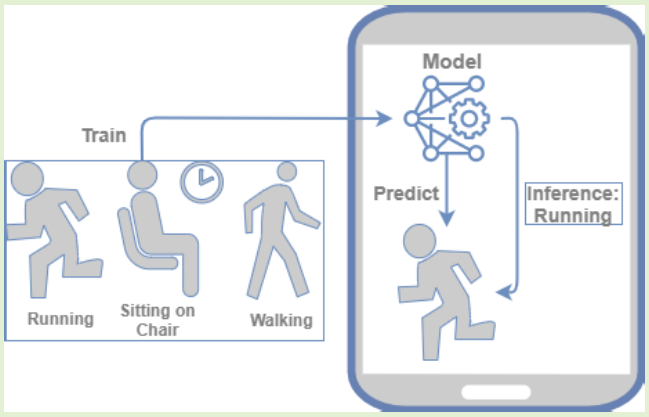
On-Device Deep Learning for Mobile and Wearable Sensing Applications: A Review

Ozlem Durmaz Incel and Sevda Özge Bursa

***Abstract*—Although running deep-learning (DL) algorithms is challenging due to resource constraints on mobile and wearable devices, they provide performance improvements compared to lightweight or shallow architectures. The widespread application areas for on-device DL include computer vision, image processing, natural language processing, and audio classification. However, mobile and wearable sensing applications are also gaining attention. They can benefit from on-device DL, given that these devices are integrated with various sensors and produce large amounts of data. This article reviews state-of-the-art studies on on-device DL for mobile and wearable devices, particularly from the sensor data analytics perspective. We first discuss the general optimization techniques of DL algorithms to meet the resource limitations of the devices. Then, we elaborate on model update and personalization techniques and review the studies by classifying them according to several aspects, including application areas, sensors, types of devices, utilized DL algorithms, mode of implementation, methods for optimizing DL algorithms for the target devices, training method, implementation toolkit/platform, performance metrics, and resource consumption analysis. Finally, we discuss the open issues and future research directions about on-device DL for mobile and wearable sensing applications.**



***Index Terms*— Deep learning (DL), resource management, sensing, wearable devices.**

I. Introduction

WEARABLE and mobile devices, including smartphones, smartwatches, and smart glasses, have become a part of our daily lives by assisting us for various purposes. In recent years, various machine-learning-based mobile applications have been developed. Particularly the use of deep-learning (DL) algorithms [1] or deep neural networks (DNNs) is emerging: image and face recognition using a camera, speech recognition using a microphone, and activity tracking using motion sensors are only a few examples.

As mentioned in [2], there are several advantages of a well-trained DL model: 1) extracting useful features from raw data (sometimes even noisy); 2) performance improvements, compared to the lightweight or shallow architectures on massive data; and 3) less requirement on feature engineering. When deploying these DL-based applications to the mobile and wearable platforms [3], there can be different configurations: 1) both training and inference in the cloud; 2) on-device inference with cloud-trained models; and 3) both training and inference on the device. Fig. 1 summarizes these modes. In the first approach, named as *online mode*, or in-cloud mode [4], mobile and wearable devices are used to collect data and then the data is transferred to the cloud, where both training and inference are performed, and the result is returned to the device. In the second approach, named *offline mode*, the training is again performed in the cloud or on a more capable device, that is, an edge device, however, the trained model is optimized to be ported to the mobile device, and the inference is performed locally on the device. Training machine-learning algorithms on resource-constrained mobile and wearable devices, particularly DL algorithms, is challenging and sometimes even impossible due to the limited computation power, storage, and, most importantly, the battery. However, we have recently seen efforts to train models directly on devices or retrain/customize the trained models on the device. In other words, these efforts handle the end-to-end learning process on the device [4]. Hence, the third mode can

Manuscript received 31 August 2022; revised 7 January 2023; accepted 17 January 2023. Date of publication 7 February 2023; date of current version 14 March 2023. The work of Ozlem Durmaz Incel was supported by the Bogazici University Research Fund under Project 19301. The associate editor coordinating the review of this article and approving it for publication was Prof. Shih-Chia Huang. *(Corresponding author: Ozlem Durmaz Incel.)*

Ozlem Durmaz Incel is with the Department of Computer Engineering, Bogazici University, 34342 Istanbul, Turkey (e-mail: ozlem.durmaz@boun.edu.tr).

Sevda Özge Bursa is with the Department of Computer Engineering, Galatasaray University, 34349 Istanbul, Turkey (e-mail: sevdaozge.bursa@ogr.gsu.edu.tr).

Digital Object Identifier 10.1109/JSEN.2023.3240854

1558-1748 © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on December 11,2024 at 06:49:30 UTC from IEEE Xplore. Restrictions apply.

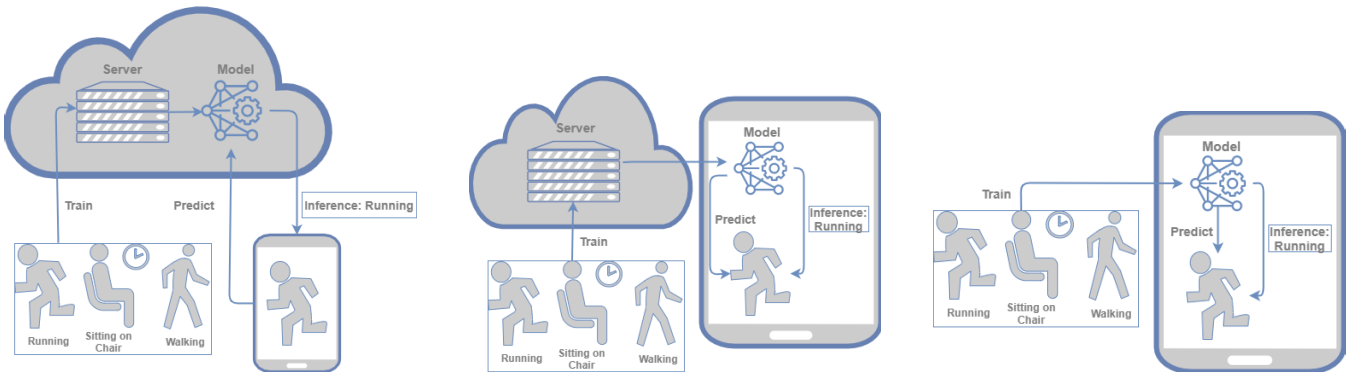


Fig. 1. Modes of running DL algorithms on mobile and wearable devices: example HAR. (a) Online mode (first mode). (b) Offline mode (second mode). (c) End-to-end mode (third mode).

be named the *end-to-end mode*. This article focuses on the second and third modes.

Besides the mentioned resource-related challenges, other challenges include a diverse computing environment, limited peak speed, limited user data, and overfitting of models, backward propagation blocking, which should be addressed by the codesign of algorithms and hardware [4]. Despite these challenges, in the literature [3], [5], efforts are focusing on the second and third modes (*offline* and *end-to-end* modes), which become possible with the introduction of dedicated hardware accelerators, the latest neural-network (NN) hardware, multicore processors, and larger memory becoming more common in mobile and wearable devices. Parallel to the hardware improvements, also there are emerging efforts to optimize, compress, and accelerate DL models without significantly reducing the accuracy [3] in a resource-constrained environment. Examples of mobile DL frameworks include TensorFlow Lite (TF Lite), Caffe2, CoreML, and so on.

The widespread application areas for on-device or mobile DL include computer vision (motion tracking, style transfer, and arts), image recognition (scene recognition, pose estimation, and face recognition), natural language processing (text classification, on-device translation, and auto-completion), and audio classification (speech recognition) [1]. This is also related to the fact that DL algorithms have been chiefly applied and proven effective on these types of massive data [6]. Mobile and wearable devices integrated with various sensors frequently generate large amounts of data. Sensor-based machine-learning apps on mobile and wearable devices are also gaining attention [1], such as in healthcare, sports, and well-being. Particularly, applications that require real-time processing, real-time responding, data privacy, and personalized models can benefit from on-device inference using DL.

This article explores on-device DL for mobile and wearable devices, particularly from the sensing applications perspective. In the literature, different recent survey papers focus on on-device DL or mobile DL [2], [3], [4], [6], [7], [8]. For example, in [3], the focus is on adapting models for mobile devices and hardware platforms but mostly on image-based data. In [4], the motivation for shifting in-cloud learning to on-device learning and possible difficulties in system implementation are discussed both from the hardware and software perspectives. Another survey [2] focused on the solutions of DL with limited resources. In [7], on-device machine learning is discussed from the perspective of algorithms and learning theory, whereas in [6], approximation methods for high-performance network inference for custom hardware implementation are discussed from the hardware perspective. A recent survey [8] also explores DL on mobile devices, however, their focus is not particularly on sensing applications but on mobile applications, including image processing, speech recognition, mobile security, and human activity recognition (HAR).

Unlike the recent surveys, we investigate on-device DL on resource-constrained mobile and wearable devices for sensing applications and review the state-of-the-art studies. Another difference is that we also review the methods to update/adapt a model with dynamic/new inputs on the device, whereas other surveys mainly review the methods to optimize DL models for on-device inference. First, we provide background information on sensing applications that can benefit from on-device DL, characteristics of sensors, sensor data, and devices. Next, we focus on the techniques for optimizing the DNNs for resource-constrained devices. We review and categorize the state-of-the-art studies according to the following aspects: 1) application areas; 2) sensors; 3) types of devices used; 4) utilized DL algorithms; 5) methods or DL toolkits for optimizing DL algorithms for the target devices; 6) implementation method (offline versus end-to-end); 7) whether the methods are validated on a device, whether an on-device model update is supported, whether resource consumption analysis is performed; and 8) performance metrics. Our contribution is to present state-of-the-art studies and guide the researchers that aim to apply DL algorithms on devices for mobile and wearable sensing applications.

While reviewing the state-of-the-art studies, we searched on Google Scholar and Semantic Scholar using keywords including "on-device DL," "wearable computing," "sensing," and "optimizing DL networks." The following criteria are used for the selection of the studies.

1) They implement the DL algorithms on mobile and wearable devices or transform/simplify them to be implemented on target devices. We exclude those that focus on the online or in-cloud mode.

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on December 11,2024 at 06:49:30 UTC from IEEE Xplore. Restrictions apply.

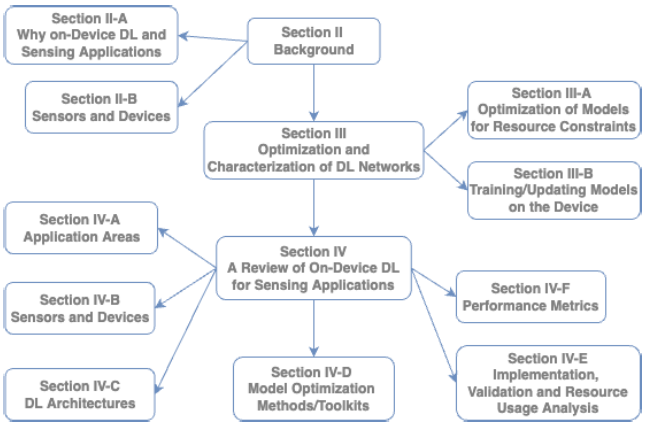


Fig. 2. Schematic of the review.

2) They use mobile and wearable sensors, excluding cameras and microphones.

The reason for excluding the camera and the microphone is that, in other surveys [1], [3], on-device DL with video, audio, and text data are analyzed. We exclude those focusing on the online mode and only apply DL on the datasets but do not consider on-device implementation. Zhou et al. [4] mentioned, "Generally, the on-device learning applications are often deployed on edge devices. Thus, on-device learning also refers to the edge intelligence [9] sometimes." In edge systems, a mesh of compute nodes can be placed close to end devices. Hence, there can be some data transfer between the edge and the end device, which does not directly match our definition of on-device learning. Therefore, we exclude edge learning but include only the studies where a smartphone is used as an edge device for an end wearable device.

Fig. 2 presents the organization of the review. In Section II, we provide background information on common DL architectures, while in Section III, we discuss the optimization techniques for DNNs on mobile and wearable devices. In Section IV, we classify and review the relevant state-of-the-art studies, and Section V includes a discussion and open research issues and finally, Section VI concludes this article.

II. Background

A. Why On-Device DL and Sensing Applications

One of the advantages of on-device DL is the *fast response time* or *limited latency*. There will be no overhead in communication time or worry about the server or link reliability with all the local computation, and this would make it easy to analyze data locally and also in real time [5].

Models built for mobile devices should be smaller and *energy-efficient* than models run on cloud servers. The cost of maintaining or renting cloud computing resources can be reduced, and the communication bandwidth between the devices and cloud computing systems can be saved. These all contribute to another advantage which is *resource saving* in terms of hardware and energy. Wearable and mobile devices are personal devices. Hence, they generate privacy-sensitive personal data. Local models can keep data on the device and significantly improve the *privacy of user data* [4]. Besides, a general machine-learning model trained with data from other users may not work well for some users, and such applications may require *personalized models* [4]. A general model can be fine-tuned/adapted or retrained on the device with personal data to deliver personalized services to enhance user experience. As these devices have become a part of our daily lives, they generate large amounts of data, and sensor data is one of the categories which rapidly growing. With the increasing capacity of computing hardware on mobile and wearable devices, such as AI chipsets, DL becomes possible on extensive sensor data.

There are various sensing application domains for mobile and wearable devices: health, sports, well-being, mood/stress recognition, activity recognition, mobility tracking, authentication, localization, rehabilitation, elderly care, sleep monitoring, augmented and virtual reality, and occupational safety. However, not all of them may require on-device learning and instead may utilize the first mode or in-cloud mode [see Fig. 1(a)]. For example, if we are interested in tracking a user's sleeping patterns, the data may be collected and processed on a server for offline analysis. Applications not generating large or real-time data may not require on-device DL and may perform well with traditional machine-learning techniques.

Privacy can be an issue for most sensor data collected from wearable and mobile devices, particularly for health, authentication, localization, and well-being recognition applications. Some applications may require personalized models and may benefit from on-device training or updating trained models. Examples of such applications are gait and activity recognition, authentication, healthcare, well-being, and mood/stress recognition. Applications that require lower latency and near real-time applications may also benefit from on-device learning, such as fall detection, authentication, augmented/virtual reality, mobility tracking, and activity recognition. We review the state-of-the-art studies with different applications in Section IV.

B. Sensors and Devices

We target sensors such as motion sensors (accelerometers, gyroscopes, and magnetic field), location sensors (GPS and wireless interfaces), pressure, thermometers, electrodermograph (EDA), electromyography (EMG), electroencephalogram (EEG), electrocardiogram (ECG), galvanic skin response (GSR), photoplethysmography (PPG), heart rate (HR), electrooculography (EOG), oximeter, and proximity (such as Bluetooth). In most sensing applications, different types of sensors, such as motion and location, are used together. Hence, the sensor data is mostly multimodal, which is different from dealing with audio, video, or textual data, and sequence information needs to be captured [10].

We particularly survey mobile and wearable devices, hence personal devices integrated with the mentioned sensors. Smartwatches, wristbands, smart eyewear, e-textiles, e-patches, smart jewelry, straps, and smartphones are devices in our focus. Resource constraints mainly characterize the devices investigated in this review. As explained in [7], processing speed affects an algorithm's throughput, latency, and response time. Another limited resource is memory. DL algorithms often require a significant amount of memory for

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on December 11,2024 at 06:49:30 UTC from IEEE Xplore. Restrictions apply.

model training. The most critical resource is power. Designing energy-efficient algorithms on battery-powered devices is essential, as we discuss in Section III-A. We present the devices used in the reviewed papers in Section IV-B.

III. Optimization and Characterization of DL Networks for Mobile and Wearable Devices

DL algorithms consist of NNs containing one or more hidden layers. What distinguishes DL from traditional machine-learning methods is its processing and powerful information extraction capabilities using considerable computational resources. The most popular architectures include the DNN, convolutional NN (CNN), and recurrent NN (RNN). Due to the page limitations, we do not elaborate on DL models; however, in Section IV-C, we present the popular architectures used in state-of-the-art studies.

The high computation cost of DL models makes it challenging to train and deploy the models on resource-constrained mobile and wearable devices. From the algorithmic point of view, compression and acceleration techniques are proposed in the literature to overcome this challenge. We discuss the algorithmic aspects in this section. Due to the page limitations, we skip the hardware aspects, but one can refer to [6] and [3], for the hardware solutions to run DL on mobile devices.

One of the popular approaches is to optimize (i.e., compress and accelerate) a network after training. In this case, the model can be trained and optimized on an external device by achieving a reduction in model size and then can be ported to a resource-constrained device. Another approach is to optimize a network during training with resource constraints in mind. Again, an external device can be used for training, or the model can be trained on a mobile device if sufficient resources are available. First, in Section III-A, we discuss the techniques that can be used after or during training. If the model is ported to a mobile device after training, such a static model will not change after optimization and will not adapt to dynamic/new inputs on the device [7]. To solve this problem, either the model can be fully trained on a mobile device, or an already trained model ported to a device can be updated. We also discuss the techniques to update a trained network in Section III-B.

A. Optimization of Models for Resource Constraints

Pruning is a commonly used technique that compresses the network by eliminating unimportant parameters. Nonstructured pruning can be done at the weight, layer, or block levels. For example, magnitude-based weight pruning identifies parameters that have a negligible impact on a model's predictions and prunes synapses and neurons to eliminate redundant connections, reducing model size with minimal reduction in accuracy [11]. A more regular network structure can be obtained in structured pruning, which is easier for parallel computation. Types of structured pruning, namely, vector-level and kernel-level pruning channel-level and filter-level pruning are explained in [2]. A different approach, which uses energy-aware pruning, is proposed in [12].

1) Low-Rank Factorization: The parameters of an NN may have redundancy and this may cause training to be slow. Low-rank factorization can be applied to avoid parameter redundancy [13]. The low-order factorization algorithm combines linearly dependent vectors and gradually reduces the size of the parameters.

One of the approaches that make mobile machine-learning possible is *quantization*. It compresses the overall size of the model by representing a number with low precision. For example, instead of representing data with 32-bit-wide precision, it is possible to use parameters with lower precision, such as 8-bit-wide fixed-point value [4], without reducing the success of the model.

Knowledge distillation (KD) is a model compression technique in which a smaller "student" NN is taught by the "teacher," the larger pretrained NN. There are different categories of KD. Response-based knowledge represents the final output layer and imitates the teacher model's prediction. With feature-based KD, both middle and final layer output can control student model training. Relation-based KD extracts relationships between different layers or data samples [14]. *Transfer learning* is another approach similar to KD. However, KD requires a predefined student model and modifies its loss function. Transfer learning is based on the idea of training a large network on a large dataset and we will discuss how it can help to update a trained model on devices in Section III-B.

2) Network Design Strategies: NNs, which have a vast design space, can be compressed, and efficient, compact, and low-cost network architecture can be achieved. Thus, reductions in storage and computational requirements can be achieved. Global average pooling with a fully connected layer, branching, depthwise separable convolution, splitting a layer into multiple layers, and decreasing the number of filters are examples of network design strategies [3].

3) Modifying Optimization Routines: While training the models, the focus is on minimizing the error rate, but we can also take the computational complexity, memory footprint, computing speed, and power consumption into account. For example, Stamoulis et al. [15] formulated the hyperparameter optimization problem by considering both minimizing error rate and energy minimization using Bayesian optimization techniques. They show that the method reduces the energy consumed for image classification on a mobile device by up to 6× in terms of energy when tested on a commercial Nvidia mobile SoC.

Hardware-aware neural architecture search (NAS) methods [16] are useful to run DNN models efficiently on devices with different hardware and tasks. NAS consists of three components. The search area represents different architectures and different value ranges for the hyperparameters. Search algorithms explore the search space to find the best architecture. The evaluator is used to obtain the accuracy of the model. With NAS algorithms, search fields are optimized according to the target hardware.

The input size in DL models directly affects the overall size of the NN, which determines the use of resources during the training and execution phases. In *data compression/transformation*, instead of reducing the model size or complexity, another approach could be to build models on compressed data by limiting memory usage. For example, data

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on December 11,2024 at 06:49:30 UTC from IEEE Xplore. Restrictions apply.

can be transformed to a lower-dimensional space [17] or can be sampled at a lower rate.

These techniques can also be combined to achieve better performance. For example, Han et al. [18] proposed a three-stage compression pipeline: pruning, quantization, and Huffman coding. In [19], AdaDeep enables an automatic selection of DNN compression techniques that balance the performance and resource constraints. Instead of creating a model from scratch, similar pretrained models can be used and fine-tuned if domain-specific data is available.

B. Training/Updating Models on the Device

Model updates can be particularly important for personalizing a general model with new incoming user data on the device. As mentioned, sensing data on wearable devices exhibit the personal characteristics of the user, and data distribution changes in time with user behavior. This is notably reported in user authentication and HAR studies [20]. Moreover, new users or classes may need to be added, requiring a model update. As mentioned in [21], deep classifiers are not good at extrapolation when the data comes from a different distribution. However, the occurrence of this is quite common in mobile applications. Hence, the models should be retrained or updated. Besides personalization, model-update helps avoid overfitting with limited data and reduces model complexity.

Model fine-tuning, transfer learning [4] is one of the common methods in updating an already trained model. It includes: 1) pretraining a large network on a large dataset and applying the model to the on-device dataset using transfer learning and 2) defining some of the layers as updatable and some frozen, and fine-tuning these layers in training. *Modelwise feature sharing* (or feature transfer) can also be applied in transfer learning [4]. Instead of or besides layerwise adaptation, we can share the features across models by transferring the corresponding parameters since different tasks can have a similar learning objective. Details of transfer learning techniques are presented in a survey paper [22].

1) Incremental/Online Learning: In the incremental or online learning approach, when new data streams, an existing model (metrics, weights) is updated to optimize the performance if a label is available. It is used when the training data points arrive sequentially or when it is not possible to train over the entire dataset. The methods are reported to be memory and run-time efficient [23]. Hence, it can be an appropriate approach for the mobile model update.

2) Continual or Lifelong Learning: Lifelong learning [23] uses continuous learning based on the idea that a model is trained with a sequence of N learning tasks and when a new class is encountered ($N + 1$), the model uses the past knowledge to help learn the $N + 1$th task. It usually focuses on optimizing the performance of the new task where a new task can be given or discovered by the system. Kwon et al. [10] introduced a continual learning framework and test its performance on Nvidia Jetson Nano and One Plus Pro smartphone platforms.

Besides these approaches, distributed learning approaches are also emerging, such as federated learning (FL) and split learning (SL) to update models on local devices. FL [24] is a collaborative learning technique where training happens across multiple devices without exchanging or storing centralized training data. Instead, the devices only share the model parameters with a central orchestrating unit. SL [25] divides a single NN into parts and distributes the lower layers across multiple devices. The devices share the parameters of the cut-layer with a server that orchestrates the training. Hence, both approaches use on-device training and can benefit from the optimization approaches on resource-constrained devices.

TABLE I
APPLICATION AREAS

Application Area	Studies
HAR	[11], [17], [19], [26]–[42]
Continuous authentication	[20], [21], [43], [44]
Gesture recogition	[40], [41], [45]–[47]
Stress detection	[48], [49]
Mobile health	[41], [50]
Sleep state monitoring	[33], [51], [52]
Mental anomaly detection	[53]
Pedestrian navigation	[54]
Context recogition	[33]
User identification	[34]
WiFi Sensing	[55]
Device identification	[56]
Cattle activity recognition	[57]
Anomaly detection	[58]
Indoor localization	[17]
Gait/step recognition	[33], [59], [60]
Road health monitoring	[61]
Walking speed	[62]

IV. REVIEW OF ON-DEVICE DL FOR SENSING APPLICATIONS

We provide a taxonomy by categorizing the studies according to: 1) application areas; 2) sensors; 3) types of devices used; 4) utilized DL algorithms; 5) methods or DL toolkits for optimizing DL algorithms for the target devices; 6) implementation method (offline versus end-to-end); 7) whether the methods are validated on a device, whether an on-device model update is supported, whether resource consumption analysis is performed; and 8) performance metrics.

A. Application Areas

Table I presents the target application areas in the reviewed studies. HAR using sensor data from wearable and mobile devices is a popular field of research in the literature and reviewed papers. Twenty papers out of 47 focused on HAR. On-device implementation of an HAR system benefits the mentioned advantages, including preserving privacy, personalized models, and real-time operation. Although recent survey papers [63], [64], [65] explore the studies on using DL algorithms for HAR datasets and highlight the importance of "online and mobile deep activity recognition," they do not focus on the on-device implementation of the DL algorithms.

The second most popular application area is continuous authentication (CA). It focuses on tracking the user's behavioral patterns, such as gait, using the sensors available on the devices and identifying the user using classification or outlier detection algorithm. CA can benefit from the privacy and real-time operation advantages of on-device implementation.

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on December 11,2024 at 06:49:30 UTC from IEEE Xplore. Restrictions apply.

TABLE II
SENSORS USED IN RELEVANT STUDIES

Sensors	Studies
IMUs	[10], [17], [19], [26]–[28], [30], [31], [33]–[41] [42], [43], [45], [46], [54], [57], [62], [66], [67]
Accelerometer	[29], [32], [58], [59], [68], [69]
EEG	[33], [51], [53]
temperature	[28], [49]
HR	[28], [48]
EMG	[20], [33]
ECG	[49], [70]
NW/channel info	[55], [56]
GSR	[49]
PPG	[50]
EOG, light, proximity	[33]
GPS	[34]
microphone	[21]

Gesture recognition is also a specific application area to benefit from on-device implementation with real-time operation avoiding delays in communicating with a cloud server. Health and well-being applications like stress, sleep state, and mental anomaly detection are applications that can benefit from privacy preservation with locally operating models. Learning pedestrian trajectories or tracking users can benefit from the real-time operation of on-device models. *The reviewed studies did not explore other application areas, such as sports, rehabilitation, elderly care, augmented and virtual reality, and occupational safety. Researchers can investigate new mobile sensing applications that can benefit from on-device DL algorithms.*

B. Hardware: Sensors and Devices

In this section, we categorize the studies according to the sensor types used and the types of target devices where DL algorithms are implemented. In Table II, we provide the list of sensors used in the reviewed studies. The most common sensors are the motion sensors, including inertial measurement units (IMUs integrating accelerometers, gyroscopes, and/or magnetometers) and only accelerometers. These sensors are primarily used in activity, gesture recognition, and motion tracking and are continuously sampled. Hence, they generate enormous data which can be better interpreted with DL algorithms. EEG, temperature, HR, GSR, and PPG sensors are commonly used in emotion/stress recognition and sleep state monitoring applications. Network and channel state information are used in device identification studies. In some papers, multiple sensors are evaluated to recognize more detailed activities or states. As the number of sensors, or inputs, increase, resource consumption and the complexity of the DL models may increase. *Hence, the tradeoff between better recognition and higher resource consumption should be addressed. Efficient merging of multimodal inputs and using architectures that can deal with such data and model the temporal relationships are crucial [34].*

Tables III and IV categorize the studies according to the target devices where DL algorithms are implemented/validated. We also provide their features/capabilities. The most commonly used platforms are the smartphones, such as Samsung Galaxy, Nexus, Google Pixel, HTC, and Huawei. Nexus 5 is integrated with Snapdragon 800 SoC. Hence, they have

TABLE III
DEVICES WITH GPU OR ACCELERATOR AND SMARTPHONES USED IN RELEVANT STUDIES

Device	Studies	CPU GPU/AI Chip
Nvidia Jetson TK1	[17], [66]	Quad-Core ARM Cortex A15 192 CUDA cores
Nvidia Jetson TX2	[20]	Hex-Core ARMv8 64-bit 256 CUDA cores
Nvidia Jetson Nano	[10], [53]	Quad-Core ARM Cortex A57 128 Core Maxwell GPU
Google Coral Edge TPU	[57]	Quad Cortex-A53 TPU coprocessor, 4 TOPS (int8)
ARM Cortex-M0 with accelerator	[40]	ARM Cortex-MO Intel DE10-Lite FPGA
PYNQ-Z1	[71]	Xilinx ZYNQ XC7Z020-1CLG400C 650MHz FPGA
Nexus 5	[27], [34], [35] [37], [72]	Quad-Core 2.3 GHz Krait 400 Adreno 330 GPU
Nexus 5X	[29], [46]	4 Cortex-A53, 2 Cortex-A57 Adreno 418 GPU
Nexus 6	[21], [43], [46]	Octa-Core 4x1.55 GHz Cortex-A53, 4x2.0 GHz Cortex-A57 Adreno 430 GPU
Nexus 6P	[21], [37], [43]	Octa-Core ARM Cortex-A53 Adreno 430 GPU
Samsung Galaxy Tab S3	[42]	Dual-core 1.2 GHz Cortex-A9 PowerVR SGX540 GPU
Samsung Galaxy S5	[27], [38]	Quad-Core 2.5 GHz Krait 400 Adreno 330 GPU
Samsung Galaxy S6	[62]	Octa-core 4x2.1 GHz Cortex-A57 4x1.5 GHz Cortex-A53 Mali-T760MP8 GPU
Samsung Galaxy S7	[45]	Octa-Core 4x2.3 GHz Mongoose, 4x1.6 GHz Cortex-A53 Mali-T880 MP12
Samsung Galaxy Nexus	[72]	Dual-core 1.2 GHz Cortex-A9 PowerVR SGX540 GPU
Samsung Galaxy Alpha	[61]	Octa-Core 4x1.8 GHz Cortex-A15, 4x1.3 GHz Cortex-A7 Mali-T628 MP6 GPU
Google Pixel	[41]	Kryo Quad-Core CPU,2.4GHz Adreno 530 GPU
Google Pixel 2	[51]	4x2.35 GHz Kryo, 4x1.9 GHz Kryo Adreno 540 GPU
Google Pixel 2XL	[41], [43]	8x Kryo 280 CPU Adreno 540 GPU
One Plus 7 Pro	[10]	Octa-Core Kryo 485 Adreno 640 GPU
Huawei Mate 8	[54], [59]	Octa-Core 4x2.3 GHz Cortex-A72, 4x1.8 GHz Cortex A53 Mali-T880 MP4 GPU
Huawei Mate 10	[43]	Octa-Core 4x2.4 GHz Cortex-A73, 4x1.8 GHz Cortex-A53
Huawei P10	[11]	Octa-Core 4x2.4 GHz Cortex-A73, 4x1.8 GHz Cortex-A53 Mali-G71 MP8 GPU
Huawei Nova 6 SE	[41]	Octa-Core 2x2.27 GHz Cortex-A76, 6x1.88 GHz Cortex-A55 Mali-G52 MP6 GPU
Huawei Honor 7A	[61]	Octa-Core 4x1.4 GHz Cortex-A53 4x1.1 GHz Cortex A53 Adreno 505 GPU
Snapdragon 800	[33], [34]	Quad-Core 2.3 GHz Krait 400 Adreno 330 GPU

the same configurations as presented in Table III. Although smartphones have GPUs, for example, Radu et al. [33], Yao et al. [34], and Zhang et al. [41] explicitly stated that only the CPU is used while running the DL algorithms. On the other hand, MobiRNN [37] framework focuses on optimization for RNNs by offloading DL tasks to the mobile GPU. It uses a mobile-specific parallelization framework, "RenderScript." The performance of the model is tested on Nexus 5 and Nexus 6p. They show that using the GPU is 3.93 times faster than the CPU, but the speed depends on the mobile device and the complexity of the model, and running a multithreaded RNN model on GPU brings an average of 32% increase in speed. Similarly, Alzantot et al. [46] utilized the RenderScript

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on December 11,2024 at 06:49:30 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
IOT DEVICES AND SMARTWATCHES IN RELEVANT STUDIES

Device	Studies	CPU/GPU
Snapdragon 400	[33], [73]	Quad-Core ARM Cortex-A7 Quad ARM Cortex-A53 GPU
Intel Edison	[27], [34]	Atom 2-Core (Silvermont) 500 MHz
Raspberry Pi 2	[48], [68]	Quad-Core ARM Cortex-A7
Raspberry Pi 3B	[36], [40], [74]	4xARM Cortex A-53
Raspberry Pi 4	[56], [75]	Quad-Core Cortex-A72
Arduino Nano 33	[58]	ARM Cortex M4 64 MHz
MinnowBoard Turbot	[28]	Intel Atom E3826 processor, 1.46 GHz
UDOO Neo Full Board	[67]	ARM Cortex-A9 core Cortex-M4 Core
LG G Watch R	[44]	Quad-Core 1.2 GHz Cortex-A7 Adreno 305 GPU
Custom bracelet	[49]	ARM Cortex M4F
Sony Smartwatch 2	[54]	1 core ARM Cortex M4
Samsung Gear S3	[41]	Dual-Core Cortex-A53 Mali™-T720 MP1 GPU
STM32F411VE	[69]	Arm Cortex-M4 32-bit RISC CPU

framework as an extension of TensorFlow to accelerate the execution of DL models on Android devices. They show that models with the multiplication of large matrices can run much faster with GPU support. *It would be interesting to evaluate further the GPU support for running DL algorithms on smartphones.*

Some studies use devices with embedded GPU or TPU platforms, such as Nvidia Jetson TK1 [17] and Nano [53]. These devices are not very different from a cloud setting [21]. However, they are used to compare the performance of smartphones with multicore CPUs. Some use FPGA accelerators [40], [71] utilizing hardware solutions to run DL algorithms efficiently. Similarly, Jiang et al. [40] designed and implemented a wearable DL system for gesture recognition by FPGA offloading. In contrast, some studies utilize typical IoT devices like Raspberry Pi versions [36], [40], [48], [56], [68], [74], Intel Edison [27], [34], and even Arduino Nano [58]. While the use of smartwatches is rare, we see example studies where Sony Smartwatch 2 and LG G Watch R are utilized [41], [54]. In MDLdroidLite [41], one of the most extensive and recent studies, the Samsung Gear S3 smartwatch is also used besides three different smartphone platforms. In another study [54], a lightweight DL framework, L-IONet (based on LSTM, RNN), is proposed to learn and reconstruct pedestrian trajectories from IMU data. Its performance is tested on different smartphones and a smartwatch (Sony Smartwatch 2) in terms of execution time and accuracy. It is shown that L-IONet achieves a real-time inference even on the smartwatch with minimal computational resources. DeepEmote [49] presents a bracelet with integrated sensors, targeting low-power design and energy efficiency on an ARM Cortex M4F microcontroller that can run multilayer NN algorithms. The authors report that with a network of 26 neurons, 100% accuracy for emotion detection is achieved using only 2% of memory. The battery lifetime is two months while performing detection every 10 min with a 600-mAh battery.

The studies in which different devices/platforms are utilized provide benchmarking results. For example, Chauhan et al. [44] focused on breathing-acoustics-based CA

TABLE V
DL ALGORITHMS USED IN RELEVANT STUDIES

Deep Learning Algorithms	Studies
DNN	[17], [20], [30], [33], [38], [49], [51], [57] [19], [55], [61], [66], [71]–[73]
CNN	[27]–[30], [32], [33], [38]–[40], [48], [53], [68] [56], [59], [62], [69], [72], [74], [75], [77]
LSTM	[20], [36]–[39], [45], [54], [56] [10], [11], [46], [52], [61], [72], [74]
CNN-LSTM	[11], [31], [39]
RNN-CNN	[34], [35]
Inception network	[28]
CONVLSTM	[28], [39]
MobileNet	[41]
LeNet	[19], [41], [43]
VGG	[19], [41], [43]
MobileNetv2	[21], [43]
Autoencoder	[58], [71]
ELM (Extreme) Learning Machine	[71]

using shallow models (SVM) and LSTM on a previously collected dataset consisting of acoustic samples of deep breathing, regular breathing, and sniffing from ten users. They performed experiments on four devices: two smartphones (Nexus 5 and Pixel), a smartwatch (LG G Watch R), and a Raspberry Pi 3 after compressing the selected RNN models using the 256-level quantization function provided in TensorFlow. They report that model loading time decreases linearly with the device's RAM, and inference time depends on the device's processing power. Although this article uses microphone data, since the application is not audio classification (speech or sound recognition), we include it in this review. It is a good reference showing the performance on different devices, including mobile, wearable, and IoT devices. In [76], a group of researchers from the TinyMLPerf working group that is comprised of over 30 organizations, presented state-of-the-art on TinyML[1] and discussed the challenges and requirements for developing a hardware benchmark for TinyML workloads, which is an important initiative. TinyML is a foundation for tiny machine learning which broadly targets machine-learning technologies and applications on extremely low-power, battery-operated devices. Various devices with different capabilities have been tested in the reviewed papers, ranging from GPU/TPU to smartwatches and IoT devices. This review targets studies tested with battery-powered smartphones and wearable devices mainly operating on CPUs. Although there are initiatives for benchmarking on different platforms, their number is still limited. *Benchmarking studies that implement the optimization methods discussed in Section III are essential for testing their resource efficiency on different hardware.*

C. DL Architectures

Table V categorizes the related studies according to the DL architectures utilized. The most popular architectures include DNNs, CNNs, and RNNs. These are the base models. However, we also see networks [37], [74], designed explicitly for the target devices or tasks utilizing these base architectures. In [68], a real-time HAR system is presented with the HARNet

[1]https://www.tinyml.org/

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on December 11,2024 at 06:49:30 UTC from IEEE Xplore. Restrictions apply.

TABLE VI
MODEL OPTIMIZATION METHODS USED

Model Optimization Toolkit or Method	Studies
TensorFlow Lite	[30], [38], [39], [58] [32], [51], [54], [57] [11], [55], [61]
TensorFlow for Mobile	[35], [72]
Optimized Fast ANN Library	[49]
DL4J	[10], [21], [43]
Data compression	[17]
Data transformation	[27], [57], [67]
Pruning	[11], [19], [21], [41], [51], [53]
Quantization	[19], [30], [38], [39], [57]
Hand-optimized	[33]
Network design strategy	[19], [33], [34], [36], [37] [59], [62], [68], [71], [72]
Modifying optimization routines	[40], [61], [66], [72]
NAS	[41]
GPU/FPGA Offloading	[37], [40], [46]
Feature transfer (training)	[21], [43], [72]
Layer freezing (training)	[11], [56]
Incremental learning (training)	[10], [20], [48], [58], [68], [74]

architecture [74] with extensive parametric optimizations and modify it as a Bayesian NN using uncertainty estimates. Cao et al. [37] trained a stacked LSTM model on a server using a previously collected dataset with TensorFlow.

There are studies proposing lightweight and efficient DL networks or focusing on resource characterization for the target resource-constrained devices [78], [79], [80], [81], [82]. However, they focus on image or speech recognition tasks. *The proposed methods can be adapted to sensing applications.*

D. Model Optimization Methods/Toolkits

In Table VI, we categorize the studies according to the model optimization/transformation techniques or DL toolkits used. The most popular toolkit used in the literature is TF Lite.[2] It offers posttraining quantization techniques to simplify DL models. For example, in [38] and [39], the trained model is transformed with TF Lite and ported to a smartphone (Samsung A5) as part of a real-time HAR app. The impact of quantizing the weights and activation functions is also explored, and it is shown that using fixed-point 8 bits instead of 32 bits reduces the memory requirements four times. Eclipse Deeplearning4j (DL4J)[3] is a tool for running DL algorithms on the Java virtual machine. It is mainly used with smartphones and also used in the reviewed papers.

As discussed in Section III-A, data compression is one of the optimization techniques for transforming DL models. Ravi et al. [27] used a data transformation technique. Specifically, spectral representations of different axes and sensors are pre-arranged and they can be processed using 1-D convolutional kernels. On the other hand, REST [51] enables NN compression through sparsity regularization where structured (channel) pruning is used. They assign a measure of importance to each filter of a CNN and prune the least important ones to achieve sparsity. Dey and Roy [53] utilized a magnitude-based weight pruning technique on CNN and multilayer perceptron (MLP) algorithms for on-device inference on Nvidia Jetson Nano. They reported a significant reduction in model size (70%) and inference time (31%). As seen in Table VI, pruning is commonly used in the relevant studies.

In [33], a shared runtime is implemented for inference using a mix of modules from the Torch supported by a set of custom C/C++ components. The authors name this as hand-optimized. Network design strategies are also commonly used in the relevant studies, as presented in Table VI. For example, DeepSense [34] integrates CNNs and RNNs to address the challenges of noisy sensor measurements and finding effective features. To simplify the network, the authors do not use residual net structures, replace the two-layer stacked gated recurrent unit (GRU) with a single-layer GRU with a larger dimension, and concatenate the input sensors. In [41], model predictive control is used as a modification of the optimization routine while training the network. Besides loss reduction, structure growth is also optimized.

Incremental learning [23] is a common method used in the relevant studies for the model update, as seen in Table VI. Tsukada et al. [71] focused on concept drift, where the distribution may change over time so that the models on the device may require an update. To enable fast training and reduce the computational cost on edge devices, ONLAD combines OS-ELM (online sequential extreme learning machine) and an autoencoder, an NN-based dimensionality reduction model. ONLAD is implemented on the PYNQ-Z1 board, a small SoC platform. The results show that the training latency is $1.95 \times 6.58\times$ faster than the other software implementations. ContAuth [20] proposes a system that adapts to new incoming data and adds new users using class incremental learning. The decision module checks for the labels of the class and decides if the framework should perform incremental learning or learn a new model (with elastic weight consolidation (EWC) and incremental classifier and representation learning (iCaRL) incremental learning algorithms). Layer freezing and feature transfer methods are also utilized in example studies, as seen in Table VI. In [11], fine-tuning is performed with a deep RNN to minimize the need for training from scratch on mobile devices. They also use magnitude base weight pruning and compression for minimizing the complexity.

Although KD [14] and NAS [16] methods are commonly used in image-processing studies, they have not been implemented in the reviewed studies, and researchers can further investigate them for sensing applications.

E. Implementation, Validation, and Resource Usage Analysis

In Table VII, we provide the implementation details of the related studies, including mode of implementation, such as offline or end-to-end (see Fig. 1), whether the implementation is validated on a device, whether the system supports model update, and finally, whether the resource consumption (e.g., CPU, battery, and memory) is analyzed or not.

The studies focusing on the end-to-end mode, where both model training and inference are performed on the devices are few. TinyDL [17] is an end-to-end example framework that uses a resource-aware signal conversion approach to map data to a collection of low-dimensional subspaces. However, the authors implemented tests on NVIDIA Jetson TK1 which

[2]https://www.tensorflow.org/lite/
[3]https://deeplearning4j.konduit.ai/

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on December 11,2024 at 06:49:30 UTC from IEEE Xplore. Restrictions apply.

TABLE VII
IMPLEMENTATION MODE, VALIDATION, MODEL UPDATE, AND RESOURCE CONSUMPTION ANALYSIS

Mode of Implementation	**End-to-End**	**Offline**
	[17], [41], [71]	[20], [21], [27], [28], [30], [33]–[35], [40], [57], [58] [29], [36]–[39], [43], [45], [51], [53], [54] [32], [48], [49], [68] [10], [11], [19], [42], [46], [55], [59], [61], [62], [67], [72], [73]
Validation on Device	**Yes**	**No**
	[10], [11], [17], [20], [21], [27], [28], [30], [33], [40], [58] [29], [32], [34]–[38], [41], [43], [45], [46], [53], [54], [57] [19], [42], [48], [49], [51], [55], [56], [59], [61], [62], [67], [71]–[74]	[26], [31], [39]
Model Update	**Yes**	**No**
	[17], [20], [21], [41], [43], [48], [56], [58], [68] [10], [11], [71], [74]	[27]–[29], [31]–[35], [37]–[40], [45], [46], [51], [53]–[55], [57] [19], [42], [49], [59], [61], [62], [66], [67], [72], [73]
Resource Consumption Analysis	**Yes**	**No**
	[10], [17], [20], [30], [33]–[35], [39], [58] [38], [40], [41], [43], [45], [51], [53], [57], [66], [71], [73]	[19], [27]–[29], [31], [32], [36], [37], [49], [54], [67], [68] [11], [46], [48], [55], [56], [59], [61], [62], [72]

is not a typical mobile device. Most of the studies validated the performance of the models on target devices but in some studies, the models were simplified but not tested on target devices, as seen in Table VII.

In most of the studies, the model update is not supported, however, the number of studies supporting model update is not few. As discussed in Section IV-D, incremental learning is a popular approach. For example, TinyOL [58] targets anomaly detection on accelerometer data and can learn from new data and update the layers' weights, besides inference on the device. TinyOL can train a layer or can change the layer structure to include new classes by using incremental learning. The system's performance is tested on an Arduino Nano 33 BLE Sense board. They show the feasibility of incremental learning on a resource-constrained device.

In [68], a real-time HAR system is presented. They use the HARNet architecture [74] with extensive parametric optimizations and modify it as a Bayesian NN using uncertainty estimates. They simulate the incremental model update on a Raspberry Pi 2 device and measure the computation time, model size, and accuracy. Model update time is reported as 14 s. Ragav and Gudur [48] explored the model update again using incremental learning but by querying the most uncertain data points from the oracle with active learning. It is deployed on Raspberry Pi 2, where model weights are updated on incoming data and it is shown that 92% accuracy can be achieved using active learning.

Another study that supports on-device model update is [21]. The authors use feature transfer and show that the models can be efficiently trained on different smartphone models (Nexus 6/6P, Huawei Mate 10, and Google Pixel2), where training takes less than 10 min and can be shortened 3–5 times with feature transfer. Wang et al. [43] focused on implementing the training phase of DL algorithms on mobile devices. As a case study, they focus on CA and reformulate classification as a deep metric learning problem in which a distance metric between similar and dissimilar samples is learned. To speed up training, they also explore the use of feature transfer. They test the performance on three datasets using the Huawei Mate10 smartphone. Kolcun et al. [56] explored how to update a pretrained model through retraining or partial retraining on an edge device, namely Raspberry Pi Model 4. They investigate how freezing a different number of layers affects the training time and accuracy of the models and show that it is feasible to retrain all models, and training time depends on the NN's type and architecture. Moreover, layer freezing decreases the training time to half for LSTM and Conv1D models.

In most of the studies, the model update is not deeply explored. However, this can be required by changing user behaviors and sensor data. This is one of the research directions that can be further explored.

F. Performance Metrics

DL algorithms' performances are usually analyzed in terms of model accuracy. Accuracy, precision, recall, and F1 scores are utilized almost in all studies targeting classification problems, whereas error rates are considered in regression or CA applications. For on-device implementation, these are still important metrics for recognition. However, metrics related to resource usage become essential on the target devices. In Table VIII, we present the performance metrics considered in the related studies. Model and parameter sizes and memory usage are important parameters for measuring a DL model's performance. Inference time/latency is an essential parameter for those applications, such as HAR, which require working with minimum latency for inference in reality. The studies supporting model updates also considered training time a critical metric. However, since devices have different computation speeds, FLOPs might be considered. *As mentioned, resource constraints mainly characterize the devices, and the most critical resource is power. Hence, energy, power, and battery lifetime measurements are essential.*

There are also studies focusing on resource characterization and management. For example, [82] is the first study (in 2015) on the performance characterization of inference with common DL models (such as CNNs and DNNs) on example mobile and embedded platforms. However, they utilize image/audio data. One of the studies focusing on energy management is the DeLight [66] framework. It automates the training and execution of DNNs based on energy sources and application data by proposing an automated customization methodology to adapt the DNN configurations to the underlying hardware characteristics. In [83], the NeuralPower framework that predicts the power, runtime, and energy of CNNs without actually executing the models on a target platform is presented. The framework models the power and runtime of the key layers that are the convolutional, the fully connected, and the pooling layers commonly used in a CNN, using a polynomial regression model. In [72], the FastDeepIoT framework is introduced to

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on December 11,2024 at 06:49:30 UTC from IEEE Xplore. Restrictions apply.

TABLE VIII
PERFORMANCE METRICS

Performance Metrics	Studies
Accuracy/ F1-score/ precision/recall	[19]–[21], [27], [58] [28], [31]–[34], [36], [38]–[42], [55], [56] [45], [48], [51], [53], [57] [10], [11], [59], [61], [66]–[68], [71], [72], [74]
Error rate	[17], [43], [54], [58], [62]
Loss	[11], [21], [43]
Inference Time	[17], [19]–[21], [58] [27], [28], [30], [31], [33], [34], [56] [37]–[42] [45], [46], [54], [68] [10], [11], [48], [49], [51], [53], [61], [66], [69], [71]–[74]
Training Time	[17], [20], [21], [58] [10], [11], [41], [56], [66], [71]
Model Size	[20], [33], [38], [39] [17], [32], [41], [51], [53], [68] [48], [56]
Memory	[21], [30], [38], [39] [10], [19], [41]–[43], [45], [49], [57], [66], [73]
Parameter Size	[31], [32], [54], [74]
Battery Lifetime	[41]
Energy, Power	[17], [21], [30], [33] [34], [38]–[40] [19], [41], [43], [45], [51], [57] [49], [61], [67], [71], [73]
CPU throughput/load	[21], [33], [39], [40] [43], [51]

explore the relationship between NN structure and execution time. The framework's profiling module uses a tree-structured linear regression model for predicting the execution time of a model using the input network structure. The compression module compresses the NN to minimize execution time by using profiling results.

V. DISCUSSION AND FUTURE RESEARCH DIRECTIONS

One of the issues about on-device learning is that there is often *small amount of training data*, particularly labeled data. Training DNNs with small datasets may lead to *overfitting*. Here, one solution could be using transfer learning [22] where a model trained with a larger dataset can be used and fine-tuned on the device [11], [26]. Another emerging technique is to use FL to deal with limited training data, as mentioned. Although FL has promising application potential in various areas, such as healthcare, it is still discussed at the conceptual level. We believe that one of the areas where it can bring advantages is wearable computing. Although there can be a small amount of training data (collecting labeled data is also a challenge), there is usually a large set of unlabeled data [3]. Here, few-shot learning, semisupervised, and transfer learning can be used, and the models can be improved later or unsupervised compression can be considered [3].

One of the other issues related to sensor data is that it can be *noisier* than well-controlled data in the laboratory. Hence, data preprocessing, such as filtering, may be required. Another issue is the *concept drift* or covariate shift. Concept drift happens when the distribution of the newly collected data differs from the original training data. This may happen with sensor data as users' behaviors change in time or due to noisy sensor readings [35]. For this purpose, continual learning or incremental learning can be used as discussed in Section III-B. However, there are few studies [58] considering the concept drift problem. This is one of the research directions that should be explored in detail for on-device learning.

As mentioned before, in sensing applications, the sensor data is mainly *multimodal*, including different types of sensors and this is different from dealing with audio, text, or video data. As mentioned in [34], it is crucial to efficiently merge multimodal inputs and use architectures that can deal with such data and model the temporal relationships among sensors.

As we discussed in Section III-B, in most of the studies, a model is trained on an external device and optimized before transporting the model to the mobile device, or an externally trained model is updated on the device with new user data, particularly for personalizing the models [11]. Although there are recent efforts on *mobile model training*, especially for image recognition, the hardware (*hardware limitation*) should be improved to train the models on-device. However, we can expect research efforts to continue in this area as we move forward. As mentioned in [3], new hardware technologies, such as hybrid memory cube and high bandwidth memory, are available on desktops, but not on mobile devices. However, they are expected to be used on mobile devices. Mainly, energy-aware memory access is essential since it is very energy-demanding.

Related to the hardware challenge and training, another challenge is the lack of studies in *energy estimation* since most of the techniques focus on the inference phase, and postprocessing techniques are proposed to reduce the energy consumption of a network. Studies at the training stage are few. There are studies on the current state of energy predictions in machine learning through energy forecasting modeling or by directly integrating power monitoring tools into existing machine-learning packages. DL models are usually trained on desktop GPUs. Energy estimation models involving mobile/embedded GPUs are needed. Estimating energy consumption is difficult due to the lack of suitable tools to measure power models in current machine-learning packages. The network compression and acceleration techniques discussed in Section III-A and fine-tuning the models on the device discussed in Section III-B include the optimization of many hyperparameters, which requires many experiments and expert knowledge. As mentioned in Section III-A, it is important to have *adaptive/automatic compression* and *acceleration* methods [3].

VI. CONCLUSION

Considering the advantages of DL, applying them to widely used mobile and wearable devices is a trend. However, this is also challenging due to the resource constraints of such devices. This article reviewed the state-of-the-art studies focusing on mobile DL, particularly those training and/or running the algorithms on the device and not using an external device. Unlike the recent surveys on the topic, we mainly focused on sensing applications. We presented the general techniques to optimize DL algorithms to meet the resource limitations of the devices and performance evaluations. We discussed updating and personalizing models with new data on such personal devices. We reviewed various sensing application areas which benefit from on-device DL and classified the studies according to several aspects, including application areas, sensors, types of devices, utilized DL algorithms, mode of implementation, methods for optimizing DL algorithms for the target devices, training method, implementation toolkit/platform, performance metrics, and resource consumption analysis. This is a relatively new field of research. Accordingly, we listed some open issues and directions for future research.

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on December 11,2024 at 06:49:30 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, "A first look at deep learning apps on smartphones," in *Proc. World Wide Web Conf.*, May 2019, pp. 2125–2136.

[2] C. Chen et al., "Deep learning on computational-resource-limited platforms: A survey," *Mobile Inf. Syst.*, vol. 2020, pp. 1–19, Mar. 2020.

[3] Y. Chen, B. Zheng, Z. Zhang, Q. Wang, C. Shen, and Q. Zhang, "Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions," *ACM Comput. Surveys*, vol. 53, no. 4, pp. 1–37, Jul. 2021.

[4] Q. Zhou et al., "On-device learning systems for edge intelligence: A software and hardware synergy perspective," *IEEE Internet Things J.*, vol. 8, no. 15, pp. 11916–11934, Aug. 2021.

[5] M. Verhelst and B. Moons, "Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to IoT and edge devices," *IEEE Solid-State Circuits Mag.*, vol. 9, no. 4, pp. 55–65, Nov. 2017.

[6] E. Wang et al., "Deep neural network approximation for custom hardware: Where we've been, where we're going," *ACM Comput. Surv.*, vol. 52, no. 2, pp. 1–39, 2019.

[7] S. Dhar, J. Guo, J. Liu, S. Tripathi, U. Kurup, and M. Shah, "A survey of on-device machine learning: An algorithms and learning theory perspective," *ACM Trans. Internet Things*, vol. 2, no. 3, pp. 1–49, 2021.

[8] T. Zhao et al., "A survey of deep learning on mobile devices: Applications, optimizations, challenges, and research opportunities," *Proc. IEEE*, vol. 110, no. 3, pp. 334–354, Mar. 2022.

[9] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[10] Y. D. Kwon, J. Chauhan, A. Kumar, P. H. Hkust, and C. Mascolo, "Exploring system performance of continual learning for mobile and embedded sensing applications," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Dec. 2021, pp. 319–332.

[11] N. Mairittha, T. Mairittha, and S. Inoue, "On-device deep personalization for robust activity data collection," *Sensors*, vol. 21, no. 1, p. 41, Dec. 2020.

[12] H. Yang, Y. Zhu, and J. Liu, "End-to-end learning of energy-constrained deep neural networks," 2018, *arXiv:1806.04321*.

[13] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 6655–6659.

[14] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, pp. 1789–1819, Mar. 2021.

[15] D. Stamoulis et al., "Designing adaptive neural networks for energy-constrained image classification," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2018, pp. 1–8.

[16] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "Hardware-aware neural architecture search: Survey and taxonomy," in *Proc. 30th Int. Joint Conf. Artif. Intell. (IJCAI)*, Z.-H. Zhou, Ed., Aug. 2021, pp. 4322–4329, doi: 10.24963/ijcai.2021/592.

[17] B. D. Rouhani, A. Mirhoseini, and F. Koushanfar, "TinyDL: Just-in-time deep learning solution for constrained embedded systems," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[18] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[19] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proc. 16th Annu. Int. Conf. Mobile Syst., Appl., Services*, Jun. 2018, pp. 389–400.

[20] J. Chauhan, Y. D. Kwon, P. Hui, and C. Mascolo, "ContAuth: Continual learning framework for behavioral-based user authentication," *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 4, no. 4, pp. 1–23, Dec. 2020.

[21] C. Wang, Y. Xiao, X. Gao, L. Li, and J. Wang, "Close the gap between deep learning and mobile intelligence by incorporating training in the loop," in *Proc. 27th ACM Int. Conf. Multimedia*, Oct. 2019, pp. 1419–1427.

[22] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Jan. 2010.

[23] Z. Chen and B. Liu, "Lifelong machine learning," *Synth. Lectures Artif. Intell. Mach. Learn.*, vol. 12, no. 3, pp. 1–207, 2018.

[24] P. Pu Liang et al., "Think locally, act globally: Federated learning with local and global representations," 2020, *arXiv:2001.01523*.

[25] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," 2018, *arXiv:1812.00564*.

[26] D. Buffelli and F. Vandin, "Attention-based deep learning framework for human activity recognition with user adaptation," *IEEE Sensors J.*, vol. 21, no. 12, pp. 13474–13483, Jun. 2021.

[27] D. Ravì, C. Wong, B. Lo, and G.-Z. Yang, "A deep learning approach to on-node sensor data analytics for mobile or wearable devices," *IEEE J. Biomed. Health Inform.*, vol. 21, no. 1, pp. 56–64, Jan. 2017.

[28] C. Xu, D. Chai, J. He, X. Zhang, and S. Duan, "InnoHAR: A deep neural network for complex human activity recognition," *IEEE Access*, vol. 7, pp. 9893–9902, 2019.

[29] I. Andrey, "Real-time human activity recognition from accelerometer data using convolutional neural networks," *Appl. Soft Comput.*, vol. 62, pp. 915–922, Jan. 2018.

[30] M. Antonini, T. H. Vu, C. Min, A. Montanari, A. Mathur, and F. Kawsar, "Resource characterisation of personal-scale sensing models on edge accelerators," in *Proc. 1st Int. Workshop Challenges Artif. Intell. Mach. Learn. Internet Things*, Nov. 2019, pp. 49–55.

[31] C. F. S. Leite and Y. Xiao, "Improving resource efficiency of deep activity recognition via redundancy reduction," in *Proc. 21st Int. Workshop Mobile Comput. Syst. Appl.*, Mar. 2020, pp. 33–38.

[32] K. Peppas, A. C. Tsolakis, S. Krinidis, and D. Tzovaras, "Real-time physical activity recognition on smart mobile devices using convolutional neural networks," *Appl. Sci.*, vol. 10, no. 23, p. 8482, Nov. 2020.

[33] V. Radu et al., "Multimodal deep learning for activity and context recognition," *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 1, no. 4, pp. 1–27, Jan. 2018.

[34] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "DeepSense: A unified deep learning framework for time-series mobile sensing data processing," in *Proc. 26th Int. Conf. World Wide Web*, Apr. 2017, pp. 351–360.

[35] S. Yao, Y. Zhao, S. Hu, and T. Abdelzaher, "QualityDeepSense: Quality-aware deep learning framework for Internet of Things applications with sensor-temporal attention," in *Proc. 2nd Int. Workshop Embedded Mobile Deep Learn.*, Jun. 2018, pp. 42–47.

[36] P. Agarwal and M. Alam, "A lightweight deep learning model for human activity recognition on edge devices," *Proc. Comput. Sci.*, vol. 167, pp. 2364–2373, 2020.

[37] Q. Cao, N. Balasubramanian, and A. Balasubramanian, "MobiRNN: Efficient recurrent neural network execution on mobile GPU," in *Proc. 1st Int. Workshop Deep Learn. Mobile Syst. Appl.*, Jun. 2017, pp. 1–6.

[38] T. Zebin, P. J. Scully, N. Peek, A. J. Casson, and K. B. Ozanyan, "Design and implementation of a convolutional neural network on an edge computing smartphone for human activity recognition," *IEEE Access*, vol. 7, pp. 133509–133520, 2019.

[39] S. O. Bursa, O. D. Incel, and G. I. Alptekin, "Transforming deep learning models for resource-efficient activity recognition on mobile devices," in *Proc. 5th Conf. Cloud Internet Things (CIoT)*, Mar. 2022, pp. 83–89.

[40] W. Jiang et al., "Wearable on-device deep learning system for hand gesture recognition based on FPGA accelerator," *Math. Biosci. Eng.*, vol. 18, no. 1, pp. 132–153, 2021.

[41] Y. Zhang, T. Gu, and X. Zhang, "MDLdroidLite: A release-and-inhibit control approach to resource-efficient deep neural networks on mobile devices," *IEEE Trans. Mobile Comput.*, vol. 21, no. 10, pp. 3670–3686, Oct. 2022.

[42] A. R. M. Forkan et al., "MobDL: A framework for profiling deep learning models: A case study using mobile digital health applications," in *Proc. 17th EAI Int. Conf. Mobile Ubiquitous Syst., Comput., Netw. Services*, Dec. 2020, pp. 405–414.

[43] C. Wang, Y. Xiao, X. Gao, L. Li, and J. Wang, "A framework for behavioral biometric authentication using deep metric learning on mobile devices," *IEEE Trans. Mobile Comput.*, vol. 22, no. 1, pp. 19–36, Jan. 2023.

[44] J. Chauhan, S. Seneviratne, Y. Hu, A. Misra, A. Seneviratne, and Y. Lee, "Breathing-based authentication on resource-constrained IoT devices using recurrent neural networks," *Computer*, vol. 51, no. 5, pp. 60–67, May 2018.

[45] Y. Liu, Z. Li, Z. Liu, and K. Wu, "Real-time arm skeleton tracking and gesture inference tolerant to missing wearable sensors," in *Proc. 17th Annu. Int. Conf. Mobile Syst., Appl., Services*, Jun. 2019, pp. 287–299.

[46] M. Alzantot, Y. Wang, Z. Ren, and M. B. Srivastava, "RSTensorFlow: GPU enabled TensorFlow for deep learning on commodity Android devices," in *Proc. 1st Int. Workshop Deep Learn. Mobile Syst. Appl.*, Jun. 2017, pp. 7–12.

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on December 11,2024 at 06:49:30 UTC from IEEE Xplore. Restrictions apply.

[47] A. Vitale, E. Donati, R. Germann, and M. Magno, "Neuromorphic edge computing for biomedical applications: Gesture classification using EMG signals," *IEEE Sensors J.*, vol. 22, no. 20, pp. 19490–19499, Oct. 2022.

[48] A. Ragav and G. K. Gudur, "Bayesian active learning for wearable stress and affect detection," 2020, *arXiv:2012.02702*.

[49] M. Magno, M. Pritz, P. Mayer, and L. Benini, "DeepEmote: Towards multi-layer neural networks in a low power wearable multi-sensors bracelet," in *Proc. 7th IEEE Int. Workshop Adv. Sensors Interfaces (IWASI)*, Jun. 2017, pp. 32–37.

[50] E. Lee and C.-Y. Lee, "PPG-based smart wearable device with energy-efficient computing for mobile health-care applications," *IEEE Sensors J.*, vol. 21, no. 12, pp. 13564–13573, Jun. 2021.

[51] R. Duggal, S. Freitas, C. Xiao, D. H. Chau, and J. Sun, "REST: Robust and efficient neural networks for sleep monitoring in the wild," in *Proc. Web Conf.*, 2020, pp. 1704–1714.

[52] J. Lai, Z. Yang, and B. Guo, "A two-stage low-complexity human sleep motion classification method using IR-UWB," *IEEE Sensors J.*, vol. 21, no. 18, pp. 20740–20749, Sep. 2021.

[53] E. Dey and N. Roy, "OMAD: On-device mental anomaly detection for substance and non-substance users," in *Proc. IEEE 20th Int. Conf. Bioinf. Bioeng. (BIBE)*, Oct. 2020, pp. 466–471.

[54] C. Chen, P. Zhao, C. X. Lu, W. Wang, A. Markham, and N. Trigoni, "Deep-learning-based pedestrian inertial navigation: Methods, data set, and on-device inference," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4431–4441, May 2020.

[55] S. M. Hernandez and E. Bulut, "Lightweight and standalone IoT based WiFi sensing for active repositioning and mobility," in *Proc. IEEE 21st Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Aug. 2020, pp. 277–286.

[56] R. Kolcun et al., "The case for retraining of ML models for IoT device identification at the edge," 2020, *arXiv:2011.08605*.

[57] S. Hosseininoorbin, S. Layeghy, B. Kusy, R. Jurdak, and M. Portmann, "Scaling spectrogram data representation for deep learning on edge TPU," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops Affiliated Events (PerCom Workshops)*, Mar. 2021, pp. 572–578.

[58] H. Ren, D. Anicic, and T. Runkler, "TinyOL: TinyML with online-learning on microcontrollers," 2021, *arXiv:2103.08295*.

[59] W. Shao, H. Luo, F. Zhao, C. Wang, A. Crivello, and M. Z. Tunio, "DePedo: Anti periodic negative-step movement pedometer with deep convolutional neural networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[60] A. S. Alharthi, S. U. Yunas, and K. B. Ozanyan, "Deep learning for monitoring of human gait: A review," *IEEE Sensors J.*, vol. 19, no. 21, pp. 9575–9591, Nov. 2019.

[61] R. Mishra, H. P. Gupta, and T. Dutta, "A road health monitoring system using sensors in optimal deep neural network," *IEEE Sensors J.*, vol. 21, no. 14, pp. 15527–15534, Jul. 2021.

[62] A. Shrestha and M. Won, "DeepWalking: Enabling smartphone-based walking speed estimation using deep learning," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.

[63] H. F. Nweke, Y. W. Teh, M. A. Al-Garadi, and U. R. Alo, "Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges," *Exp. Syst. Appl.*, vol. 105, pp. 233–261, Sep. 2018.

[64] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognit. Lett.*, vol. 119, pp. 3–11, Mar. 2019.

[65] E. Ramanujam, T. Perumal, and S. Padmavathi, "Human activity recognition with smartphone and wearable sensors using deep learning techniques: A review," *IEEE Sensors J.*, vol. 21, no. 12, pp. 13029–13040, Mar. 2021.

[66] B. D. Rouhani, A. Mirhoseini, and F. Koushanfar, "DeLight: Adding energy dimension to deep neural networks," in *Proc. Int. Symp. Low Power Electron. Design*, New York, NY, USA, Aug. 2016, pp. 112–117.

[67] A. Machidon and V. Pejovic, "Enabling resource-efficient edge intelligence with compressive sensing-based deep learning," in *Proc. 19th ACM Int. Conf. Comput. Frontiers*, May 2022, pp. 141–149.

[68] G. K. Gudur, P. Sundaramoorthy, and V. Umaashankar, "ActiveHARNet: Towards on-device deep Bayesian active learning for human activity recognition," in *Proc. 3rd Int. Workshop Deep Learn. Mobile Syst. Appl.*, Jun. 2019, pp. 7–12.

[69] Y. L. Coelho, F. D. A. S. D. Santos, A. Frizera-Neto, and T. F. Bastos-Filho, "A lightweight framework for human activity recognition on wearable devices," *IEEE Sensors J.*, vol. 21, no. 21, pp. 24471–24481, Nov. 2021.

[70] M. Z. Uddin, "A wearable sensor-based activity prediction system to facilitate edge computing in smart healthcare system," *J. Parallel Distrib. Comput.*, vol. 123, pp. 46–53, Jan. 2019.

[71] M. Tsukada, M. Kondo, and H. Matsutani, "A neural network-based on-device learning anomaly detector for edge devices," *IEEE Trans. Comput.*, vol. 69, no. 7, pp. 1027–1044, Jul. 2020.

[72] S. Yao et al., "FastDeepIoT: Towards understanding and optimizing neural network execution time on mobile and embedded devices," in *Proc. 16th ACM Conf. Embedded Networked Sensor Syst.*, Nov. 2018, pp. 278–291.

[73] A. Mathur et al., "Using deep data augmentation training to address software and hardware heterogeneities in wearable and smartphone sensing devices," in *Proc. 17th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2018, pp. 200–211.

[74] P. Sundaramoorthy, G. K. Gudur, M. R. Moorthy, R. N. Bhandari, and V. Vijayaraghavan, "HARNet: Towards on-device incremental learning using deep ensembles on constrained devices," in *Proc. 2nd Int. Workshop Embedded Mobile Deep Learn.*, Jun. 2018, pp. 31–36.

[75] N. Phukan, S. Mohine, A. Mondal, M. S. Manikandan, and R. B. Pachori, "Convolutional neural network-based human activity recognition for edge fitness and context-aware health monitoring devices," *IEEE Sensors J.*, vol. 22, no. 22, pp. 21816–21826, Nov. 2022.

[76] C. R. Banbury et al., "Benchmarking TinyML systems: Challenges and direction," 2020, *arXiv:2003.04821*.

[77] A. L. Machidon and V. Pejovic, "Deep learning techniques for compressive sensing-based reconstruction and inference—A ubiquitous systems perspective," 2021, *arXiv:2105.13191*.

[78] H. Phan, D. Huynh, Y. He, M. Savvides, and Z. Shen, "MoBiNet: A mobile binary network for image classification," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2020, pp. 3453–3462.

[79] S. Bhattacharya and N. D. Lane, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," in *Proc. 14th ACM Conf. Embedded Netw. Sensor Syst.*, Nov. 2016, pp. 176–189.

[80] S. Yang, Z. Gong, K. Ye, Y. Wei, Z. Huang, and Z. Huang, "EdgeRNN: A compact speech recognition network with spatio-temporal features for edge computing," *IEEE Access*, vol. 8, pp. 81468–81478, 2020.

[81] Y. Wei, Z. Gong, S. Yang, K. Ye, and Y. Wen, "EdgeCRNN: An edge-computing oriented model of acoustic feature enhancement for keyword spotting," *J. Ambient Intell. Humanized Comput.*, vol. 13, no. 3, pp. 1525–1535, Mar. 2022.

[82] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and Internet-of-Things devices," in *Proc. Int. Workshop Internet Things Towards Appl.*, Nov. 2015, pp. 7–12.

[83] L. Yang and B. Murmann, "SRAM voltage scaling for energy-efficient convolutional neural networks," in *Proc. 18th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2017, pp. 622–637.

Ozlem Durmaz Incel received the Ph.D. degree in computer science from the University of Twente, Enschede, The Netherlands, in 2009.

She was a Faculty Member with Galatasaray University, Istanbul, Turkey, from 2012 to 2021. She is currently an Associate Professor with the Computer Engineering Department, Bogazici University, Istanbul. Her research interests include wearable computing, the Internet of Things, and applied machine learning.

Sevda Özge Bursa received the B.Sc. degree in mathematical engineering from Istanbul Technical University, Istanbul, Turkey, in 2016, and the M.Sc. degree in computer engineering from Galatasaray University, Istanbul, in 2022.

Her current research interests include human activity recognition, deep learning, and wearable devices.

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on December 11,2024 at 06:49:30 UTC from IEEE Xplore. Restrictions apply.