First Year Report

Young D. Kwon
Churchill College

UNIVERSITY OF
CAMBRIDGE

Efficient On-device Systems that can Sense, Learn, and Optimize Continually in the Wild

University of Cambridge
Department of Computer Science and Technology
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: ydk21@cam.ac.uk

June 28, 2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Recent advances in deep learning have made unprecedented progress in various applications such as Computer Vision (CV) [1], natural language processing [2], and ubiquitous computing [3]. In particular, in mobile sensing applications that hinge on a fine-grained and continuous collection of a data stream from sensors, many studies adopt a deep learning approach and demonstrated its effectiveness and feasibility. With the rise of mobile, wearable devices, and the Internet of Things (IoT), the proliferation of sensory type data fostered the breakthroughs of deep neural networks in many mobile sensing applications. For example, researchers use deep neural networks based on sensory data in such mobile sensing tasks as human activity recognition [4, 5], gesture recognition [6], tracking and localization [7], mental health and wellbeing [8], and audio sensing applications [9]. However, a vital component common to the above applications is the need for the deployed model to adapt to a dynamically changing environment and to accommodate new user inputs and classes over time. In other words, the ability to learn *continually* [10, 11, 12] (i.e., to learn consecutive tasks without forgetting previously learned tasks) becomes crucial in scenarios where a learned model is deployed in real world.

In general, Machine Learning (ML) models typically suffer from Catastrophic Forgetting (CF) [13, 14], that is, a learned model experiences performance degradation on previously learned task(s) (e.g., task A) as information relevant to a new task (e.g., task B) is incorporated. Hence, many researchers have developed various Continual Learning (CL) methods designed to solve the CF issue by learning new tasks/classes over time while retaining previously learned knowledge [15].

However, enabling CL in mobile computing imposes unique challenges. First of all, there exists a need to conduct comprehensive empirical evaluations of various CL methods in

mobile sensing tasks. Since many CL methods are developed in the CV area [16, 17, 15], it is unclear whether the same techniques can be applied to another data modality different from images. For example, Human Activity Recognition (HAR) relies on Inertial Measurement Unit (IMU) data [18], Gesture Recognition (GR) relies on Surface Electromyography (sEMG) [19], Emotion Recognition (ER) relies on audio and speech signals [20]. As argued in [9], the signal steams on which mobile sensing data is primarily based consist of a one-dimensional time-series signal, fundamentally different from two-dimensional images. Moreover, it is essential to capture the signal's sequence information for representing such signal inputs. Therefore, It is crucial to examine how different categories of existing CL methods perform in mobile sensing tasks and then investigate the trade-offs among them.

Next, the nature of limited computational resources on mobile systems is another challenge that I address in my thesis. Unlike conventional model training and updates where the learning happens on powerful GPU clouds with large memory and storage sizes (e.g., 10 GB RAM and 1 TB storage), the system resources on a mobile system are severely limited. For example, many mobile and wearable devices contain memory and storage sizes of around 512 MB or moderately larger sizes. Furthermore, with the rise of the IoT, where billions of tiny microcontrollers (MCUs) have been deployed in the real world, we need a system that can address the extremely resource-constrained device [21, 22]. For example, tiny MCUs such as STMF767ZI have only 512 KB Static Random-Access Memory (SRAM) for intermediate data and 2 MB on-chip embedded flash (eFlash) memory for program storage. Hence, CL on MCUs, where it has a few KiloBytes (KB) of memory and storage size, is a very challenging and essential task for facilitating billions of tiny mobile and embedded devices (i.e., on-device systems) being able to adapt to frequently changing environments without the help of powerful servers. Besides, updating models manually from scratch would be very tedious and time-consuming, considering a massive number of the deployed devices. Hence, more intelligent on-device systems must be deployed in the real world and updated to the environment continuously.

Among CL methods, the exemplar-based method shows superior performance with an additional storage budget for saving some representative samples (can be raw data or intermediate representation) of the learned classes [17, 23]. However, on resource-constrained devices with small memory and storage sizes, it has not been fully understood how CL methods ensure high performance under such extremely resource-constrained environments. Hence, it is critical to investigate the feasibility of CL under limited resources with respect to compute overhead, memory and storage requirements. Based on this study, it is needed to propose a better approach to ensure the high performance of CL and to reduce its computational overhead and memory/storage requirements.

Finally, CL is also limited in a way that it requires a relatively large amount of labeled data to learn new tasks/classes [17, 23, 15]. In particular, in mobile sensing tasks, it is drastically more difficult to collect a labeled data from an incoming stream of the user

inputs than images that can be labeled after they are collected [24]. In addition, the more training data is required to do CL, the more computational resources are needed. For example, larger memory and storage sizes are needed to accommodate more training data, and more computation is required to update the model eventually. Thus, it is essential to enable CL with a small amount of labeled data to minimize the expensive manual labeling, computational costs, and system requirements.

1.2 Objectives

Motivated by the aforementioned limitations and challenges of applying CL in mobile computing, the main objective and scope of this thesis are *to develop an efficient on-device system that can sense, learn, and optimize Continually in the wild.* I attempt to achieve this objective by first investigating the advantages and disadvantages of current CL methods in mobile sensing tasks. Then, I plan to overcome the strict resource constraints of mobile and embedded systems by optimizing the computational costs and memory and storage usages and developing a better compression technique. Finally, I aim at enabling an adaptive CL system by incorporating meta-learning [25] so that the system relies on minimal human intervention.

Exploration of CL in mobile sensing applications

As described in Chapter 1.1, prior works in the literature are primarily focused on image modality, making it unknown how the existing CL methods perform in different modalities of mobile sensing applications based on a stream of signals (e.g., accelerometers, speech, and sEMG). Therefore, it is crucial to ask the following research questions: *how accurately do existing CL methods of different categories perform on various mobile sensing tasks?, what are the trade-offs among different categories of CL methods?* In addition, to address the resource constraints of the mobile and embedded systems, I also asked the following research question: *is CL feasible under a strict resource constraint in terms of computational overhead, memory, and storage size?*

Hence, I performed the first systematic study to examine the CF problem on mobile and embedded sensing applications using various CL methods. First of all, I employed six CL methods from three different CL categories to evaluate their effectiveness and efficiency on six datasets from different modalities of mobile sensing tasks. Second, To investigate the system limitations imposed by different configurations of CL, I implemented the CL framework on two types of devices with different specifications – an Nvidia Jetson Nano GPU (used in mobile robotics and tablets) and a smartphone (One Plus 7 Pro) CPU – with respect to computational costs, storage, and memory footprint. Finally, I examined and discussed trade-offs of studied CL methods regarding their performance, storage footprint, computational costs, and the peak memory limit to consider the feasibility and

applicability of the CL methods on mobile and embedded devices.

More detailed description, results, and findings of this work (exploration of CL in mobile sensing applications) are presented in Chapter 3 and in Appendix A. I summarized the results and findings of this work and submitted it to SEC '21 (The Sixth ACM/IEEE Symposium on Edge Computing). Currently, the paper is under review.

Efficient CL systems on resource-constrained devices

Based on the work (Appendix A) introduced above, I identified that although CL methods can operate on Jetson Nano and a smartphone, they are computationally very costly. In particular, I found that a representative exemplar-based method (iCaRL), which outperforms other CL methods, requires a large budget for storing raw data samples. Thus, I asked the following question: *is there a better way to reduce the overhead of resources and computations in enabling CL without sacrificing its accuracy?*

Thus, I developed an end-to-end framework, FastICARL, by enabling efficient and accurate on-device CL. FastICARL improves upon iCaRL by reducing the CL time and alleviating the storage requirements to store samples. First, FastICARL utilizes a k-nearest-neighbor and a max heap data structure to search exemplars more efficiently. After that, I further optimized FastICARL by applying quantization on exemplars to reduce the storage requirement. In addition, I implemented FastICARL on two mobile and embedded devices, Jetson Nano and a smartphone (Google Pixel 4). I experimented with FastICARL in two audio sensing applications to show its effectiveness and efficiency as a case study.

More detailed description, results, and findings of this work (FastICARL) are presented in Chapter 3 and in Appendix B. Moreover, this work is published at INTERSPEECH '21 (Conference of the International Speech Communication Association).

Exploration of better compression techniques for CL systems of multiple heterogeneous networks

In the work introduced in the subsection above (Appendix B), I have conducted an initial investigation of how and to what extent a quantization of exemplars affects the overall performance of CL (for exemplar-based CL methods). However, it has not been fully understood about the relationships between various compression techniques (see Chapter 2.3 for details) and CL methods in terms of performance, latency, and memory/storage overheads. Furthermore, compression of both the CL model's weights and its stored exemplars can severely deteriorate the model's performance, which is a challenge that I plan to tackle. Moreover, mobile and embedded systems deployed in the wild often need to run multiple applications. Since such systems' memory is limited, retaining a pre-trained model for each application is not scalable nor practical, although applying compression on each model can mitigate memory/storage issues. Sharing network structure [26] can be a solution for *correlated and similarly structured models*, however, it is not practical when systems want

to operate *multiple heterogeneous models*. Therefore, I ask research questions as follows: *is it feasible to compress multiple heterogeneous models without sacrificing accuracy?* and *how can I enable compression of heterogeneous CL models and their exemplars without losing the ability to adapt to new classes and to remember previous knowledge?*

I plan to tackle this problem with the following steps.

1. I start by exploring a variety of techniques that can compress multiple heterogeneous models such as weight memory virtualization [27] or product quantization (PQ) [28] and identifying their benefits and disadvantages in compressing both multiple models and exemplars.
2. I plan to develop better approaches to further compress the multiple models and their exemplars by extending the existing compression techniques. For example, PQ can be further optimized by removing redundant items in its codebook, which contains prototypical weight vectors of model weights. One idea is to apply a pruning framework [29, 30] to the found codebook.
3. I plan to incorporate this work of compressing multiple heterogeneous models into the CL framework. Then, the resulting CL system can operate multiple models with less memory/storage burden and update its models at any time.

Adaptive CL systems with minimal human intervention

Many prior works in CL require a relatively large amount of labeled data to learn new tasks/classes [17, 23, 15]. For mobile sensing tasks, it is more complicated to collect a labeled data different from image data. This point leads me to ask the following questions: *how can I reduce the amount of labeled data for learning new incoming classes to enable CL while ensuring high performance? To what extent can I decrease the labeled data size, and what are the trade-offs between the amount of labeled data and the CL model's performance?*

Meta-learning (i.e., few-shot learning) [25, 31, 32, 33, 34] gains an increasing interest in the machine learning community due to its fast adaptation speed and impressive performance on unseen classes. Also, some initial explorations [33, 34] attempt to combine two fields, meta-learning and continual learning, called Meta Continual Learning (Meta CL). Among them, Neuromodulated Meta-Learning Algorithm (ANML) [34] shows an impressive performance. It retains high accuracy until the model is fine-tuned up to 600 classes in the meta-testing phase.

Meta CL opens up a new door for solving a major bottleneck of enabling a CL system (i.e., expensive computational costs for model update) and continuously adapting to new user inputs with minimal human intervention. Since Meta CL can learn new classes/tasks with only a few labeled samples, updating the model does not take much time compared

to a conventional CL. However, since Meta CL relies on only a few samples for the model update for learning new classes, it could have a very high variation of performance in that the model can fail to learn a new class. Thus, to ensure to produce a reliable performance (i.e., low variance) is also a significant challenge to tackle.

Hence, I want to tackle this research direction as follows.

1. I begin by exploring ANML and identify its benefits and disadvantages in the area of mobile sensing.
2. I plan to resolve any disadvantages of the Meta CL method and also extend it from a regularization-based method [16] to an exemplar-based method [17] (see Chapter 2.1 for more details about different CL categories) by saving some representative samples in the form of hidden features of an intermediate representation. By doing so, it is expected that the model is able to ensure higher performance and potentially lower variance since it is trained with not only new inputs but also stored samples. Also, as observed in Appendix A and [17, 35, 36] exemplar-based methods outperform regularization-based methods.
3. However, storing some samples comes with a storage/memory overhead, which can severely burden resource-constrained devices like embedded systems, wearables, and MCUs. Thus, I plan to incorporate compression techniques, explored in the above subsection, into the Meta CL framework so that my new method can obtain the best of both worlds: (i) high and reliable performance and (ii) low memory/storage overhead.

1.3 Structure of the Report

In this chapter, I describe the motivations, objectives, and scope of this thesis. Chapter 2 summarizes the related work and Chapter 3 presents the works that I have conducted in the first year of my Ph.D. After that, Chapter 4 presents the tentative timeline and summaries of future work that I plan to pursue for the rest of my Ph.D.

Chapter 2

Related Work

2.1 Continual Learning

Continual Learning (CL) studies the ability to learn over time from a coming stream of data by incorporating new knowledge while retaining previously learned knowledge [15]. CL is also called incremental learning (IL) [17], lifelong learning [15], and sequential learning [13]. In a CL setup, learning methods typically suffer from CF [13, 14], that is, a learned model experiences performance degradation on previously learned task(s) (e.g., task A) as information relevant to a new task (e.g., task B) is incorporated. It is because the learned parameters of the network that are optimized to perform well in task A (i.e., important weights to task A) are changed to maximize/minimize the objective/loss of task B. In addition, there exist various fields that are related to CL in the ML literature. First of all, *Multi-Task Learning (MTL)* attempts to solve the problem of learning across a variety of tasks [37, 38] at the same time while exploiting commonalities and differences across tasks. MTL assumes that simultaneous access of the model to all the tasks at once is possible, and MTL does not include continuous adaptation after the model has been deployed in contrast to CL. *Transfer Learning* aims to improve the performance of a target task by using knowledge learned from a source task or domain [39, 40]. Similarly, in *Domain Adaptation*, the source and target tasks are the same, but the data is drawn from different domains [41]. Khan et al. [40] used domain adaptation to allow HAR models trained in one context to be readily adapted to a different contextual domain. However, the key difference of domain adaptation and transfer learning compared to CL is that the performance on the source task(s) is not considered during training in both transfer learning and domain adaptation. Furthermore, there are *Zero-shot Learning* [42, 43], *One-shot Learning* [44, 45, 46], or *Few-shot (Meta) Learning* paradigms [25, 31] which aim to improve the performance of a model on new or unseen tasks but does not consider the performance of the model on previously learned tasks.

In recent years, many researchers have focused on solving the CF issue by proposing a range of CL approaches. The first group of approaches is a *regularization-based* method [16, 47, 48, 49, 50, 51] where regularization terms are added to the loss function to minimize changes to important weights of a model for previous tasks to prevent forgetting. Kirkpatrick et al. [16] proposed Elastic Weight Consolidation (EWC) which uses the Fisher matrix to estimate the importance of each weight of a model and thus adjust the update of the weights based on its importance. Besides, Zenke et al. [47] proposed Synaptic Intelligence (SI) which maintains an online estimate of the weights' importance with respect to its contribution to the change in the loss function. Another group of approaches is a *replay-based* method [52, 53] where model parameters are updated by complementing the training data for each new task with "pseudo-data" representative of the previous task. Li and Hoiem [52] developed a Learning without Forgetting (LwF) that labels the input data of the current task using the model trained on the previous tasks and uses them pseudo-data. The replayed data can help the model's output stay close to that of the model trained on the previous tasks. Besides, Shin et al. [53] proposed a Generative Adversarial Networks (GAN) [54] based CL method that utilizes a GAN to generate data from past experiences when learning on new data. Lastly, *replay with exemplars-based* methods [17, 55, 36, 23, 35] require training data from the new class and also few training samples from earlier classes to update the model. Rebuffi et al. [17] proposed an Incremental Classifier and Representation Learning (iCaRL) that stores a small set of raw data of previous tasks as exemplars to prevent forgetting and learn a new task. In addition, Lopez-Paz and Ranzato [55] proposed a new CL method, Gradient Episodic Memory (GEM), that utilized episodic memory to enable CL by minimizing negative backward transfer (i.e., forgetting).

The proposed CL methods to solve CF are empirically evaluated using small and large datasets [11, 12]. However, these empirical studies either adopt only a few methods [11, 12] or neglect resource constraints of mobile and embedded devices with respect to storage and latency [12]. To fill this gap, we perform a systematic study on six most cited (or state-of-the-art) CL methods from three representative categories of CL approaches (see Chapter 3 and Appendix A for details).

2.2 Mobile and Embedded Sensing Applications

Deep learning is increasingly being applied in mobile and embedded systems as it achieves state-of-the-art performances on many sensing applications such as activity recognition [18, 56], gesture recognition [57], and audio sensing [58]. First of all, one of the most widely studied mobile sensing application is HAR [18, 59, 60], where the aim is to determine various human activities automatically (e.g., from a simple activity like walking to a complex activity like cooking a meal) based on smartphone and wearable (body-worn)

IMU sensors. Hammerla et al. [18] experimented with three variants of deep learning approaches such as feed-forward, convolutional, and recurrent neural networks on HAR datasets, and then presented guidelines for training neural networks. Moreover, Ordóñez and Roggen [61] proposed the DeepConvLSTM model in which convolutional layers extract the features from raw IMU data, and Long-Short Term Memory (LSTM) recurrent layers capture temporal dynamics of feature activations to improve the performance of HAR.

Another application frequently used in mobile sensing is to recognize hand gestures (e.g., fist and open palm) using sEMG signals generated during muscle contractions [62, 63, 6, 64, 19]. sEMG signals are measured quantitatively in a non-invasive fashion by estimating the electrical potential differences between muscle and ground electrodes. sEMG signal is used for medical [65], rehabilitation [66], human-computer interactions [67], and upper-limb prostheses control [68]. Zhai et al. [69] proposed a self-recalibrating framework that can be updated to maintain the model's performance so that it does not need users' additional labels for re-training. Shatilov et al. [70] developed a low-cost and adaptable system for prosthetic hand by incorporating an sEMG sensor, a mobile phone, a cloud component, and a 3D printed arm. Finally, Becker et al. [64] used sEMG of the forearm to classify finger touches with their proposed neural architecture combining convolutional, feed-forward, and LSTM layers.

Last but not least, the audio sensing application are also one of the foundational mobile sensing applications that much research has focused on to deliver behavioral insights to users. The audio sensing tasks include Emotion Recognition (ER) [20], Speaker Identification [71], Speaker Counting [72], Environmental Sound Classification (ESC) [73], and Conversation Analysis [74], and Keyword Spotting (KWS) [75, 76]. Lu et al. [77] proposed a scalable framework, called SoundSense, to recognize meaningful sound events that occur in users' everyday lives. The authors designed the general-purpose sound sensing system to work on resource-limited phones. Lee et al. [74] developed SocioPhone, a new mobile platform for face-to-face interaction monitoring to facilitate group conversations by utilizing meta-linguistic contexts of conversation. Moreover, Georgiev et al. [78] proposed a deep learning modeling and optimization framework that explicitly targets various audio sensing tasks in resource-constrained embedded systems.

In contrast to these works, I investigated to what extent current CL methods can enable a practical CL system for mobile and embedded sensing applications on-device and examined the implications of such systems regarding performance, efficiency, generalizability. In addition, throughout my thesis, I plan to continue to examine and improve upon the capability of CL systems on-device in the context of mobile sensing applications in terms of both performance and efficiency (e.g., smaller memory and storage footprint with less latency).

2.3 Model Compression

While more and more deep learning models show impressive performance, it comes with the peril of a tremendous amount of deeper network architecture and training data [79]. Although higher performance is critical for many applications, high model complexity demands much more computation resources and often hits the hardware memory limit on resource-constrained devices such as mobile and embedded devices or MCUs [80]. Hence, many researchers focus on developing a method to improve efficiency without sacrificing the model's accuracy.

First of all, many researchers have focused on designing and hand-drafting more efficient network architectures, namely, SqueezeNets [81, 82], ShuffleNets [83, 84], and MobileNets [85, 86]. In particular, Howard et al. [85] proposed MobileNet, a new network architecture, that utilizes depth-wise separable convolutions. MobileNet is based on the depth-wise separable convolutions which factorize a standard convolution into a depthwise convolution and a 1×1 convolution (pointwise convolution), making MobileNet consumes $8 - 9$ times less computation than standard convolutions. Sandler et al. [86] designed a novel layer called the inverted residual with a linear bottleneck on which MobileNetV2 is based and further improved the performance and efficiency over MobileNet. Recently, Neural Architecture Search (NAS) methods [87, 88, 80, 89, 90, 91, 92, 93, 94] gain popularity in designing efficient neural architectures by automating the process of finding network structures. NAS often achieves better efficiency than hand-drafted neural architectures. For example, Liu et al. [87] introduced a novel algorithm for differentiable NAS based on bilevel optimization which can be used to both convolutional and recurrent layers. Also, Cai et al. [90] developed a progressive shrinking algorithm that first trains a large neural network and then progressively fine-tune the network to support smaller sub-networks that share weights with the larger ones.

Secondly, another thread of research is weight pruning methods that leverage the inherent redundancy in the weights of neural networks [95, 29, 96, 97, 98, 30, 99, 100, 101, 102]. Han et al. [95, 29] proposed a iterative pruning framework that removes weights having small magnitudes and retrain the model until the model reaches a certain threshold in terms of performance or model sizes. In addition, Yang et al. [96] introduced the energy-aware pruning, and Liu et al. [98] utilized a structural regularity of the neural networks and Alternating Direction Method of Multipliers (ADMM)-based pruning framework [99], showing an impressive result.

Furthermore, quantization of model weights and activiations has been an active area of research. Many prior works quantize the weights and activations from 32-bit float to 8-bit integer [103], ternary values (2-bit) [104, 105], binary values (1-bit) [106, 107, 108, 109], and mixed precision [110, 111]. Also, weight clustering methods are proposed to group weights into several clusters to compress a model. Chen et al. [112] proposed a hashing

trick to group each weight, and Han et al. [29] incorporated the weight clustering into the iterative pruning framework. Moreover, researchers studied techniques that quantize an array of scalars of the weights to compress a model or a particular layer. Some works extended a sparse coding [113] to learn a compact representation that covers the feature space of weights of a model [114, 115]. Bagherinezhad et al. [115] proposed LCNN, lookup-based convolutional neural network, encoding convolutions by a few lookups to a learned dictionary that covers weights' feature space. However, while sparse coding-based methods could well represent a feature space of weights, it requires non-trivial memory to store the learned dictionary as well as its indices of weights to the dictionary. Also, many researchers examined vector quantization-based methods. For example, Gong et al. [116] conducted an empirical study to compare binarized networks, scalar quantization using k-means (i.e., weight clustering), Product Quantization (PQ) [28]. The authors found that PQ shows the best result among the studied methods. Besides, Wu et al. [79] showed $4-6\times$ speed-up and $15-20\times$ compression with one percent loss of accuracy by applying PQ together with the error-correction mechanism. In addition, Hayes et al. [23] utilized PQ to compress an intermediate layer of a network as exemplars to maintain the high performance of CL. Chen et al. [117] proposed a differentiable PQ to compress an embedding layer in Natural Language Processing which is typically very large in terms of size.

As shown in the works described above, the vector quantization technique (PQ) demonstrates the effectiveness and potential in weight compression. Nevertheless, all the prior works are limited in the utilization of PQ to a single model or a layer (which is on a scale of tens of millions of weights). Given that PQ is typically used to index billions of vectors for approximate nearest neighbor search in database community [28, 118, 119], I argue that PQ has not been fully utilized in model compression. Also, PQ can be used to compress multiple heterogeneous networks [27]. Finally, the learned PQ codebook might be able to generalize well to unseen model weights. Those are the questions that I will attempt to answer in my future work. Note that the scope of my work is limited to lossy compression to maximize the compression rate (although lossy compression can incur accuracy loss) as Ko et al. [120] showed that the lossless compression technique alone achieves a relatively low compression rate ($\sim 2\times$) without accuracy loss.

2.4 Meta Continual Learning

In actual deployment scenarios, it is very tough to obtain a large number of labeled training data to update deployed models or add new classes to them. It is because users are only willing to annotate a few samples for the new classes. As we discussed in Chapter 1, the annotation problem is even more challenging in mobile sensing applications since it is also not straightforward to label a stream of signals. However, conventional CL methods still require much training data to prevent CF while learning new classes.

Thus, Meta CL or Few-shot Class-Incremental Learning (FSCIL) [34, 33, 121, 122, 123, 124] has been proposed to overcome the limitation mentioned above. This method aims to solve two challenges: (i) to avoid CF of old classes, (ii) to prevent overfitting to few samples of new classes. Tao et al. [123] proposed a novel FSCIL framework that employs a neural gas (NG) network [125] to learn the topology of feature space to represent learned knowledge and solves FSCIL problem by adjusting the update of the NG network. In addition, Javed and White [33] demonstrated that the proposed method, OML, can successfully avoid interference and ensure learning of new knowledge up to 200 classes by continually learning through a meta-objective. Then, Beaulieu et al. [34] extended the work of Javed and White [33] using a neuromodulatory (NM) network that learns selective activation of another network that performs a prediction. Beaulieu et al. showed that their method could successfully solve CF problems up to 600 classes.

In my thesis, I plan to improve upon existing works on Meta CL and FSCIL by utilizing additional storage/memory to store representative samples or intermediate layers (i.e., extend regularization-based methods to exemplar-based methods) to ensure high performance. After that, to minimize the resource overhead, I will incorporate model compression techniques in my Meta CL framework.

Chapter 3

First-year Contributions

As I have described in Chapter 1, I began with analyzing the feasibility and applicability of CL methods in a variety of mobile sensing applications while considering the limits poised by mobile and edge platforms, namely, low computational power, smaller memory and storage. Then, I employed six CL methods from three different CL categories to evaluate their effectiveness and efficiency on six datasets from different modalities of mobile sensing tasks. In detail, I employed three datasets from the widely researched application of Human Activity Recognition (HAR) [126] based on accelerometer, gyroscope, and magnetometer data. Next, I included two datasets from Gesture Recognition (GR) [69] based on surface electromyography (sEMG). Finally, I further incorporated an Emotion Recognition (ER) dataset [20] based on speech among audio sensing tasks to make our results generalizable to different modalities across diverse applications. Also, the six CL methods fall under three paradigms: *regularization* ((1) Elastic Weight Consolidation: EWC [16], (2) Synaptic Intelligence: SI [47], and (3) Online EWC [48]), *replay* ((4) Learning without Forgetting: LwF [52]), and *replay with exemplars* ((5) Incremental Classifier and Representation Learning: iCaRL [17] and (6) Gradient Episodic Memory: GEM [55]). After that, I implemented an end-to-end CL framework on two kinds of devices with different specifications: Jetson Nano and an off-the-shelf smartphone. Through the extensive evaluation, I identified that all the CL methods can largely mitigate the CF issue in relatively easier CL scenarios. Whereas, in the difficult CL scenarios, only exemplars-based CL method can successfully solve CF problem while other CL methods are not effective in preventing CF. This work became a good starting point helping me understand the challenges and limitations of current CL methods when applied to mobile sensing applications on resource-constrained devices. The result of the first work is presented in Appendix A. Currently, the paper is under review at SEC '21 (The Sixth ACM/IEEE Symposium on Edge Computing)

Based on the findings of the first work, I realized that one of the major bottlenecks of

enabling end-to-end CL on-device is an expensive computational requirement to learn new user inputs/classes (e.g., activities in HAR, gestures in GR). Furthermore, the high-performing CL method, iCaRL (exemplar-based method), requires a large storage budget to store representative samples of learned classes. Motivated by these limitations of prior works, I proposed a novel CL method, FastICARL, that improves upon iCaRL by reducing the CL time as well as alleviating the storage requirements to store samples. To develop FastICARL, I first optimized the construction process of an exemplar set (which takes most of the CL time) to shorten the CL time to tackle the limitation on computational overhead. Specifically, to find the informative exemplars that can best approximate feature vectors over all training examples, ICARL relies on herding which contains inefficient double for loops. Instead, FastICARL utilizes a k-nearest-neighbor and a max heap data structure to search exemplars more efficiently. In addition, to address the limitation on storage burden in resource-constrained devices, I further optimized FastICARL by applying quantization on exemplars to reduce the storage requirement. I converted the 32-bit float data type into 16-bit float and 8-bit integer data types. Furthermore, I implemented our end-to-end CL framework on mobile and embedded devices of two different specifications: Jetson Nano and a smartphone (Google Pixel 4). To demonstrate its effectiveness and efficiency, I experimented with it in two audio sensing applications: Emotion Recognition (ER) task and an Environmental Sound Classification (ESC) as a case study. This work is published at INTERSPEECH '21 (Conference of the International Speech Communication Association)

Papers Submitted

Exploring System Performance of Continual Learning for Mobile and Embedded Sensing Applications
Young D. Kwon, Jagmohan Chauhan, Abhishek Kumar, Pan Hui, and Cecilia Mascolo.
The Sixth ACM/IEEE Symposium on Edge Computing, 2021. (SEC '21)

Papers Published

FastICARL: Fast Incremental Classifier and Representation Learning with Efficient Budget Allocation in Audio Sensing Applications
Young D. Kwon, Jagmohan Chauhan, Cecilia Mascolo.
Conference of the International Speech Communication Association, 2021. (INTERSPEECH '21)

Chapter 4

Thesis Proposal

My research focuses on building an efficient on-device system that is exceptionally lightweight and capable of updating itself to changing environments and user inputs continually with minimal human intervention. The following is a proposed timeline outlining the targets and milestones for my research.

1. Literature review: Months 1-4 (Done.)
    - Review the literature regarding CL and improve my knowledge about deep neural networks.
    - Understand the knowledge about mobile sensing, embedded systems.
2. Initial exploration of CL in mobile sensing. Months 5-9 (Done.)
    - Conduct the foundational research to identify the feasibility and applicability of applying various CL methods in mobile sensing applications. (under review at SEC '21)
    - Learn practical guidelines for applying CL in mobile sensing tasks on resource-constrained devices.
    - Propose a novel exemplar-based CL method called FastICARL. FastICARL reduces the latency for the CL time and requires less storage than the original iCaRL (published at INTERSPEECH '21).
3. Implementation of existing works and further literature review: Months 10-12 (In Progress.)
    - Extensively review model compression literature, e.g., NAS, pruning, quantization, weight sharing.

- Implement some of the prior works to fully understand them and later use them as baselines of my future work.
- Review meta-learning and meta CL literature.
- Implement representative meta CL works to use them as baselines.

4. Exploration of better compression techniques for CL systems of multiple heterogeneous networks: Months 13-18
    - Start with applying the basic PQ to a single model to see if it retain the accuracy and to what extent it can reduce a storage and memory footprint.
    - Extend PQ to be applied to multiple models and evaluate the performance of the PQ with respect to the accuracy, memory, storage footprint, and energy consumption.
    - Come up with optimization techniques based on PQ or on other compression techniques to further shrink the memory/storage footprint of multiple heterogeneous models.
    - Evaluate the optimized compression technique to what extent it can improve upon the baseline methods in terms of performance and system aspects.
    - Evaluate my method in various application scenarios. For example, when a user wants to include new models in my weight sharing method, it can represent the unseen models without learning from scratch.
5. Adaptive CL systems with minimal human intervention: Months 19-24
    - Analyze the advantages and disadvantages of the current state-of-the-art meta CL method (ANML).
    - Extend the regularization-based meta CL framework to exemplar-based meta CL to maximize the performance with some additional overheads on memory and storage.
    - Explore the possible optimizations based on quantization, pruning, and joint optimization to reduce resource usages of the exemplar-based meta CL.
6. Combination of the new compression technique and Meta CL in the wild: Months 25-30
    - Explore the possibility of combining my new compression technique and the exemplar-based meta CL to enable continual update rapidly with less memory and storage overhead in the context of multiple heterogeneous neural networks.

- Conduct extensive evaluations of the advantages and disadvantages of the CL system built with the two techniques in the wild.

7. Thesis writing: Months 31-36

Appendix A

Exploring System Performance of Continual Learning for Mobile and Embedded Sensing Applications

Abstract

Continual learning approaches help deep neural network models adapt and learn incrementally by trying to solve catastrophic forgetting. However, whether these existing approaches, applied traditionally to image-based tasks, work with the same efficacy to the sequential time series data generated by mobile or embedded sensing systems remains an unanswered question.

To address this void, we conduct the first comprehensive empirical study that quantifies the performance of three predominant continual learning schemes (i.e., regularization, replay, and replay with examples) on six datasets from three mobile and embedded sensing applications in a range of scenarios having different learning complexities. More specifically, we implement an end-to-end continual learning framework on edge devices. Then we investigate the generalizability, trade-offs between performance, storage, computational costs, and memory footprint of different continual learning methods.

Our findings suggest that replay with exemplars-based schemes such as iCaRL has the best performance trade-offs, even in complex scenarios, at the expense of some storage space (few MBs) for training examples (1% to 5%). We also demonstrate for the first time that it is feasible and practical to run continual learning on the device with the tight memory budget. In particular, the latency on two types of mobile and embedded devices suggests that both incremental learning time (few seconds - 4 minutes) and training time (1 - 75 minutes) across datasets are acceptable, as training could happen on the device when the embedded device is charging thereby ensuring complete data privacy. Finally, we present some guidelines for practitioners who want to apply a continual learning paradigm for mobile sensing tasks.

A.1 Introduction

Deep learning has revolutionized the performance of various disciplines, including mobile and embedded systems applications. This is particularly true for applications relying on continuous streams of sensor data such as activity recognition [4], mental health, and wellbeing [8], gesture recognition [6], tracking and localization [7]. However, a crucial characteristic common to the above applications is the need for a trained model to adapt to accommodate new classes and to a dynamically changing environment. In these settings, the ability to *continually* learn [11], that is, to learn consecutive tasks without forgetting how to perform previously learned tasks, becomes essential. Let us consider an example. Alice has a deep learning model deployed on her smartphone for human activity recognition (HAR) to recognize simple activities such as sitting, and standing. As time passes, the model might want to learn new activities such as walking to be more beneficial to a very active Alice. A static model will learn new activities but will fail to predict older activities correctly due to *catastrophic forgetting* (CF) [13], i.e., the abrupt and near-complete loss of knowledge obtained from previous tasks when the model learns new tasks. Continual learning allows deep learning models to learn incrementally (adapt or accommodate new classes/behaviors) and obviates the need to be trained every time from scratch, which might waste useful resources on Alice's device.

In practice, enabling deep learning models to continually learning is very challenging due to the CF problem. Since CF was first identified in Multi-Layer Perceptrons (MLPs), many researchers have proposed methods to mitigate it [52, 16, 17, 127, 55] and evaluate it using small and large datasets [11, 12, 128]. However, the proposed methods are mainly evaluated in the field of computer vision with MLPs or Convolutional Neural Networks (CNN) based deep learning models. *It is unclear whether these methods are viable in sensor-based applications, where the modality of the data is significantly different from images, and sequence information needs to be captured [9].* Moreover, *most of the existing Incremental Learning (IL)*[1] *techniques [129] do not take into account the resource requirements of these devices*, which may make them inapplicable to embedded and mobile systems deployments. There is a clear need to understand the resource consumption limitations of existing continual learning methods to see if they are applicable to resource-constrained edge platforms.

To address the aforementioned limitations of prior work, we conduct the first systematic study to investigate the CF problem on mobile and embedded sensing applications using various IL methods. **First,** we employ three datasets from the widely researched application of Human Activity Recognition (HAR) [126] based on accelerometer, gyroscope, and magnetometer data. Next, we include two datasets from Gesture Recognition (GR) [69] based on surface electromyography (sEMG). We further incorporate an Emotion Recogni-

[1] In this work, we use continual learning (CL) and incremental learning (IL) interchangeably.

tion (ER) dataset [20] based on speech among audio sensing tasks to make our results generalizable to different modalities across diverse applications. **Second,** we examine trade-offs of studied IL methods in terms of their performance, storage footprint, computational costs, and the peak memory limit to consider the feasibility and applicability of the IL methods on mobile and embedded devices. To investigate the system limitations imposed by different configurations of IL, we implemented the IL framework on two types of devices with different specifications – an Nvidia Jetson Nano GPU (used in mobile robotics and tablets) and a smartphone (One Plus 7 Pro) CPU – with respect to computational costs, storage, and memory footprint.

Overall, the major contributions and findings of this paper are:

First, we conduct a systematic investigation of the CF problem on mobile and embedded applications using six state-of-the-art IL methods falling under three paradigms: **regularization** ((1) Elastic Weight Consolidation: EWC [16], (2) Synaptic Intelligence: SI [47], and (3) Online EWC [48]), **replay** ((4) Learning without Forgetting: LwF [52]), and **replay with exemplars** ((5) Incremental Classifier and Representation Learning: iCaRL [17] and (6) Gradient Episodic Memory: GEM [55]). In addition, to make our study generalizable across different modalities of data, we perform analysis on six datasets of three different sensing applications (HAR, GR, and ER).

Second, to evaluate CF in real-life scenarios, we employ Sequential Learning Tasks (SLTs), successively learning two or more sub-tasks $D_1, ..., D_k$, instead of learning a single task D [12]. Learning new tasks continuously becomes vital since the number of classes (activities or users) and the environments of edge applications often change over time. We adopt a class-incremental learning setup where each task contains distinct classes, which fits well with practical application scenarios (see §A.3.1 for detail). Specifically, we try three scenarios: adding only one class to a base classifier (simple), adding half of the classes, $N/2$, to a base classifier at once (mildly complex), and a very practical (complex) scenario where half of the classes, $N/2$, are added incrementally to a base classifier one by one, where N is the total number of classes. Through extensive experiments, we find that all IL methods perform well when presented with simple scenarios but fail in the complex scenario, except for iCaRL. The main reason for iCaRL's strong performance is its use of exemplar samples. To the best of our knowledge, we are the first to train and implement IL methods to run on mobile and embedded systems, with the aim to build an end-to-end on-device continual learning system and to evaluate trade-offs of studied IL methods in terms of their performance, storage, and computational costs, as well as the peak memory usage.

Third, we find that iCaRL and GEM require a modest amount of storage, which seemingly is not an issue on many modern devices as they support a large amount of storage (in order of a few GBs). Even at a maximum number of stored exemplars (i.e., 20% - 40%

of training samples), iCaRL and GEM require only 2 MB–115 MB. Besides, the average IL time taken by EWC based algorithms varies from 39.5–1,160 seconds on Jetson Nano. For all other algorithms, it ranged from 0.5–342 seconds on both Jetson Nano and a smartphone. iCaRL, in particular, needs less than a minute on a smartphone to do IL on a per-task basis and operates within a reasonable peak memory overhead (196–2,127 MB). Our study also shows that simple deep learning architectures such as one and two-layer long short-term memory (LSTM) [130] can be trained entirely on the smartphone, thereby ensuring complete user privacy.

Finally, based on our findings, we present a series of lessons and guidelines to help practitioners and researchers in their use of continual deep learning for mobile sensing applications.

In addition to the above contributions, we adapt the experimental protocol proposed in [12] which considers learning only two tasks. We extend this protocol so that it can incorporate any number of tasks ($D_1, ..., D_k$) in an incremental manner and identify the best performing IL model by permutating a set of hyper-parameters and IL-method-specific parameters (see §A.3.3 for detail). Finally, we believe that our work and findings open the door to the use of continual learning in edge devices and applications.

A.2 Related Work

We begin by reviewing continual learning approaches and empirical studies to evaluate them, followed by applications of deep learning in the mobile and edge sensing domain.

A.2.1 Continual Learning

Continual learning studies the ability to learn over time from a coming stream of data by incorporating new knowledge while retaining previously learned experiences [15]. Continual learning is also called incremental learning (IL) [17], lifelong learning [15], and sequential learning [13]. In a continual learning setup, learning methods typically suffer from CF [13, 14], That is, a learned model experiences performance degradation on previously learned task(s) (e.g., task A) as information relevant to a new task (e.g., task B) is incorporated. It is because the learned parameters of the network that are optimized to perform well in task A (i.e., important weights to task A) are changed to maximize/minimize the objective/loss of task B. In recent years, many researchers have focused on solving the CF issue by proposing a range of IL approaches. The first group of approaches is a *regularization-based* method [16] where regularization terms are added to the loss function to minimize changes to important weights of a model for previous tasks to prevent forgetting. Another group of approaches is a *replay-based* method [52] where model parameters are updated for learning a representation by using training data of the

currently available classes, which is different from *replay with exemplars-based* method [17, 55] where updating the model requires training data from the new class and also few training samples from earlier classes.

The proposed IL methods to solve CF are empirically evaluated using small and large datasets [11, 12]. However, these empirical studies either adopt only a few methods [11, 12] or neglect resource constraints of mobile and embedded devices with respect to storage and latency [12]. To fill this gap, we perform a systematic study on six most cited (or state-of-the-art) IL methods from three representative categories of IL approaches with three continual learning scenarios with different difficulties. Also, we conduct the first comprehensive study of generalizability and trade-offs between performance, storage, and computational costs among the studied IL methods on mobile and embedded devices.

A.2.2 Deep Learning for Mobile Sensing Systems

Deep learning is increasingly being applied in mobile and embedded systems as it achieves state-of-the-art performances on many sensing applications such as activity recognition [18, 56], gesture recognition [57], and audio sensing [58]. [18] experimented with three variants of deep learning approaches such as feed-forward, convolutional, and recurrent neural networks on HAR datasets, and present guidelines for training neural networks. [61] proposed the DeepConvLSTM model in which convolutional layers extract the features from raw IMU data, and Long-Short Term Memory (LSTM) recurrent layers capture temporal dynamics of feature activations to improve the performance of HAR.

Deep neural networks have also helped applications that need to recognize hand gestures using surface electromyographic (sEMG) signals generated during muscle contractions [64, 19]. [69] proposed a self-re-calibrating framework which can be updated to maintain the model's performance so that it does not need users' additional labels for re-training. [64] used sEMG of the forearm to classify finger touches with their proposed neural architecture combining convolutional, feed-forward, and LSTM layers.

Many works have investigated using deep learning for audio sensing tasks including Emotion Recognition, Speaker Identification [131], and Keyword Spotting. [78] proposed a deep learning modeling and optimization framework that specifically targets various audio sensing tasks in resource-constrained embedded systems. Keyword recognition [76] achieved 45% relative improvement with a deep learning model compared to a competitive Hidden Markov Model-based system.

In contrast to these works, we investigate whether current IL methods can enable a practical continual learning system for mobile and embedded sensing applications on-device and what the performance implications of such systems are. In addition, to fully understand the issue of CF in mobile sensing where the modality of the data is significantly different from image datasets [9] with which the IL methods are typically evaluated, we implement

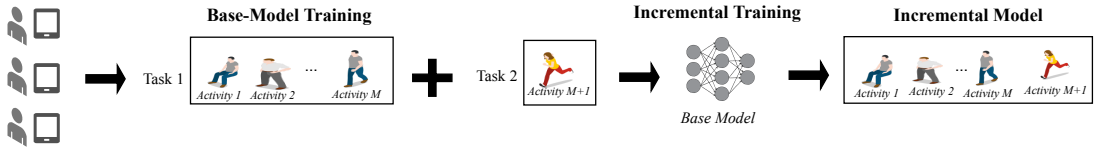


Figure A.1: Overview of our continual learning system.

an end-to-end continual learning framework that evaluates various IL methods in three embedded sensing applications (e.g., HAR, GR, and ER) with different data modalities (e.g., accelerometer, sEMG, and speech).

A.3 Continual Learning for Mobile and Embedded Sensing Framework

We now present our framework to comprehensively evaluate the performance of various IL methods for three mobile and embedded applications (HAR, GR, and ER). We first explain the continual learning setup and three scenarios adopted in our experiments (§A.3.1). Then, we present six IL methods evaluated in this work (§A.3.2). We then describe the hyper-parameters of the LSTM based deep learning model and the different IL methods (§A.3.3). After that, we propose our novel IL model training process in §A.3.4. Next, we describe the datasets used in this study (§B.3.1). Finally, we provide brief details about our implementation (§A.3.5)

A.3.1 Continual Learning Setup and Three Scenarios

In this work, we focus on Sequential Learning Tasks (SLTs) from the mobile and embedded systems domain where new classes can emerge over time. Thus, the learning model has to continuously learn to accommodate new classes without CF, as would happen in real-life scenarios. Learning tasks of this type, called SLTs, indicates that a model continuously learns two or more tasks $D_1, ..., D_k$, one after another instead of learning a single task D once [12]. Figure A.1 shows an overview of our continual learning system for sensing applications using HAR as an illustrative example. A user starts with a model containing a fixed set of classes on their devices which is then incrementally updated over time as new classes arrives.

We introduce three scenarios of different levels of difficulties for models to learn continuously (from easy to difficult scenarios). First of all, inspired by Pfulb et al. [12], we adopt the SLTs consisting of two tasks: D_1 and D_2. Hence, Scenario 1 consists of two tasks, where the first task contains the $N-1$ classes, and the second task contains the other one class (N is the total number of classes). Scenario 2 includes two tasks where the first task

contains half of the classes, $N/2$, and the second task contains the remainder of the classes. Finally, Scenario 3 deals with a more realistic situation where many tasks are to be learned sequentially [11]. In the third scenario, we first train a model in the first task with $N/2$ classes and then incrementally train the model by adding subsequent tasks with one class (essentially $N/2 + 1$ tasks). Unlike the first scenario (which has only N different cases of task permutations), it is not practical to consider every random permutation of classes to be included in different tasks for the second and third scenarios. Hence, we consider ten variations by randomly choosing classes in each task for the last two scenarios. Note that each task consists of disjoint groups of classes as we adopt class-incremental learning [132].

A.3.2 Incremental Learning Methods

As described in the related work section, various methods exist that can mitigate CF in IL. We describe them in depth as they form the basis of our exploration. To mitigate CF, there exist three main categories of IL approaches: (1) Regularization, (2) Replay, and (3) Replay with Exemplars. We select at least one representative method for each of the above categories. These methods are the state of the art methods (most cited) for IL and are most often used in machine learning papers for comparison. We now describe the employed methods.

LSTMs [130]: LSTMs are a type of recurrent neural networks widely used for a sequence classifier in many applications, specifically for time-series data. We use LSTMs as a base neural network.

EWC [16]: Elastic Weight Consolidation (EWC) is a regularization based method which adds a penalty to regular loss function when learning a new task (i-th task), i.e.,

$$L(\theta) = L_i(\theta) + \lambda/2 \sum_{j=0}^{i-1} F_j(\Theta_i - \Theta_j^*)^2 \tag{A.1}$$

where $L(\theta)$ is the total loss, θ is the network's parameters, $L_i(\theta)$ is the loss for the new task, and Θ_j^* are the important parameters of all previous tasks. λ is a hyperparameter that controls how much importance should be given to previous tasks compared to the new task. F is the Fisher matrix used to constrain the parameters important to previously learned tasks to stay close to their old values to retain the knowledge of previous tasks and to be able to learn new tasks simultaneously.

Online EWC [48]: It is a variation of EWC method where the loss function is represented as,

$$L(\theta) = L_i(\theta) + \lambda/2(\Theta_i - \Theta_{i-1}^*)^2 \sum_{j=0}^{i-1} F_j \tag{A.2}$$

Online EWC eliminates the need to store mean and fisher matrices for each previous task and only requires the latest mean and running sum of fisher matrices to calculate the current task's total loss.

SI [47]: It is another regularization method which is similar to EWC where the loss function is calculated in the following way,

$$L(\theta) = L_i(\theta) + \lambda \sum_k \Omega_k^i (\theta_k^* - \theta_k)^2 \tag{A.3}$$

where k is the subscript for the parameters of the models, λ is the strength parameter, θ_k^* is the parameter value at the end of the previous task, and Ω_k^i represents the per-parameter regularization strength taking into account all previous tasks, calculated as:

$$\sum_{j=0}^{i-1} \frac{w_k^j}{(\triangle \theta_k^j)^2 + \varepsilon} \tag{A.4}$$

parameter distance $\triangle \theta_k^j$ determines how much a parameter moved between tasks during the entire trajectory of training. ε is the dampening parameter to prevent division by zero errors. The main difference between SI and EWC is that SI weights importance, w_k, is continuously updated online during training. In contrast, in EWC, the Fisher matrices (weights importance) are calculated at the end of each task.

LwF [52]: This method relies on adding loss for the replayed data to the loss of the current task. The replayed data is the input data of the current task which is labeled using the model trained on the previous tasks to generate target probabilities. The ultimate aim of the replayed data is to match the probabilities predicted by the model being trained to the target probabilities (a form of data distillation) and is termed as the loss for replayed data.

iCaRL [17]: Incremental Classifier and Representation Learning (iCaRL) store data from previous tasks (i.e., exemplars) to alleviate the CF problem. The exemplars are a representative set of the small number of samples from a distribution, and those that can approximate the average feature vector over all training examples are selected as exemplars (based on herding [133]). The classification is done based on a nearest-class-mean (NCM) rule using features extracted from the deep learning model, where the class means are calculated from the stored examples. When new tasks (classes) arrive, iCaRL creates a new training set combining the exemplars from all the previous tasks with the data samples of the new task. Then, the model parameters are updated by minimizing a loss function which encourages the model to output the correct class for the new task (classification loss) and to reproduce the scores stored in the previous step for the old tasks (distillation loss) using data samples from the new training set.

GEM [55]: Gradient Episodic Memory (GEM) stores exemplars from the previous tasks like iCaRL and solves CF as a constrained optimization problem. A parameter update while doing IL is made depending on whether it will lead to an increase in loss for the previous tasks. This is calculated by computing the angle between loss gradient vectors of stored examples and the proposed parameter update. If the calculation suggests no loss, then the update is done straight away. Otherwise, the parameter is updated by projecting gradient in such a way that it will incur a minimal loss for the previous tasks.

Our Contribution: It is worth noting that the above six IL methods are known in the machine learning literature from a theoretical point of view. Yet, they are not off-the-shelf methods that can be simply used to any dataset to enable continual learning. As will be shown in Section A.5, there exist many factors affecting the performance and applicability of the IL methods in real-world deployment such as the complexity of the continual learning scenario, resource availability of mobile and embedded devices, and choice of hyper-parameters. Thus, a distinctive contribution of our work is a comprehensive evaluation and comparison study of the IL methods in diverse sensing applications and is to develop an end-to-end and on-device IL framework that can investigate trade-offs between performance, storage requirements, and latency.

A.3.3 Characterization of Hyper-parameters

We categorize hyper-parameters into three types and IL-method-specific parameters. First of all, we use architectural hyper-parameters which cannot be changed when learning new tasks, e.g., the number of hidden layers L and its size S. We then use learning and regularization hyper-parameters which can be adaptable when learning new tasks. For example, a learning rate ϵ and λ term in L2-regularization can be modified during training over time. We denote the set of hyper-parameters as $\mathcal{P}$.

IL-method-specific parameters: Each IL method has method-specific parameters to control the behaviors of the model. For example, in regularization-based methods [16, 134, 47, 52], importance parameter λ is often utilized to modulate how much importance a model puts on previous tasks or a current task. The importance parameter can be adaptable while learning new tasks in our IL model training process (Algorithm 1). In addition, in replay with exemplars-based methods [17, 55], the size of the storage budget is used to balance between storage requirements and the performance of a model. Since the budget size is difficult to be adaptable after completion of the first task, it is given as an input in our experimental protocol (Algorithm 1).

Hyper-parameter setting for experiments: We first fix several hyper-parameters as default values. We set dropout rates for all tasks as 0.2 and 0.5 in input and hidden layers of a model, respectively [135] and a batch size of 32 with Adam optimizer set to a default learning rate of 0.001 for task 1 (D_1). After that, we vary hyper-parameters

Algorithm 1: IL model training process to determine the best model by incrementally learning tasks up to task k

Input: Tasks $D_1, ..., D_k$, model m, budget $\mathcal{B}$, epochs $\mathcal{E}$
Input: The number of hidden layers L, Hidden layer size S
Input: IL-method-specific parameters $\mathcal{P}_{IL}$, learning rate ϵ
Output: The best model with hyper-parameter vector p^*

for $p \in (L \cup S)$ **do**
  for $t = 1, \mathcal{E}$ **do**
    Train model m_1 using training set of D_1 with p
    Test model m_1 using test set of D_1
    Store performance $q_{1,t}$
Update the model m_{1,p^*} with max q_1
for $p \in (\mathcal{P}_{IL} \cup \epsilon)$ **do**
  Initialize model m_2 with m_{1,p^*}
  for $j = 2, k$ **do**
    for $t = 1, \mathcal{E}$ **do**
      Train model m_2 using training set of D_j with p
      Test model m_2 using test set of $\cup_{l=1}^{j} D_l$
      Store performance $q_{j,t}$
Update the model m_{k,p^*} with max q_k

for all models in each dataset. Specifically, in the task 1 (D_1), we vary architectural hyper-parameters as follows: $L \subset \{1, 2\}$, $S \subset \{32, 64\}$. In subsequent tasks from task 2 to k ($D_2, ..., D_k$), we fix architectural hyper-parameters but vary adaptable hyper-parameters and IL-method-specific parameters as follows: (1) $\epsilon \subset \{0.001, 0.0001\}$ for all models, (2) $\lambda \subset \{1, 10, 10^2, 10^3, 10^4, 10^5, 10^6\}$ for both EWC and Online EWC, (3) $\gamma \subset \{0.5, 1.0\}$ for Online EWC, (4) $c \subset \{0.2, 0.4, 0.6, 0.8, 1.0\}$ for SI. We denote varying IL-method-specific parameters as $\mathcal{P}_{IL}$. For replay-based methods, the losses of the current and replayed data are weighted according to the number of tasks a model has learned so far by following [132]. Note that budget size, $\mathcal{B} \subset \{1\%, 5\%, 10\%, 20\%\}$, is given as an input and fixed for replay with exemplars-based methods while other hyper-parameters are permuted. Since the total number of samples for each dataset is different, we use a ratio from the total training samples rather than a fixed number of samples for the budget size.

A.3.4 Model Training Process

We extend protocol [12] to incorporate multiple tasks up to task k ($D_1, ..., D_k$) in an incremental manner based on our characterization of hyper-parameters and IL-method-specific parameters. Algorithm 1 describes our protocol in which we only utilize training

data of a current task j ($\leq k$) for model learning and test data of previously learned tasks up to task j for evaluation.

Given an SLT consisting of $D_1, D_2, ..., D_k$ and a model m, the goal is to find a vector of hyper-parameters p^* which produces the best performance q after incrementally training all tasks up to task k. For the first step, we find the best performing hyper-parameters in task 1 (D_1) by searching among the set of architectural hyper-parameters (lines 1-5 in Algorithm 1) and update the model m_{p^*} with the found hyper-parameters (line 6). The next step is to find the best model by searching among the set of learning hyper-parameters and IL-method-specific parameters in subsequent tasks from task 2 to k (lines 7-13). Finally, we select the best model which shows the highest performance based on test sets after incrementally trained up to task k (line 14). Note that to facilitate the extensive experiments performed in our study and to make a fair comparison among the IL methods (Section A.5), we first identify the best architectural hyper-parameter (from $L \subset \{1, 2\}$ and $S \subset \{32, 64\}$) and then use the found hyper-parameter across the different IL methods. The final LSTM architecture we used for each dataset are reported in Table A.1.

A.3.5 Implementation

We implemented our continual learning framework on Nvidia Jetson Nano and One Plus Pro smartphone platforms. All the IL algorithms were explored on Nano GPU, and we used PyTorch 1.1 to implement the framework. Keeping in mind that Scenario 3 is the most practical continual learning scenario and iCaRL is the best performing IL approach, we only implemented iCaRL for Scenario 3 on the smartphone's CPU (as an Android app) using the DeepLearning4j library. The smartphone app size is 134 MB. We choose CPU on the smartphone as it provides an upper bound on the performance of any system and is more challenging to implement. We envisage that if a system can work (or at least feasible) on a CPU, then it would be much easier and faster to run similar systems on accelerators such as GPU. When working on a dataset, we first loaded the training data pertaining to all the tasks in the memory to make the continual learning process work faster. As a limited amount of memory is allocated to each Android app, we set large heap property in the app to True to use larger heaps for our app. We still encountered memory issues, especially when working with large datasets such as Skoda, which we solved by using memory-mapped files.

In addition, we employ a weighted F1-score which is more resilient to class imbalances as the employed datasets (see §B.3.1 for details) are not balanced [136, 18]. As in [137], we applied a weighted loss to all evaluated methods by estimating the inverse class distribution which gives more importance to the loss of a class with fewer samples. Also, as deep learning models can overfit to small datasets such as EmotionSense, with our framework, we experimented with shallow and deep neural network architectures and found that deep architectures show marginal improvement over shallow architectures, indicating that the

Table A.1: Overview of the employed datasets.

Application	Dataset	Dimension	# Train Data	# Test Data	# Classes	*Layer/Size*
HAR	HHAR	20 × 120	59,403	7,721	6	2/64
	PAMAP2	33 × 52	35,263	5,209	12	1/64
	Skoda	33 × 60	10,047	1,193	10	1/64
GR	Ninapro (Per Subject)	40 × 12	3,118	639	10	1/64
	Ninapro (LOUO)	40 × 12	30,488	3,759	10	1/64
ER	EmotionSense	20 × 24	2,011	224	14	2/64

overfitting is not an issue.

A.4 Experimental Setup

Before we present the findings of this work in Section A.5, we describe experimental setup for conducting a comprehensive evaluation of three continual learning schemes in mobile and embedded sensing applications. We first describe six datasets in three different sensing applications (§B.3.1) and evaluation metrics adopted for systematic comparison of the IL techniques and their trade-offs between system aspects (e.g., storage and computational costs) (§A.4.2).

A.4.1 Datasets

We focus on three sensing applications (e.g., HAR, GR, and ER) as they are some of the most popular applications in the mobile sensing. Table A.1 shows the overview of the employed datasets.

Human Activity Recognition (HAR)

For the HAR application, we used three datasets: (1) HHAR [136], (2) PAMAP2 [138], and (3) Skoda [139]. These datasets contain many real-life activities (e.g., walking, sitting, and cycling) obtained using Inertial Motion Units (IMUs), which contain accelerometer, gyroscope, and magnetometer data of mobile and wearable devices. We next present the detailed summaries of the three datasets.

HHAR: This dataset considers six different daily activities of users The data was recorded from nine participants, where they followed a scripted set of activities with eight smartphones and four smartwatches of different brands and models. Having various devices for recording makes HHAR an excellent benchmark to study heterogeneity of HAR (i.e., sensor biases, sampling rate heterogeneity, and sampling rate instability). We follow the preprocessing steps as proposed by Yao et al. [140]. Raw measurements of both

accelerometer and gyroscope are segmented into 5-second samples. Each sample is divided into time intervals of 0.25s. After that, we apply a Fourier transform to each time interval. It produces $d \times 2f$ dimensional vectors per time interval, where d is the dimension for each measurement and f is the frequency with magnitude and phase pairs, resulting in 120 dimensions. We adopt leave-one-user-out (LOUO) for evaluation [140]. One user (i.e., the first participant) is used for testing, and the remaining users are left for training.

PAMAP2: In this dataset, nine subjects carried out various daily living activities and sportive exercises. IMU data (accelerometer, gyroscope, magnetometer), heart rate, and temperature data were recorded from body-worn sensors attached to the hand, chest, and ankle. The resulting dataset has 52 dimensions, and more than 10 hours of data were collected. We follow a preprocessing protocol used by Hammerla et al. [18]. The sensor data are downsampled to 33Hz. After that, all samples are normalized to zero mean and unit variance. Also, to be consistent with the previous works [18, 4, 141], we use runs 1 and 2 from the sixth participant for testing and remaining data for training.

Skoda: The Skoda dataset contains activities of assembly-line workers in a manufacturing scenario. One subject wore 20 3D accelerometers on both arms. Following the preprocessing steps [61, 4], we employ raw and calibrated data from ten accelerometers placed on the right arm, resulting in input data of 60 dimensions. The data are downsampled to 33Hz and normalized to zero mean and unit variance. For experiments, the last 10% of each class is used as the test data and the remaining as the training data. Note that Skoda consists of one subject, i.e., subject dependent evaluation.

Gesture Recognition (GR)

We employ the Non Invasive Adaptive Prosthetics (Ninapro) database [142] for the GR application in our experiments as it consists of surface electromyography (sEMG) signals and thus can provide different sensor modalities than IMU sensors present in HAR datasets.

Ninapro (Per Subject): The Ninapro database is widely used in research on the hand movement recognition application. We employ Ninapro Database2 (DB2) in this study. It includes sEMG data recordings from 40 subjects while performing several repetitive gestures such as wrist movements, grasping and functional movements, and force patterns. Following, Li et al. [143], we select ten types of hand gestures commonly used in daily life. After that, we downsample the sEMG data to 200 Hz and normalize them to zero mean and unit variance. We used a sliding window size of 200 ms with a 50% overlap [69, 35]. We select a subject who has the most amount of data samples for subject dependent (i.e., per subject) evaluation. After that, we use the fifth repetition for a test set and the remainder for training.

Ninapro (LOUO): To have consistent evaluation with the HAR application we adopt LOUO evaluation for the GR application using the Ninapro dataset. We select the top

ten subjects having more data samples than others. After that, we use a subject with the least data samples for testing and the remainders for training. The preprocessing steps are the same as in Ninapro (Per Subject).

Audio Sensing Task

We pick Emotion Recognition (ER) since it is one of the most widely adopted audio sensing tasks. We employ the EmotionSense dataset [20] which was collected by recording human participants' emotions as well as proximity and patterns of conversation using an off-the-shelf smartphone. This dataset has been used in multiple studies to understand the correlation and impact of interactions and activities on the emotions and behavior of individuals in various settings [144][58][8].

EmotionSense: The EmotionSense dataset contains audio signals which represent 14 different emotions. In the EmotionSense dataset, each measurement corresponding to a particular emotion (or class) is based on a 5-second context window. Following Georgiev et al. [78], we extract 24 log filter banks [145] from each audio frame over a time window of 30 ms with 10 ms stride. Each sample contains 500*24 = 12,000 features where 1–24 features are filter banks from the first 10 ms, and 25–48 features are filter banks for the next 10 ms and so on. After that, as our preprocessing steps, we downsample each sample measurement by averaging corresponding 24 filter banks of every 250 ms (or 25 consecutive windows) without any overlap to reduce the length of the input sequence for a learned neural network. We normalize each window to zero mean and unit variance.

A.4.2 Evaluation Metrics

We consider how much an IL method forgets previous tasks and learns new tasks after it was trained from task 1 to k to assess the actual performance of IL methods [49] by considering the following metrics.

Average Performance Measure (A): We denote the performance measure of a model on the j-th task ($j \leq k$) as $a_{k,j} \in [0, 1]$ after the model is trained from task 1 to k. The average performance measure at task k is defined as follows:

$$A_k = \frac{1}{k} \sum_{j=1}^{k} a_{k,j} \tag{A.5}$$

The output space consists of $\cup_{j=1}^{k} \boldsymbol{y}^j$, and $a_{k,j}$ is based on a weighted F1-score in this work. Note that $a_{k,j}$ can be used to indicate an accuracy, proportion of correctly classified activities or gestures.

Forgetting Measure (F): The forgetting measure provides an estimate of how much a

model forgets about the task given its present state. The forgetting for the j-th task after the model has been trained up to task $k > j$ can be quantified as:

$$f_j^k = \max_{l \in 1,\ldots,k-1} a_{l,j} - a_{k,j}, \quad \forall j < k \tag{A.6}$$

The average forgetting at k-th task is denoted as $F_k = \frac{1}{k-1} \sum_{j=1}^{k-1} f_j^k$ by normalizing the number of tasks seen previously. The lower the F_k, the less forgetting on previous tasks.

Intransigence Measure (I): Intransigence is defined as the inability of a model to learn new tasks. To quantify the inability to learn, the joint model, often considered upper bound, which has access to all the datasets seen so far ($\cup_{l=1}^{k} D_l$) is compared and its performance is denoted as a_k^*. We then denote the intransigence for the k-th task as:

$$I_k = a_k^* - a_{k,k} \tag{A.7}$$

where $a_{k,k}$ represents the performance of a model on the k-th task trained up to task k. Lower I_k implies that a model performs as close as a joint model or performs even better than the joint model when intransigence is negative ($I_k < 0$). Note that we use $a_{k,k}$ and I_k as the main performance indicators of a model since we are interested in the current performance of the model on all learned tasks from 1 to k.

Note that in addition to metrics mentioned above, we also report **storage** and **latency** required to execute each IL method.

A.5 Findings

We now present the results of our evaluation. Firstly, we compare the performances of different IL methods on HAR, GR, and ER tasks using two basic scenarios (Scenario 1 and 2) in §A.5.1. Then, we study the performance of IL methods for Scenario 3 in §A.5.2. We examine the generalizability of IL methods across different datasets (§A.5.3). Then, we discuss the trade-offs of IL methods with respect to the storage, computational costs, and memory footprint. (§A.5.4). Finally, in §A.5.5, we investigate the effect of iCaRL specific parameters on the performance.

A.5.1 Performance on Simple and Mildly Difficult Tasks

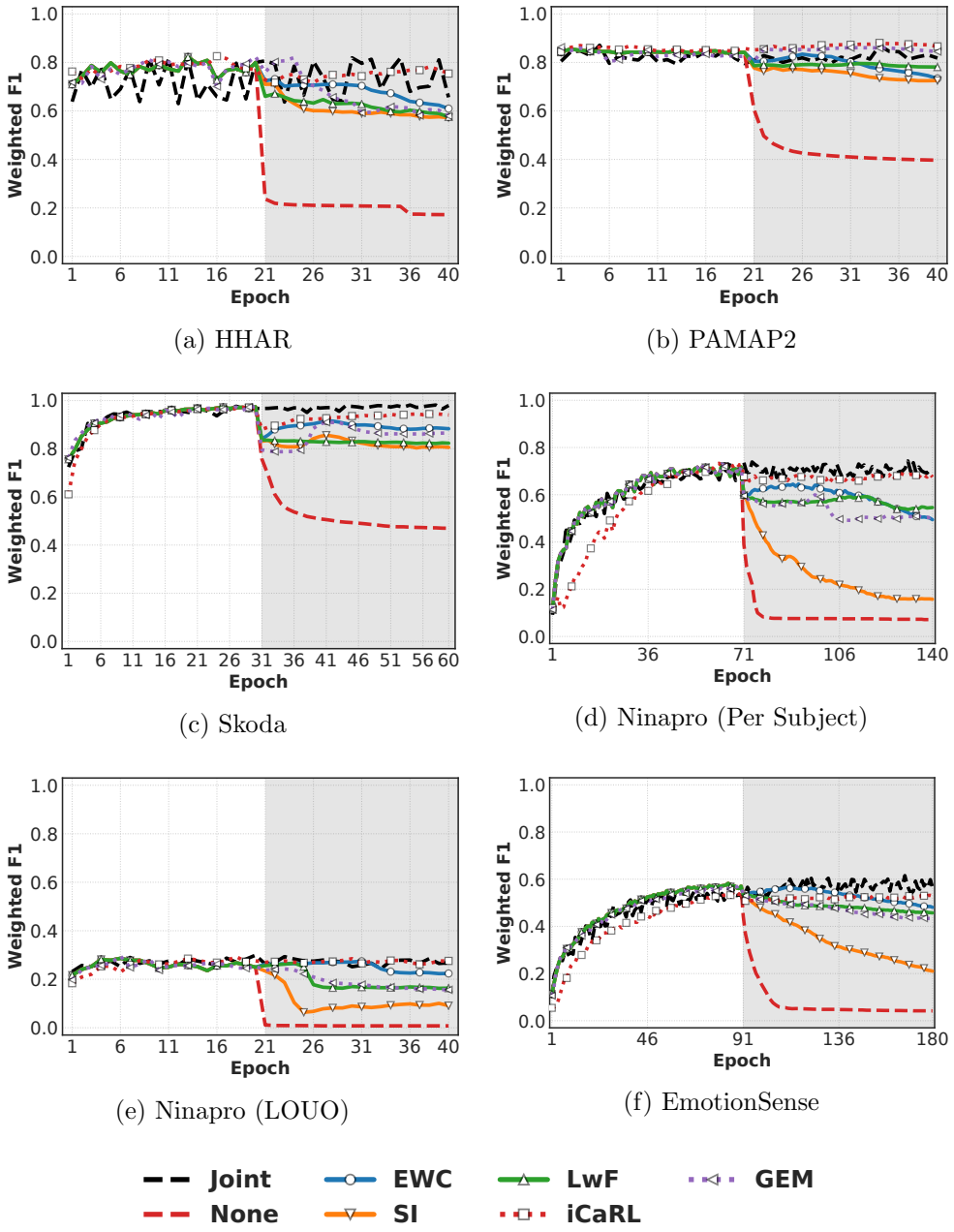
We show the best average weighted F1-scores across all runs for different IL methods for Scenario 1 and 2 for different datasets in Figure A.2 and Figure A.3, respectively. For HAR and GR applications, the results of iCaRL/GEM with the budget size of 20% are shown in Figure A.2 and Figure A.3 since the models with the budget size of 20% show the best performance. Then, for the ER application, the results of iCaRL/GEM with the

budget size of 40% are shown since the EmotionSense dataset has the least number of training samples, requiring more budget size in ER than the other two applications. **Joint** refers to the case when training data is available for all the classes from the beginning. It is a classic case to train a model with all data at once and serves as the upper bound in many cases. **None** refers to the case when no IL method is applied to solve CF. The white part in the figure shows performance on Task 1, and the grey part shows performance for Task 2.

The results show that without any IL method (None), the performance drops sharply as soon as a new task is encountered. The decline in performance is as drastic as 60% in both scenarios. iCaRL provides the best performance in Scenario 1, which stays very close to the performance obtained with the joint model. It is because iCaRL stores representative exemplars and relies on a nearest-class-mean (NCM) rule that is robust against changes in the data representation [17]. In fact, all the IL methods effectively solve the CF problem and achieve comparable performance to the joint model (between 5% and 15%) after only running for a few epochs (5 or less in many cases). *One can conclude that, in general, the existing IL methods we analyzed can solve the CF issue on mobile and embedded sensing applications for simple scenarios.*

However, the same cannot be said for the performance in Scenario 2. *Except iCaRL, none of the other methods seems to solve the CF issue for the mildly complex scenario (i.e., Scenario 2).* The performance drop is up to 60% when the performances between IL methods and the joint model are compared. iCaRL remains the best performing method with its weighted F1-score close to that of the joint model (within 10%). GEM performs the second-best (within few epochs) on HAR datasets while EWC performs well for GR and ER datasets. Although GEM is a replay with exemplars-based approach like iCaRL, it never matches the performance of iCaRL due to its reliance on using gradients and not the actual examples themselves. Another reason might be that iCaRL selects best examples to be stored based on herding (a sort of prioritization), while GEM employs selecting examples randomly which can be less informative. A regularization-based method such as SI and a replay only approach such as LwF perform poorly across all datasets. The weighted F1-score degrades roughly 40–50% of what can be achieved by the joint model. As indicated by [146], the performance of LwF significantly decreases when the model learns a sequence of tasks drawn from different distributions. In other words, when tasks learned by LwF are not sufficiently related, enforcing the new model to give similar outputs for the old task may hurt the model's performance. SI relies on the weight changes in a batch gradient descent which can overestimate the importance of the weights and thereby leads to lower performance.

Note that iCaRL employs a different way (i.e., NCM rule) to classify data samples (perform inference) than other methods (including None and Joint) which use cross-entropy based classification. Also, for GEM, it minimizes the loss on the current task by using inequality



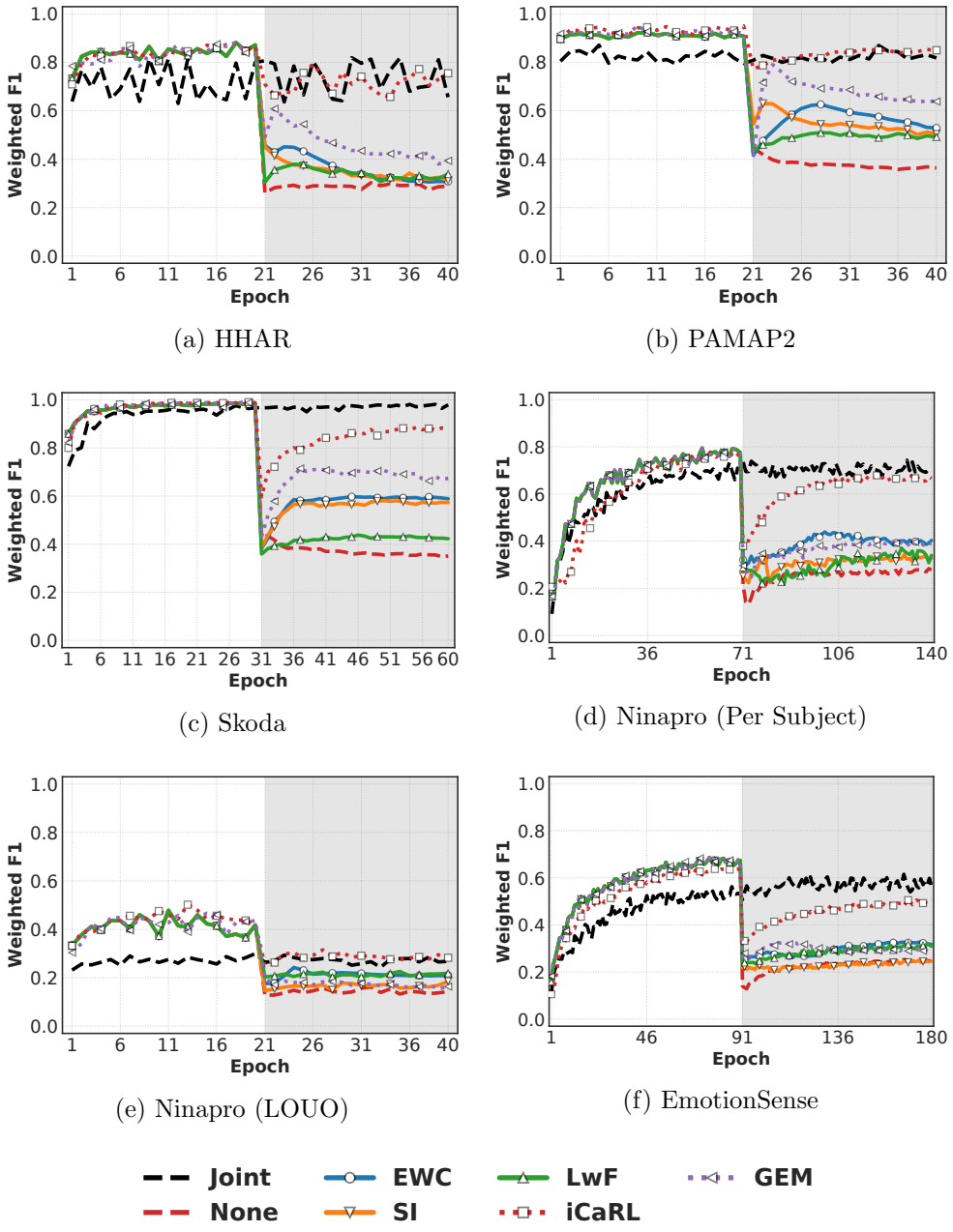
(a) HHAR (b) PAMAP2

(c) Skoda (d) Ninapro (Per Subject)

(e) Ninapro (LOUO) (f) EmotionSense

Figure A.2: The performance comparison of the five IL methods including two baselines in Scenario 1 on each dataset.

constraints, avoiding its increase but allowing its decrease. Therefore, iCaRL and GEM can obtain different weighted F1-scores than the other methods in task 1. Otherwise, ideally one would assume all methods (e.g., None, EWC, SI, LwF in our study) to get the



(a) HHAR (b) PAMAP2

(c) Skoda (d) Ninapro (Per Subject)

(e) Ninapro (LOUO) (f) EmotionSense

Figure A.3: Performance comparison in Scenario 2.

same performance in the first task as it only involves learning a baseline LSTM model without any IL. Also worth mentioning is that initially (especially task 1) IL methods can achieve higher weighted F1-scores than the joint model. It is because their performance is based on classifying the smaller number of classes than the joint model, where all classes

need to be classified from the first epoch.

A.5.2 Performance on Many Sequential Tasks

Figure A.4 shows results for Scenario 3. Recall that Scenario 3 presents the case when classes are added one by one to an already existing deep learning model, which will happen in real-life scenarios and is the most challenging task for any IL method. Note that this graph is shown differently than the graphs for Scenario 1 and 2 (epoch based) as in epoch based graph, we would have only two data points to show as there were only two tasks. In Scenario 3 the number of tasks will be $N/2 + 1$ for N classes. Without the IL method (None), CF happens, and the weighted F1-score almost always lies between 0%–10%. *iCaRL is the best method and appears to solve the CF issue for the challenging third scenario.* Its performance is nearly equal to the joint model in most of the cases. *All other methods do not solve the CF issue, and the performance suffers severely as more tasks are added to the system especially with LwF and SI.*

A.5.3 Generalization

Table A.2 shows the results in a summarized way for all the datasets and IL methods evaluated in our study. A_k refers to average performance on all tasks while $a_{k,k}$ shows the weighted F1-score at the end of learning all tasks. F_k tells us how good an IL method is in retaining old knowledge about previous tasks. Whereas I_k means how much an IL method is good at learning new tasks. Note that the higher the values of A_k and $a_{k,k}$, the better the model is. However, for F_k and I_k, a low value indicates a better model since low F_k and I_k means that the model forgets knowledge of previous tasks less and performs as close as a joint model, respectively. iCaRL is one of the best-performing methods on all metrics across all datasets. iCaRL can learn new classes (tasks) while retaining old knowledge and maintain high performance even in the most challenging scenario. Given that small errors are allowed when performing HAR, GR and ER, iCaRL alleviates the issue of CF to a large extent. The same is not true for all other IL methods. Although LwF allows previous knowledge to be largely retained (low F value), it does not learn new tasks easily and thus has low performance in general. SI is neither good at learning new tasks (high I) nor at remembering old knowledge (high F). EWC and online EWC offer a decent alternative to iCaRL without needing extra storage on-device but at the expense of lower performance than iCaRL. The overall takeaway is that *iCaRL can enable a system to learn incrementally (continuously) in the mobile and embedded sensing domain (if storage is not such a constraint on a device).*

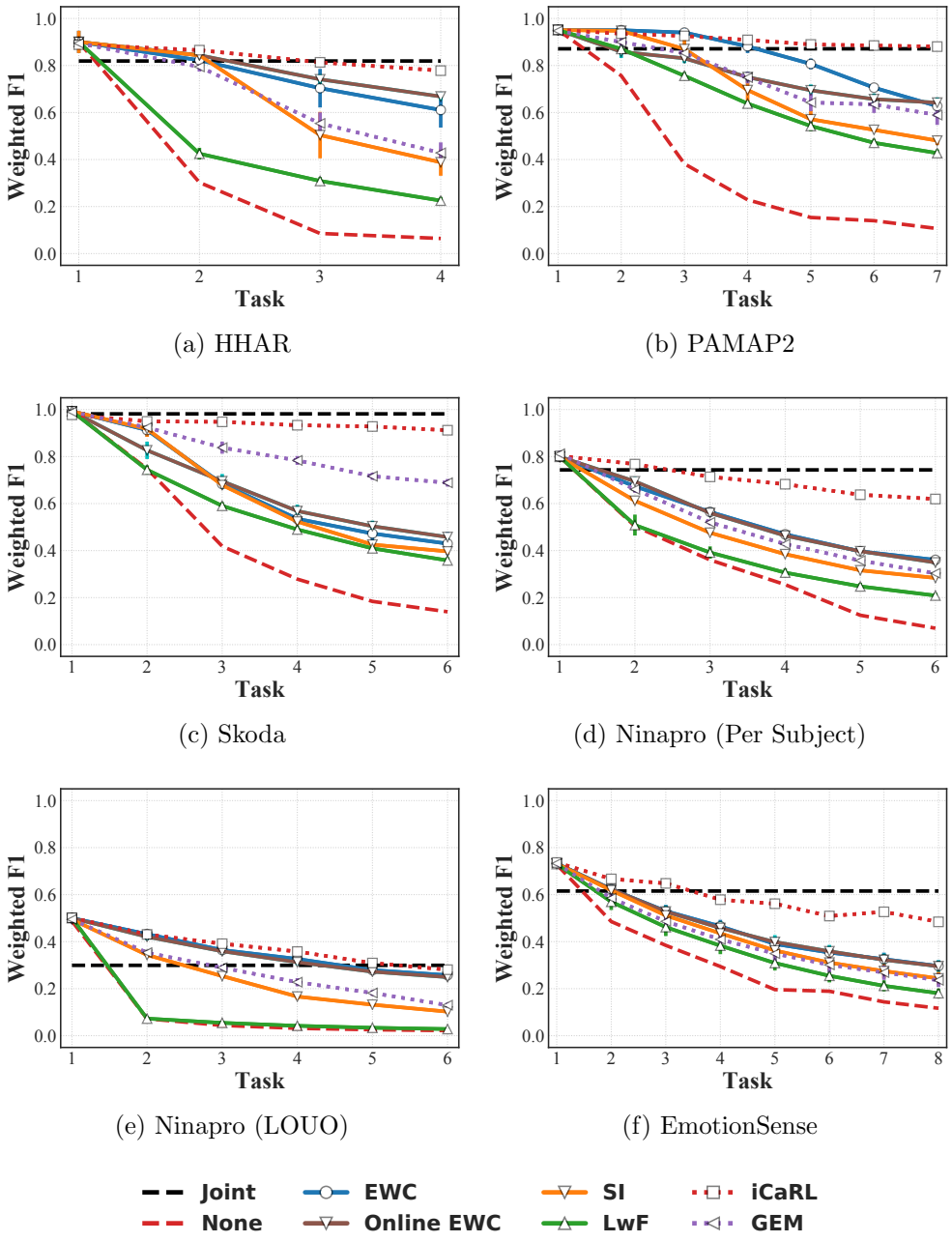


Figure A.4: The performance comparison in Scenario 3. All reported results are averaged over 10 trials, and standard-error intervals are depicted.

Table A.2: Average performance of different methods in all scenarios on HAR, GR, and ER.

Scenario	Methods	HAR				GR				ER			
		A_k	F_k	$a_{k,k}$	I_k	A_k	F_k	$a_{k,k}$	I_k	A_k	F_k	$a_{k,k}$	I_k
1	None	0.55	0.35	0.54	0.36	0.22	0.29	0.20	0.32	0.47	0.11	0.44	0.16
	EWC	0.88	0.01	0.86	0.03	0.49	0.01	0.47	0.05	0.60	0.01	0.58	0.03
	SI	0.85	0.03	0.812	0.07	0.45	0.04	0.42	0.10	0.57	0.01	0.54	0.07
	LwF	0.84	0.02	0.79	0.10	0.47	0.02	0.44	0.09	0.57	0.01	0.54	0.07
	iCaRL	0.89	0.01	0.88	0.01	0.51	0.01	0.49	0.03	0.57	0.02	0.56	0.05
	GEM	0.88	0.01	0.87	0.02	0.46	0.03	0.43	0.09	0.57	0.01	0.54	0.06
2	None	0.30	0.76	0.41	0.48	0.20	0.48	0.23	0.29	0.27	0.45	0.27	0.34
	EWC	0.77	0.06	0.65	0.24	0.47	0.06	0.35	0.17	0.55	0.01	0.39	0.22
	SI	0.64	0.26	0.60	0.29	0.31	0.31	0.29	0.23	0.38	0.27	0.32	0.29
	LwF	0.70	0.03	0.48	0.41	0.45	0.06	0.31	0.21	0.52	0.04	0.35	0.26
	iCaRL	0.89	0.05	0.86	0.03	0.53	0.09	0.51	0.02	0.57	0.07	0.53	0.08
	GEM	0.77	0.13	0.71	0.18	0.39	0.19	0.31	0.21	0.51	0.08	0.37	0.24
3	None	0.22	0.21	0.10	0.79	0.09	0.17	0.05	0.48	0.18	0.16	0.12	0.49
	EWC	0.75	0.01	0.56	0.34	0.44	0.01	0.31	0.21	0.46	0.01	0.30	0.49
	Online EWC	0.72	0.03	0.59	0.30	0.44	0.01	0.30	0.22	0.45	0.01	0.30	0.31
	SI	0.59	0.10	0.42	0.47	0.32	0.07	0.22	0.31	0.42	0.02	0.24	0.36
	LwF	0.53	0.06	0.34	0.55	0.20	0.08	0.12	0.40	0.29	0.11	0.18	0.43
	iCaRL	0.86	0.01	0.79	0.10	0.53	0.01	0.45	0.07	0.62	0.12	0.48	0.13
	GEM	0.70	0.07	0.57	0.32	0.33	0.08	0.22	0.31	0.33	0.02	0.16	0.44
-	Joint	-	-	0.89	-	-	-	0.52	-	-	-	0.61	-

A.5.4 Storage, Latency, and Memory Footprint

Storage: We report the storage overhead of each IL method, as shown in Table A.3. We first specify the mathematical formulas used to calculate the overall storage requirements of each IL method to show how much storage the IL method needs with respect to the number of tasks ($\mathcal{T}$) added, the model parameters (M), and the budget size ($\mathcal{B}$). This point would help practitioners and researchers easily understand how much storage overhead occurs when they want to deploy their models with a particular IL method. First of all, LwF requires no extra storage other than the storage needed to store the model parameters (M). Then, SI requires a running estimate (w_k), the cumulative importance measures (Ω_k^i), and reference weights (θ_k^*) of importance weights of the current task. EWC stores fisher matrices and means for each task. Unlike EWC, Online EWC is only required to store one fisher matrix and running means across tasks. Thus, the required storage for Online EWC does not increase as the number of learned tasks increases. Similar to LwF, iCaRL also requires the previous task model for knowledge distillation. For GEM, it stores the gradient of the exemplar set for each learned task. As both iCaRL and GEM rely on stored examples, their storage demands are mainly driven by the number of examples to

Table A.3: Storage requirements of IL methods. $\mathcal{M}$ refers to the number of model parameters, $\mathcal{T}$ represents number of tasks and $\mathcal{B}$ is the storage budget.

Category	**Method**	**Required Storage**
Reg-based	EWC	$2 \times \mathcal{M} \times \mathcal{T}$
	Online EWC	$2 \times \mathcal{M}$
	SI	$3 \times \mathcal{M}$
Replay-based	LwF	$\mathcal{M}$
Replay+Exemplars	iCaRL	$\mathcal{M} + \mathcal{B}$
	GEM	$\mathcal{T} \times \mathcal{M} + \mathcal{B}$

Table A.4: Storage requirements of IL methods for all datasets - Scenario 3. Units are measured in MB.

IL Method	**HHAR**	**PAMAP2**	**Skoda**	**Ninapro (Per Subject)**	**Ninapro (LOUO)**	**EmotionSense**
EWC	2.601	3.599	3.177	2.587	2.587	3.663
Online EWC	0.650	0.514	0.529	0.431	0.431	0.458
SI	0.975	0.771	0.794	0.647	0.647	0.687
LwF	0.325	0.257	0.265	0.216	0.216	0.229
iCaRL (1%)	5.990	2.676	1.051	0.270	0.805	0.257
iCaRL (5%)	28.838	12.341	4.187	0.512	3.190	0.407
iCaRL (10%)	57.350	24.421	8.179	0.805	6.179	0.607
iCaRL (20%)	114.374	48.658	16.162	1.410	12.141	0.981
iCaRL (40%)	-	-	-	-	-	1.755
GEM (1%)	6.989	4.205	2.350	1.351	1.884	1.862
GEM (5%)	29.817	13.874	5.537	1.583	4.278	2.016
GEM (10%)	58.372	25.996	9.532	1.884	7.274	2.217
GEM (20%)	115.444	50.240	17.476	2.485	13.266	2.603
GEM (40%)	-	-	-	-	-	3.374

be stored (i.e., budget size, $\mathcal{B}$).

Numerical model sizes (i.e., $\mathcal{M} + \mathcal{B}$) are shown in Table A.4 for all the employed datasets in Scenario 3. Note that we do not add tables containing the results of Scenario1 and 2 due to the page limit. However, by reporting the results of Scenario 3 where the storage requirements of various IL methods are greater than or equal to those of Scenario 1 and 2, we aim to present the upper bound of the required storage. Besides, the reported

numerical sizes of storage requirements in Table A.4 are based on IL methods with the largest model in our experiments (i.e., number of LSTM layers ($L = 2$) and the number of hidden units ($S = 64$)) to capture the upper bound to practically operate IL methods on embedded and mobile devices. Here we take the Skoda dataset to further explain our findings as it represents an ideal use case scenario where IL methods need to be applied to personal mobile devices (single-user scenario with modest dataset size). In the Skoda dataset, replay with exemplars methods such as iCaRL and GEM requires at most around 17 MB, and other IL methods have even smaller storage requirements. For EmotionSense dataset where we use up to 40% budget, iCaRL needs less than 2 MB, and GEM needs less than 3.4 MB at most. Even with the largest dataset of HHAR in our experiments, the storage requirements are constrained within less than about 115 MB, which falls well within the storage capacity of modern embedded devices and smartphones. Many modern mobile and embedded devices already support a large amount of storage (in order of GBs). *In summary, the amount of storage required to practically enable continual learning on many modern edge platforms such as Nvidia Jetson or Raspberry PIs and smartphones is not excessive, as evident from Table A.4.* Note that tuning appropriate parameters in the IL method would still allow IL to perform effectively, i.e., ensuring good performance with a reasonable budget size (discussed in §A.5.5).

Latency: The average training and incremental learning time to execute different IL methods are illustrated in Table A.5 for all the employed datasets in Scenario 3 on Jetson Nano[2] which is an edge platform having four cores, 4 GB RAM and a GPU and often used in mobile robotics and can be used in tablets. Training time represents the usual training time involved in learning a neural network including updating weights, back-propagation, etc. EWC and Online EWC take the highest time. In small datasets of Ninapro (Per Subject) and EmotionSense datasets, incremental learning time is around 39.5–58.9 seconds. In the largest dataset of HHAR, IL takes over 1,160 seconds. This is surprising as EWC is a simple method. However, the time complexity comes from calculating and updating the Fisher matrices, which is a computationally expensive process, after every task. SI (mostly relying on running estimates) and LwF (replay only, calculating distillation loss) are two of the top three fastest IL methods but come at the peril of very low accuracy which is not suitable for IL in mobile and embedded applications. iCaRL, the best performing IL method, is also very fast and takes only a few seconds (e.g., 0.94–2.8 seconds) in the Emotion datasets and Ninapro (Per Subject) to complete. In the HHAR dataset, the average latency of IL time of iCaRL with the largest budget size (i.e., 20%) is relatively small of 133 seconds compared to and its training time (i.e., 633 seconds) and the IL time of EWC (i.e., 1,160 seconds). In reality, the majority of the time is taken by actual training (except EWC and Online-EWC), which depends on the number of epochs to be performed and is independent of the IL method. Across scenarios, we observe that the

[2]By reporting the results of Scenario 3 where the latency of IL methods is greater than or equal to that of Scenarios 1 and 2, we aim to capture the upper bound of the latency.

Table A.5: Average Latency (Training Time/IL Time) in seconds for IL methods on different datasets - Scenario 3 on Jetson Nano.

IL Method	HHAR	PAMAP2	Skoda	Ninapro (Per Subject)	Ninapro (LOUO)	EmotionSense
EWC	613/1160	185/567	98.2/161	222/55.2	103/544	190/45.6
Online-EWC	575/1156	163/552	85.2/159	213/58.9	93.5/529	190/39.5
SI	597/51.6	171/19.5	94.8/6.2	192/2.2	151/17.1	164/2.6
LWF	591/13.9	182/7.2	73.0/2.1	198/0.86	117/6.7	161/0.73
iCaRL (1%)	522/21.7	153/15.8	146/3.8	224/1.1	160/11.9	235/0.94
iCaRL (5%)	577/40.4	168/22.9	132/4.9	228/1.4	145/20.2	188/1.1
iCaRL (10%)	581/60.5	175/34.4	113/6.4	265/1.9	115/25.8	210/1.3
iCaRL (20%)	633/133	181/53.4	120/9.6	234/2.6	162/42.0	212/1.9
iCaRL (40%)	-	-	-	-	-	214/2.8
GEM (1%)	663/60.9	160/35.2	78.9/7.9	191/2.6	80.8/26.5	152/3.6
GEM (5%)	790/118.2	192/53.1	80.0/9.5	206/2.8	97.1/37.5	148/3.6
GEM (10%)	1054/192	220/80.5	81.2/11.1	211/2.8	98.1/55.1	149/3.9
GEM (20%)	1081/342	322/136	99.1/14.5	206/3.2	136/90.7	149/4.0
GEM (40%)	-	-	-	-	-	156/4.2

average training time can range from one to 15 minutes in general.

Having realized that iCaRL is the most promising method in terms of accuracy and latency, we wanted to check if iCaRL can also effectively work on modern smartphone CPUs. For this, we have implemented iCaRL on OnePlus 7 Pro for three datasets: Skoda, Ninapro (Per Subject), and EmotionSense as they represent datasets where IL needs to be applied to personal mobile devices (single-user case) and Scenario 3 (most practical scenario). The smartphone has eight cores and 12 GB of RAM. To reiterate, we used DeepLearning4j library to implement iCaRL. The smartphone app size is 134 MB. The results are shown in Table A.6. Similar to Jetson Nano, iCaRL takes minimal time (0.5–212 seconds) for all the tasks for every dataset. This does not only mean that IL is feasible on modern smartphones but even if a very high number of tasks are to be learned even in the most challenging scenario, iCaRL can do end-to-end IL in a few minutes. The training time slows down the whole process and ranges from 20–75 minutes on the CPU of the smartphone for different datasets. Also note that the training time taken by the tasks after the first task (actual incremental tasks after the initial model is trained) is very small: one to four minutes. This is a relevant result as one can train a baseline model on a powerful machine first and can then move it to a mobile and embedded device to learn incrementally over time. Regardless, we show that the complete incremental learning

Table A.6: Average Latency (Training Time/IL Time) in seconds for iCaRL on three datasets - Scenario 3 on Smartphone.

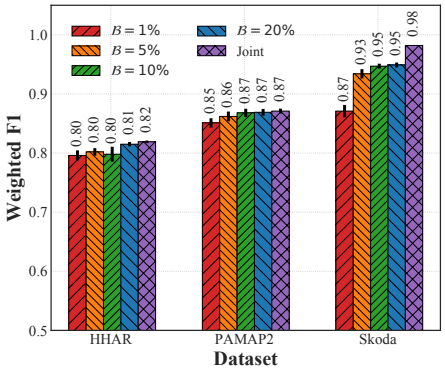
IL Method	**Skoda**	**Ninapro (Per Subject)**	**EmotionSense**
iCaRL (1%)	4400/9	1956/1.28	1568/0.5
iCaRL (5%)	3894/29	1974/3	1388/1.91
iCaRL (10%)	3869/72	2312/4.5	1535/2.6
iCaRL (20%)	3902/212	2008/5.1	1517/4.7
iCaRL (40%)	-	-	1506/8.1

process can still be done entirely on the smartphone CPU, especially given that the phone can be charged overnight. *This is an interesting result as this suggests that our continual learning framework can be deployed on a smartphone CPU. It is also encouraging because the performance can be further improved by exploiting GPU and NPU once support for training them programmatically starts to emerge.*

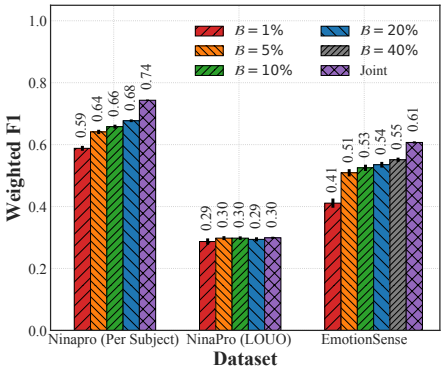
Memory footprint: We further examine the peak memory usage of iCaRL with its largest budget size of 20-40% on all the datasets to evaluate whether or not it can fit the tight memory budget of Jetson Nano. The peak memory overheads of running the end-to-end IL range from 196 MB for our smallest dataset of EmotionSense to 1,194 MB for our largest dataset of HHAR, when the CPU is used for IL. Then, when we use GPU for running iCaRL, it incurs 1,782-2,127 MB peak memory and requires an additional swap space of 750-3,523 MB. We observed that the reduction of latency using GPU over CPU is largely consistent between 80-86%, indicating that the swap space has minimal impacts on the speed-up of the IL using GPU compared to using CPU on Jetson Nano. *This result confirms that IL in the mobile and embedded sensing domain is applicable on resource-constrained devices within a reasonable memory overhead.*

A.5.5 Performance with IL parameters

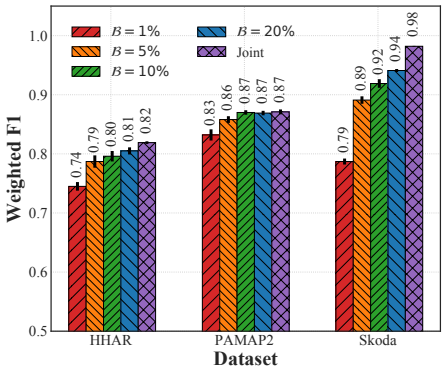
We study the importance of the storage budget parameter for iCaRL as it is the best performing IL method. Figure A.5 shows the weighted F1-score with changing storage budgets of 1%, 5%, 10%, and 20% of total training samples (up to 40% storage budget for the case of ER). In general, more samples are needed to avoid CF as the complexity of the scenario increases. In Scenario 1, only 1% of total samples are needed to achieve similar performance as the joint model. Moreover, in Scenario 2 and 3, the results show that the budget size of 5% is enough to achieve the high performance which is quite close to that of the joint model, although the difficulty of the task increases compared to Scenario 1. In contrast, 10% of samples are required to achieve near joint model's performance (i.e.,



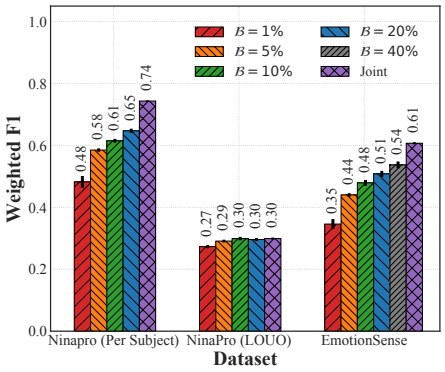
(a) Scenario 1



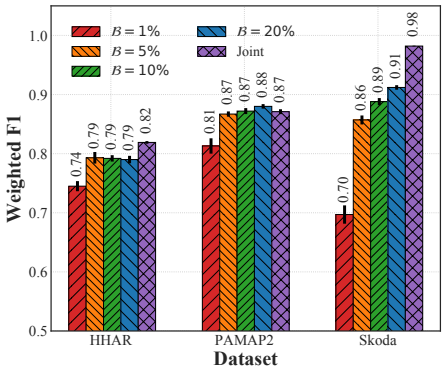
(b) Scenario 1 (GR & ER)



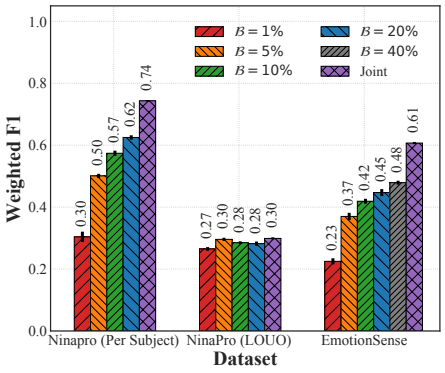
(c) Scenario 2



(d) Scenario 2 (GR & ER)



(e) Scenario 3



(f) Scenario 3 (GR & ER)

Figure A.5: The parameter analysis of the best performing model, iCaRL, in all tasks (HAR, GR, and ER) for all scenarios according to its storage budgets. Reported results are averaged over 10 trials. Standard-error intervals are depicted.

upper bound performance) in the most challenging setup (Scenario 3).

Note that the performances of iCaRL with the budget size of 5% are often very close to those of iCaRL with budget sizes of 10%, 20%, and 40%. This result indicates that iCaRL enables us to achieve close to the performance of the joint model without requiring excessive storage (less than 30 MB in all datasets in our experiment when a budget size is 5%). Specifically, the required storage of iCaRL with 5% budget size for each dataset (HHAR, PAMAP2, Skoda, Ninapro (Per Subject), Ninapro (LOUO), and EmotionSense corresponds to 28.84, 12.34, 4.19, 0.51, 3.19, and 0.41 MB, respectively. This is an interesting finding, making iCaRL a good candidate to perform IL on many embedded devices and smartphones with reasonable storage as only a few samples are required to be stored.

A.6 Discussion

We discuss the potential guidelines ($\mathcal{G}$) for researchers and practitioners in the mobile and embedded systems community based on our findings of this work. The readers should take our results and guidelines with a pinch of salt as we did not compare all the existing IL methods due to reasons mentioned earlier (Section A.3) and These findings are based on a few prominent IL methods we analyzed in our study.

($\mathcal{G}_1$): If storage is not an issue on the device, one can choose to use the iCaRL method since it performs best across all datasets in different sensing applications. As many modern computing platforms including smartphones and embedded devices have large storage capacity, the issue of storing a proportion of training samples can be minor. iCaRL is also not very computationally expensive on the modern embedded devices and the smartphone. Also, the process can be sped up by using GPUs although it incurs higher peak memory than CPUs.

($\mathcal{G}_2$): GEM, although being a replay with exemplars-based method like iCaRL, should not be preferred over iCaRL as its performance remains inferior to those of iCaRL. Also, GEM requires more storage than iCaRL.

($\mathcal{G}_3$): In a severely resource-constrained environment, EWC and Online EWC can be a reasonable alternative to iCaRL since these methods require less additional storage. Although EWC is a computationally expensive method, however the computational cost can be manageable as the IL process is only performed once per task, and one can reduce the number of samples used to compute fisher matrices, which account for the majority of the IL time.

($\mathcal{G}_4$): LwF and SI should be avoided as they offer minimal protection against CF on mobile sensing applications.

($\mathcal{G}_5$): If the resources such as storage available are constrained on the device, we suggest using iCaRL with the budget size of 1%–5% of training samples as using higher budget size does not always provide enough benefits in case the training dataset size is large (HHAR, PAMAP2, and NinaPro (LOUO)). For datasets having smaller training sizes such as Ninapro (Per Subject) and EmotionSense datasets, having a higher budget of 20%–40% helps to a large extent.

A.7 Conclusions and Future Work

In this paper, we studied the CF problem using six prominent IL methods based on three representative sensing applications (i.e., HAR, GR, and ER) in three continual learning scenarios with varying complexities. With our end-to-end IL framework implemented on Nvidia Jetson Nano and a smartphone (OnePlus 7 Pro), we conducted extensive experiments to investigate IL methods' performance, generalizability, and trade-offs of storage, computational costs, and memory footprints. We first identified that CF occurs in mobile and embedded sensing applications when IL methods are not used. We also found that while most IL methods solve the CF in simple scenarios, only iCaRL among the compared methods can successfully alleviate CF issues in more challenging scenarios across the employed datasets. Furthermore, we demonstrated that the IL approaches incur minor to modest storage, peak memory usage, and latency overheads (a minute per task in general), thereby saving a considerable amount of computational resources on-device compared to a case when training is done from scratch whenever a new class/task is added to the system. Finally, based on those findings, we discuss potential guidelines for practitioners and researchers interested in applying IL to edge platforms.

As future work, we believe that it would be worthwhile to further investigate continual learning on more severely resource-constrained devices such as microcontrollers as they have smaller storage, limited memory, and low computational power to apply IL methods. Moreover, we want to study how model compression techniques such as quantization affect IL methods' performance. Similarly, combining binary neural networks with IL methods can be interesting future work. The other key point our study highlighted is that the major bottleneck comes from the training during the IL process. In this context, techniques such as Mixed Precision Training (MPT) [147] and quantization using only 16 or 8-bit floating-point representation [148] for weights might be useful.

Appendix B

FastICARL: Fast Incremental Classifier and Representation Learning with Efficient Budget Allocation in Audio Sensing Applications

Abstract

Various incremental learning (IL) approaches have been proposed to help deep learning models learn new tasks/classes continuously without forgetting what was learned previously (i.e., avoid catastrophic forgetting). With the growing number of deployed audio sensing applications that need to dynamically incorporate new tasks and changing input distribution from users, the ability of IL on-device becomes essential for both efficiency and user privacy.

However, prior works suffer from high computational costs and storage demands which hinders the deployment of IL on-device. In this work, to overcome these limitations, we develop an end-to-end and on-device IL framework, FastICARL, that incorporates an exemplar-based IL and quantization in the context of audio-based applications. We first employ k-nearest-neighbor to reduce the latency of IL. Then, we jointly utilize a quantization technique to decrease the storage requirements of IL. We implement FastICARL on two types of mobile devices and demonstrate that FastICARL remarkably decreases the IL time up to 78-92% and the storage requirements by 2-4 times without sacrificing its performance. FastICARL enables complete on-device IL, ensuring user privacy as the user data does not need to leave the device.

B.1 Introduction

A recent development of deep learning has revolutionized various audio-based applications such as emotion recognition (ER) [20], environmental sound classification (ESC) [149], and keyword spotting [150, 151]. However, in a real-world setting where a deployed audio classification models may need to dynamically incorporate new tasks (i.e., new classes or inputs) from users [35] and changing input distribution [39], current supervised learning approaches are severely limited due to the constrained nature of available resources on the edge devices and the catastrophic forgetting (CF) issue [13]. That is, a deep learning model becomes able to recognize a new task but forgets previously learned knowledge.

Many researchers proposed a range of Incremental Learning (IL) methods [15] to solve the CF problem. The first group of the IL approaches is a *regularization-based method* [16, 47, 48] where regularization terms are added to the loss function to minimize changes to important weights of a model for previous tasks to prevent forgetting. Kirkpatrick et al. [16] proposed a regularization-based method, Elastic Weight Consolidation (EWC), which uses the Fisher information matrix to identify important weights to the previous tasks and update less on those weights while learning a new task. Another group of the IL approaches is *exemplars-based method* [17, 55] where the method requires to store important samples from previous tasks to prevent from forgetting learned tasks. Rebuffi et al. [17] proposed a representative exemplar-based method, ICARL, that first utilizes herding [133] to search for exemplars (informative samples) and then uses knowledge distillation loss on the previously learned classes and classification losses on a new class to prevent forgetting and learn the new class. However, prior works are limited in two ways. First, It is challenging to enable IL on-device since IL methods are computationally heavy. Second, exemplar-based methods require storing exemplars, which can impose a considerable burden on resource-constrained systems.

Moreover, many techniques have been proposed to facilitate efficient machine learning systems on resource-constrained devices. Quantization and low-bit precision of model parameters are utilized to reduce the size of the model [103, 107]. Low-rank factorization [152, 153] and pruning [29] have been proven effective in reducing model size, while retaining accuracy. IL with optimizations that allow its use on-device, however, has never been explored in the context of audio-based applications.

In this work, an end-to-end framework, FastICARL, is developed to enable efficient and accurate on-device IL in two audio sensing applications, an ER task and an ESC task. Also, FastICARL is a new IL method devised to improve upon the representative exemplar-based IL method, ICARL, as we observed that ICARL consistently outperforms EWC and other regularization-based IL methods [47, 48]. However, it has computational and storage issues. Thus, FastICARL solves these limitations while maintaining accuracy. First, we optimize the construction process of an exemplar set (which takes most of the IL time)

to shorten the IL time to tackle the first limitation. Specifically, to find the informative exemplars that can best approximate feature vectors over all training examples, ICARL relies on herding which contains inefficient double for loops. Instead, FastICARL utilizes a k-nearest-neighbor and a max heap data structure to search exemplars more efficiently. In addition, to address the second limitation, we further optimize FastICARL by applying quantization on exemplars to reduce the storage requirement. We convert the 32-bit float data type into 16-bit float and 8-bit integer data types. Furthermore, we implement our end-to-end IL framework on mobile and embedded devices of two different specifications: Jetson Nano and a smartphone (Google Pixel 4). For a smartphone implementation, we employ MNN [154] and our implementation enables complete on-device training of new tasks/classes unlike TensorFlow Lite [155] or PyTorch Mobile [156] where only on-device inference is enabled.

Overall, the major contributions and findings of this paper are as follows. We design, implement, and evaluate FastICARL, which overcomes the limitations of the prior work. First of all, FastICARL shows that it can effectively solve the CF issues happening in audio-based datasets by achieving 69% and 71% weighted F1-scores for ER and ESC, respectively. FastICARL reduces the latency of exemplar set selection up to 78% on Jetson Nano and 92% on Google Pixel 4. Moreover, FastICARL decreases the storage requirement by 2-4 times without sacrificing its performance. In addition, we demonstrate that FastICARL can enable on-device IL without the support of the cloud. Hence, FastICARL ensures complete data privacy as user data does not need to leave the device. Finally, to the best of our knowledge, FastICARL is the first end-to-end and on-device framework that incorporates exemplar-based IL and quantization techniques in the context of audio sensing applications.

B.2 Methodology

In this section, we formulate our problem (§B.2.1) and describe the important prior work (§B.2.2). After that, we propose our IL method, FastICARL (§B.2.3).

B.2.1 Problem Formulation

We focus on Sequential Learning Tasks (SLTs) [12] from the audio sensing tasks, where new classes (e.g., different sounds in ESC) can emerge over time. Thus, the learning model has to continuously learn to accommodate new classes without CF, as would happen in real-life scenarios. Learning tasks of this type, called SLTs, indicates that a model continuously learns two or more tasks $D_1, ..., D_k$, one after another instead of learning a single task D once (i.e., multi-task learning). Note that each task consists of disjoint groups of classes as we adopt class-incremental learning [132]. Formally, we are given training samples, $X^1, X^2, ...,$, where X^y is a set of samples of class y. Inspired by prior works [11, 35], we

first train a model on the first task with $N/2$ classes and then incrementally train the model by adding subsequent tasks with one class ($N/2 + 1$ tasks).

B.2.2 ICARL

ICARL is the representative exemplar-based IL method in the literature that attempts to solve the CF problem of class-incremental setting. At the high level, ICARL maintains a set of exemplar samples for each observed class (see Algorithm 2). An exemplar set is a subset of all samples of the class to carry the most representative information of the class. When new tasks (classes) become available, ICARL first creates a new training set by joining all exemplar sets and the data of the new class. Then, it updates its weight parameters by minimizing a classification loss of the new task (class) as well as the distillation loss of the previous tasks (classes). Then, ICARL builds an exemplar set for the new class and trims the existing exemplars for previous classes. Finally, the classification is performed by finding the nearest-class-mean of exemplars to a given test sample in a feature space extracted from the learned representation.

B.2.3 FastICARL

Although ICARL provides impressive performance, it is limited by high computational costs and large storage requirements to maintain sufficient budget size to perform reasonably well. To begin with, ICARL's high computational loads comes from its herding operation (find an exemplar set that has a min distance between the class mean and exemplars mean in feature space), i.e., exemplar selection procedure which is based on the inefficient double for loops (Lines 2-4), resulting in the $O(nm^2)$ complexity (which takes up 70 - 90% of the total IL time). n is the number of examples in a class, and m represents the target number of exemplars. Note that in this work, training time indicates the usual training time with respect to back-propagation, updating weights, while the rest of the time in learning a new task or adding a new class is considered IL time. Thus, instead of relying on herding, FastICARL employs a k-nearest-neighbor search to identify the representative examples to construct exemplar sets. This enables FastICARL to accelerate the process of exemplar construction without performance degradation, as shown in Section B.3. By jointly utilizing the max heap as in Algorithm 2, FastICARL remarkably reduces the complexity of finding m exemplars out of n samples to $O(n(1 + log(m)) + mlog(m)) = O(nlog(m))$. In detail, the computation of feature distance and the insertion of max heap cost $1 + log(m)$ which is performed on n samples in total. After that, the sorting on m identified exemplars in a max heap costs another $mlog(m)$.

Furthermore, ICARL requires as much as 69 MB (see §B.3.4). To alleviate this storage demand, we apply quantization on exemplar sets on the fly. Note that since budget sizes take up 72-99% of the storage requirements of FastICARL, we apply quantization only

Algorithm 2: Construction and quantization of exemplar sets for ICARL/FastICARL

Input: Feature Extractor $\mathcal{F}()$, The number of exemplars to be stored m, Quantization bit b, IL method

Output: Quantized Exemplar set Q

Data: $X = \{x_1, ..., x_n\}$ of class y

$\mu \leftarrow \frac{1}{n} \sum_{i=1}^{n} \mathcal{F}(x_i)$ // calculate class mean

/* find m exemplars out of n samples */

if IL method is *ICARL* **then**

  for $k = 1, ..., m$ **do**

    $p_k \leftarrow \underset{x \in X}{\operatorname{argmin}} \left\| \mu - \frac{1}{k}(\mathcal{F}(x) + \sum_{i=1}^{k-1} \mathcal{F}(p_i)) \right\|$

if IL method is *FastICARL* **then**

  /* calculate feature distance between each sample and class mean */

  for $i = 1, ..., n$ **do**

    $d_i = \mathcal{F}(x_i) - \mu$

  /* build max heap with size k */

  create max heap H of pair {d, index}

  for $k = 1, ..., m$ **do**

    H.insert(d_k, k)

  /* loop over the remaining samples while updating the max heap */

  for $k = m + 1, ..., n$ **do**

    if $d_k < H$.extractMaxDist() **then**

      H.pop() // delete one item from H

      H.insert(d_k, k)

  /* build a sorted exemplar set P */

  for $k = m, ..., 1$ **do**

    $i \leftarrow H$.extractMaxDistIndex(), H.pop()

    $p_k \leftarrow x_i$

for $k = 1, ..., m$ **do**

  $q_k \leftarrow \text{Quantize}(p_k, b)$

$Q \leftarrow (q_1, ..., q_m)$ // Quantized exemplar set

on exemplars in this work. While constructing exemplar sets, FastICARL converts 32-bit float data to 16-bit float or 8-bit integer types and store them with a smaller budget. When converting between 32-bit float and 8-bit integer, we use quantization scheme used in [103] to minimize the information loss in quantization. The scheme utilizes an affine mapping of integers q to real numbers r, i.e.,

$$r = S(q - Z) \tag{B.1}$$

for some constant quantization parameters S and Z. S denotes the scale of an arbitrary positive real number, and Z denotes zero-point of the same type as quantized values q and corresponds to the real value 0.

B.3 Evaluation

B.3.1 Datasets

We experiment with our method on two audio applications.

EmotionSense: For emotion recognition (ER) application, we employ the EmotionSense dataset [20] as it is used in multiple studies in audio sensing [144, 8, 58]. It contains audio signals which are clustered into five standard broader emotion groups, generally used by social psychologists [157] such as (1) Happy, (2) Sad, (3) Fear, (4) Anger, and (5) Neutral. This dataset has 2,235 samples, and each measurement corresponds to a particular emotion based on a 5-second context window. Following [78], we extract 24 log filter banks [145] from each audio frame over a time window of 30 ms with 10 ms stride. After that, as our preprocessing steps, we downsample each sample measurement by averaging corresponding 24 filter banks of every 250 ms (or 25 consecutive windows) without any overlap to reduce the length of the input sequence for a learned neural network. We normalize each window to zero mean and unit variance. As a result, we created an input of size 20×24.

UrbanSound8K: For environment sound classification (ESC) application, we adopt the UrbanSound8K dataset [149] as it is a large dataset that can test the effectiveness of our method on resource-limited devices. UrbanSound8K contains 9.7 hour-long data with 8,732 labeled urban sounds collected in real-world settings. This dataset consists of 10 audio event classes such as car horn, drilling, street music, etc. Following [73], we extracted four different audio features ((1) Log-mel spectrogram, (2) chroma, (3) tonnets, (4) spectral contrast) for each sound clip, sampled at 22 kHz. Using the first 3-seconds of sound, we created an input of size 128×85, where 128 represents the number of frames and 85 represents aggregated feature size of the four audio features.

B.3.2 Experimental Setup

Task: As described in §B.2.1, we adopt class-incremental learning. Hence, for EmotionSense, two classes are selected as task 1 for training a base model, and then the other three classes are added to the model one by one sequentially. For UrbanSound8K, five classes are used as the first task, and the other five classes are learned incrementally. Note that all reported results in §B.3.4 are averaged over five times of experiments.

Model Architecture: We adopt a convolutional neural networks (CNN) architecture from prior work [73] to construct the ER and ESC models. To identify a high-performing

Table B.1: Average weighted F1-score of baselines and FastICARL according to the budget size ($\mathcal{B} = 5\%, 10\%, 20\%$) in EmotionSense and UrbanSound8K datasets.

	EmotionSense (ER)			**UrbanSound8K (ESC)**		
	5%	10%	20%	5%	10%	20%
ICARL (32 bits)	0.57	0.60	0.70	0.67	0.69	0.69
ICARL (16 bits)	0.55	0.63	0.70	0.66	0.67	0.71
ICARL (8 bits)	0.59	0.62	0.68	0.65	0.68	0.70
FastICARL (32 bits)	0.57	0.62	**0.67**	0.67	0.69	0.70
FastICARL (16 bits)	0.58	0.65	0.68	0.66	0.69	**0.71**
FastICARL (8 bits)	**0.60**	**0.63**	**0.69**	**0.65**	**0.68**	**0.69**
Joint (Upper Bound)		0.83			0.89	
None (Lower Bound)		0.41			0.02	

and yet lightweight CNN model to operate on embedded and mobile devices, we conducted hyper-parameter search with different number of convolutional layers {2,3,4}, number of convolutional filters {8,16,32}, pooling layer type {max pooling, average pooling}, number of fully-connected (FC) layers {0,1} and its hidden units {128,512,1024}. A basic convolutional layer consists of 3×3 convolution, batch normalization, and Rectified Linear Unit (ReLU). We found that although the best performing model is a 4-layered CNN with 32 Conv filters followed by an FC layer (Weighted F1-score of 86% for ER and 90% for ESC), the performance degradation without the FC layer is minimal (see Table B.1) while the majority of the model parameters are consumed in the FC layer as shown in [73]. Hence, as our final CNN architecture, we use [Conv: {32,32,64,64}] for ER and [Conv: {16,16,32,32}] for ESC. We omit an FC layer in both applications, and average pooling layers and a 0.5 dropout probability are adopted for the second and fourth Conv layers. ADAM optimizer [158] and learning rate of 0.001 are used.

Evaluation Protocol: Following prior works [20, 73], the 10% of each class is used as the test set and the remaining as the training data. In addition, we report the performance of a model trained up to task k incrementally. Also, we report the results based on a weighted F1-score which is more resilient to class imbalances as the employed datasets are not balanced.

Baselines: To evaluate the effectiveness of FastICARL, we include various baselines in our experiments. First, we include a *Joint* model which represents a scenario when the model is trained with training data of all classes available from the beginning. *Joint* serves as a performance upper bound. Second, a *None* model represents a case where a model is fine-tuned incrementally by adding classes to the model without any IL method. *None* can be regarded as a performance lower bound. Thirdly, we include ICARL with three quantization levels (32, 16, and 8 bits). Finally, FastICARL (32, 16, and 8 bits) is compared.

Table B.2: Average Latency (IL Time) in seconds for ICARL and FastICARL on Jetson Nano and a smartphone (Google Pixel 4) for both datasets according to the budget size ($\mathcal{B} = 5\%, 10\%, 20\%$).

	Embedded Device (Jetson Nano)						**Smartphone (Google Pixel 4)**					
	EmotionSense (ER)			**UrbanSound8K (ESC)**			**EmotionSense (ER)**			**UrbanSound8K (ESC)**		
	5%	10%	20%	5%	10%	20%	5%	10%	20%	5%	10%	20%
ICARL (32 bits)	6.25	7.24	9.35	102	144	271	1.41	1.98	2.73	41.5	75.5	138
ICARL (16 bits)	6.30	7.40	9.25	100	144	270	1.48	1.99	2.74	44.6	78.8	139
ICARL (8 bits)	6.27	7.40	9.25	120	178	292	1.43	1.99	3.04	45.4	77.7	146
FastICARL (32 bits)	5.10	5.18	5.18	**60.6**	60.8	60.7	0.88	0.90	**0.83**	10.5	10.8	**10.4**
FastICARL (16 bits)	**4.96**	4.98	5.22	61.1	61.5	**60.6**	0.87	0.89	0.84	10.7	11.2	10.9
FastICARL (8 bits)	5.01	5.07	5.24	67.1	66.3	61.5	0.90	0.91	0.87	10.7	10.7	10.6

B.3.3 Implementation

To evaluate our framework on resource-constrained devices, we implemented it on an embedded (Jetson Nano) and a mobile device (Google Pixel 4). The Jetson Nano is an embedded mobile platform with four cores and 4 GB RAM. It is often utilized in mobile robotics. We use PyTorch 1.6 to develop and evaluate FastICARL on Jetson Nano. The Google Pixel 4 phone has eight cores and 6 GB RAM. We develop FastICARL based on C++ on the Android smartphone using mobile deep learning framework, MNN, and the Android Native Development Kit. Note that our implementation of FastICARL on the smartphone enables complete on-device training of new tasks/classes incrementally, unlike other deep learning frameworks on mobile platforms (e.g., PyTorch Mobile) where only on-device inference is supported. The binary size of our implementation on a mobile platform is only 3.8 MB which drastically reduces the burden of integrating the IL functionality into mobile applications given that ICARL requires as much as 69 MB for UrbanSound8K.

B.3.4 Results

Performance: We first show the average weighted F1-score across all runs for different baselines and IL methods for the EmotionSense and UrbanSound8K datasets in Table B.1. For both datasets, we present the performance according to the size of the budgets storing exemplars (5%, 10%, and 20%) to analyze trade-offs between the performance and storage requirement of the studied IL methods. Note that the weighted F1-score of the models after all tasks are trained incrementally is reported.

To begin with, the *None* model allows us to confirm that CF occurs without the IL method. Its weighted F1-score drops sharply to 41% for ER and 2% for ESC. In contrast, the *Joint* model achieves as high as 83% and 89% weighted F1-scores for ER and ESC, respectively. ICARL (32 bits) and our proposed IL method, FastICARL (32 bits), can largely mitigate the CF issues observed in the None model. With a budget size of 20%, ICARL provides a

high weighted F1-score of 70% for ER and 69% for ESC. Likewise, FastICARL achieves a similar performance (67% for ER and 70% for ESC) to that of ICARL, which stays close to the upper bound performance of the *Joint* model. Furthermore, we find that the impact of the information loss due to the quantization of the saved exemplars for both ICARL and FastICARL is minimal. As shown in Table B.1, all four variants, such as ICARL (16 and 8 bits) and FastICARL (16 and 8 bits), achieve similar performance to their original counterparts.

Finally, we study the importance of the storage budget parameter. We present the performance of our IL method according to its budgets of 5%, 10%, and 20% of total training samples. In general, the more samples are used as exemplars, the higher the weighted F1-score the IL method can achieve. We also find that our method (FastICARL) needs only 5% budget size to achieves a weighted F1-score of 60-64% and successfully retain its weighted F1-score even after losing some information by applying quantization up to 8 bits on its exemplars.

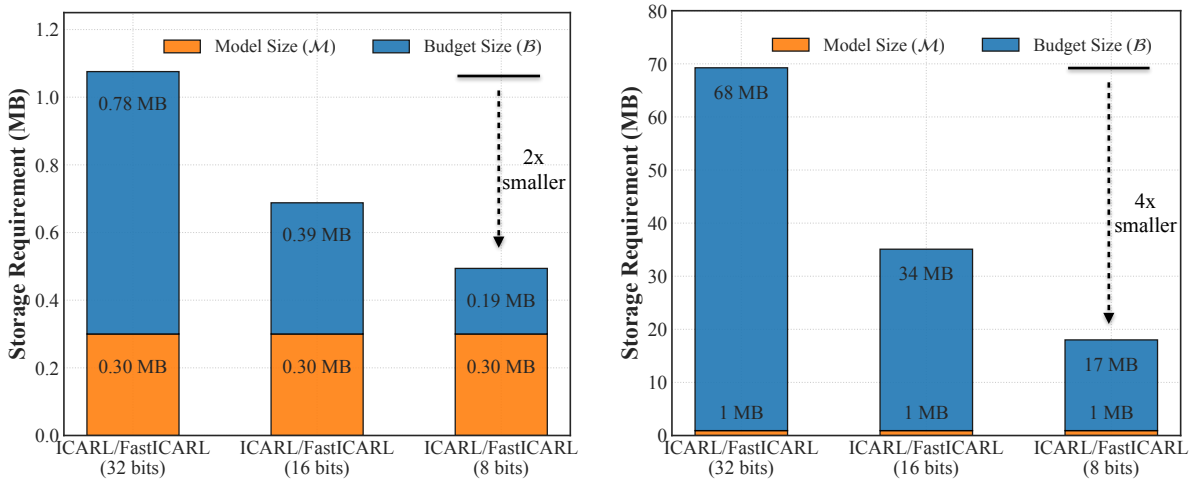
Latency: We measure the computational costs of sequentially learning additional classes based on a pre-trained model. The average IL time to run different IL methods is presented in Table B.2. The IL time of FastICARL (32, 16, and 8 bits) ranges 4.96-67.1 seconds on Jetson Nano and 0.83-11.2 seconds on Google Pixel 4 depending on the budget and datasets. FastICARL remarkably reduces the IL time by 18-78% on Jetson Nano and 37-92% on Google Pixel 4 compared to ICARL. Note that the training time of ICARL and FastICARL is approximately the same (these results are omitted for brevity). Also, FastICARL (16 and 8 bits) shows substantial improvement in IL time: this indicates that the additional operation of quantizing exemplars does not impose a meaningful burden on the system.

Storage: We now show the storage overhead of the IL method. The size of FastICARL is composed of the model parameter size ($\mathcal{M}$) and budget size ($\mathcal{B}$). As FastICARL relies on stored exemplars, its storage demand is primarily driven by the number of exemplars to be stored, i.e., budget size ($\mathcal{B}$). As shown in Figure B.1, FastICARL requires at most 0.49 MB for the EmotionSense dataset and 18 MB for the UrbanSound8K dataset, decreasing the storage requirement 2 to 4 folds over ICARL. Model sizes for EmotionSense and UrbanSound8K datasets are fixed as 0.3 MB and 1 MB, respectively.

Based on the results in this section, we have demonstrated that FastICARL enables faster IL by reducing the IL time and storage requirements by applying quantization.

B.4 Conclusions

In this paper, we developed an end-to-end and on-device IL framework, FastICARL, that enables efficient and accurate IL in mobile sensing applications. We implemented



(a) EmotionSense (b) UrbanSound8K

Figure B.1: Comparison of the storage requirement ($\mathcal{M} + \mathcal{B}$) for ICARL and FastICARL (32, 16, and 8 bits) based on 20% budget size in each dataset.

FastICARL on two resource-constrained devices (Jetson Nano and Google Pixel 4) and demonstrated its effectiveness and efficiency. FastICARL decreases the IL time up to 78-92% by optimizing the exemplar construction procedure and also reduces the storage requirements by 2-4 times by quantizing its exemplars without sacrificing the performance.

There are many interesting directions that deserve further research. First of all, we want to extend our work to enable a higher degree of quantization (such as using 2 or 3 bits) and apply pruning on a model to reduce the model parameters and speed up the training process, which is another bottleneck of the IL. Furthermore, it is worth investigating IL methods on more severely resource-constrained devices such as micro-controller units having meager system resources.

Bibliography

[1] Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S. Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, April 2016.

[2] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent Trends in Deep Learning Based Natural Language Processing [Review Article]. *IEEE Computational Intelligence Magazine*, 13(3):55–75, August 2018.

[3] Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T. Campbell. A survey of mobile phone sensing. *IEEE Communications Magazine*, 48(9):140–150, September 2010.

[4] Yu Guan and Thomas Plötz. Ensembles of Deep LSTM Learners for Activity Recognition Using Wearables. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(2):11:1–11:28, June 2017.

[5] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 119:3–11, March 2019.

[6] Junjun Fan, Xiangmin Fan, Feng Tian, Yang Li, Zitao Liu, Wei Sun, and Hongan Wang. What is That in Your Hand?: Recognizing Grasped Objects via Forearm Electromyography Sensing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(4):161:1–161:24, December 2018.

[7] Yifei Jiang, Xin Pan, Kun Li, Qin Lv, Robert P. Dick, Michael Hannigan, and Li Shang. ARIEL: Automatic Wi-fi Based Room Fingerprinting for Indoor Localization. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 441–450, 2012.

[8] Hong Lu, Denise Frauendorfer, Mashfiqui Rabbi, Marianne Schmid Mast, Gokul T. Chittaranjan, Andrew T. Campbell, Daniel Gatica-Perez, and Tanzeem Choudhury. StressSense: Detecting Stress in Unconstrained Acoustic Environments Using Smartphones. In *Proc. UbiComp*, pages 351–360, 2012.

[9] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. Deep Learning for Audio Signal Processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, May 2019.

[10] Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks. *arXiv:1312.6211 [cs, stat]*, December 2013.

[11] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[12] B. Pfülb and A. Gepperth. A comprehensive, application-oriented study of catastrophic forgetting in DNNs. In *ICLR*, 2019.

[13] Michael McCloskey and Neal J. Cohen. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In *Psychology of Learning and Motivation*, volume 24, pages 109–165. January 1989.

[14] James L. McClelland, Bruce L. McNaughton, and Randall C. O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3):419–457, 1995.

[15] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, May 2019.

[16] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proc. National Academy of Sciences*, 114(13):3521–3526, March 2017.

[17] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proc. CVPR*, pages 2001–2010, 2017.

[18] Nils Y. Hammerla, Shane Halloran, and Thomas Plötz. Deep, Convolutional, and Recurrent Models for Human Activity Recognition Using Wearables. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 1533–1540, 2016.

[19] Young D. Kwon, Kirill A. Shatilov, Lik-Hang Lee, Serkan Kumyol, Kit-Yung Lam, Yui-Pan Yau, and Pan Hui. MyoKey: Surface Electromyography and Inertial

Motion Sensing-based Text Entry in AR. In *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 1–4, March 2020.

[20] Kiran K. Rachuri, Mirco Musolesi, Cecilia Mascolo, Peter J. Rentfrow, Chris Longworth, and Andrius Aucinas. EmotionSense: a mobile phones based adaptive platform for experimental social psychology research. In *Proc. UbiComp*, pages 281–290, September 2010.

[21] Biyi Fang, Xiao Zeng, and Mi Zhang. NestDNN: Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, pages 115–127, 2018.

[22] Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9017–9028. 2018.

[23] Tyler L. Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. REMIND Your Neural Network to Prevent Catastrophic Forgetting. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pages 466–483, Cham, 2020. Springer International Publishing.

[24] Aaqib Saeed, Tanir Ozcelebi, and Johan Lukkien. Multi-task Self-Supervised Learning for Human Activity Detection. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 3(2):61:1–61:30, June 2019.

[25] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1126–1135, Sydney, NSW, Australia, August 2017. JMLR.org.

[26] Xiaoxi He, Zimu Zhou, and Lothar Thiele. Multi-task zipping via layer-wise neuron sharing. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[27] Seulki Lee and Shahriar Nirjon. Fast and scalable in-memory deep multitask learning via neural weight virtualization. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, MobiSys '20, pages 175–190, New York, NY, USA, June 2020. Association for Computing Machinery.

[28] H. Jégou, M. Douze, and C. Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, January 2011.

[29] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv:1510.00149 [cs]*, February 2016.

[30] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient ConvNets. November 2016.

[31] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical Networks for Few-shot Learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4077–4087. Curran Associates, Inc., 2017.

[32] Taesik Gong, Yeonsu Kim, Jinwoo Shin, and Sung-Ju Lee. MetaSense: Few-shot Adaptation to Untrained Conditions in Deep Mobile Sensing. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, SenSys '19, pages 110–123, New York, NY, USA, 2019. ACM.

[33] Khurram Javed and Martha White. Meta-Learning Representations for Continual Learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 1820–1830. Curran Associates, Inc., 2019.

[34] Shawn Beaulieu, Lapo Frati, Thomas Miconi, Joel Lehman, Kenneth O. Stanley, Jeff Clune, and Nick Cheney. Learning to Continually Learn. *arXiv:2002.09571 [cs, stat]*, March 2020.

[35] Jagmohan Chauhan, Young D. Kwon, Pan Hui, and Cecilia Mascolo. ContAuth: Continual Learning Framework for Behavioral-based User Authentication. *Proc. IMWUT*, 4(4):122:1–122:23, December 2020.

[36] Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning. *arXiv preprint arXiv:1912.01100*, 2019.

[37] Xu Sun, Hisashi Kashima, and Naonori Ueda. Large-scale personalized human activity recognition using online multitask learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(11):2551–2563, 2012.

[38] Liangying Peng, Ling Chen, Zhenan Ye, and Yi Zhang. Aroma: A deep multi-task learning based simple and complex human activity recognition method using

wearable sensors. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(2):74, 2018.

[39] Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010.

[40] Md Abdullah Al Hafiz Khan, Nirmalya Roy, and Archan Misra. Scaling human activity recognition via deep learning-based domain adaptation. In *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–9. IEEE, 2018.

[41] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1):151–175, May 2010.

[42] Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 951–958, June 2009.

[43] Mark Palatucci, Dean Pomerleau, Geoffrey E Hinton, and Tom M Mitchell. Zero-shot Learning with Semantic Output Codes. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1410–1418. 2009.

[44] Li Fe-Fei, Fergus, and Perona. A Bayesian approach to unsupervised one-shot learning of object categories. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1134–1141 vol.2, October 2003.

[45] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, koray kavukcuoglu, and Daan Wierstra. Matching Networks for One Shot Learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3630–3638. 2016.

[46] Luca Bertinetto, João F. Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 523–531. 2016.

[47] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual Learning Through Synaptic Intelligence. In *Proc. ICML*, pages 3987–3995, 2017.

[48] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pages 4535–4544, 2018.

[49] Arslan Chaudhry, Puneet K. Dokania, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence. pages 532–547, 2018.

[50] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational Continual Learning. February 2018.

[51] Sandra Servia-Rodriguez, Cecilia Mascolo, and Young D. Kwon. Knowing when we do not know: Bayesian continual learning for sensing-based analysis tasks. *arXiv:2106.05872 [cs]*, June 2021.

[52] Z. Li and D. Hoiem. Learning without Forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, December 2018.

[53] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual Learning with Deep Generative Replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2990–2999. 2017.

[54] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[55] David Lopez-Paz and Marc\textquotesingle Aurelio Ranzato. Gradient Episodic Memory for Continual Learning. In *Proc. NIPS*, pages 6467–6476. 2017.

[56] Henry Friday Nweke, Ying Wah Teh, Mohammed Ali Al-garadi, and Uzoma Rita Alo. Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges. *Expert Systems with Applications*, 105:233–261, September 2018.

[57] Angkoon Phinyomark and Erik Scheme. EMG Pattern Recognition in the Era of Big Data and Deep Learning. *Big Data and Cognitive Computing*, 2(3):21, September 2018.

[58] Nicholas D. Lane, Petko Georgiev, and Lorena Qendro. DeepEar: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments Using Deep Learning. In *Proc. UbiComp*, pages 283–294, 2015.

[59] Ming Zeng, Haoxiang Gao, Tong Yu, Ole J. Mengshoel, Helge Langseth, Ian Lane, and Xiaobing Liu. Understanding and Improving Recurrent Networks for Human Activity Recognition by Continuous Attention. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers*, ISWC '18, pages 56–63, 2018.

[60] Vishvak S. Murahari and Thomas Plötz. On attention models for human activity recognition. pages 100–103, October 2018.

[61] Francisco Javier Ordóñez and Daniel Roggen. Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition. *Sensors*, 16(1):115, January 2016.

[62] Christoph Amma, Thomas Krings, Jonas Böer, and Tanja Schultz. Advancing Muscle-Computer Interfaces with High-Density Electromyography. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 929–938, 2015.

[63] Faizan Haque, Mathieu Nancel, and Daniel Vogel. Myopoint: Pointing and Clicking Using Forearm Mounted Electromyography and Inertial Motion Sensors. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 3653–3656, 2015.

[64] Vincent Becker, Pietro Oldrati, Liliana Barrios, and Gábor Sörös. Touchsense: Classifying Finger Touches and Measuring Their Force with an Electromyography Armband. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers*, ISWC '18, pages 1–8, 2018.

[65] Jamileh Yousefi and Andrew Hamilton-Wright. Characterizing EMG data using machine-learning tools. *Computers in Biology and Medicine*, 51:1–13, August 2014.

[66] Brent D. Winslow, Mitchell Ruble, and Zachary Huber. Mobile, Game-Based Training for Myoelectric Prosthesis Control. *Frontiers in Bioengineering and Biotechnology*, 6, 2018.

[67] T. Scott Saponas, Desney S. Tan, Dan Morris, Ravin Balakrishnan, Jim Turner, and James A. Landay. Enabling Always-available Input with Muscle-computer Interfaces. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '09, pages 167–176, 2009.

[68] Erik Scheme and Kevin Englehart. Electromyogram pattern recognition for control of powered upper-limb prostheses: state of the art and challenges for clinical use. *Journal of Rehabilitation Research and Development*, 48(6):643–659, 2011.

[69] Xiaolong Zhai, Beth Jelfs, Rosa H. M. Chan, and Chung Tin. Self-Recalibrating Surface EMG Pattern Recognition for Neuroprosthesis Control Based on Convolutional Neural Network. *Frontiers in Neuroscience*, 11, 2017.

[70] Kirill A. Shatilov, Dimitris Chatzopoulos, Alex Wong Tat Hang, and Pan Hui. Using Deep Learning and Mobile Offloading to Control a 3d-printed Prosthetic Hand. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 3(3):102:1–102:19, September 2019.

[71] Hong Lu, A. J. Bernheim Brush, Bodhi Priyantha, Amy K. Karlson, and Jie Liu. SpeakerSense: energy efficient unobtrusive speaker identification on mobile phones. In *Proceedings of the 9th international conference on Pervasive computing*, Pervasive'11, pages 188–205, San Francisco, USA, June 2011.

[72] Chenren Xu, Sugang Li, Gang Liu, Yanyong Zhang, Emiliano Miluzzo, Yih-Farn Chen, Jun Li, and Bernhard Firner. Crowd++: unsupervised speaker count with smartphones. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, UbiComp '13, pages 43–52, Zurich, Switzerland, September 2013.

[73] Yu Su, Ke Zhang, Jingyu Wang, and Kurosh Madani. Environment Sound Classification Using a Two-Stream CNN Based on Decision-Level Fusion. *Sensors*, 19(7):1733, January 2019.

[74] Youngki Lee, Chulhong Min, Chanyou Hwang, Jaeung Lee, Inseok Hwang, Younghyun Ju, Chungkuk Yoo, Miri Moon, Uichin Lee, and Junehwa Song. SocioPhone: everyday face-to-face interaction monitoring platform using multi-phone sensor fusion. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, MobiSys '13, pages 375–388, Taipei, Taiwan, June 2013.

[75] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello Edge: Keyword Spotting on Microcontrollers. *arXiv:1711.07128 [cs, eess]*, November 2017.

[76] Guoguo Chen, Carolina Parada, and Georg Heigold. Small-footprint keyword spotting using deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4087–4091. IEEE, 2014.

[77] Hong Lu, Wei Pan, Nicholas D. Lane, Tanzeem Choudhury, and Andrew T. Campbell. SoundSense: scalable sound sensing for people-centric applications on mobile phones. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, MobiSys '09, pages 165–178, Kraków, Poland, June 2009.

[78] Petko Georgiev, Sourav Bhattacharya, Nicholas D. Lane, and Cecilia Mascolo. Low-resource Multi-task Audio Sensing for Mobile and Embedded Devices via Shared Deep Neural Network Representations. *Proc. IMWUT*, 1(3):50:1–50:19, September 2017.

[79] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized Convolutional Neural Networks for Mobile Devices. pages 4820–4828, 2016.

[80] Mingxing Tan and Quoc Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, May 2019.

[81] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv:1602.07360 [cs]*, November 2016.

[82] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. SqueezeNext: Hardware-Aware Neural Network Design. pages 1638–1647, 2018.

[83] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, June 2018.

[84] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. pages 116–131, 2018.

[85] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*, April 2017.

[86] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, June 2018.

[87] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[88] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. pages 2820–2828, 2019.

[89] Mingxing Tan and Quoc V. Le. EfficientNetV2: Smaller Models and Faster Training. *arXiv:2104.00298 [cs]*, April 2021.

[90] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-All: Train One Network and Specialize it for Efficient Deployment. *arXiv:1908.09791 [cs, stat]*, April 2020.

[91] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[92] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for MobileNetV3. pages 1314–1324, 2019.

[93] Igor Fedorov, Ryan P. Adams, Matthew Mattina, and Paul Whatmough. SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers. pages 4977–4989, 2019.

[94] Mohamed S. Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D. Lane. Zero-Cost Proxies for Lightweight NAS. *arXiv:2101.08134 [cs]*, March 2021.

[95] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[96] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. pages 5687–5695, 2017.

[97] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[98] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. AutoCompress: An Automatic DNN Structured Pruning Framework for Ultra-High Compression Rates. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):4876–4883, April 2020.

[99] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers. pages 184–199, 2018.

[100] Hongjia Li, Ning Liu, Xiaolong Ma, Sheng Lin, Shaokai Ye, Tianyun Zhang, Xue Lin, Wenyao Xu, and Yanzhi Wang. ADMM-based Weight Pruning for Real-Time Deep Learning Acceleration on Mobile Devices. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, GLSVLSI '19, pages 501–506, New York, NY, USA, May 2019. Association for Computing Machinery.

[101] Tianyun Zhang, Shaokai Ye, Xiaoyu Feng, Xiaolong Ma, Kaiqi Zhang, Zhengang Li, Jian Tang, Sijia Liu, Xue Lin, Yongpan Liu, Makan Fardad, and Yanzhi Wang. StructADMM: Achieving Ultrahigh Efficiency in Structured Pruning for DNNs. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15, 2021.

[102] Ao Ren, Tianyun Zhang, Shaokai Ye, Jiayu Li, Wenyao Xu, Xuehai Qian, Xue Lin, and Yanzhi Wang. ADMM-NN: An Algorithm-Hardware Co-Design Framework of

DNNs Using Alternating Direction Methods of Multipliers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pages 925–938, New York, NY, USA, April 2019. Association for Computing Machinery.

[103] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. volume abs/1712.05877, 2017.

[104] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained Ternary Quantization. November 2016.

[105] Fengfu Li, Bo Zhang, and Bin Liu. Ternary Weight Networks. *arXiv:1605.04711 [cs]*, November 2016.

[106] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. *Advances in Neural Information Processing Systems*, 28, 2015.

[107] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv:1602.02830 [cs]*, March 2016.

[108] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 525–542, Cham, 2016.

[109] Milad Alizadeh, Javier Fernández-Marqués, Nicholas D. Lane, and Yarin Gal. An Empirical study of Binary Neural Networks' Optimisation. September 2018.

[110] Diwen Wan, Fumin Shen, Li Liu, Fan Zhu, Jie Qin, Ling Shao, and Heng Tao Shen. TBN: Convolutional Neural Network with Ternary Inputs and Binary Weights. pages 315–332, 2018.

[111] Shihui Yin, Zhewei Jiang, Jae-Sun Seo, and Mingoo Seok. XNOR-SRAM: In-Memory Computing SRAM Macro for Binary/Ternary Deep Neural Networks. *IEEE Journal of Solid-State Circuits*, 55(6):1733–1743, June 2020.

[112] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing Neural Networks with the Hashing Trick. In *International Conference on Machine Learning*, pages 2285–2294. PMLR, June 2015.

[113] Jingyuan Zhao, Zhang Sihao, and Zeng Jing. Review of the sparse coding and the applications on image retrieval. In *2016 International Conference on Communication and Electronics Systems (ICCES)*, pages 1–5, October 2016.

[114] Tiezheng Ge, Kaiming He, and Jian Sun. Product Sparse Coding. pages 939–946, 2014.

[115] Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. LCNN: Lookup-Based Convolutional Neural Network. pages 7120–7129, 2017.

[116] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing Deep Convolutional Networks using Vector Quantization. *arXiv:1412.6115 [cs]*, December 2014.

[117] Ting Chen, Lala Li, and Yizhou Sun. Differentiable Product Quantization for End-to-End Embedding Compression. In *International Conference on Machine Learning*, pages 1617–1626. PMLR, November 2020.

[118] Y. Kalantidis and Y. Avrithis. Locally Optimized Product Quantization for Approximate Nearest Neighbor Search. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2329–2336, June 2014.

[119] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4):744–755, April 2014.

[120] Yousun Ko, Alex Chadwick, Daniel Bates, and Robert Mullins. Lane Compression: A Lightweight Lossless Compression Method for Machine Learning on Embedded Systems. *ACM Transactions on Embedded Computing Systems*, 20(2):16:1–16:26, March 2021.

[121] Chi Zhang, Nan Song, Guosheng Lin, Yun Zheng, Pan Pan, and Yinghui Xu. Few-Shot Incremental Learning with Continually Evolved Classifiers. *arXiv:2104.03047 [cs]*, April 2021.

[122] Ali Cheraghian, Shafin Rahman, Pengfei Fang, Soumava Kumar Roy, Lars Petersson, and Mehrtash Harandi. Semantic-aware Knowledge Distillation for Few-Shot Class-Incremental Learning. *arXiv:2103.04059 [cs]*, March 2021.

[123] Xiaoyu Tao, Xiaopeng Hong, Xinyuan Chang, Songlin Dong, Xing Wei, and Yihong Gong. Few-Shot Class-Incremental Learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12180–12189, Seattle, WA, USA, June 2020. IEEE.

[124] Mengmi Zhang, Tao Wang, Joo Hwee Lim, Gabriel Kreiman, and Jiashi Feng. Variational Prototype Replays for Continual Learning. *arXiv:1905.09447 [cs]*, February 2020.

[125] Bernd Fritzke. A growing neural gas network learns topologies. In *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, page 625–632, Cambridge, MA, USA, 1994. MIT Press.

[126] Andreas Bulling, Ulf Blanke, and Bernt Schiele. A Tutorial on Human Activity Recognition Using Body-worn Inertial Sensors. *ACM Comput. Surv.*, 46(3):33:1–33:33, January 2014.

[127] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong Learning with Dynamically Expandable Networks. February 2018.

[128] Monika Schak and Alexander Gepperth. A study on catastrophic forgetting in deep LSTM networks. page 14.

[129] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming Catastrophic Forgetting by Incremental Moment Matching. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4652–4662. 2017.

[130] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[131] Sourav Bhattacharya and Nicholas D Lane. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, pages 176–189, 2016.

[132] Gido M. van de Ven and Andreas S. Tolias. Three scenarios for continual learning. *arXiv:1904.07734 [cs, stat]*, April 2019.

[133] Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1121–1128, Montreal, Quebec, Canada, June 2009.

[134] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & Compress: A scalable framework for continual learning. In *Proc. ICML*, pages 4528–4537, July 2018.

[135] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580 [cs]*, July 2012.

[136] Allan Stisen, Henrik Blunck, Sourav Bhattacharya, Thor Siiger Prentow, Mikkel Baun Kj\a ergaard, Anind Dey, Tobias Sonne, and Mads Møller Jensen. Smart Devices Are Different: Assessing and MitigatingMobile Sensing Heterogeneities for Activity Recognition. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, pages 127–140, 2015.

[137] Gary King and Langche Zeng. Logistic Regression in Rare Events Data. *Political Analysis*, 9(2):137–163, 2001.

[138] A. Reiss and D. Stricker. Introducing a New Benchmarked Dataset for Activity Monitoring. In *2012 16th International Symposium on Wearable Computers*, pages 108–109, June 2012.

[139] Thomas Stiefmeier, Daniel Roggen, Georg Ogris, Paul Lukowicz, and Gerhard Tröster. Wearable Activity Tracking in Car Manufacturing. *IEEE Pervasive Computing*, 7(2):42–50, April 2008.

[140] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. DeepSense: A Unified Deep Learning Framework for Time-Series Mobile Sensing Data Processing. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 351–360, Republic and Canton of Geneva, Switzerland, 2017.

[141] Harish Haresamudram, David V. Anderson, and Thomas Plötz. On the Role of Features in Human Activity Recognition. In *Proceedings of the 23rd International Symposium on Wearable Computers*, ISWC '19, pages 78–88, 2019.

[142] Manfredo Atzori, Arjan Gijsberts, Claudio Castellini, Barbara Caputo, Anne-Gabrielle Mittaz Hager, Simone Elsig, Giorgio Giatsidis, Franco Bassetto, and Henning Müller. Electromyography data for non-invasive naturally-controlled robotic hand prostheses. *Scientific Data*, 1:140053, December 2014.

[143] G. Li, A. E. Schultz, and T. A. Kuiken. Quantifying Pattern Recognition—Based Myoelectric Control of Multifunctional Transradial Prostheses. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 18(2):185–192, April 2010.

[144] Petko Georgiev, Nicholas D Lane, Kiran K Rachuri, and Cecilia Mascolo. Dsp.ear: Leveraging co-processor support for continuous audio sensing on smartphones. In *Proc. SenSys*, pages 295–309, 2014.

[145] M. Smith and T. Barnwell. A new filter bank theory for time-frequency representation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):314–327, March 1987.

[146] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert Gate: Lifelong Learning With a Network of Experts. pages 3366–3375, 2017.

[147] Dipankar Das, Naveen Mellempudi, Dheevatsa Mudigere, Dhiraj Kalamkar, Sasikanth Avancha, Kunal Banerjee, Srinivas Sridharan, Karthik Vaidyanathan, Bharat Kaul, Evangelos Georganas, et al. Mixed precision training of convolutional neural networks using integer operations. *arXiv preprint arXiv:1802.00930*, 2018.

[148] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Advances in neural information processing systems*, pages 7675–7684, 2018.

[149] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. A Dataset and Taxonomy for Urban Sound Research. In *Proc. ACM MM*, pages 1041–1044, November 2014.

[150] Akhil Mathur, Nadia Berthouze, and Nicholas D. Lane. Unsupervised Domain Adaptation Under Label Space Mismatch for Speech Classification. In *Proc INTERSPEECH*, pages 1271–1275, October 2020.

[151] Ashish Mittal, Samarth Bharadwaj, Shreya Khare, Saneem Chemmengath, Karthik Sankaranarayanan, and Brian Kingsbury. Representation Based Meta-Learning for Few-Shot Spoken Intent Recognition. In *Proc INTERSPEECH*, pages 4283–4287, October 2020.

[152] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices. In *Proc. IPSN*, pages 1–12, April 2016.

[153] Ruoming Pang, Tara Sainath, Rohit Prabhavalkar, Suyog Gupta, Yonghui Wu, Shuyuan Zhang, and Chung-Cheng Chiu. Compression of end-to-end models. In *Proc. INTERSPEECH*, pages 27–31, 2018.

[154] Xiaotang Jiang, Huan Wang, Yiliu Chen, Ziqi Wu, Lichuan Wang, Bin Zou, Yafeng Yang, Zongyang Cui, Yu Cai, Tianhang Yu, Chengfei Lyu, and Zhihua Wu. MNN: A Universal and Efficient Inference Engine. *Proc. MLSys*, 2:1–13, March 2020.

[155] Juhyun Lee, Nikolay Chirkov, Ekaterina Ignasheva, Yury Pisarchyk, Mogan Shieh, Fabio Riccardi, Raman Sarokin, Andrei Kulik, and Matthias Grundmann. On-Device Neural Net Inference with Mobile GPUs. *arXiv:1907.01989 [cs, stat]*, July 2019.

[156] PyTorch. https://www.pytorch.org.

[157] Lisa Feldman Barrett and James A Russell. Independence and bipolarity in the structure of current affect. *Journal of personality and social psychology*, 74(4):967, 1998.

[158] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017.