2019.

[15] F. M. Castro *et al.*, "End-to-end incremental learning," in *ECCV*, 2018.

[16] L. Pellegrini *et al.*, "Latent replay for real-time continual learning," in *IROS*, 2020, pp. 10 203–10 209.

[17] L. Mei *et al.*, "Train-on-request: An on-device continual learning workflow for adaptive real-world brain machine interfaces," in *BioCAS*, 2024.

[18] H. Ren *et al.*, "Tinyol: Tinyml with online-learning on microcontrollers," in *IJCNN*, 2021.

[19] G. Shi *et al.*, "Overcoming catastrophic forgetting in incremental few-shot learning by finding flat minima," in *NeurIPS*, 2021.