IEEE Open Journal of the Communications Society

Received 7 February 2024; accepted 27 February 2024. Date of publication 5 March 2024; date of current version 26 March 2024.

Digital Object Identifier 10.1109/OJCOMS.2024.3373177

TinyML Empowered Transfer Learning on the Edge

ALI M. HAYAJNEH [1], MARYAM HAFEEZ [2] (Member, IEEE), SYED ALI RAZA ZAIDI [2], AND DES MCLERNON [2]

[1]Department of Electrical Engineering, Faculty of Engineering, The Hashemite University, Zarqa 13133, Jordan

[2]School of Electronic and Electrical Engineering, University of Leeds, LS2 9JT Leeds, U.K.

CORRESPONDING AUTHOR: S. A. R. ZAIDI (e-mail: S.A.Zaidi@leeds.ac.uk)

This work was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) CHEDDAR under Grant EP/X040518/1; in part by the UK Research and Innovation (UKRI) under Grant EP/X039161/1; in part by MSCA Horizon EU under Grant 101086218; and in part by the Abdul Hameed Shoman Foundation under Grant 230800328.

ABSTRACT Tiny machine learning (TinyML) is a promising approach to enable intelligent applications relying on Human Activity Recognition (HAR) on resource-limited and low-power Internet of Things (IoT) edge devices. However, designing efficient TinyML models for these devices remains challenging due to computational resource constraints and the need for customisation to unique use cases. To address this, we propose a novel approach that utilises transfer learning (TL) techniques on edge micro-controller units (MCUs) to accelerate TinyML development. Our strategy involves pre-training generalised ML models on large-scale and varied datasets and fine-tuning them on-device for specific applications using TL. We demonstrate the effectiveness of our approach for HAR by experimenting with a convolutional neural network, long short-term memory TL (CNN-LSTM-TL) model architecture and visualisation techniques like t-distributed stochastic neighbour embedding (t-SNE) for dimensionality reduction. To further validate our model's proficiency and adaptability, we conducted extensive testing using two distinct datasets: MotionSense and UCI. This dual-dataset approach allowed us to assess the robustness of our model across different data domains, showcasing its versatility and effectiveness in various HAR scenarios. Our results show significant model accuracy and reduced training time while maintaining high inference rates and low MCU memory footprint. We also provide insights into best practices for implementing TL on edge MCUs and evaluate classification performance metrics such as accuracy, precision, recall, F1 score, and categorical cross-entropy loss. Our work lays a solid foundation for faster and more efficient TinyML deployments through TL framework across different application domains and types of edge IoT devices.

INDEX TERMS TinyML, transfer learning, machine learning, deep neural networks.

I. INTRODUCTION

HUman activity recognition (HAR) systems and techniques are essential for safety and efficiency in various modern day settings like public transport, supermarkets, sports, medical care and factories. HAR normally uses sensor data for automatically recognizing and classifying human actions [1]. They reduce risks to personal and property safety through anomaly detection as well as enhance real-time surveillance to reduce the need for manual analysis. Additionally, HAR aids behavior analysis and user experience, as well as providing guidance, tracking progress, and preventing failures.

Using machine learning (ML) approaches for HAR has become increasingly popular in recent years [2]. The complex pattern recognition capabilities inherent in ML algorithms can significantly enhance the performance of HAR. In HAR, these patterns correspond to the distinctive movements, behaviors, or signals associated with different human activities. ML utilizes data driven approaches leveraging integration of multimodal data from multiple sources, for example, sensors like accelerometers, gyroscopes, etc., providing a more comprehensive and accurate understanding of activities. The efficiency, scalability, and ability to adapt to user-specific behavior makes ML an increasingly powerful tool for HAR.

Existing HAR implementations using ML techniques have some drawbacks. Various sensors on smart devices, which are the primary source of HAR data, lack the intelligence

to extract valuable information from acquired data and learn from it. Typically, data collected by current sensors must be transmitted to the cloud for further ML processing, leading to increased energy consumption, communication latency, and potential security and privacy risks. Designing truly intelligent nodes for such tools presents challenges in size and lifespan, mainly due to limited batteries powering these smart microcontroller unit (MCU) powered sensor nodes. The limited battery life significantly limits the widespread adoption of smart sensor nodes for tool monitoring, impacting usability due to the difficulty, cost, and time involved in replacing batteries. Achieving a satisfactory lifespan necessitates meticulous low-power system design, including the careful selection of components to meet the small battery size requirements and ensure long operational periods lasting months or years.

In order to overcome the above challenges, edge processing for HAR has become increasingly popular as it bypasses the need for data to be relayed to the cloud [3]. This tactic not only enhances privacy and expedites response times but also alleviates network bandwidth pressures, introducing an additional degree of efficiency to IoT infrastructures. Moreover, to truly benefit from such edge-based solutions, it is important to devise learning algorithms that are energy and resource efficient. In this context, transfer learning (TL) proves to be an advantageous strategy for edge devices by facilitating the efficient and successful implementation of machine learning models [4]. It enhances performance, optimizes resource utilization, and effectively deals with the unique challenges inherent to edge architectures. This approach empowers edge devices to harness the advantages of deep learning while adhering to their specific limitations and needs.

With these developments, a crucial area of interest has formed around the requirement for reliable tools to implement ML models on edge IoT devices. To further optimise the edge-based HAR implementations, light-weight ML solutions are the key and are referred to as Tiny Machine Learning (TinyML) approaches [5], [6]. TinyML specifically addresses space and computational constraints in low-end smart devices and presents a framework of implementing ML solutions for operation on the edge. As a result, frameworks like TFLite-μ, PyTorch mobile, and Edge Impulse have emerged as important facilitators in this area [7], [8], [9]. TinyML Foundation recently appeared and is working to promote the development of ultra-low-power ML technology, that complements these tools [10]. Their cumulative use in the field of edge computing serves as an example of how ML models can be significantly more effective and useful in environments with limited resources. By enabling effective ML model execution on low-power micro-controller units (MCUs) and fostering the quick adaption of pre-trained models to address specific use cases, TFLite-μ in particular synergistically empowers this integration. This harmonious interplay significantly mitigates the complexity of deploying intricate ML models on edge devices, paving the way for a progressively intelligent IoT landscape.

The fusion of TinyML and TL presents itself as a promising solution [5], [6]. In fact, the joint implementation of TinyML and TL brings a robust approach that seamlessly balances model performance and energy efficiency. This opens up new ventures for the integration of the ML capabilities for edge IoT devices which as a result leads to smarter and more responsive IoT ecosystems.

The focal point of this study is an in-depth analysis and development of a TinyML-as-a-service (TinyMLaaS) framework. Our focus is centred around the ML development and operations (MLDevOps) workflow of TL for edge devices, where our objectives lie in expediting model deployment and enabling smooth integration within IoT systems. Also, we address the progress of TinyML development in conjunction with TL for edge MCUs. By harnessing TL techniques, our ambition is to facilitate a streamlined implementation of resource-demanding ML models, such as long short-term memory (LSTM) and convolutional neural networks (CNNs), on edge devices. This will trigger a rapid model deployment, thereby boosting the overall performance of IoT applications from multiple performance aspects. Finally, we aim to explore the effectiveness of TL in augmenting the precision of HAR models that are trained on limited labelled data. In conjunction with this, we aim to address how edge computing can be harnessed to improve the efficiency of HAR models deployed on resource-constrained devices. We believe that this research offers invaluable insights into the potential of TL and edge computing in amplifying the performance and practicality of HAR systems and other edge inference and classification tasks.

A. RELATED WORKS

Human activity recognition (HAR) has captivated research interest for more than a decade, owing to its applications across various domains such as healthcare, sports, and security. HAR encompasses the identification of human activities grounded in sensor data gathered from wearable gadgets, smartphones, or cameras [11], [12], [13], [14], [15].

Starting with the traditional transformation approaches, methods utilising Two-Dimensional Fast Fourier Transform (2-D FFT) and Wigner-Ville Transform (WVT) for time-frequency representations have been proposed [16]. These approaches utilise CNNs to achieve exceptional performance in HAR tasks. In the age of deep learning (DL), a significant shift has been observed. For example, an adaptive batch size-based CNN-LSTM model has been developed for efficiently handling imbalanced classes and non-normalised data was introduced in [17]. Another paper presents the CapsLSTM model, a framework which leverages spatiotemporal information to recognise multiple human activities [18]. This method displays resilience in data-scarce situations and offers more reliable performance even with a smaller fraction of training data.

Another work proposed a semi-supervised DL framework, which efficiently employs weakly labelled sensor data with a deep Q-network (DQN) based intelligent auto-labelling scheme and a multi-sensor data fusion mechanism in a reinforcement learning (RL) approach [19]. Taking a step further, a temporal-aware and modality-aware (TAMA) attention mechanism has been proposed in [20], which highlights the importance of varying temporal steps or modalities, increasing the interoperability of HAR tasks without any additional computational burdens.

A shift in the HAR tasks involves the use of wireless fidelity (WiFi) channel state information (CSI) by utilising the orthogonal frequency division multiple access (OFDMA) sub-carriers CSI. The approach in this domain is the deployment of a DL-based model called attention-based bi-directional LSTM (ABLSTM) [21]. This model, utilising CSI measurements, exhibits superior performance in HAR tasks. It assigns varying weights to learned features, leading to substantial improvements in recognition performance.

Further to this, the challenge of non-uniformly distributed unlabelled data has been addressed through the application of a generative adversarial network (GAN) integrated with a multi-modal generator [22]. This innovative approach augments the diversity of generated data, thereby enhancing the recognition of specific activities across diverse environmental settings. Thus, the application of CSI in HAR signifies a promising direction in the field, showing the potential for future advancements.

Finally, the importance of survey-oriented studies in this field is well recognised. Comprehensive reviews have been conducted, covering sensor-based activity recognition, challenges, future research directions, and multi-sensor applications [11], [12], [13].

Hence, the domain of HAR has seen a remarkable progression from traditional transformation approaches to modern DL-based techniques. However, with the growing complexity of these ML models, there emerges a substantial need for lightweight frameworks capable of executing on-board computations, especially on-edge devices. This is where TinyML comes into play, providing a new frontier for developing and deploying lightweight ML models. The dynamic nature of HAR research and the continual emergence of innovative approaches, including TinyML, testify to the vast potential and exciting future prospects of this field.

Transitioning to TinyML: TinyML technologies have brought a fresh perspective to edge computing, with a particular focus on HAR [23], [24], [25]. Several innovative techniques have emerged to balance the use of the available resources with the need for data bearing in mind the inherited challenges from the conventional ML task [3], [26], [27], [28], [29].

The study in [26] shows a binarized neural network that successfully reduced the memory requirements and inference time. Another work introduced an adaptive neural network, capable of modifying its structure according to available resources and increasing efficiency across different HAR scenarios [27]. On the other hand, advancements were made by designing CNNs specifically for HAR, considering aspects such as the influence of hyper-parameters on performance and the energy efficiency of the models by assessing the inference time and memory footprint [3], [28].

These efforts were further emphasised by a proposal for lightweight and energy-efficient DL models suitable for wearable devices in [29]. Another work introduced a framework called BandX, which utilised DL inference on the edge using TinyML and shows how the network traffic can be reduced [30]. These advancements also include extraction techniques for stride-specific information from accelerometry data [31].

Moving forward, studies are exploring how to deploy efficient deep-learning models on resource-constrained devices. Notably, one study demonstrated the potential of TensorFlow Lite compression techniques on CNNs and LSTM networks, yielding considerable improvements in energy savings and inference latency [32].

A shift in research direction was observed towards integrating TinyML and edge computing for HAR using various sensing techniques and solutions like presence detection with ultra wideband radar technology [33]. Another work also emphasises the utilisation of a TinyML-based radar solution for sensing and detecting human activity [34].

Despite the strides in TinyML for HAR, challenges like data scarcity and privacy concerns persist. The laborious and costly process of gathering annotated data sets in various deployment scenarios, especially in edge computing, prompts an interest in techniques like TL. TL effectively utilises limited labelled data by leveraging knowledge from related tasks, making it increasingly vital in HAR applications.

Bridging these domains is TL, offering an effective solution for HAR in edge computing scenarios. By leveraging pre-existing knowledge from related tasks, TL reduces the necessity for a large volume of labelled data—an often expensive and labour-intensive process, thereby reducing the cost of collecting a large volume of labelled data [35], [36], [37], [38], [39], [40].

TL is an ML technique where a model developed for one task is reused as the starting point for a model on a second task. This approach is particularly beneficial in scenarios where labeled data is scarce or expensive to obtain. TL involves two main phases: pre-training and fine-tuning. In the pre-training phase, a model is trained on a large, often generic dataset to learn a wide range of features. This model is then fine-tuned by re-training it on a smaller, task-specific dataset, allowing the model to adapt these learned features to the new task. This methodology not only accelerates the training process but also enhances the model's performance, especially in specialized applications like HAR where data collection can be challenging. The efficiency of TL lies in its ability to leverage learned patterns and knowledge from one domain and apply them to another, thereby reducing the need for extensive training. In the context of our work, we apply

TL to expedite the development of TinyML applications on edge MCUs. This is achieved by first training our models on extensive datasets and then adapting them to specific tasks in edge computing, ensuring efficient performance despite the limited computational resources of MCUs. Our focus on TinyML further emphasizes the need for optimized models that can operate within the constraints of edge devices, making TL an ideal approach for developing sophisticated yet resource-efficient applications.

Several methods apply to this concept. For instance, one uses a three-phase framework with a CNN and LSTM [35]. Another study combined TL with WiFi's CSI for activity recognition, emphasising TL's potential in HAR [36].

Furthermore, TL can improve cross-domain activity recognition, as seen in the stratified TL framework [37]. The Deep Multi-scale TL (DMSTL) model and class augmentation method further develop TL's capabilities [38], [39]. TL also has applications in privacy-focused environments, as exemplified by the FedHealth framework for wearable healthcare [40], and the subject adaptor GAN (SA-GAN) for wearable sensor-based HAR [41].

On the algorithmic front, ensemble models have proven to be promising for TL. For example, a study integrated AdaBoost with CNN to design an ML method that could handle large imbalanced datasets with high accuracy [42]. The AdaBoost-CNN method significantly reduced computational costs compared to classical AdaBoost and achieved better accuracy on various datasets. Another work presents the adaptive spatial-temporal TL (ASTTL) approach that addresses challenges in cross-dataset activity recognition [43]. This approach leverages spatial-temporal features for cross-dataset HAR and can be utilised for source domain selection and accurate activity transfer.

In summary, TL has proved to be an indispensable tool for HAR, offering potential solutions to the challenges of data collection and annotation, data privacy, and accurate knowledge transfer across various domains. The continued exploration and advancement in TL methodologies are paving the way for enhanced HAR in various real-world scenarios.

In this paper, we seek to develop an approach for accelerating TinyML development with TL on-edge MCUs, addressing key aspects such as inference rate, model adaptation, and rapid deployment. By leveraging TL techniques, we aim to simplify the implementation of resource-intensive ML models, like LSTM and CNNs, on edge devices, promoting rapid model deployment and enhancing the overall performance of IoT applications. Furthermore, we investigate the potential of TinyML, emphasising the role of TFLite-μ in empowering this paradigm shift.

In conjunction with examining the establishment of a TinyML-TL infrastructure, our investigation seeks to expand upon preceding efforts in HAR and scrutinise the efficacy of TL in enhancing the precision of HAR models trained on scarce labelled data, as well as its influence on edge inference performance.

B. CONTRIBUTIONS AND ORGANISATION

This paper makes significant strides in the domain of HAR using DL, particularly suited for resource-limited edge devices. We utilise an advanced CNN-LSTM deep neural network (CNN-LSTM-DNN) TL model architecture, demonstrating enhanced feature extraction and classification performance. Furthermore, a comparative analysis of the model's edge inference footprint is provided, offering insights into the model's real-time processing capabilities and low-latency requirements. Our main contributions in this paper are as follows:

- *Advanced Model Architecture:* This study presents an efficient deep-learning model for motion classification, leveraging a CNN-LSTM-DNN architecture for superior feature extraction and classification performance.
- *Denoisng and Dimensionality Reduction:* The research employs principal component analysis (PCA) to reduce the dimensionality of the problem space. This not only improves computational efficiency but also enhances the model's edge inference potential for TinyML applications using TFLite-μ.
- *Transfer Learning:* The use of TL in the model design leverages its adaptability to new data, addressing overfitting issues and reducing the computational resources needed for training. This makes the model suitable for deployment on edge devices with limited processing power, such as MCUs.
- *Comparative Analysis of Edge Inference Footprint:* This paper includes a comprehensive comparative analysis of the model's inference footprint on edge devices. The results highlight the model's effectiveness and efficiency in real-time processing and low-latency requirements of motion classification tasks, thereby showcasing its suitability for implementation on resource-restricted edge devices.
- *Pathway for Future Research:* By demonstrating the effectiveness of sophisticated DL models and dimensionality reduction methods for motion classification tasks, this work paves the way for future research into further optimisation strategies, alternative model architectures, and broader application areas within the domains of motion classification and edge computing.

The rest of this paper is organised as follows: In Section II, we describe the methodology employed, including data preprocessing, model development, and evaluation metrics used in our study. The results and discussion of the performance of various models are presented in Section III, highlighting the strengths and weaknesses of each approach. Finally, Section V concludes the paper, summarising our findings and providing insights into future research directions in this field.

II. DL MODEL FOR MOTION CLASSIFICATION ON EDGE DEVICES

A. THE DATASET

In this study, we designed a DL model for motion classification using the MotionSense dataset, which includes a set

of inertial signals essential for understanding the motion and orientation of a device [44]. The dataset consists of twelve time-series inertia signals collected from a smartphone worn by 24 volunteers, with diverse demographic distribution in terms of age, gender, and weight. The participants performed various activities, such as walking, climbing stairs, running, lying, and sitting, while carrying an iPhone in their front pocket.

The smartphone's inherent sensors, encompassing the accelerometer and gyroscope, were employed to capture the data at a sampling frequency of 50 Hz. These signals (features) exemplify diverse facets of the device's motion, including its orientation (attitude), the influence of gravity on its movement, its rotational velocity (rotation rate), and the acceleration perceived by the user, each spanning three spatial dimensions. In this investigation, we utilise 50Hz 2-second samples to categorise five selected labels for our classification task. The dataset is partitioned into 100×12 time series samples for every activity.

One of the challenges in motion classification using the MotionSense dataset is the variability in how individuals perform activities, which can result in different motion patterns. This will be tackled by the sophisticated DNN architecture that we will be using, as described in the subsequent sections. Furthermore, the dataset is imbalanced, with some activities having fewer samples compared to others. Hence, before performing any pre-processing or training we balance the dataset to have a similar probability of occurrence for each of the five chosen classes for our task.

Dataset for TL workflow: In the context of setting up the MotionSense dataset for the TL workflow, we harness data augmentation strategies. The primary objective here is to boost the diversity and volume of our training data by introducing minor changes, thereby bolstering our DL model's generalizbility.

Considering the data in our study exhibits a time-series character, a range of time-series data augmentation techniques are relevant. These methodologies include time warping which either expands or shrinks the series; jittering which introduces mild noise to the series; scaling which alters the time series' amplitude; and window slicing which generates new instances from diverse time windows. Each augmentation technique engenders fresh and varied motion patterns. Moreover, given the dataset's imbalanced nature as mentioned before, data augmentation can also contribute to further balancing the dataset. Additional samples of the less-represented activities can be artificially generated employing the data augmentation strategies, thereby guaranteeing an equal chance of occurrence for every activity class.

Given our training data sample $\mathbf{X}$ of size $M \times N$, our objective is to jointly augment the data to produce $\mathbf{X}_{\text{augmented}}$. The augmentation process comprises two main steps: first, we manipulate the time axis either by oversampling or downsampling and subsequently, we slice the data by extracting random consecutive samples of length M.

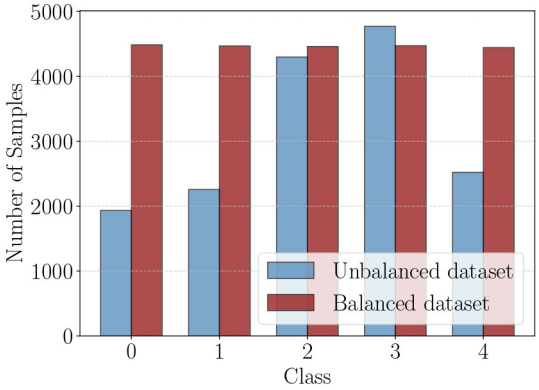

FIGURE 1. Class Distribution Comparison: Before and After Balancing.

Initially, we modify the time axis. Depending on whether we wish to elongate or condense the time duration, we can opt for oversampling (increasing data points) or downsampling (decreasing data points). This results in the modified data sample $\mathbf{X}_{\text{mod}} = [\mathbf{x}_{\text{mod},1}, \mathbf{x}_{\text{mod},2}, \ldots, \mathbf{x}_{\text{mod},N}]$ of the size $(M \times R) \times N$, where R is the oversampling or downsampling ratio.

Post to the time-axis modification, we proceed to slice the data. To extract slices, we pick a random starting point j and retrieve M consecutive samples from $\mathbf{x}_{\text{mod},i}$. This procedure is encapsulated in the following mathematical representation[1]:

$$\mathbf{x}_{\text{slice},i}[j] = \mathbf{x}_{\text{mod},i}[j : j + M], \tag{1}$$

$$\mathbf{X}_{\text{augmented}} = [\mathbf{x}_{\text{slice},1}, \mathbf{x}_{\text{slice},2}, \ldots, \mathbf{x}_{\text{slice},N}], \tag{2}$$

where, j is randomly chosen for each slice from the interval [0, (length of $\mathbf{x}_{\text{mod},i}$ - M)] ensuring that the extracted slice originates entirely from the same data sample and does not overlap with other samples in the time series data stream. Here, $\mathbf{x}_{\text{slice},i}[j]$ represents the i-th sliced data sample starting at the j-th position within $\mathbf{x}_{\text{mod},i}$, where $\mathbf{x}_{\text{mod},i}$ denotes the i-th sample of the modified data $\mathbf{X}_{\text{mod}}$. This slicing process is captured in (1), where $\mathbf{x}_{\text{slice},i}[j]$ is defined as the subset of $\mathbf{x}_{\text{mod},i}$ starting from index j and extending for M elements. Subsequently, $\mathbf{X}_{\text{augmented}}$ in (2) is constructed by assembling these sliced samples $\mathbf{x}_{\text{slice},i}[j]$ across all N modified data signals, thus creating the augmented dataset. The notation $\mathbf{x}_{\text{slice},i}$ in the context of $\mathbf{X}_{\text{augmented}}$ implies a collection of such slices for the i-th sample, all starting at the same randomly chosen index j.

Subsequent to the slicing process, we need to synchronise the labels to the newly formed slices. Given that every slice has a direct correlation to its original sample, the label of the source sample is replicated for each corresponding slice.

The results of the dataset augmentation using slicing window and time axis modifications (i.e., downsampling or oversampling) are illustrated in Figure 1. The Figure shows how the number of samples of each class is increased. This

[1]In the subsequent discussions, we will slightly abuse the notation $\mathbf{X}$ to refer to the augmented data sample $\mathbf{X}_{\text{augmented}}$.

will allow for a more generalisation accuracy. In the TL phase, the same augmented dataset will undergo a more augmentation process by squeezing and stretching the motion window and by also adding some noise, to make sure that a relatively new dataset is generated to test the TL framework against. This augmentation process in the TL phase will be performed on portions of the test dataset that are not seen in the initial training phase.

TL with the UCI dataset: In addition to the TL workflow that emphasizes data augmentation on the MotionSense dataset, as mentioned earlier, it is crucial to test the fluency of the TL workflow on a new dataset. Another dataset that shares similar HAR activities is the UCI HAR dataset [45], which presents an ideal opportunity to assess the adaptability and generalizability of our deep learning model.

The UCI dataset, while similar in its focus on HAR activities, introduces a distinct set of challenges compared to the MotionSense dataset. These differences primarily manifest in sensor configurations, data collection protocols, and participant demographics. Understanding and accounting for these variations is critical in ensuring that our model, trained initially on MotionSense, can effectively transfer its learned knowledge to the UCI dataset.

A key step in this process involves aligning the datasets for the TL workflow. From the UCI dataset, we have selected six activities that are analogous to those in the MotionSense dataset, ensuring a consistent basis for activity recognition. Additionally, while the MotionSense dataset comprises twelve inertial signals, the UCI dataset contains nine corresponding signals. This overlap forms the basis of our TL model's training, focusing exclusively on these nine matching signals.

The application of our model to the UCI dataset aims to test the model's fluency in adapting to new data characteristics. To accommodate the reduced number of signals, we will perform PCA as will be discussed in the following section. This step is critical in ensuring that the model not only adapts to the structural differences between the datasets but also efficiently processes the available data for optimal performance.

This involves evaluating how effectively the model can apply its pre-learned patterns and behaviors to a dataset with a different structure and distribution while maintaining or improving its performance metrics. The result of this tailored TL approach will provide valuable insights into the model's versatility and generalizability across different HAR scenarios.

B. PRE-PROCESSING AND FEATURES EXTRACTION

The MotionSense dataset, comprising 12 distinct signals (features), presents a multidimensional feature space that requires a systematic approach to extract the most pertinent components for ML modelling. The PCA algorithm is employed for both feature extraction and dimensionality reduction [46], [47]. The algorithm begins its process by standardising the data. Let $\mathbf{X}$ be the original data matrix (i.e., a one-time series sample of the entire dataset of 12×100 times series readings of the motion signals). The standardisation ensures each feature i has zero mean and unit variance:

$$\mathbf{x}_{s_i} = \frac{\mathbf{x}_i - \text{mean}(\mathbf{x}_i)}{\text{std}(\mathbf{x}_i)}, \text{ for } i \in \{1, 2, \ldots, 12\}, \tag{3}$$

where $\mathbf{x}_{s_i}$ is the standardised feature, $\text{mean}(\mathbf{x}_i)$ represents the mean of the feature, and $\text{std}(\mathbf{x}_i)$ denotes the standard deviation of the feature. Then by concatenating the standardised features we build the standardised data sample as:

$$\mathbf{X}_s = [\mathbf{x}_{s_1}, \mathbf{x}_{s_2}, \ldots, \mathbf{x}_{s_N}], \tag{4}$$

for $N = 12$ the total number of the data sample features. Thereafter, the covariance matrix Σ is computed to encapsulate the interrelationships between the features:

$$\Sigma = \frac{1}{M-1}\mathbf{X}_\text{s}^T\mathbf{X}_\text{s}, \tag{5}$$

where M stands for the number of readings for each feature.

The eigenvectors $\mathbf{v}_i$ and eigenvalues λ_i of the covariance matrix, defining the principal directions and their significance respectively, are then deduced:

$$\Sigma\mathbf{v}_i = \lambda_i\mathbf{v}_i \tag{6}$$

Following the computation, the eigenvectors are ordered in descending fashion based on the magnitude of their corresponding eigenvalues. The top k eigenvectors, signifying the most substantial directions of data variance, are cherry-picked. Subsequently, the data undergoes transformation to yield the transformed data matrix $\mathbf{X}_\text{PCA}$:

$$\mathbf{X}_\text{PCA} = \mathbf{X}_s\mathbf{W}_k \tag{7}$$

where $\mathbf{W}_k = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k]$ is the matrix formed by the k eigenvectors corresponding to the k largest eigenvalues in (6).

Applying PCA to both the MotionSense and UCI datasets facilitates the extraction of the most significant information while compressing the data. This reduction in dimensionality is crucial in streamlining the complexity of the problem for both datasets and mitigates concerns such as overfitting and computational demands, especially when adapting the ML model from one dataset to the other in the transfer learning workflow. Evaluations of the ML model's performance pivot around various iterations of the selected PCA components, ensuring that the model efficiently captures the essential features from both datasets.

Figure 2 illustrates the proportion of the dataset's total variance as represented by each of the top k principal components that is illustrated in the previous discussion. The graphic employs a bar plot; the x-axis signifies the principal components, whereas the y-axis correlates to their explained variance ratios.

The paramount value of PCA within this context is underscored by its potential to amplify edge inference, especially when integrated with TinyML via TensorFlow Lite

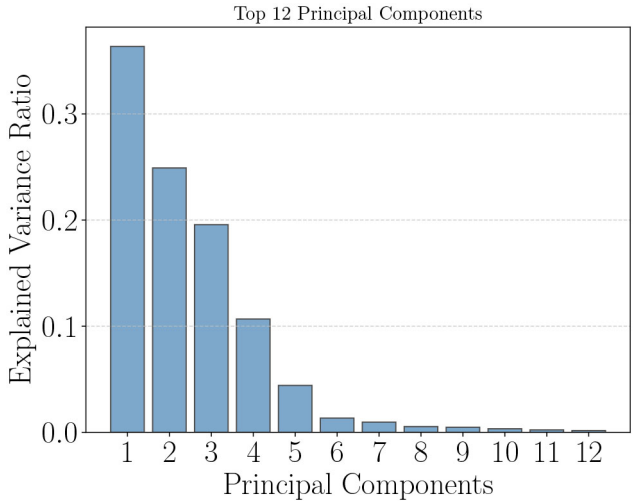

FIGURE 2. Explained Variance Ratio of the Top *N* Principal Components.

Micro (TFLite-μ). Diminishing the data's dimensionality paves the way for the generation of compact, efficient models demanding minimal memory and computational resources. Such enhancements facilitate swifter inference rates, energy efficiency, and a reduced memory footprint, proving invaluable for the integration of sophisticated ML paradigms on edge devices bounded by resource constraints. Particularly for tasks like motion classification, where immediate processing and minimal latency are indispensable.

C. CNN-LSTM-DNN-TL MODEL ARCHITECTURE

As illustrated in Figure 3, the neural network design begins with a Conv1D layer (N_1 filters, kernel size of 3, rectified linear unit (ReLU) activation, and L2-regularisation) to uncover local patterns within the input data. Following this, a Normalisation layer is employed to enhance the model's training speed and stability. Subsequently, a second Conv1D layer with N_2 filters, a kernel size of 3, ReLU activation, and L2-regularisation delves into extracting advanced feature sets. This is paired with another Normalisation layer.

To mitigate overfitting, a Dropout layer with a rate of 0.2 is employed. The features are then reshaped using a Reshape layer to prepare them for the subsequent LSTM layers. The first LSTM layer, having N_3 hidden units and configured to return sequences, is key to recognising temporal relationships. A subsequent LSTM layer with N_4 hidden units condenses these sequences into a more compact representation. Another Dropout layer with a rate of 0.2 is utilised post the LSTM layers to further avert overfitting.

The output of the LSTM is processed by a Dense layer with N_5 hidden units, ReLU activation, and L2-regularisation to learn a nonlinear transformation. Finally, a Dense layer with 5 units equipped with a soft-max activation function discerns the ultimate output classifications.

We also extend the evaluation of the implications of modifying the number of Conv1D layers (N_1 and N_2), the hidden units in the LSTM layers (N_3 and N_4), and the units in the Dense layer (N_5). The goal here is to show the interplay between model performance metrics and resource demands, especially with an emphasis on edge computing applications with TinyML. Hence, tuning these hyper-parameters, can compile trade-offs between the model complexity and computational limits.

LSTM Layers Architecture: Following the model architecture we discussed in Figure 3, it is noteworthy to highlight that we only utilise unidirectional LSTM layers which is designed to process the input sequence in the forward direction only. The choice of unidirectional LSTMs (but not bi-directional) is to facilitate inference on edge devices using TinyML and TFLite-μ, since TFLite-μ only supports unidirectional LSTM layers as bi-directional architectures require multiple graphs which are not yet supported. For a given time step t, the LSTM equations are as follows [48]:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{8}$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{9}$$
$$\tilde{C}t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \tag{10}$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \tag{11}$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \tag{12}$$
$$h_t = o_t \odot \tanh(C_t), \tag{13}$$

where x_t represents the input vector at time step t. The hidden state vector is denoted by h_t, and the cell state vector is represented by C_t. The functions σ and tanh are the sigmoid and hyperbolic tangent activation functions, respectively. f_t, i_t, and o_t are the forget, input, and output gates of the LSTM layer, which control the flow of information through the cell state. $\tilde{C}_t$ is the candidate cell state, a temporary value that assists in updating the cell state. The weight matrices W_f, W_i, W_C, and W_o correspond to the forget, input, output, and candidate cell state, while the bias vectors b_f, b_i, b_C, and b_o are the learnable parameters associated with these gates in the LSTM layer. The symbol $\odot$ denotes element-wise multiplication, while the symbol $(\cdot)$ represents the dot product operation between vectors.

The unidirectional LSTM model captures the past information effectively by maintaining the cell state vector, which helps in mitigating the vanishing gradient problem to some extent. However, it does not take into account future information as bidirectional models do. Nevertheless, the unidirectional LSTM is suitable for deployment on edge devices due to its lower complexity, and its compatibility with TFLite-μ makes it an attractive option for real-world applications that require on-device processing. To gain a better understanding, consider that, while bidirectional LSTMs have the potential to provide greater contextual information by processing data from both past and future states, their adoption in edge environments is limited due to higher computational demands and increased model complexity (i.e., they require double the number of parameters compared to unidirectional LSTMs, leading to a larger model size and more memory consumption. Additionally, bidirectional

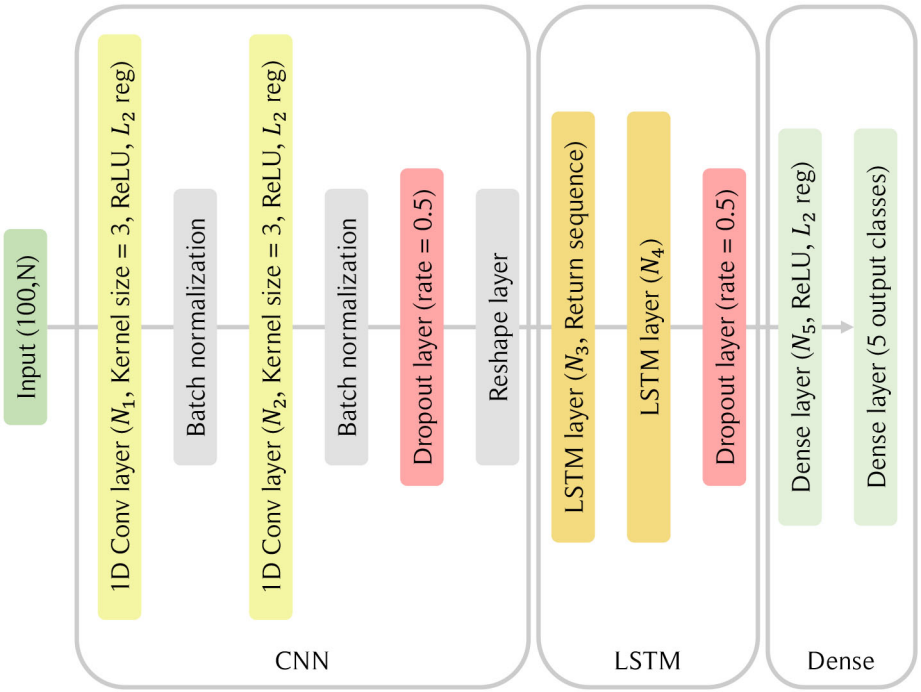

FIGURE 3. CNN-LSTM-DNN network architecture.

LSTMs necessitate the processing of data in both directions, which can result in the need for multiple graphs and a more complex control flow, further straining the limited resources of edge devices). This makes them less appropriate for configurations with strict resource constraints. Our focus on unidirectional LSTMs is thus an intentional choice, aiming at combining the demand for temporal data processing efficiency with the practicalities of deployment on edge devices.

During the initial training phase, the entire architecture is trained on the HAR motion sense dataset to learn the feature extraction capabilities. While in the TL workflow, the model is fine-tuned on the target task or domain by re-training the model to result in new model weights for certain number of the ML model layers (e.g., we may choose to train the LSTM layers and dense layers next to them will and keep the CNN layers weights untouched). This fine-tuning process allows the model to adapt to the specific characteristics of the new data, mitigating the risk of overfitting and reducing the computational resources required for training, making it particularly well-suited for edge devices with limited processing capabilities, such as TL training with Raspberry Pi4. Overall, the combination of an advanced DL model and the benefits of dimensionality reduction through PCA provides a powerful solution for motion classification tasks, particularly when deployed on resource-constrained edge devices using TFLite-μ.

Our model architecture, while seemingly a straightforward fusion of CNN and LSTM, is tailored for the specific requirements of motion classification in edge computing environments. The integration of CNN and LSTM is deliberate, leveraging the strengths of both architectures: CNNs for their superior feature extraction capabilities, particularly in handling spatial data, and LSTMs for their effectiveness in processing time-series data, crucial for understanding temporal dynamics in motion. This strategic combination allows our model to efficiently capture both spatial and temporal features, a necessity for accurate motion classification. Moreover, the model is designed with a focus on lightweight and efficient computation, making it suitable for deployment on edge devices with limited processing power. In this context, 'advanced' refers not just to the complexity of the model, but also to its optimization for specific use-cases, balancing performance and computational efficiency, which is a significant consideration in the field of edge computing and TinyML. By fine-tuning this hybrid model through TL, we ensure it remains adept at handling the nuances of motion data while being deployable in resource-constrained environments, a key contribution in our approach.

D. CLASSIFICATION PERFORMANCE METRICS

In this article, we measure multiple classification performance metrics to evaluate the classification performance of the adopted DNN model:

1) **Accuracy**: the accuracy is the ratio of correctly predicted instances to the total instances in the dataset:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{14}$$

where TP is the true positive rate, TN is the true negative rate, FP is the false positive rate, and FN is the false negative rate.

2) **Precision**: is the ratio of correctly predicted positive instances to the total predicted positive instances:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{15}$$

3) **Recall**: also known as sensitivity or true positive rate, is the ratio of correctly predicted positive instances to the total actual positive instances:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{16}$$

4) **F1 Score**: is the harmonic mean of precision and recall:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{17}$$

5) **Categorical cross-entropy loss (CCEL)**: measures the dissimilarity between the predicted probability distribution and the true probability distribution of the target classes. The categorical cross-entropy loss can be measured by:

$$\text{CCEL} = -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij}) \tag{18}$$

where N is the number of samples, C is the number of classes, y_{ij} is the true probability of sample i belonging to class j, and $\hat{y}_{ij}$ is the predicted probability of sample i belonging to class j. Lower values of categorical cross-entropy loss indicate better classification performance.

By considering accuracy, precision, recall, F1 score and the categorical cross-entropy loss, we can better assess the model's ability to classify instances while minimising false positives and false negatives correctly. These metrics together will be able to provide a general and deeper view of the ML model performance and hence help in overcoming any overfitting issues or biases towards any of the selected HAR classes.

E. DEVELOPMENT METHOD FOR MODEL TUNING WITH TL

In our proposed framework, we utilize a Raspberry Pi as an intermediary edge platform for fine-tuning the pre-trained CNN-LSTM models using TL before deploying them on the edge MCU (i.e., ESP32S3). This approach aims to efficiently optimize the model while minimizing the resource constraints typically faced by edge IoT devices. Post-fine-tuning, the optimized model is transferred to the ESP32S3 MCU for low-power, on-device inference, enabling real-time performance and responsiveness across various application domains.

To facilitate seamless model deployment and integration into the IoT edge devices, we employ the message queuing telemetry transport (MQTT) protocol. MQTT, a lightweight and efficient messaging protocol, is widely adopted in IoT applications for its low bandwidth usage and ease of implementation, making it particularly suitable for constrained environments like edge devices (see [49] for a detailed discussion on MQTT). It is publish/subscribe model ensures efficient and reliable communication between the Raspberry Pi and the edge MCU for model updates.

It is important to note that in our work, the process of updating the TL model via MQTT is streamlined through the use of a custom-developed pipeline. This pipeline facilitates the transfer of TFLite-μ DNN model hex files. The procedure involves sequentially numbering the characters in the hex file and orchestrating their collection by the edge device. This approach is complemented by an integrated model version control mechanism. This method is especially beneficial for edge controllers with limited bandwidth, as it optimizes data transmission and provides precise model version control. While this method proves effective, it is worth mentioning that alternative approaches exist. One such approach is the implementation of over-the-air (OTA) updates, which can be used to update the entire firmware of the ESP32S3. Additionally, MQTT file uploads can be managed more directly, as exemplified by the methods detailed in the repository [50].

With this pipeline, we enable continuous adaptation and improvement of TinyML models as new data becomes available, further enhancing the performance and flexibility of the proposed TL-based TinyML development method. As a result, we aim to combine the strengths of the Raspberry Pi as an average edge computer and the ESP32S3 MCU (or any IoT MCUs) to accelerate TinyML model development. This ensures efficient power consumption and real-time performance, making it a promising solution for a wide range of edge applications that are not limited to HAR.

Figure 4 shows the comprehensive pipeline employed for harnessing the MotionSense dataset. The journey starts with raw data undergoing a standardisation process, setting the groundwork for the subsequent PCA for dimensionality reduction without sacrificing crucial information. This processed data is then fed onto the initial phase of model training. Once this model finishes, we augment the dataset, to make sure that the model will be able to distinguish classes from new unseen datasets, and utilise it in the TL process executed on a Raspberry Pi. The result of these processes is a fine-tuned model, which can be seamlessly transferred to the ESP32S3 MCU via MQTT IoT protocol. The final stage then is to collect the inference metrics from the ESP32S3 MCU and build our conclusions on the framework parameters for successful and efficient edge deployment.

III. RESULTS AND DISCUSSION

This section provides a detailed evaluation of our model's performance during both the training and TL stages, as well as the appraisal of the resource-constrained edge devices. In the initial training phase, the model is trained and tested on a powerful computing setup that includes an 11th Generation Intel Core i7-11800H CPU operating at a 2.30GHz clock speed and supporting 8 cores, a substantial 16.0 GB of

TABLE 1. Model classification test results with TL test results for different network architectures.

	Initial testing results (Epochs = 75)					TL testing results (Epochs = 15)				
CNN-LSTM-DNN architecture	Trainable parameters	F1	Precision	Recall	Accuracy	Trainable parameters	F1	Precision	Recall	Accuracy
$N_1 = 64$, $N_2 = 32$, $N_3 = 32$, $N_4 = 16$, $N_5 = 16$	19,397	0.9853	0.9854	0.9853	0.9853	11,813	0.9411	0.9457	0.9414	0.9423
$N_1 = 48$, $N_2 = 24$, $N_3 = 24$, $N_4 = 12$, $N_5 = 8$	11,933	0.9843	0.9844	0.9843	0.9842	7,397	0.9433	0.9464	0.9436	0.9445
$N_1 = 40$, $N_2 = 20$, $N_3 = 18$, $N_4 = 9$, $N_5 = 8$	8,105	0.9798	0.9798	0.9799	0.9798	4,805	0.9485	0.9515	0.9490	0.9498
$N_1 = 32$, $N_2 = 16$, $N_3 = 16$, $N_4 = 8$, $N_5 = 4$	6,253	0.9703	0.9705	0.9703	0.9703	3,997	0.9425	0.9450	0.9427	0.9434
$N_1 = 20$, $N_2 = 10$, $N_3 = 12$, $N_4 = 6$, $N_5 = 4$	3,719	0.9679	0.9692	0.9677	0.9678	2,669	0.9279	0.9324	0.9285	0.9294

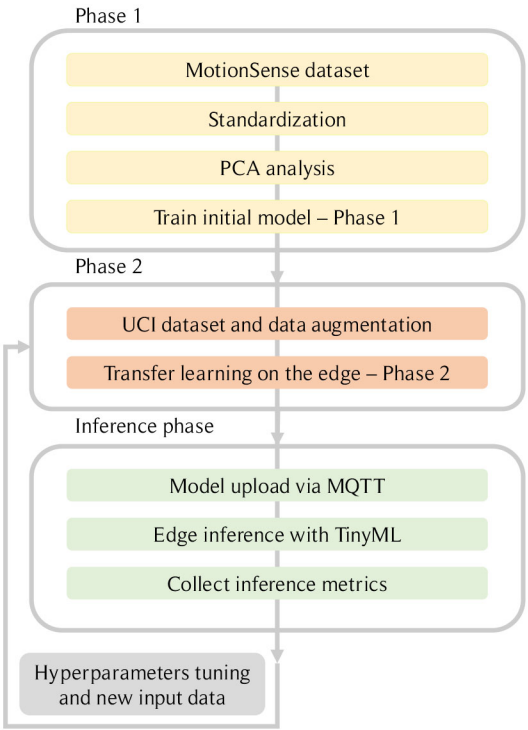

FIGURE 4. End-to-End Workflow from Data Collection to Inference on ESP32S3 using the MotionSense Dataset and TL.

RAM, and an Nvidia RTX 3050 Ti graphics processing unit equipped with 4GB of memory. For the TL phase, as mentioned earlier, we carry out the training on a Raspberry Pi 4 device equipped with a 4GB RAM variant, underlining the feasibility of our approach even on devices with limited computational resources.

A. VISUALISING FEATURE REPRESENTATIONS WITH 3D T-DISTRIBUTED STOCHASTIC NEIGHBOUR EMBEDDING (T-SNE)

Here, we use t-distributed stochastic neighbour embedding (t-SNE) which is a nonlinear dimensionality reduction technique to visualise the high-dimensional information in lower dimensions (i.e., 3D space). We utilise t-SNE to diminish the dimensionality of the feature representations from the final dense layer of our CNN-LSTM-DNN model to three dimensions. We aim here to examine the ML model's capacity to differentiate between distinct classes within the dataset. By doing this, we add an extra layer in the pipeline for testing the model's immunity against overfitting. With only 3D space, we can show the degree of separation between data points related to the different classes. Thus, we provide extra evidence of the model's proficiency in learning discriminative features. The t-SNE algorithm accomplishes this by keeping the local structure of the data in the high-dimensional space while minimising the divergence between pairwise similarities in both high-dimensional and low-dimensional spaces. With this visualisation technique, we prove the performance of our CNN-LSTM-DNN model and impart insights into its capability to accurately classify the data.

As illustrated in Figures 5 and 6, we show the features of the test dataset derived from the final dense layer. These representations in 2D and 3D allow an intensive examination of the class separation within the test dataset. It is evident from the figures that the CNN-LSTM-DNN model has a significant ability to distinguish between the classes, thus signifying the measured accuracy of data classification.

In addition, sub-figures (a) in Figures 5 and 6 illustrate the interplay of components 0 and 1. The distinct class separation resonates with the model's capacity to recognise varying types of motion patterns and its readiness to accurately categorise them in real-life scenarios. Also, sub-figures (b) in Figures 5 and 6, present the t-SNE components 0 and 2 of the 3D dimensional T-SNE dimensionality reduction, where the formed clusters further endorse the model's effectiveness in classifying individual motion types. Sub-figures (c) in Figures 5 and 6 showcase the interaction between components 1 and 2, once again revealing the model's strong classification performance, capable of handling the interplay of these specific components. Also, sub-figures (d) in Figures 5 and 6 render a 3D perspective of the last dense layer features, maintaining the clarity of cluster separation even in this expanded view. This persistence of distinct classification across dimensions solidifies the model's classification capabilities.

B. ML-TL MODEL EVALUATION

1) TL WITH DATA AUGMENTATION

Table 1 presents a comparative analysis of various CNN-LSTM-DNN architectures and their performance in initial and TL training scenarios. The analysis includes five configurations of CNN 1D and LSTM layers. In the initial training phase, the models were trained for 75 epochs. Remarkably, the architecture with Conv1D(64), Conv1D(32), LSTM(32), LSTM(16), and Dense(16) layers demonstrated the highest accuracy, recall, precision, and F1 score of approximately

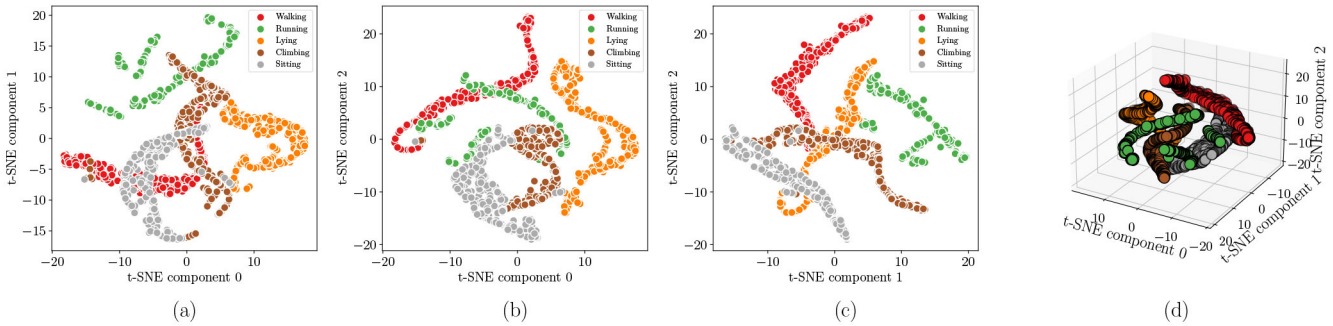

FIGURE 5. T-SNE visualisation of the last dense layer features. (a) Components 0 and 1, (b) components 0 and 2, (c) components 1 and 2. (d) 3D visualisation of the last dense layer features. Summarised network architecture: Conv1D(64), Conv1D(32), LSTM(32), LSTM(16) and Dense(16).

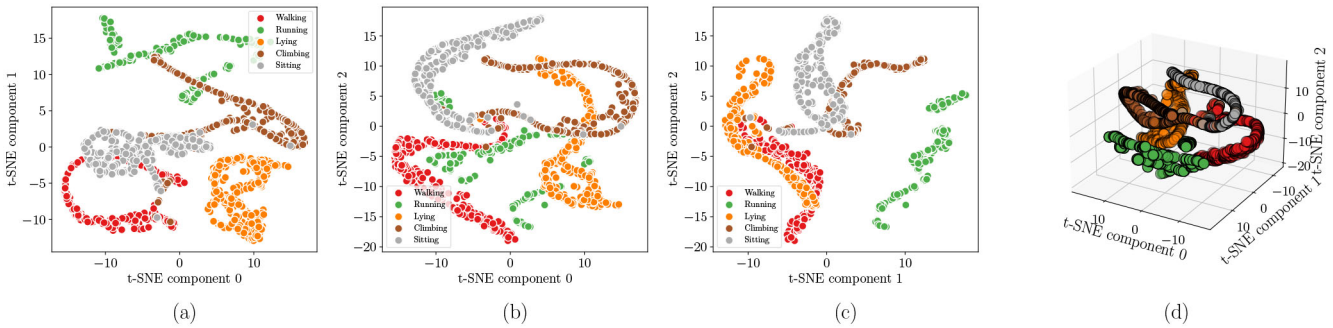

FIGURE 6. T-SNE visualisation of the last dense layer features. (a) Components 0 and 1, (b) components 0 and 2, (c) components 1 and 2. (d) 3D visualisation of the last dense layer features. Summarised network architecture: Conv1D(20), Conv1D(10), LSTM(12), LSTM(6) and Dense(4).

0.98 but with most trainable parameters (19,397). These results show the correlation between the complexity of the model and its performance and hence provide more insight into the performance of the chosen model against overfitting or under-fitting. Also, the architecture with Conv1D(40), Conv1D(20), LSTM(18), LSTM(9), and Dense(8) layers achieved almost similar performance, despite having significantly fewer trainable parameters (8,105). This could be an indication that further refinement of the model configuration could yield similar performance with fewer parameters and hence improve the efficiency.

In the TL training phase, the models were trained for only 15 epochs with dataset augmentation using both time stretching and additive white Gaussian noise of 0.01 variance. The Conv1D(40), Conv1D(20), LSTM(18), LSTM(9), and Dense(8) architecture again demonstrated outstanding performance, maintaining accuracy, recall, precision, and F1 scores close to the initial testing phase, even though the number of trainable parameters was dramatically reduced (3,719). This highlights the effectiveness of TL in maintaining high performance with fewer epochs and parameters. Also, it is important to emphasise that the model with the most minor complexity (i.e., Conv1D(20), Conv1D(10), LSTM(12), LSTM(6), and Dense(4)) has the lowest scores in both the initial and TL training phases. This again reinforces the notion that a certain level of model complexity is necessary for optimal performance.

The findings in this section are pivotal in the context of TinyML. That is, despite the performance edge of more complex models, it is evident that well-structured and efficient models can achieve comparable results. This is a critical observation in the context of TinyML, where the computational capacity and power availability are typically limited. Also, the effectiveness of TL as shown earlier in maintaining high performance with fewer epochs and parameters is particularly relevant. TL reduces the need for extensive training, thus improving the efficiency of the ML process and more widely for the ML MLOps and accelerates the operation in our framework. This is again pivotal for TinyML applications, where minimising computational load and memory footprint without compromising on performance is our primary objective.

As demonstrated in Figures 8 and 9, specifically within sub-figure (a) in both of them, we witness the customary progression during the training process. With the advancement of epochs, there is a concurrent enhancement in the model's training accuracy and a decrease in training loss. In parallel, we note an enhancement in validation accuracy and a decrease in validation loss, underscoring the model's capacity to extend its learning to unseen data. When we turn our attention to the training and validation curves following TL training shown in sub-figures (b) of Figures 8 and 9, a similar trend is observed. However, what is noteworthy here is the rapid convergence of the curves, signifying faster learning and generalisation capabilities. The

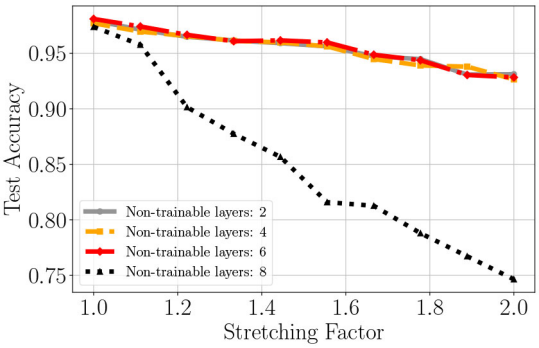

FIGURE 7. TL-Test accuracy for different augmentation stretching values and varying numbers of non-trainable TL layers, using 6 PCA components. Parameters: $N_1 = 64$, $N_2 = 32$, $N_3 = 32$, $N_4 = 16$, and $N_5 = 16$.

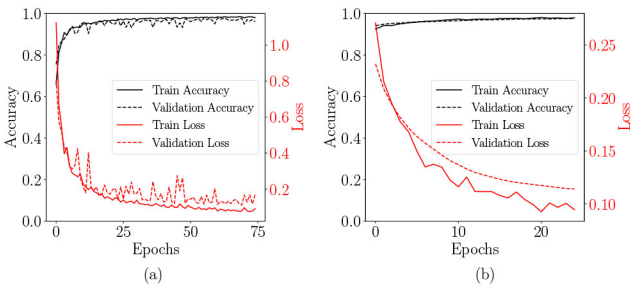

FIGURE 8. Accuracy and loss for pre-trained model and TL training (a) initial model training, (b) TL model training. TL dataset augmentation with 1.2 times stretching factor and 0.02 additive white Gaussian noise variance. Summarised network architecture: Conv1D(64), Conv1D(32), LSTM(32), LSTM(16) and Dense(16).

rapid convergence is indicative of the accelerated training process that TL brings about due to the utilisation of previously learned features. This also implies that the model equipped with TL reaches a state of balanced bias-variance trade-off faster, ultimately leading to computational efficiency.

Moreover, the proximity of the training and validation curves after convergence suggests that the model is not overfitting the training data, thus maintaining a commendable level of generalisation. This convergence, coupled with the notable performance metrics shown in Table 1, affirms the efficacy of TL in maintaining robust performance whilst reducing the computational resources needed for training. In essence, the convergence of training and validation curves for both pre and post-TL reinforces the argument that our model achieves a harmonious blend of efficiency and effectiveness. It presents an appealing case for deploying such models in resource-constrained environments, such as those typically found in IoT applications.

Figure 7 illustrates the performance of the TL model under different configurations of non-trainable layers, specifically for sets of 2, 4, 6, and 8 layers made non-trainable starting from the initial layers of the model to study the effect of data augmentation (i.e., time axis stretching in particular). The depicted model is comprised of a series of layers: two convolutional layers each followed by batch normalization, then dropout and reshaping operations, subsequently two

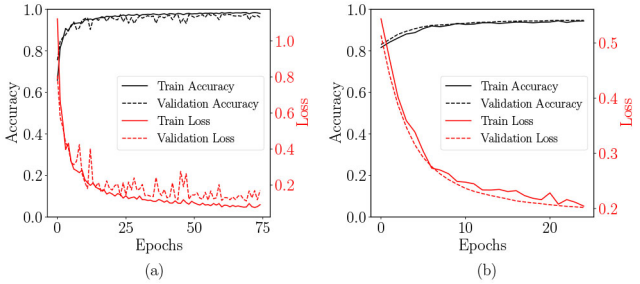

FIGURE 9. Accuracy and loss for pre-trained model and TL training (a) initial model training, (b) TL model training. TL dataset augmentation with 1.2 times stretching factor and 0.02 additive white Gaussian noise variance. Summarised network architecture: Conv1D(20), Conv1D(10), LSTM(12), LSTM(6) and Dense(4).

LSTM layers, another dropout, and finally two dense layers. This structure forms the basis for our TL experiments.

From the curves, we observe a trend of robust performance when the first 2, 4, or 6 layers are rendered non-trainable. The performance degradation is subtle across these configurations, indicating the model's resilience in preserving accuracy despite fewer trainable parameters. This suggests that in applications where computational efficiency is crucial, one might choose to train fewer layers without a significant compromise in accuracy. However, the curve corresponding to 8 non-trainable layers exhibits a pronounced drop in accuracy. This scenario leaves only the final dense layers for training. This decline in the performance emphasises the importance of the preceding layers in capturing relevant features and patterns in the data hence selecting the trainable layers in the TL phase has a strong implication on the overall performance. This understanding informs our subsequent investigation into the TL efficacy from the MotionSense dataset to the UCI dataset, providing a nuanced perspective on the adaptability of our model across diverse data domains. By examining the transition in performance through the lens of TL layer selection, we aim to shed light on the strategic nuances of optimizing model architectures for enhanced TL outcomes, especially in contexts where the balance between computational efficiency and predictive accuracy is paramount.

2) TL FROM MOTIONSENSE TO UCI DATASET

In this subsection, we extend the TL evaluation to assess the fluency of the TL framework for classification tasks on a similar yet distinct dataset. Specifically, we focus on the evaluation of the MotionSense-UCI TL workflow. It is pertinent to note the differences in this task compared to previous classification tasks. For this analysis, the time-series HAR activity signals from both datasets have been extended to encompass 128 data points, maintaining the same sampling rate of 50 Hz. This adjustment effectively translates to classifying activities based on a task length of 2.6 seconds.

Additionally, we limit the analysis to the first six PCA components for both datasets. This approach is chosen to ensure a consistent and streamlined dimensionality reduction

TABLE 2. Model classification test results for MotionSense-UCI TL.

	Initial training phase testing results: MotionSense dataset (Epochs = 75)			TL phase testing results: UCI dataset (Epochs = 20)			
CNN-LSTM-DNN architecture	Trainable parameters	F_1	Accuracy	Trainable Layers	Trainable parameters	F_1	Accuracy
$N_1 = 64$, $N_2 = 32$, $N_3 = 32$, $N_4 = 16$, $N_5 = 16$	31,173	0.9853	0.9883	CNN	10,720	0.9297	0.9308
$N_1 = 32$, $N_2 = 16$, $N_3 = 16$, $N_4 = 8$, $N_5 = 8$	12,197	0.9759	0.9730	CNN	3,056	0.8940	0.8962
$N_1 = 64$, $N_2 = 32$, $N_3 = 32$, $N_4 = 16$, $N_5 = 16$	31,173	0.9853	0.9883	LSTM	23,424	0.9420	0.9432
$N_1 = 32$, $N_2 = 16$, $N_3 = 16$, $N_4 = 8$, $N_5 = 8$	12,197	0.9759	0.9730	LSTM	9,920	0.8839	0.8829
$N_1 = 64$, $N_2 = 32$, $N_3 = 32$, $N_4 = 16$, $N_5 = 16$	31,173	0.9853	0.9883	Dense	3,685	0.8805	0.8811
$N_1 = 32$, $N_2 = 16$, $N_3 = 16$, $N_4 = 8$, $N_5 = 8$	12,197	0.9759	0.9730	Dense	1,013	0.8076	0.8083

TABLE 3. Performance metrics and inference details of models with varying architecture configurations and PCA components, tested on ESP32S3.

		Testing results (Epochs = 75)					Inference Metrics		
Number of PCA Components	CNN-LSTM-DNN architecture	Trainable parameters	F1	Precision	Recall	Accuracy	Flash memory size (Bytes)	Inference Rate (Hz)	Min Arena Size
6	$N_1 = 64$, $N_2 = 32$, $N_3 = 32$, $N_4 = 16$, $N_5 = 16$	19,397	0.9734	0.9739	0.9734	0.9733	34,672	5.88	N/A
	$N_1 = 40$, $N_2 = 20$, $N_3 = 18$, $N_4 = 9$, $N_5 = 8$	8,105	0.9853	0.9854	0.9853	0.9853	21,784	16.67	N/A
	$N_1 = 20$, $N_2 = 10$, $N_3 = 12$, $N_4 = 6$, $N_5 = 4$	3,719	0.9736	0.9736	0.9736	0.9735	16,416	33.33	N/A
5	$N_1 = 64$, $N_2 = 32$, $N_3 = 32$, $N_4 = 16$, $N_5 = 16$	19,397	0.9839	0.9839	0.9839	0.9839	34,568	5.88	21 Kb
	$N_1 = 40$, $N_2 = 20$, $N_3 = 18$, $N_4 = 9$, $N_5 = 8$	8,105	0.9712	0.9714	0.9714	0.9713	21,744	16.67	14 Kb
	$N_1 = 20$, $N_2 = 10$, $N_3 = 12$, $N_4 = 6$, $N_5 = 4$	3,659	0.9765	0.9765	0.9766	0.9765	16,440	50.00	9 Kb
4	$N_1 = 64$, $N_2 = 32$, $N_3 = 32$, $N_4 = 16$, $N_5 = 16$	19,013	0.9722	0.9722	0.9723	0.9722	34,088	5.88	21 Kb
	$N_1 = 40$, $N_2 = 20$, $N_3 = 18$, $N_4 = 9$, $N_5 = 8$	7,865	0.9768	0.9768	0.9768	0.9767	21,656	16.67	14 Kb
	$N_1 = 20$, $N_2 = 10$, $N_3 = 12$, $N_4 = 6$, $N_5 = 4$	3,599	0.9547	0.9559	0.9551	0.9547	16408	50.00	9 Kb
3	$N_1 = 64$, $N_2 = 32$, $N_3 = 32$, $N_4 = 16$, $N_5 = 16$	18,821	0.9820	0.9821	0.9821	0.9821	34,184	5.88	21 Kb
	$N_1 = 40$, $N_2 = 20$, $N_3 = 18$, $N_4 = 9$, $N_5 = 8$	7,865	0.9681	0.9682	0.9682	0.9681	20,728	16.66	14 Kb
	$N_1 = 20$, $N_2 = 10$, $N_3 = 12$, $N_4 = 6$, $N_5 = 4$	3,539	0.9511	0.9526	0.9511	0.9507	16,240	50.00	9 Kb

process across the datasets, thereby facilitating a more direct comparison in the TL context. By doing so, we maintain the integrity of the data and maximize the efficiency of our model in extracting relevant features for classification.

In the examination of the TL efficacy from the MotionSense dataset to the UCI dataset, as shown in Table 2, we observe a notable paradigm of performance variation across different architectures of a CNN-LSTM-DNN model. Initially, the model exhibits high proficiency on the MotionSense dataset, with F_1 scores and accuracy rates indicating robust learning and generalization capabilities. This is particularly evident in the larger model configuration (with higher N values), which demonstrates marginally superior performance compared to the smaller configuration. Such results underscore the initial training phase's effectiveness and set a benchmark for the model's learning capacity, crucial for subsequent TL applications.

Upon transitioning the model to the UCI dataset during the TL phase, a noticeable decline in performance metrics, including F_1 scores and accuracy, is evident across all configurations and layers, as illustrated in the table. This decrease should not be interpreted solely as a shortfall in the TL process but rather as indicative of the unique challenges posed by the UCI dataset. Literature suggests that the UCI dataset typically results in lower accuracies, ranging from 0.8920 to 0.95, in comparison to the MotionSense dataset. For instance, several studies employing the UCI dataset, including [51] with a gated recurrent neural network, [52] with an inception network architecture, and [53] with a residual bi-directional LSTM, have observed similar performance trends. While these studies achieve results comparable to our findings, they concurrently underscore the inherent intricacies and limitations of the UCI dataset. Particularly noteworthy is the relevance of these findings to the deployment of such models in edge computing scenarios. The architectural complexity and memory demands of these sophisticated models often pose significant challenges for practical applications on edge devices.

C. EDGE INFERENCE PERFORMANCE EVALUATION

1) EDGE INFERENCE RESULTS

Table 3 sheds light on the compromises and challenges associated with deploying various DNN architectures on edge devices, using the ESP32S3 controller as a reference.

At first glance, the use of PCA serves to reduce input data's dimensionality. However, the relationship between PCA components and performance isn't straightforward due to the choice of the model complexity trade-off. For instance, with 6 PCA components, the model with $N_1 = 40$, $N_2 = 20$, $N_3 = 18$, $N_4 = 9$, $N_5 = 8$ achieved a higher accuracy of 0.9853 compared to the model with more complex architecture ($N_1 = 64$), which had an accuracy of 0.9733. This suggests that simply adding more components doesn't guarantee superior performance and the choice needs to be jointly with deep measures from the complexity of the model and hence optimisation is required.

In terms of model complexity, while intuition might suggest that a higher number of trainable parameters should result in better performance, the results provide some anomalies. For instance, for 4 PCA components, the model with $N_1 = 20$, $N_2 = 10$, $N_3 = 12$, $N_4 = 6$, $N_5 = 4$ has only 3,599 trainable parameters, yet its accuracy, at 0.9547, isn't

drastically lower than the model with 19,013 parameters, which achieved an accuracy of 0.9722. This discrepancy emphasises that more isn't always better and highlights the importance of efficient parameter usage.

The inference rate, a crucial metric for real-time applications, presents some notable findings. With 3 PCA components, the simpler model ($N_1 = 20$) achieved a rate of 50.00 Hz, significantly outpacing the complex model ($N_1 = 64$) with a rate of just 5.88 Hz. This underscores the potential advantages of smaller models in real-time scenarios. This also required attention to the other edge inference metrics like the required tensor arena size.

The 'Min Arena Size' is a red flag for certain models. Several models marked 'N/A' are not deployable on the ESP32S3 and thus lead to a severe limitation. For instance, three out of three models with 6 PCA components cannot be used on the ESP32S3 due to this constraint. This highlights that, in edge computing, deployability can overshadow raw performance. This can be referred to the more HAR array size (sequence length and size) that is required when using 6 PCA components at which the memory of the controller cannot help to host at the run-time.

In summary, the table presents a critical lesson: While increasing complexity and PCA components can enhance performance metrics like accuracy, these improvements might come at tangible costsslower inference rates and, crucially, non-deployability on certain platforms. Balancing these factors is essential for effective TinyML deployment.

2) TRADE-OFFS IN SEQUENCE SIZE, SAMPLING RATE, NUMBER OF PCA COMPONENTS, AND ARENA SIZE

In the context of inference on edge devices, key parameters that impact both the model performance and computational feasibility are the sequence size, sampling rate, and the number of PCA components. The sequence size corresponds to the multiplication of the number of PCA components and the window size, where the window size refers to the time steps the model utilises to make a prediction.

Sequence Size: A larger sequence size could potentially capture more intricate and subtle temporal patterns in the data, improving the prediction performance. However, this will be on the account of the computational complexity of the model and may lead to a critical issue on resource-constrained edge devices. In particular, if the sequence size surpasses the available memory, it could result in stack overflow errors. Conversely, a smaller sequence size is computationally efficient and feasible for edge devices but limits the model's ability to capture complex temporal patterns and affects the prediction performance.

Sampling Rate: The sampling rate of the data is another critical parameter. In our particular investigation, we used a time series of 50 Hz sampling rate which is sufficient to capture the HAR patterns. However, a higher sampling rate provides more data points for the model to learn from and potentially improves the performance. However, this also increases the sequence size and the computational complexity. Decreasing the sampling rate reduces sequence length and makes the model more computationally efficient but possibly at the cost of model performance because of the potentially removed information from the time-series sequence.

Number of PCA Components: The number of PCA components pertains to the count of different principal components at each time step. While a greater number of PCA components could potentially provide the model with more information for making predictions, it increases the computational complexity. Additionally, it is worth emphasising the that the number of PCA components that is chosen for the model training at the dataset preparation phase is a key parameter in optimising the edge TL workflow.

Trade-off: Therefore, a trade-off exists between model performance and computational feasibility. The ideal sequence size is one that balances these two factors - it should be small enough to prevent stack overflow errors and other computational issues on the edge device, but also large enough to enable the model to capture necessary temporal patterns for accurate prediction. One approach to manage this trade-off is to commence with a smaller sequence size and a lower sampling rate that are computationally feasible for your edge device, then gradually enlarge the size and increase the rate until computational issues, such as stack overflow errors, begin to emerge.

Arena Size: The arena size pertains to the memory pool that TensorFlow Lite for MCUs uses. It needs to be sufficiently large to accommodate the model's weights, input, output, and intermediate arrays. If the model requires more memory than what's allocated to the arena, it can result in a stack overflow or out-of-memory errors. By increasing the arena size, these errors can be mitigated. However, this also consumes more of the device's limited memory resources.

To summarise, designing ML models for edge devices requires careful consideration of the trade-offs between model performance and computational feasibility. The sequence size, sampling rate, number of PCA components, and the arena size are all crucial factors in these trade-offs.

IV. SUMMARY OF KEY FINDINGS AND DISCUSSION

In our study of employing CNN-LSTM networks for HAR classification tasks and TL for TinyML applications on edge devices, several key findings have emerged that highlight the role of TL in studying the computational constraints of edge computing and the ways of pushing the boundaries for practical usage through model and workflow tuning.

At the center of our work is the strategic use of TL, which significantly enhances model adaptability and computational efficiency. By pre-training models on comprehensive datasets and fine-tuning them for specific HAR tasks, we observed notable improvements in both performance and training efficiency. This approach is particularly beneficial for edge devices, where computational resources are limited, underscoring TL's potential to streamline the deployment of

effective models in resource-constrained environments. Thus paving the way for more edge real-time applications beyond the HAR.

Also, our work presents a balance between model complexity and computational efficiency achieved through TL by focus for practical deployment on edge devices. Future investigations could address optimizing this balance, examining how adjustments in the architecture and TL's application impact model performance on edge devices as vital direction for advancing TinyML's practicality and effectiveness.

The application of our model across diverse datasets, including MotionSense and UCI, demonstrates TL's contribution to enhancing model robustness and versatility. This broad assessment suggests avenues for future research focused on extending TL's benefits to a wider array of HAR tasks and datasets, potentially exploring untapped areas within real-world applications.

Lastly, Addressing the inherent challenges of deploying sophisticated ML models on highly constrained edge devices remains a key area for innovation. Future work could explore more TL techniques that further reduce computational demands, including innovative fine-tuning approaches or specialized architectures designed for edge computing's unique requirements.

V. CONCLUSION

In conclusion, this paper showcases an efficient deep-learning model for motion classification utilising the MotionSense dataset. Combining CNN, LSTM, and DNN layers in our CNN-LSTM-DNN-TL model architecture, we attain powerful feature extraction and enhanced classification performance. Employing PCA for dimensionality reduction streamlines the problem while addressing overfitting and computational efficiency concerns.

Furthermore, the dimensionality reduction bolsters edge inference potential in TinyML applications using TFLite-μ, enabling the implementation of advanced ML algorithms on resource-restricted edge devices. This is particularly advantageous for motion classification tasks necessitating real-time processing and minimal latency.

The TL approach adopted in this study allows the model to adapt to new data, mitigating overfitting and lessening computational resources needed for training. This renders the proposed model apt for deployment on edge devices with limited processing abilities, such as MCUs.

In summary, our research exhibits the promise of DL models and dimensionality reduction methods in effectively tackling motion classification tasks, particularly when deployed on resource-limited edge devices. This work lays the groundwork for future investigations into further optimisation strategies, alternative model architectures, and diverse application areas within motion classification and edge computing realms.

REFERENCES

[1] Z. Sun, Q. Ke, H. Rahmani, M. Bennamoun, G. Wang, and J. Liu, "Human action recognition from various data modalities: A review," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 3, pp. 3200–3225, Mar. 2023.

[2] 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, Jun. 2021.

[3] N. Rashid, B. U. Demirel, and M. A. Al Faruque, "AHAR: Adaptive CNN for energy-efficient human activity recognition in low-power edge devices," *IEEE Internet Things J.*, vol. 9, no. 15, pp. 13041–13051, Aug. 2022.

[4] B. Yang et al., "A joint energy and latency framework for transfer learning over 5G industrial edge networks," *IEEE Trans. Ind. Informat.*, vol. 18, no. 1, pp. 531–541, Jan. 2022.

[5] S. A. Zaidi, A. M. Hayajneh, M. Hafeez, and Q. Z. Ahmed, "Unlocking edge intelligence through tiny machine learning (TinyML)," *IEEE Access*, vol. 10, pp. 100867–100877, 2022.

[6] A. M. Hayajneh, S. Aldalahmeh, S. A. R. Zaidi, D. McLernon, H. Obeidollah, and R. Alsakarnah, "Channel state information based device free wireless sensing for IoT devices employing TinyML," in *Proc. 4th IEEE Middle East North Afr. Commun. Conf. (MENACOMM)*, 2022, pp. 215–222.

[7] "PyTorch mobile." 2023. Accessed: Jun. 11, 2023. [Online]. Available: https://pytorch.org/mobile/home/

[8] "Edge impulse documentation." Edge Impulse. 2023. Accessed: Jun. 11, 2023. [Online]. Available: https://docs.edgeimpulse.com/docs/

[9] "TensorFlow lite for microcontrollers." TensorFlow Team. 2023. Accessed: Jun. 11, 2023. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers

[10] "TinyML foundation." TinyML foundation. 2023. Accessed: Jun. 11, 2023. [Online]. Available: https://www.tinyml.org/

[11] K. Chen, D. Zhang, L. Yao, B. Guo, Z. Yu, and Y. Liu, "Deep learning for sensor-based human activity recognition: Overview, challenges, and opportunities," *ACM Comput. Surveys*, vol. 54, no. 4, pp. 1–40, 2021.

[12] S. R. Ramamurthy and N. Roy, "Recent trends in machine learning for human activity recognition—A survey," *Wiley Interdiscip. Rev. Data Min. Knowl. Disc.*, vol. 8, no. 4, 2018, Art. no. e1254.

[13] S. Qiu et al., "Multi-sensor information fusion based on machine learning for real applications in human activity recognition: State-of-the-art and research challenges," *Inf. Fusion*, vol. 80, pp. 241–265, Apr. 2022.

[14] E. Ayodele et al., "Grasp classification with weft knit data glove using a convolutional neural network," *IEEE Sensors J.*, vol. 21, no. 9, pp. 10824–10833, May 2021.

[15] E. Ayodele et al., "A weft knit data glove," *IEEE Trans. Instrum. Meas.*, vol. 70, pp. 1–12, 2021.

[16] S. Zebhi, "Human activity recognition using wearable sensors based on image classification," *IEEE Sensors J.*, vol. 22, no. 12, pp. 12117–12126, Jun. 2022.

[17] N. A. Choudhury and B. Soni, "An adaptive batch size-based-CNN-LSTM framework for human activity recognition in uncontrolled environment," *IEEE Trans. Ind. Informat.*, vol. 19, no. 10, pp. 10379–10387, Oct. 2023.

[18] P. Khan, Y. Kumar, and S. Kumar, "CapsLSTM-based human activity recognition for smart healthcare with scarce labeled data," *IEEE Trans. Comput. Social Syst.*, vol. 11, no. 1, pp. 707–716, Feb. 2024.

[19] X. Zhou, W. Liang, K. I.-K. Wang, H. Wang, L. T. Yang, and Q. Jin, "Deep-learning-enhanced human activity recognition for Internet of Healthcare Things," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6429–6438, Jul. 2020.

[20] C. Han et al., "Understanding and improving channel attention for human activity recognition by temporal-aware and modality-aware embedding," *IEEE Trans. Instrum. Meas.*, vol. 71, pp. 1–12, 2022.

[21] Z. Chen, L. Zhang, C. Jiang, Z. Cao, and W. Cui, "WiFi CSI based passive human activity recognition using attention based BLSTM," *IEEE Trans. Mobile Comput.*, vol. 18, no. 11, pp. 2714–2724, Nov. 2019.

[22] D. Wang, J. Yang, W. Cui, L. Xie, and S. Sun, "Multimodal CSI-based human activity recognition using GANs," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17345–17355, Dec. 2021.

[23] L. Dutta and S. Bharali, "Tinyml meets IoT: A comprehensive survey," *Internet Things*, vol. 16, Dec. 2021, Art. no. 100461.

[24] R. Sanchez-Iborra and A. F. Skarmeta, "Tinyml-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits Syst. Mag.*, vol. 20, no. 3, pp. 4–18, 3rd Quart., 2020.
[25] P. P. Ray, "A review on TinyML: State-of-the-art and prospects," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 4, pp. 1595–1623, 2022.
[26] F. Luo, S. Khan, Y. Huang, and K. Wu, "Binarized neural network for edge intelligence of sensor-based human activity recognition," *IEEE Trans. Mobile Comput.*, vol. 22, no. 3, pp. 1356–1368, Mar. 2023.
[27] X. Wang et al., "Deep convolutional networks with tunable speed–accuracy tradeoff for human activity recognition using wearables," *IEEE Trans. Instrum. Meas.*, vol. 71, pp. 1–12, 2021.
[28] 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.
[29] Y. L. Coelho, F. D. A. S. dos 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.
[30] B. Saha, R. Samanta, S. Ghosh, and R. B. Roy, "BandX: An intelligent IoT-band for human activity recognition based on TinyML," in *Proc. 24th Int. Conf. Distrib. Comput. Netw.*, 2023, pp. 284–285.
[31] R. Jain, V. B. Semwal, and P. Kaushik, "Stride segmentation of inertial sensor data using statistical methods for different walking activities," *Robotica*, vol. 40, no. 8, pp. 2567–2580, 2022.
[32] C. Contoli and E. Lattanzi, "A study on the application of TensorFlow compression techniques to human activity recognition," *IEEE Access*, vol. 11, pp. 48046–48058, 2023.
[33] M. Pavan, A. Caltabiano, and M. Roveri, "TinyML for UWB-radar based presence detection," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2022, pp. 1–8.
[34] S. S. Yadav, R. Agarwal, K. Bharath, S. Rao, and C. S. Thakur, "tinyRadar: MmWave radar based human activity classification for edge computing," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2022, pp. 2414–2417.
[35] Y. Abdulazeem, H. M. Balaha, W. M. Bahgat, and M. Badawy, "Human action recognition based on transfer learning approach," *IEEE Access*, vol. 9, pp. 82058–82069, 2021.
[36] S. Arshad, C. Feng, R. Yu, and Y. Liu, "Leveraging transfer learning in multiple human activity recognition using WiFi signal," in *Proc. IEEE 20th Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)*, 2019, pp. 1–10.
[37] J. Wang, Y. Chen, L. Hu, X. Peng, and S. Y. Philip, "Stratified transfer learning for cross-domain activity recognition," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. (PerCom)*, 2018, pp. 1–10.
[38] Y. Zhu, H. Luo, S. Guo, and F. Zhao, "DMSTL: A deep multi-scale transfer learning framework for unsupervised cross-position human activity recognition," *IEEE Internet Things J.*, vol. 10, no. 1, pp. 787–800, Jan. 2023.
[39] K. Kondo and T. Hasegawa, "Deep transfer learning using class augmentation for sensor-based human activity recognition," *IEEE Sens. Lett.*, vol. 6, no. 10, pp. 1–4, Oct. 2022.
[40] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "FedHealth: A federated transfer learning framework for wearable healthcare," *IEEE Intell. Syst.*, vol. 35, no. 4, pp. 83–93, Jul./Aug. 2020.
[41] E. Soleimani and E. Nazerfard, "Cross-subject transfer learning in human activity recognition systems using generative adversarial networks," *Neurocomputing*, vol. 426, pp. 26–34, Feb. 2021.
[42] A. Taherkhani, G. Cosma, and T. M. McGinnity, "AdaBoost-CNN: An adaptive boosting algorithm for convolutional neural networks to classify multi-class imbalanced datasets using transfer learning," *Neurocomputing*, vol. 404, pp. 351–366, Sep. 2020.
[43] X. Qin, Y. Chen, J. Wang, and C. Yu, "Cross-dataset activity recognition via adaptive spatial-temporal transfer learning," *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 3, no. 4, pp. 1–25, 2019.
[44] M. Malekzadeh, R. G. Clegg, A. Cavallaro, and H. Haddadi, "Mobile sensor data anonymization," in *Proc. Int. Conf. Internet Things Design Implement.*, 2019, pp. 49–58.
[45] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," in *Proc. Esann*, vol. 3, 2013, p. 3.
[46] G. T. Reddy et al., "Analysis of dimensionality reduction techniques on big data," *IEEE Access*, vol. 8, pp. 54776–54788, 2020.
[47] S. Ayesha, M. K. Hanif, and R. Talib, "Overview and comparative study of dimensionality reduction techniques for high dimensional data," *Inf. Fusion*, vol. 59, pp. 44–58, Jul. 2020.
[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[49] A. Banks and R. Gupta. "MQTT version 3.1.1: OASIS standard." Oct. 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
[50] V. Alkılıç. "MQTT-file-uploader." 2023. Accessed: Dec. 22, 2023. [Online]. Available: https://github.com/volkanalkilic/Mqtt-File-Uploader
[51] Y. Wang et al., "A novel deep multifeature extraction framework based on attention mechanism using wearable sensor data for human activity recognition," *IEEE Sensors J.*, vol. 23, no. 7, pp. 7188–7198, Apr. 2023.
[52] M. Ronald, A. Poulose, and D. S. Han, "iSPLInception: An inception-ResNet deep learning architecture for human activity recognition," *IEEE Access*, vol. 9, pp. 68985–69001, 2021.
[53] Y. Zhao, R. Yang, G. Chevalier, X. Xu, and Z. Zhang, "Deep residual bidir-LSTM for human activity recognition using wearable sensors," *Math. Problems Eng.*, vol. 2018, pp. 1–13, Dec. 2018.

ALI M. HAYAJNEH received the B.Sc. and M.Sc. degrees from the Jordan University of Science and Technology, Irbid, Jordan, in 2010 and 2014, respectively, and the Ph.D. degree from the University of Leeds, Leeds, U.K. He is an Assistant Professor with the Department of Electrical Engineering, Faculty of Engineering, The Hashemite University, Zarqa, Jordan, where he also serves as the Director of the Innovation and Entrepreneurial Projects Center. His research has been supported by the Royal Academy of Engineering under two programs: the Transfer Systems through Partnerships and the Distinguished International Associate, focusing on smart agriculture, drone-assisted micro-irrigation, and tiny machine learning on edge IoT devices. Additionally, his work has received funding from the Abdul Hameed Shoman Foundation, Jordan, further enabling his research pursuits in AI driven digital twins for smart agriculture applications. His research interests include edge computing, drone-assisted wireless communications, public safety communication networks, backscatter communication, deep learning, power harvesting, stochastic geometry, device-to-device and machine-to-machine communications, modeling of heterogeneous networks, and reinforcement learning.

MARYAM HAFEEZ (Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Leeds, U.K., in 2015. From 2015 to 2018, she was a Research Fellow with the Institute of Robotics, Autonomous Systems and Sensing, University of Leeds, U.K, where she is currently an Associate Professor of Communication Networks and Systems with the School of Electronic and Electrical Engineering. Her current research is funded by the EU Horizon 2020 programme. Her research interests include the design and analysis of protocols for next generation green intelligent wireless networks by employing tools from game theory and stochastic geometry along with Internet of Things and industry 4.0 related research. She has worked in the area of dynamic spectrum access had received the best paper award at the IEEE International Conference on Communications. She is also serving as a member of the Editorial Board for *Frontiers in Communications and Networks*.

SYED ALI RAZA ZAIDI received the Doctoral degree from the School of Electronic and Electrical Engineering. He is an Associate Professor with the University of Leeds in the broad area of communication and sensing for robotics and autonomous systems. Earlier from 2013 to 2015, he was associated with the SPCOM Research Group working on US ARL funded project in the area of network science. From 2011 to 2013, he was associated with the International University of Rabat working as a Research Associate. He was also a Visiting Research Scientist with Qatar Innovations and Mobility Centre from October to December 2013 working on QNRF funded project QSON. He has published 90+ papers in leading IEEE conferences and journals. His current research interests are at the intersection ICT, applied mathematics, mobile computing, and embedded systems implementation. Specifically, his current research is geared towards: (i) design and implementation of communication protocols to enable various applications (rehabilitation, healthcare, manufacturing, and surveillance) of future RAS; and (ii) design, implementation and control of RAS for enabling future wireless networks (for, e.g., autonomous deployment, management, and repair of future cellular networks). He was awarded the G. W. and F. W. Carter Prizes for best thesis and best research paper. He has been awarded COST IC0902, Royal Academy of Engineering, EPSRC, Horizon EU, and DAAD grants to promote his research outputs. From 2014 to 2015, he was an Editor of IEEE COMMUNICATION LETTERS and also a Lead Guest Editor for *IET Signal Processing* Special Issue on Signal Processing for Large Scale 5G Wireless Networks. He is also an Editor of IET ACCESS, Front haul and Backhaul books. He is currently serving as an Associate Technical Editor for *IEEE Communication Magazine*.

DES MCLERNON received the B.Sc. degree in electronic and electrical engineering and the M.Sc. degree in electronics from the Queen's University of Belfast, Ireland, and the Ph.D. degree in signal processing from the Imperial College, University of London, U.K. He was working on radar systems research with Ferranti Ltd., Edinburgh, U.K. He is currently a Reader of Signal Processing with the University of Leeds, U.K. His research interests are broadly within the domain of signal processing for wireless communications, in which field he has around 350 research publications and also supervised over 50 Ph.D. students. Finally, in the little spare time that remains, he plays jazz piano in restaurants and bars.