ON-DEVICE DEEP LEARNING FOR MOBILE AND WEARABLE COMPUTING

(MOBİL VE GİYİLEBİLİR HESAPLAMA İÇİN CİHAZ ÜZERİNDE DERİN ÖĞRENME)

by

Sevda Özge BURSA, B.S.

Thesis

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

in the

GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

of

GALATASARAY UNIVERSITY

June 2022

ACKNOWLEDGMENTS

I am deeply indebted to my supervisors, Assoc. Prof. Dr. Gülfem IŞIKLAR ALPTEKİN and Assoc. Prof. Dr. Özlem DURMAZ İNCEL for their constant supports in the process. This research could have reached that extent with the guidance of my supervisors' immense knowledge.

June 2022

Sevda Özge BURSA

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Mobile and wearable sensor technologies have gradually extended their usability into a wide range of applications. The amount of collected sensor data can quickly become immense to be processed. The time and resource-consuming computations on such require efficient methods of machine learning and analysis, where deep learning is a promising technique. However, it is challenging to train and run deep learning algorithms on mobile devices due to resource constraints, such as limited battery power, memory, and computation units. In this thesis, we have focused on evaluating the performance of four different deep architectures when optimized with the Tensorflow Lite platform to be deployed on mobile devices in the field of human activity recognition (HAR). We have used two datasets from the literature (WISDM and MobiAct). We have compared the performance of the original models in terms of model accuracy, model size, and resource usages, such as CPU, memory, and energy usage, with their optimized versions. As a result of the experiments, we observe that the model sizes and resource consumption were significantly reduced when the models are optimized compared to the original models. Another focus is on personalizing the deep learning model of HAR tasks considering the MobiAct and OpenHAR datasets with a model trained on a larger dataset and with transfer learning that can be fine-tuned on the device with that user if possible. We observed that transfer learning can increase the accuracy rate compared to a general model without user-specific data, or a user-specific model trained with small and user-specific data.

Keywords : Human Activity Recognition, Deep Learning, Mobile Computing, Transfer Learning

RÉSUMÉ

Les technologies de capteurs mobiles et portables ont progressivement étendu leur convivialité à un large gamme d'applications. La quantité de données collectées à traiter peut rapidement devenir immense. Ces calculs gourmands en temps et en ressources nécessitent des méthodes de classification et d'analyse, où l'apprentissage en profondeur est une technique prometteuse. Cependant, il est difficile de former et d'exécuter des algorithmes d'apprentissage en profondeur sur des appareils mobiles, en raison de contraintes de ressources, telles que la puissance de la batterie, la mémoire et les unités de calcul. Nous nous sommes concentrés sur l'évaluation des performances de quatre architectures profondes différentes. Les architectures sont optimisées en utilisant la plateforme Tensorflow Lite pour être déployé sur les dispositifs mobiles dans le domaine de la reconnaissance de l'activité humaine (RAH). Nous avons utilisé deux ensembles de données de la littérature (WISDM et MobiAct). Nous avons comparé les performances des modèles originaux en termes de précision du modèle, de taille du modèle et d'utilisation des ressources, telles que le processeur, la mémoire et l'utilisation de l'énergie, avec leurs versions optimisées. A la suite des expériments, nous avons observé que les tailles de modèles et la consommation de ressources ont été significativement réduite lorsque les modèles sont optimisés par rapport aux modèles d'origine. Une autre étude était la personnalisation du modèle d'apprentissage en profondeur des tâches RAH en tenant compte des ensembles de données MobiAct et OpenHAR avec un modèle formé sur une ensemble de données plus large et avec transfert d'apprentissage qui peut être affiné sur l'appareil avec cet utilisateur si possible. Un modèle général obtenu sans données des utilisateurs ou un modèle spécifique à l'utilisateur formé avec un ensemble de données de taille petite ne peuvent pas obtenir de bons résultats. Nous avons observé que le transfert d'apprentissage peut augmenter la taux de précision.

Mots Clés : Reconnaissance de l'Activité Humaine, Apprentissage en Profondeur, Informatique Mobile, Transfert d'Apprentissage

ÖZET

Son senelerde, mobil ve giyilebilir algılayıcı teknolojileri, kullanılabilirliklerini artırarak, kademeli olarak geniş bir uygulama yelpazesine ulaştı. Bunula paralel olarak, toplanan veri miktarı hızla işlenmek için çok büyük hale gelmeye başladı. Büyük verilerin işlenmesi uzun zaman ve büyük kaynak tüketen hesaplamalar içerdiğinden, derin öğrenmenin umut verici bir teknik olduğu ve sınıflandırma ve analiz süreçlerini etkinleştirdiği ortaya çıktı. Ancak sınırlı pil gücü, bellek ve hesaplama birimleri gibi kaynak kısıtları nedeniyle, mobil cihazlarda derin öğrenme algoritmalarını eğitmek ve çalıştırmak oldukça zorlu olmaktadır. Bu çalışmada, Tensorflow Lite platformuyla optimize edilen dört farklı İnsan Etkinliği Tanıma (İET) mimarisi oluşturup, mobil cihazlardaki performanslarını değerlendirmeye odaklanılmıştır. Akademik yazından iki farklı veri kümesi (WISDM ve MobiAct) alınmıştır. Orijinal modellerin performansı, model doğruluğu, model boyutu ve işlemci, bellek ve enerji kullanımı gibi kaynak kullanımları açısından optimize edilmiş halleriyle karşılaştırılmıştır. Deneyler sonucunda, modeller orijinal modellere göre optimize edildiğinde, model boyutlarının ve kaynak tüketiminin önemli ölçüde azaldığı gözlemlenmiştir. Diğer bir odak noktası, MobiAct ve OpenHAR veri kümeleri üzerinde, İET görevlerinin derin öğrenme modelini, daha büyük bir veri kümesi üzerinde eğitilmiş bir model ve mümkünse o kullanıcıyla cihazda ince ayar yapılabilen öğrenme aktarımı ile kişiselleştirmek olmuştur. Kullanıcı verileri olmadan elde edilen genel bir model veya küçük veri grubuyla eğitilen kullanıcıya özel bir modelin iyi sonuçlar vermeyebileceği saptanmıştır. Ayrıca, öğrenme aktarımının doğruluk oranını artırabildiği gözlemlenmiştir.

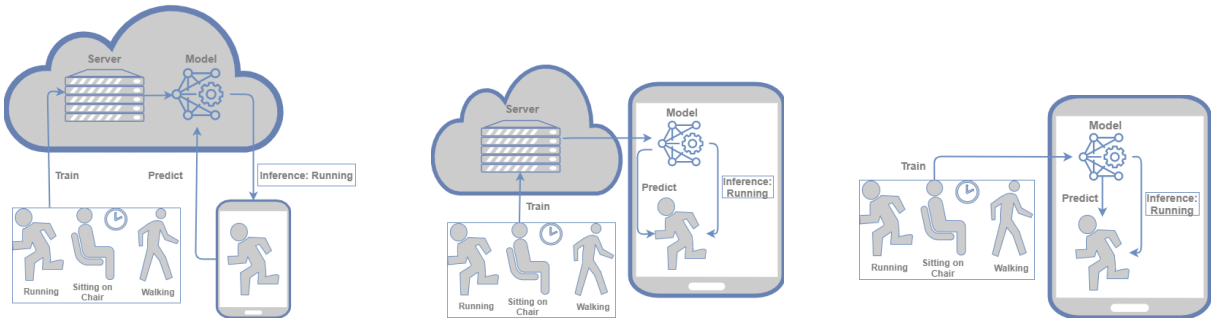
Anahtar Kelimeler : İnsan Etkinliği Tanıma, Derin Öğrenme, Mobil Hesaplama, Öğrenme Aktarımı

1 INTRODUCTION

Wearable and mobile devices, including smartphones, smartwatches, and smart glasses, have become a part of our daily lives by assisting us for various purposes. The widespread use of mobile and wearable devices among consumers has recently increased the expansion of different types of mobile artificial intelligence applications based on traditional machine learning (ML) techniques (Dai et al., 2020). Even the applications that are utilizing deep learning (DL) algorithms (Xu et al., 2019) are developed, as smartphones have become capable platforms for running these algorithms. Image and face recognition using a camera, speech recognition using a microphone, and activity tracking using motion sensors are only a few examples.

As mentioned in (Chen, Zhang, Zhang, Dai, Yi, Zhang and Zhang, 2020), there are several advantages of a well-trained DL model: i) extracting useful features from raw data (sometimes even noisy), ii) performance improvements, compared to the lightweight or shallow architectures on massive data, iii) less requirement on feature engineering. When deploying these DL-based applications to the mobile and wearable platforms (Chen, Zheng, Zhang, Wang, Shen and Zhang, 2020), there can be different configurations: i) both training and inference in the cloud, ii) on-device inference with cloud-trained models, iii) both training and inference on the device. Figure 1.1 summarizes these modes. In the first approach, named as *online mode*, or in-cloud mode (Zhou et al., 2021), mobile and wearable devices are used to collect data, and then the data is transferred to the cloud where both training and inference are performed, and the result is returned to the device. In the second approach, named *offline mode*, the training is again performed in the cloud or on a more capable device, i.e., an edge device, however, the trained model is optimized to be ported to the mobile device, and the inference is performed locally on the device. Training machine learning algorithms on resource-constrained mobile and wearable devices, particularly DL algorithms, is challenging and sometimes even impossible due to the limited computation power, storage, and, most importantly, the battery. However, recently we also see efforts to train models directly on devices or re-train/customize the trained models on the device. In

other words, these efforts handle the end-to-end learning process on the device (Zhou et al., 2021). Hence, the third mode can be named the *end-to-end mode*.



(a) Online mode (first mode) (b) Offline mode (second mode) (c) End-to-end mode (third mode)

FIGURE 1.1 – Modes of Running Deep Learning Algorithms on Mobile and Wearable Devices: example human activity recognition

Besides the mentioned resource-related challenges, other challenges include diverse computing environment, limited peak speed, limited user data, and overfitting of models, backward propagation blocking, which should be addressed by the co-design of algorithms and hardware (Zhou et al., 2021). Despite these challenges, in the literature (Verhelst and Moons, 2017; Chen, Zheng, Zhang, Wang, Shen and Zhang, 2020), efforts are focusing on the second and third modes (*offline* and *end-to-end* modes), which become possible with the introduction of dedicated hardware accelerators, the latest neural-network (NN) hardware, multi-core processors, and larger memory becoming more common in mobile and wearable devices. Parallel to the hardware improvements, also there are emerging efforts to optimize, compress and accelerate DL models without significantly reducing the accuracy (Chen, Zheng, Zhang, Wang, Shen and Zhang, 2020) in a resource-constrained environment. Examples of mobile DL frameworks (Ren et al., 2021) include TensorFlow Lite (TF Lite), Caffe2, CoreML, etc. Most of these frameworks aim to implement the inference part on smartphones, while some of them, such as CoreML, also aim to implement the training part on mobile devices.

The widespread application areas for on-device DL include computer vision (motion tracking, style transfer, arts), image recognition (scene recognition, pose estimation, face recognition), natural language processing (text classification, on-device translation, auto-completion/Smart Reply), audio classification (speech recognition) (Xu et al., 2019).

This is also related to the fact that DL algorithms have been chiefly applied and proven to be effective on these types of massive data (Wang, Davis, Zhao, Ng, Niu, Luk, Cheung and Constantinides, 2019). Mobile and wearable devices integrated with a variety of sensors frequently generate large amounts of data. Sensor-based machine learning apps on mobile and wearable devices are also gaining attention (Xu et al., 2019), such as in healthcare, sports, and well-being.

Although deep learning faces challenges due to resource constraints, such as limited battery power, memory, and computation units on mobile devices, deep learning on mobile devices provides several advantages. One of the advantages is the fast response time. With all the computation performed locally, there will be no overhead in communication time or worry about the server reliability. The cost of cloud computing resources can be reduced. Hence, the cost of maintaining or even renting cloud computing resources can be a disadvantage for some applications. It is possible to save the communication bandwidth between mobile devices and cloud computing systems. The more computing is done on the mobile device; the fewer data are sent to the cloud. Using deep learning on mobile keeps data on the device and dramatically improves user data privacy. Another advantage of using deep learning models is the performance improvements and less requirement on feature engineering than the lightweight or shallow architectures on massive data. Additionally, since these devices are personal, a machine learning model trained with data from other users may not work well for some users, and applications, such as human activity recognition (HAR), may require personalised models. A general model can be fine-tuned/adapted and or re-trained on the device with personal data.

Deep learning models may cause high resource consumption. In Section 3, we investigate solutions that significantly reduce the resource consumption of deep learning models on mobile and wearable devices. This study focuses on HAR using deep learning algorithms considering two different datasets from the literature. We mainly analyze the performance of deep learning models when optimized with the Tensorflow Lite platform to be deployed on mobile devices. Tensorflow Lite uses optimization techniques, such as quantization. WISDM dataset (*WISDM : WIreless Sensor Data Mining*, n.d.) includes 6 different activities collected from 36 participants, while the MobiAct dataset (*MobiAct*, n.d.) includes 11 activities from 61 volunteers. We train different types of deep architectures, including Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), CONVLSTM2D (with a 2D convolutional LSTM layer), and CNN + LSTM models. After training and evaluating these

models on both datasets, the developed models are transformed with Tensorflow Lite, and then the performance of the models is evaluated. We compare the performance of the original models with the converted ones in terms of model accuracy, model size, and resource usage, including CPU, memory, and energy. An average of 0.97 accuracy is achieved for WISDM data, and an average of 0.92 accuracy in MobiAct data, which is larger. We show that all optimized versions of Tensorflow Lite models running different algorithms reveal dramatic improvements in computation duration, memory usage, and energy consumption. Although we do not evaluate the performance of the models on a smartphone or another wearable device, we aim to present comparative results of original and modified models on the same device. Hence, our contribution is the comparison of the performance and resource consumption of original and modified models on two different state-of-the-art HAR datasets.

In Section 4, we focus on personalizing deep learning models trained for HAR tasks using transfer learning. And we focus on offline (Fig 1.1b) and end-to-end (Fig 1.1c) approaches of on-device learning. In offline learning, usually a general model is trained with available data on the cloud or in some cases only with user's data. In the first case, such a general model trained with data from other users may not perform well for some users with diverse characteristics. In such applications, such as human activity recognition, we need *personalized models* (Zhou et al., 2021) where a general model can be fine-tuned/adapted, or re-trained on the device with personal data to deliver personalized services to enhance user experience (Zhang et al., 2020). In the second case where only user's data is used to train the model, often, small amount of training data, particularly labeled data is available and training DNNs with small data sets may lead to overfitting. In both problems, one solution could be using transfer learning (Pan and Yang, 2010) where a model trained with a larger dataset can be used and fine-tuned for that user and if possible on the device (Buffelli and Vandin, 2021; Mairittha et al., 2021a). A transfer learning study was performed on HAR considering MobiAct and OpenHAR (Siirtola et al., 2018) datasets. When transfer learning is used, we analyzed the performance of deep learning models according to the accuracy rate. The impact of unfreezing the last layer or other layers was investigated. Higher accuracy rates were achieved with transfer learning models compared to the base model or user-specific models. The impact of the amount of user training data on transfer learning was analyzed. Although we have limited data from the user, better results can be obtained with a accuracy rate of over 90% in transfer learning. Transfer learning performance was examined with new datasets containing new classes not available in the base model. The model can learn new

classes even if the data used in transfer learning is new, there is no need to change the base model. Models can be converted to TensorFlow Lite after transfer learning. It will provide us with a personalized model and reduce resource consumption.

The rest of the thesis is organized as follows: In Section 2, we present the related studies that focus on on-device deep learning with resource-constrained devices. In Section 3, we explain the details of our methodology with Tensorflow Lite, and in Section 4, we explain the details of our methodology with Transfer Learning. Section 5 concludes the thesis.

2 LITERATURE REVIEW

2.1 On-device Deep Learning and Sensing Applications

One of the advantages of on device DL is the *fast response time* or *limited latency*. There will be no overhead in communication time or worry about the server or link reliability with all the local computation and this would make it easy to analyze data locally and also in real time (Verhelst and Moons, 2017).

Models built for mobile devices should be smaller and *energy-efficient* than models run on cloud servers. The cost of maintaining or renting cloud computing resources can be reduced, and the communication bandwidth between the devices and cloud computing systems can be saved. These all contribute to another advantage which is *resource saving* in terms of hardware and energy. Besides these advantages, wearable and mobile devices are personal devices. Hence they generate privacy-sensitive personal data. Local models can keep data on the device and significantly improve the *privacy of user data* (Zhou et al., 2021). Besides, a general machine learning model trained with data from other users may not work well for some users, and such applications may require *personalized models* (Zhou et al., 2021). A general model can be fine-tuned/adapted, or re-trained on the device with personal data to deliver personalized services to enhance user experience (Zhang et al., 2020). As these devices have become a part of our daily lives, they generate large amounts of data, and sensor data is one of the categories which rapidly grow. With the increasing capacity of computing hardware on mobile and wearable devices, such as AI chipsets, DL becomes possible on extensive sensor data.

There are various sensing application domains for mobile and wearable devices: health, sports, well-being, mood/stress recognition, activity recognition, mobility tracking, authentication, localization, rehabilitation, elderly care, sleep monitoring, augmented and virtual reality, occupational safety. However, not all of them may require on-device learning and instead may utilize the first mode or in-cloud mode (Figure 1.1a). For example, if we are interested in tracking a user's sleeping patterns, the data may be collected and

processed on a server for offline analysis. Applications not generating large or real-time data may not require on-device DL and may perform well with traditional machine learning techniques.

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

In survey (Ma et al., 2019), ubiquitous smart objects produce large amounts of data every day and thanks to advances in deep learning, this data is used quickly and efficiently. Various IoT applications of deep learning have been mentioned including smart health, smart home, smart transportation and smart industry applications. While these applications are presented, information about which models are used in the literature is also given. Healthcare applications examined under the topics of health monitoring and disease analysis. Smart home applications examined under the topics of indoor localization and home robotics. Smart transportation applications examined under the topics of traffic prediction, traffic monitoring and automonous driving. Smart industry applications examined under the topics of manufacture inspection and fault asessment.

In (Deng, 2019) the advantages of deep learning studies that can be carried out on mobile devices, hardware architectures, mobile deep learning platforms and libraries, optimization of deep learning on mobile devices and applications are introduced. Google Tensorflow Lite; runs Tensorflow models on mobile and embedded devices. Small model size provides low latency. Facebook Caffe2 is a lightweight, scalable and modular deep learning framework. It supports models trained using CNTK and PyTorch. IOS Core ML; it provides support for Keras, Caffe, Scikit-Learn, XGBoost, and LibSVM machine learning tools. Snapdragon; runs models trained in Caffe, Caffe2, Tensorflow. DeepLearningKit; it is a deep learning framework for Apple devices. It supports CNN models trained in Caffe. In order to use deep learning models in mobile applications, optimization can be with little or no compromise on accuracy. CNN architecture can be applied with pruning, quantization and Huffman

Coding techniques for optimizations. Creating a model from scratch is not always a good approach. Similar pre-trained models can be used. Pre-trained models can be fine-tuned if domain-specific data is available.

In (Verhelst and Moons, 2017), an overview of algorithmic and processor architecture optimization techniques for embedded devices are presented, in order to improve efficiency and enable the inference of deep neural networks. The focus is on the energy-efficient execution of convolutional layers, which create the bulk of the workload during inference, and techniques applied to fully connected layers. By developing customized hardware accelerators instead of general purpose processors, the energy efficiency of embedded network evaluation is improved. Enhancing and exploiting methods such as network structure, fault tolerance and network sparsity are examined. Network structure includes algorithmic techniques for deep neural network models. Network sparsity contains processor architecture techniques. Fault tolerance includes both algorithmic and processor architecture techniques. These techniques are considered separately as memory and computational bottleneck solutions. The hardware and algorithmic layers must be optimized together, by taking advantage of deep neural network's many cross-layer opportunities.

2.2 Sensors, Devices and Resource Constraints

Sensors other than the camera and microphone are available on the target devices; such as motion sensors (accelerometer, gyroscope, magnetic field), location sensors (GPS and wireless interfaces), pressure, thermometer, electrodermograph (EDA), electromyography (EMG), electroencephalogram (EEG), electrocardiogram (ECG), oximeter and proximity (such as Bluetooth) are in the focus of this thesis. In most of the sensing applications, different types of sensors are used together, such as motion sensors, location sensors, etc. Hence, the sensor data is mostly multi-modal, which is different from dealing with audio, video or textual data and sequence information needs to be captured (Kwon et al., 2021).

Integrated sensors have been surveyed in personal devices such as mobile and wearables. Smartwatches, wrist bands, smart eye-wear (glasses, head-mounted displays), e-textiles, e-patches, smart jewelry, straps, and smartphones are the example devices that are in our focus.

The devices investigated in this review are mainly characterized by the resource constraints. As explained in (Dhar et al., 2019), processing speed affects the throughput, latency and response time of an algorithm. Another limited resource is the memory. DL algorithms often require significant amount of memory for model training. The most critical resource is the power.

2.3 Transfer Learning for Activity Recognition

A survey on transfer learning is presented in (Pan and Yang, 2010). Training and test data are taken in the same feature space, when the distribution changes, it is necessary to train the model from scratch with the new training dataset, it is a costly process to rebuild the models. Knowledge transfer provides a solution for this. Transfer learning significantly saves effeort for tagging data. This allows for different assignments and distributions of data used in training and testing. Figure 2.1 presents a comparison between, traditional machine learning and transfer learning.

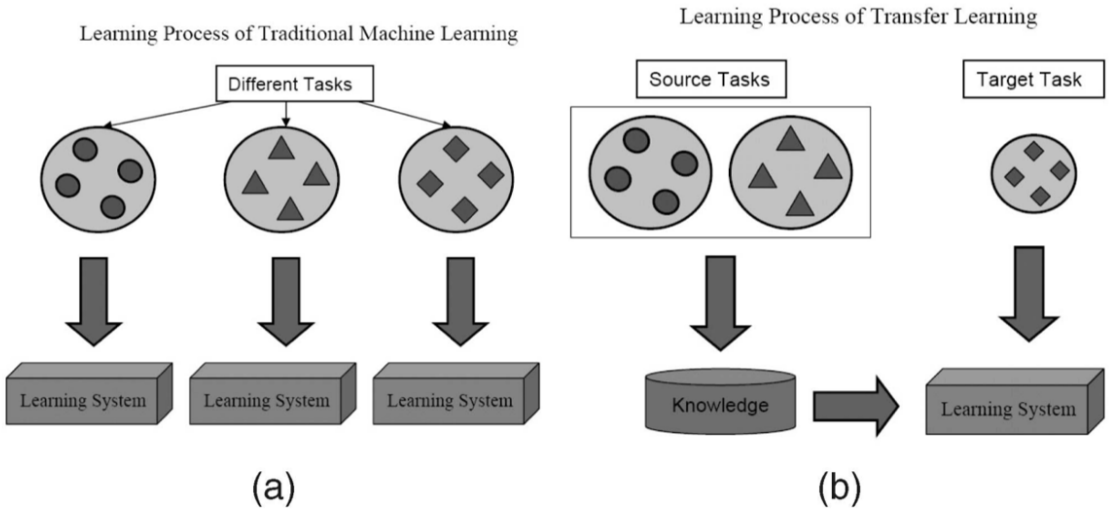


FIGURE 2.1 – (a) Traditional machine learning (b) Transfer learning (Pan and Yang, 2010)

DeepWear (Xu et al., 2020) framework transfers deep learning tasks from wearable device to handheld device paired with bluetooth local network connection (offloading). It does not require an Internet connection, consumes less energy and provides privacy for users. It provides context-aware offloading, strategic model partitioning and pipelining support techniques to efficiently utilize the processing capacity of the paired handheld device. It is

as simple as deep learning libraries like Tensorflow. Deep learning on wearables is difficult to support due to the computational load of deep learning and limited processing capacity in wearables. The reason why it is preferred to transfer the wearable device to the handheld device; handheld devices such as smartphones are powerful with multi-core CPU, GPU and GB memory. Privacy issues for users are minimized because they do not have access to the Internet. Users carry both wearable and paired handheld devices, so this offloading is possible anywhere. In context-aware offloading scheduling, the appropriate offloading is decided by examining many different factors such as deep learning model structure, network connection status, latency requirement of the application. Although handheld device is more powerful in resources than wearable devices, it still has limited resources and battery life. In partial offloading, a smaller intermediate output can be given instead of the original input size of the model. Optimized data streaming continuously optimizes the model for stream data such as video frames and audio particles. It also uses pipelined processing. The same APIs can be used as deep learning libraries like Tensorflow. Developers can set the latency requirement and energy consumption preferences. It has been implemented for Android OS and Android Wear OS. There is a tradeoff between end-to-end latency, wearable power consumption, and handheld power consumption. Offloading decisions can be dynamically adjusted based on factors such as battery life, network status, and CPU/GPU workload. Partial downloading dynamically selects the most appropriate split point with predictive models. It takes as input the possible pods created in the previous step. It decides by estimating the delay and the energy consumption for both devices. As a result of the experiments, according to the shared values, the best partition point was selected in 47 of the 48 tested cases of partition selection accuracy. Energy saving was generally achieved for both devices. The battery levels of both devices, the bluetooth bandwith, the load level of the processor of the handheld device, the delays determined by the developers are taken into account. In this study, DeepWear is used in the inference phase as opposed to the training phase. DeepWear makes its decisions according to the end-to-end latency and energy consumption.

In (Mairittha et al., 2021b), personalized activity estimates for a specific individual user were used as feedback to motivate user contribution and improve the quality of labeled data. Fine-tuning was performed with the deep recurrent neural network to eliminate the lack of sufficient training dataset and to minimize the need for training from scratch on mobile devices. Fine-tuning is a technique used in transfer learning to overcome the lack of sufficient training data. Transfer learning can reduce the need for very large labeled

data. Model pruning is used to reduce the computational cost of on-device customization without reducing accuracy. Magnitude base weight pruning is a technique for minimizing the complexity of optimizing deep learning inference. Magnitude base wight pruning works by subtracting parameters that have negligible impact on a model's predictions. Compression; prunes synapses and neurons to eliminate redundant connections, reducing model size with minimal reduction in accuracy. Thus, an activity data labeling system was created with these two techniques. On-device has three basic features: fine tuning, model optimization and personalized feedback. LSTM and CNN-LSTM models were used in the experiments. An accuracy of 98% was achieved with both models. The Simple-LSTM model is faster than the CNN-LSTM model in terms of training and inference time, and it consumes less resources. To solve the imbalance problem in real data, upsampling was done using the SMOTE (*SMOTE*, n.d.) algorithm. Results show that this proposed system improves activity recognition for individual users and reduces inference cost and latency on mobile devices. The recognition accuracy of users' average activity increased from 82% to 90%. Accuracy values also increase in the recognition of all activities.

Transfer of information focuses on what to transfer, how to transfer and when to transfer. While traditional machine learning techniques try to learn each task from scratch, transfer learning tries to transfer the information from the previous task to the target task while having less training data. Transfer learning can be divided into three headings: inductive transfer learning, transductive transfer learning, and unsupervised transfer learning. Most studies have focused on the first two other than unsupervised transfer learning. Unsupervised transfer learning should be given more focus in the future. In Figure 2.2 , the settings of transfer are presented (Pan and Yang, 2010).

In (Buffelli and Vandin, 2021), TRASEND, an attention-based DL framework for HAR with user adaptation, is proposed. TRASEND integrates a personalization module based on a lightweight transfer learning approach. Instead of using the pre-trained weights in the base, they extract the output layer from a trained model and use transfer learning only on the output layer. As a separate network, the output layer receives the output of the temporal layer as input, and it is trained with the data generated by the user. Hence, in a practical implementation, the system can ask the user for the activity labels; the output layer can be trained on-device using these samples. They present the effectiveness of the personalization step, which provides 6.2% increase on the F1 score. However, the authors do not provide an on-device implementation.

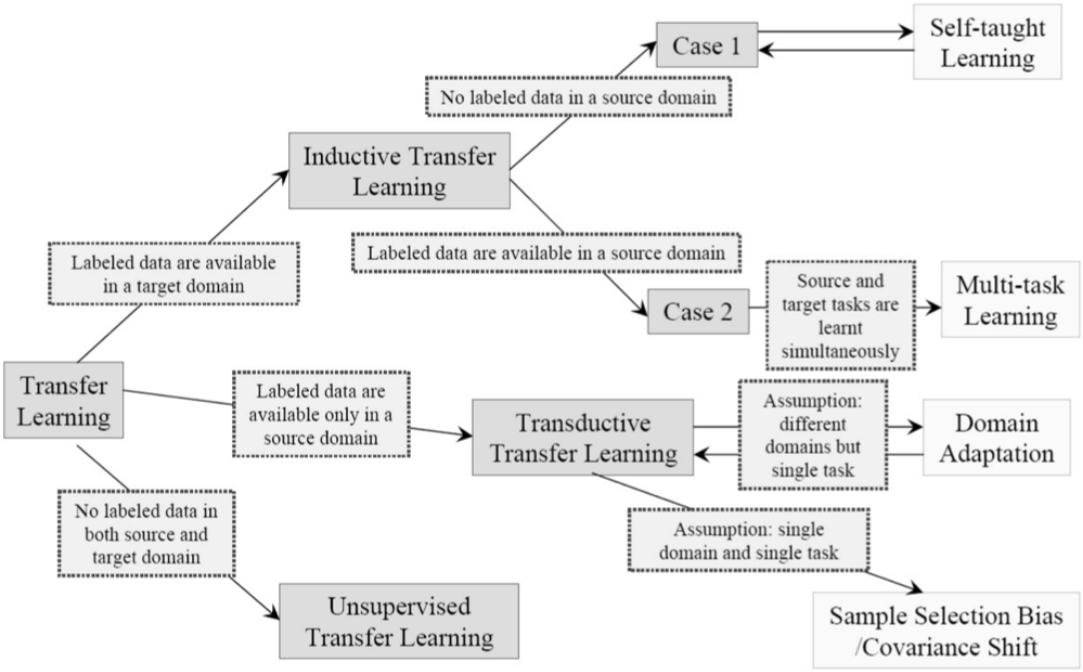


FIGURE 2.2 – Settings of transfer (Pan and Yang, 2010)

2.4 Optimization and Characterization of Networks for Mobile and Wearable Devices

DL algorithms consist of NNs containing one or more hidden layers. What distinguishes DL from traditional machine learning methods is its processing and powerful information extraction capabilities using considerable computational resources. The most popular architectures include DNN, Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN). CNN is commonly used in image and video processing, and examples include AlexNet, LeNet, ResNet. RNN, on the other hand, is primarily used in NLP and audio processing tasks.

The high computation cost of DL models makes it challenging to train and deploy the models on resource-constrained mobile and wearable devices. From the algorithmic point of view, compression and acceleration techniques are proposed in the literature to overcome this challenge.

One of the popular approaches is to optimize (i.e., compress, accelerate) a network after training. In this case, the model can be trained and optimized on an external device by

achieving a reduction in model size and then can be ported to a resource-constrained device. Another approach is to optimize a network during training with resource constraints in mind. Again, an external device can be used for training, or the model can be trained on a mobile device if sufficient resources are available. This section contains deep learning reviews in the Section 2.4.1. Secondly, in Section 2.4.2, we discuss the techniques that can be used after or during training. If the model is ported to a mobile device after training, such a static model will not change after optimization and will not adapt to dynamic/new inputs on the device (Dhar et al., 2019). To solve this problem, either the model can be fully trained on a mobile device, or an already trained model ported to a device can be updated. We also discuss the techniques to update a trained network in Section 2.4.3.

2.4.1 A Review on Deep Learning Methods

Deep learning is a neural network model with parameters and layers between input and output. Models using traditional machine learning techniques; it includes several sequential steps including preprocessing, feature extraction, intelligent feature selection, learning and then classification or regression. Unlike traditional ML methods, learning in DL methods can be achieved automatically without applying many sequential steps (Alzubaidi et al., 2021). Popular deep learning types are presented in this section: Convolutional neural networks (CNNs) and recurrent neural network (RNNs).

2.4.1.1 Convolutional Neural Network

A convolutional neural network (CNN) is a neural network algorithm to classify patterns in data. In general, neural networks consist of a series of neurons, each with their own learnable weights and biases, arranged in layer. CNNs are mathematical structures that consist of three types of layers: access, pooling, and fully connected layers. The convolution and pooling layers are the first two layers and extract the features. The third, fully connected layer, maps the resulting attributes to the output, such as classification. There are differences between feature extraction techniques, traditional machine learning classifiers, and CNN. Handmade feature extraction in traditional methods is not required for CNN. CNN is computationally more expensive than others, it needs larger data due to millions of learnable parameters. CNNs give very accurate results in image processing. Pixel values in digital images are

two-dimensional. It uses the kernel with feature extractor for each image position. One layer feeds the next layer, the extracted features become more complex. It uses optimization algorithms such as back propagation and gradient descent (Yamashita et al., 2018). CNN can reduce the parameters of the network and obtain the spatial correlation in the data to reduce the risk of overfitting (Wang et al., 2020). In Figure 2.3, basic architecture of CNN are presented (Gu et al., 2019). It consists of layers. The input part is the input layer, the output part is the output layer. In between, there are layers such as convolution layer, pooling layer, fully connection.

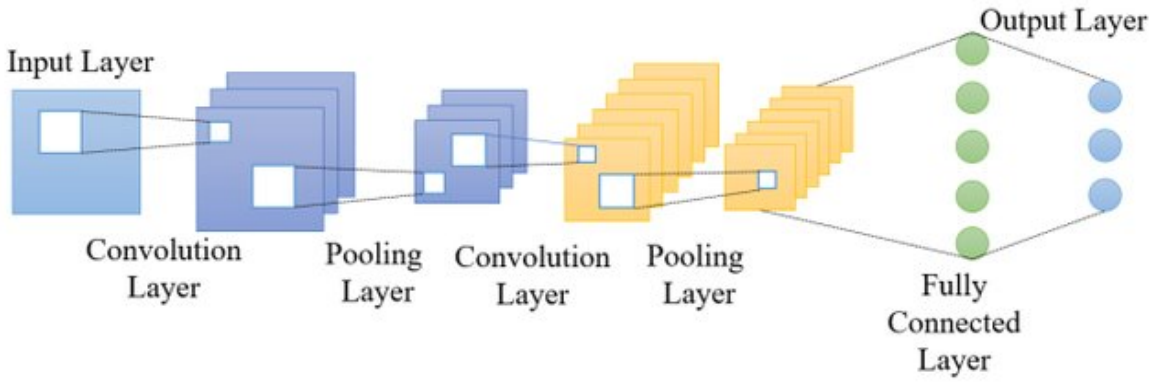


FIGURE 2.3 – Basic architecture of CNN (Gu et al., 2019)

2.4.1.2 Recurrent Neural Network

Multilayer feedforward network exhibits static behavior while recurrent neural networks exhibit dynamic behavior. RNNs have been used in many areas such as optimization, generalization and associative memories. RNNs feed the feedback and current value back to the network, thus improving its performance in classification in time series (Singh et al., 2018).

Recurrent neural networks (RNNs) are designed for sequential or time series data. CNNs, on the other hand, are good at abstracting spatial features. The RNN also receives the state of the previous timesteps, each neuron of the layer does not just output the previous layer. Thus, it uses previous information to use it with current information. But RNNs can only look back a few steps. Long Short Term Memory (LSTM) from RNN models can go back more than a few steps. LSTM uses a forget gate to decide what to keep in memory. Calculations stored throughout the learning process remain intact, resulting in better performance. RNN and LSTM are used in sequential scenarios. Examples of sequential scenarios are natural

language processing and activity recognition (Wang et al., 2020). In Figure 2.4, basic architecture of RNN are presented (Tai and Liu, 2016). The usual RNN structure is present on the left. The unfold version is on the right, and it transforms the previous information to the next time step.

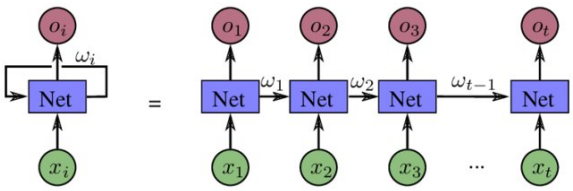


FIGURE 2.4 – Basic architecture of RNN (Tai and Liu, 2016)

2.4.2 Optimization of Models for Resource Constraints

Pruning is a commonly used technique that compresses the network by eliminating unimportant parameters. In non-structured pruning, unimportant weights could be pruned without affecting accuracy, but this may cause an irregular structure in the network (Chen, Zhang, Zhang, Dai, Yi, Zhang and Zhang, 2020). Non-structured pruning can be done at the weight, layer, or block levels. For example, magnitude-based weight pruning identifies parameters that have a negligible impact on a model's predictions and prunes synapses and neurons to eliminate redundant connections, reducing model size with minimal reduction in accuracy (Mairittha et al., 2021a). A more regular network structure can be obtained in structured pruning, which is easier for parallel computation. Types of structured pruning, namely, vector-level and kernel-level pruning, channel-level and filter level pruning, are explained in (Chen, Zhang, Zhang, Dai, Yi, Zhang and Zhang, 2020). A different approach, which uses energy-aware pruning, was proposed in (Yang et al., 2018).

Low-rank factorization: The parameters of an NN may have redundancy and this may cause training to be slow. Low-rank factorization can be applied to avoid parameter redundancy (Sainath et al., 2013). The low-order factorization algorithm combines linearly dependent vectors and gradually reduces the size of the parameters.

One of the approaches that make mobile machine learning possible is *quantization*. It reduces the overall size of the model by representing a number with low precision. For example,

it is not necessary to represent data with 32-bit wide precision, and it is possible to use parameters with lower precision, such as 8-bit wide fixed-point value (Zhou et al., 2021), without reducing the success of the model.

Knowledge distillation (KD), is a model compression technique in which a smaller "student" neural network is taught by the "teacher", the larger pre-trained neural network. There are different categories of KD. Response-based knowledge represents the final output layer and imitates the teacher model's prediction. With feature-based KD, both middle and final layer output can control student model training. Relation-based KD extracts relationships between different layers or data samples (Gou et al., 2021). *Transfer learning* is another approach similar to KD. However, KD requires a predefined student model and modifies its loss function. Transfer learning is based on the idea of training a large network on a large dataset and we will discuss how it can help to update a trained model on devices in Section 2.4.3.

Network design strategies: NNs, which have a vast design space, can be compressed and efficient, compact and low-cost network architecture can be achieved. Thus, reductions in storage and computational requirements can be achieved. Global average pooling with a fully connected layer, branching, depth-wise separable convolution, splitting a layer into multiple layers and decreasing the number of filters are examples of network design strategies (Chen, Zheng, Zhang, Wang, Shen and Zhang, 2020).

Modifying optimization routines: While training the models, the focus is on minimizing the error rate, but we can also take the computational complexity, memory footprint, computing speed, and power consumption into account. For example, in (Stamoulis et al., 2018), the authors formulate the hyperparameter optimization problem by considering both minimizing error rate and energy minimization using Bayesian optimization techniques. They show that the method reduces the energy consumed for image classification on a mobile device by up to 6× in terms of energy when tested on a commercial Nvidia mobile SoC.

Hardware-aware neural architecture search (NAS) methods (Benmeziane et al., 2021) are useful to run DNN models efficiently on devices with different hardware and tasks. NAS consists of three components. Search area represents different architectures and different value ranges for the hyperparameters. Search algorithms explore the search space to find the best architecture. Evaluator is used to obtain the accuracy of the model. With NAS algorithms, search fields are optimized according to the target hardware.

The input size in DL models directly affects the overall size of the NN, which determines the use of resources during the training and execution phases. In *data compression*, instead of reducing the model size or complexity, another approach could be to build models on compressed data by limiting the memory usage. For example, data can be transformed to a lower-dimensional space or can be sampled at a lower rate. TinyDL (Darvish Rouhani et al., 2017) is an end-to-end framework that uses a resource-aware signal conversion approach to map data to a collection of low-dimensional sub-spaces. Resource usage; it can be evaluated in terms of runtime, power, energy, and memory. The signal conversion here minimally affects the model's accuracy and provides network compression via data compression. It uses an adaptive preprocessing step for large dynamic datasets. The lower-dimensional subspace data highlights the most informative parts of the data, thus reducing the workload on the model. It provides reductions in power and energy consumption. TinyDL uses the formula $|A - DC|$ signal conversion for primary resource constraints and aims to minimize the difference. Here, A is the input data, D is the dictionary matrix, and C is the block-sparse coefficient matrix. It adapts the dictionary size according to the memory bandwidth, customizes the user-defined runtime budget, and adjusts the algorithmic parameters. The authors implemented tests on NVIDIA Jetson TK1 and measured energy, runtime, and network size. It has been reported that the DL training process has been dramatically accelerated.

These techniques can also be combined to achieve better performance. Creating a model from scratch is not always a good approach. Similar pre-trained models can be used and fine-tuned if domain-specific data is available.

The most common metrics (Zhou et al., 2021) in the evaluations are listed as: model size or compression ratio, inference latency and throughput, training speed, arithmetic operation overhead, memory footprint, model quality, energy efficiency, accuracy. For example, (Lane et al., 2015) is the first study (in 2015) on the performance characterization of inference with common DL models (such as CNNs and DNNs) on example mobile and embedded platforms: Qualcomm Snapdragon 800, Intel Edison, Nvidia Tegra K1. However, they utilize image/audio data. In another study (Mathur et al., 2018) authors investigate sensing system heterogeneity using deep learning classifiers and a data-augmented training process. The performance is tested on Snapdragon 400 processor, commonly in smartwatches, for activity recognition and speaker identification tasks and energy and memory overhead of the deep models are within reasonable limits.

One of the studies focusing on energy management is the DeLight (Rouhani et al., 2016) framework. It automates the training and execution of DNNs based on energy sources and application data by proposing an automated customization methodology to adapt the DNN configurations to the underlying hardware characteristics. The size of the data directly affects the overall size of the deep neural network model. Model size also directly determines resource usage. DeLight uses a small subset of the input data, such as %5, to decrease the model size. A close estimate of the underlying data dependencies can be obtained using a small random subset of the data without losing these dependencies. Contrary to this approach, PCA and SVD have limitations. The computational complexity of these methods is quadratic, costly in big data. It uses data geometry and platform customization to improve energy efficiency using the data projection approach recommended as a preprocessing step. Experiments have shown an energy increase of up to 100 times over previous best solutions.

In (Cai et al., 2017), the NeuralPower framework which predicts the power, runtime, and energy of CNNs without actually executing the models on a target platform. The framework models the power and runtime of the key layers that are the convolutional, the fully connected, and the pooling layers commonly used in a CNN, using a polynomial regression model.

In (Yao et al., 2018), the FastDeepIoT framework is introduced to explore the relationship between NN structure and execution time. The framework is composed of two main modules. The profiling module uses a tree-structured linear regression model for predicting the execution time of a model using the input network structure. The compression module compresses the NN to minimize execution time by using profiling results. The performance of the two modules is evaluated on Nexus 5 and Galaxy Nexus, with the TensorFlow for Mobile library with three sensing-related tasks. It is reported that FastDeepIoT can speed up the NN execution time by 48% to 78% and improve energy consumption by 37% to 69%.

In (Banbury et al., 2020) a group of researchers from TinyMLPerf working group that is comprised of over 30 organizations, present state of the art on TinyML [1] and discuss the challenges and requirements for developing a hardware benchmark for TinyML workloads which is an important initiative. TinyML is a foundation for tiny machine learning which is broadly targets machine learning technologies and applications on extremely low power, battery operated devices.

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

In the paper (Zhou et al., 2021), the motivation for shifting in-cloud learning to on-device learning and possible difficulties in system implementation are discussed. Model-level neural network design and algorithm-level training optimization are used together to optimize the performance of edge applications. Hardware-level instruction acceleration provides domain-specific accelerators to fully improve system efficiency. Knowledge distillation and model fine-tuning approaches are proposed to solve the insufficient user data challenge in on-device learning. A hardware implementation approach is proposed for the blocking solution that can be encountered in backward propagation while the DL model is being trained. In order to reduce the computational overhead, parameter pruning, loss regularization for network sparsification and data quantization approaches are proposed.

2.4.3 Training/Updating Models on the Device

Model update can be particularly important for personalizing a general model with new incoming user data on the device. As mentioned, sensing data on wearable devices exhibit personal characteristics of the user, and data distribution changes in time with user behavior. This is notably reported in user authentication and HAR studies (Chauhan et al., 2020). Moreover, new users or classes may need to be added, requiring a model update. As mentioned in (Wang, Xiao, Gao, Li and Wang, 2019), deep classifiers are not good at extrapolation when the data comes from a different distribution. However, the occurrence of this is quite common in mobile applications. Hence, the models should be re-trained or updated. Besides personalization, model-update helps avoid overfitting with limited data and reducing model complexity.

Model Fine Tuning, Transfer Learning (Zhou et al., 2021) is one of the common methods in updating an already trained model. It includes: i) pre-training a large network on a large dataset and applying the model to the on-device dataset using transfer learning, ii) defining some of the layers as updatable and some frozen, and fine-tuning these layers in training. *Model-wise feature sharing* can also be applied in transfer learning (Zhou et al., 2021). Instead of or besides layer-wise adaptation, we can share the features across models by transferring the corresponding parameters since different tasks can have a similar learning objective. Details of transfer learning techniques are presented in a survey paper (Pan and Yang, 2010).

Incremental/Online Learning: In the incremental or online learning approach, when new data streams, an existing model (metrics, weights) is updated to optimize the performance if a label is available. It is used when the training data points arrive sequentially or when it is not possible to train over the entire dataset. The methods are reported to be memory and run-time efficient (Chen and Liu, 2018). Hence, it can be an appropriate approach for mobile model update.

Continual or Lifelong Learning: Lifelong learning (Chen and Liu, 2018) uses continuous learning based on the idea that a model is trained with a sequence of N learning tasks and when a new class is encountered $(N+1)$, the model uses the past knowledge to help learn the $N+1^{th}$ task. It usually focuses on optimizing the performance of the new task where new task can be given or discovered by the system. In (Kwon et al., 2021), authors introduce a continual learning framework and test its performance on Nvidia Jetson Nano and One Plus Pro smartphone platforms.

Besides these approaches, distributed learning approaches are also emerging, such as federated learning (FL) and split learning (SL) to update models on local devices. FL (Liang et al., 2020) is a collaborative learning technique where training happens across multiple devices without exchanging or storing centralised training data. Instead, the devices only share the model parameters with a central orchestrating unit. SL (Vepakomma et al., 2018) divides a single NN into parts and distributes the lower layers across multiple devices. The devices share the parameters of the cut-layer with a server which orchestrates the training. Hence, both approaches use on-device training and can benefit from the optimization approaches on resource-constrained devices.

2.5 Deep Learning for Resource-Efficient Activity Recognition on Mobile Devices

The survey study in (Chen, Zhang, Zhang, Dai, Yi, Zhang and Zhang, 2020) focused on the solutions of DL with limited resources. For current resource limits, deep learning solutions are presented in three categories: algorithmic, computational, and hardware innovation solutions. Algorithmic designs allow reducing resource consumption by mathematically re-tuning the deep neural network model and algorithm. Simplification techniques include deeply separable convolution, matrix factorization, weight matrix distribution, weight matrix compression, data size reduction, and mathematical optimization. Computational

optimization techniques include data caching, verbose memory usage, code parallelization, and fine-tuning of parallel code.

The widespread application areas of on-device DL include computer vision, image recognition, natural language processing, and audio classification; since DL algorithms have been chiefly applied and proven to be effective on extensive data, such as image, video, and text (Wang, Davis, Zhao, Ng, Niu, Luk, Cheung and Constantinides, 2019). However, mobile and wearable devices are integrated with various sensors and frequently generate large amounts of data. One of the popular application areas is human activity recognition (HAR) using motion sensors (Wang, Chen, Hao, Peng and Hu, 2019). Although the performance of DL algorithms has been evaluated on datasets of human activity recognition (Wang, Chen, Hao, Peng and Hu, 2019), primarily the evaluations are performed offline, not on a resource-constrained device. Some recent studies (Yao et al., 2017; Ignatov, 2018; Radu et al., 2018; Peppas et al., 2020) have focused on porting the deep learning models and improving the performance on resource-constrained mobile and embedded devices.

In (Yao et al., 2017), a deep-learning framework integrating convolutional and recurrent neural networks is proposed, and it is tested on sensor data from three different application areas: car tracking, HAR, and user identification. For HAR, they utilized a dataset for training the models and tested the performance of the models on two platforms: Nexus 5 with Qualcomm Snapdragon 800 SoC and Intel Edison Compute Module. In (Ignatov, 2018), an architecture using Convolutional Neural Networks for local feature extraction together with simple statistical features is proposed for real-time HAR. The performance of the model is evaluated on two datasets. Authors import the model on a mid-range Nexus 5X Android smartphone and report that the model could classify about 28 samples per second which should be sufficient for real-time HAR. In (Radu et al., 2018), the performance of four types of deep neural networks are explored on four different HAR datasets, and the performance is tested on two embedded platforms, Snapdragon 400 and Snapdragon 800 SoCs.

A deep learning methodology combined with complementary information from the shallow feature for activity classification from inertial sensor data is proposed in (Ravi et al., 2016). This research proposes combining shallow and deep features that can be implemented in real-time on a wearable device. By limiting the connections between the network nodes and using several hidden layers, features are computed efficiently, thus reducing the

computational cost.

Peppas et al., (Peppas et al., 2020) propose a real-time physical activity recognition method using CNNs where the models are trained on two datasets. The trained models are exported with the TensorFlow Lite framework to a mobile device. Their performance is measured in terms of throughput and size, and it is shown that the number of parameters and the size of the model are five to eight times smaller than the compared model. This work is the most similar one to our study and uses the same WISDM dataset. However, they only use two CNN architectures and report the performance in terms of throughput, model size, and the number of model parameters. In contrast, we use four different architectures and compare the performance of original and optimized models in terms of CPU, memory, and battery usage.

Energy consumption is not prioritized, as most research focuses primarily on achieving a high level of accuracy without any computational constraints. In (García-Martín et al., 2019), the aim is to provide guidelines for the use and creation of specific energy estimation methods for ML algorithms. Energy consumption analysis is divided into two, as software-level and hardware-level. At the software level, developers are concerned with the energy consumption of the application. The software level is divided into two categories: application and instruction. At the hardware level, developers deal with the energy consumption of hardware components. In most studies for energy estimation in DL, energy estimation techniques are used in the inference phase, and post-processing techniques are applied to reduce the energy consumption of the neural network. Few studies focus on the training stage. There are studies in the current state-of-the-art energy predictions in machine learning through energy forecasting modeling or directly integrating power monitoring tools into existing ML packages. Deep learning models are usually trained on desktop GPUs. Hence, energy estimation models involving GPUs are needed. However, estimating energy consumption is difficult due to the lack of suitable tools to measure power models in current ML packages.

In a performance analysis study (Grzymkowski and Stefański, 2020), convolutional neural networks implemented in RISC-based processor architecture were evaluated. In order to shorten the inference time, it is necessary to simplify the network by using higher computational capabilities or lower depth. This may result in an increase in power consumption or a decrease in accuracy. Besides the energy constraint, another constraint

is dynamic memory. Insufficient memory size constrains the depth and size of the neural network, thus having a significant impact on performance. Performance is also limited by computing resources. These include factors such as whether the system is single or multi-core, instruction set and parallelism, and core frequency. It is analyzed how constraints on computing resources, cache access and availability of DSP intructions running on multiple data points simultaneously affect the performance of convolutional neurals running on the device. RAM usage is highly dependent on the number of layers, parameters, size of buffers for intermediate results, and input data. The inference time is directly related to the number of layers and parameters. By migrating the model to Tensorflow Lite, reduction in inference time can be achieved. By optimizing the model, there is more gain in inference time. As a result of the analysis, most of the computation time (about 95%) is spent doing convolutions. Optimizing convolutions will have a significant impact on performance. In order to evaluate the positive effect of cache and core frequency on performance, the inference time is extended when the test is performed by disabling the cache and lowering the core frequency. Memory access is the limiting factor for the kernel to run at full capacity. The cache size must be large enough for the kernel to be used at full capacity efficiently and to process data continuously. Core frequency and cache should be set together. Apart from these, there are many factors to consider. For this reason, all elements should be evaluated together.

SplitEasy (Palanisamy et al., 2021) trains complex deep learning models on resource constrained mobile devices. It runs the computationally intensive layers on the server, while the sensitive layers run on the mobile device. With minimal changes, deep learning models work under split learning. The reason why transfer learning is not used instead of split learning, it is still necessary to train the models. The reason why federated learning is not used, users' privacy is not protected. As shown in the figure, the first and last parts of the model are processed by the mobile device and a server is processing the middle part. The middleware outputs are manipulated without passing any information about the input data and tags. In the data sent to the server, the outside party does not know what the input data and the model are trying to learn and protects the privacy of the users. SplitEasy is flexible and works with simple architectures. Graphs are disconnected as the recommended split learning will split a network between multiple models. Therefore, it does not rely on the overall loss calculated in the final model, as it will update the model. It creates an approach based on gradients from the previous section. A permanent socket connection must also be

established between the client and the server. The location of the server and the connection to the server can affect the overall uptime. The server and mobile device will need to be on the same network in order to work faster. Split learning techniques are one of the solutions for training complex deep learning models on resource-constrained mobile devices.

3 ON-DEVICE HUMAN ACTIVITY RECOGNITION

Wearable devices allow massive amounts of sensor data to be collected, including movement, location, physiological signals, and environmental information. Recognition of human activity interprets the properties obtained from these data to understand human behavior. HAR is the estimation of what a person is doing mostly based on motion sensors to track their movements. Movements, including standing, sitting, walking, and climbing stairs, are usually everyday indoor activities. The sensors are typically embedded in a smartphone or a watch. Commonly used motion sensors include accelerometers, gyroscopes, and magnetometers. There are many fields, where human activity recognition can be used, such as health, remote monitoring, gaming, security, surveillance and human-computer interaction. In this section, we examine strategies that significantly reduce resource consumption of deep learning models on mobile and wearable devices. This thesis investigates the use of deep learning for HAR, incorporating two different datasets from the literature as examples. We focus on how to optimize deep learning models for mobile devices with the Tensorflow Lite platform. Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), CONVLSTM2D (with a 2D convolutional LSTM layer), and CNN + LSTM models are among the deep learning architectures we train. The created models are transformed with Tensorflow Lite and then their performance is evaluated on both datasets. In terms of model accuracy, model size, and resource utilization, such as CPU, memory, and energy, we compare the performance of the original and converted models.

3.1 Datasets

Two different datasets were utilized. The first one is the Wireless Sensor Data Mining (WISDM) laboratory (*WISDM : WIreless Sensor Data Mining*, n.d.) dataset. It was collected in a controlled laboratory environment. This dataset consists of accelerometer data collected from a smartphone. The dataset contains 1,098,207 rows and 6 columns. 6 activities are targeted: Walking, running, upstairs, downstairs, sitting, standing, and 36 people participated

in the data collection process. The WISDM dataset has the following columns: user id information, description of the activity, timestamp, information of x, y, and z axes from the accelerometer, presented in Table 3.1. The sampling rate is 20 Hz. The size of the data collected from each user is different from the other. The user with the fewest data has 11,371 samples, while the user with the most data has 56,632 samples. The distribution of the activities in the dataset is shown in Figure 3.1.

TABLE 3.1 – WISDM dataset format (*WISDM : WIreless Sensor Data Mining*, n.d.)

user_id	activity	timestamp	x_axis	y_axis	z_axis
33	Jogging	49105962326000	-0.694638	12.680544	0.503953
33	Jogging	49106062271000	5.012288	11.264028	0.953424
33	Jogging	49106112167000	4.903325	10.882658	-0.081722
33	Jogging	49106222305000	-0.612916	18.496431	3.023717
33	Jogging	49106332290000	-1.184970	12.108489	7.205164

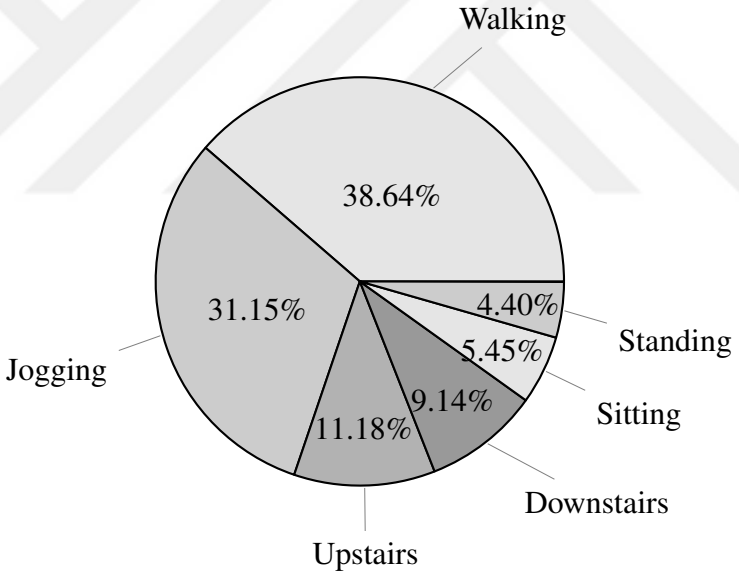


FIGURE 3.1 – Distribution of activities in WISDM (*WISDM : WIreless Sensor Data Mining*, n.d.)

As the second dataset, MobiAct (*MobiAct*, n.d.) data were used. This dataset consists of accelerometer data, and it has 5,275,371 rows. 11 activities were performed: standing, walking, jogging, jumping, stairs up, stairs down, sit chair, sitting-on-chair, chair up, car step-in, and car step-out. There are 61 participants, and there is an almost even distribution. The user with the fewest data has 70844 events, and the user with the largest data has 97815 samples. The distribution of the activities is shown in Figure 3.2.

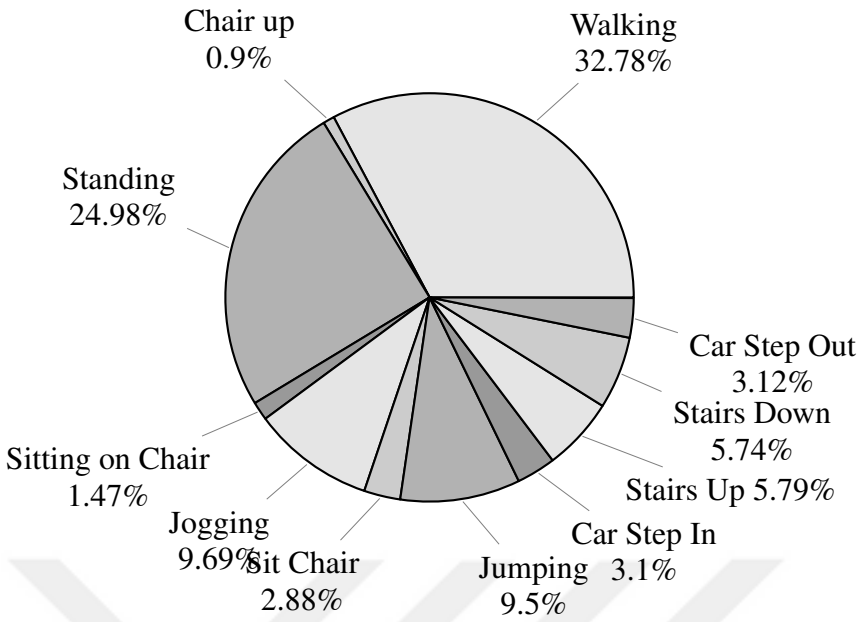


FIGURE 3.2 – Distribution of activities in MobiAct (*MobiAct*, n.d.)

3.2 Model Training

For comparison, before converting the models to light/mobile versions, deep learning models were trained on the datasets. We measure both the accuracy and resource consumption of these models to evaluate the success of methods.

Before training, the same pre-processing steps were performed for both datasets. The data were scaled with *StandardScaler* in ScikitLearn [1] for accelerometer values. It is a scaling method for standardizing features by removing the mean and scaling to unit variance. Since target (y) values are string data, they had to be enumerated, and we used *OneHotEncoder* in the ScikitLearn platform. Since the sensor data is streaming, we applied windowing using 80 samples per window size with a 50% overlapping ratio.

Experiments with DL models were performed for both datasets. CNN, LSTM, ConvLSTM2D, and CNN + LSTM models were trained. *GridSearchCV* in ScikitLearn was used for hyperparameter tuning in all models. With this library, different models were created by using all combinations for the parameters and values to be tested in the model. The best parameters of the model were determined according to the model that gave the best result according to the determined success criteria. In the CNN experiments, 2D

1. https://scikit-learn.org

convolutional layers were created in Keras. *'sparse_categorical_ crossentropy'* is used as a loss function. 'Adam' as the optimizer, 'Accuracy' as the metric, '0.001' as the learning rate, and '100' as the epoch were used.

LSTM is an artificial recurrent neural network (RNN) architecture. In the LSTM experiment, we used a simple bi-directional LSTM model. 'categorical_crossentropy' is used as a loss function, 'Adam' as the optimizer, 'Accuracy' as the metric, '0.01' as the learning rate, '64' as batch size and '50' as the epoch were used.

ConvLSTM2D uses a 2D convolutional LSTM layer. A convolutional LSTM is similar to an LSTM, but the input and recurrent transformations are both convolutional. In the ConvLSTM2D experiments, 'categorical_crossentropy' is used as a loss function. 'Adam' as the optimizer, 'Accuracy' as the metric, '0.01' as the learning rate, '64' as batch size, and '50' as the epoch were used.

CNN layers for feature extraction on input data are paired with LSTMs to facilitate sequence prediction in the CNN+LSTM architecture. 'categorical_crossentropy' is used as a loss function, 'Adam' as the optimizer, 'Accuracy' as the metric, '64' as batch size and '100' as the epoch were used. Models are trained for both datasets with the same parameters.

3.3 Performance Evaluation of Deep Learning Models

In this section, we first present the accuracy results of the models on both datasets. In Table 3.2 and Table 3.3 , we summarize the accuracy values for WISDM and MobiAct datasets, respectively. On average, for the WISDM data, the CNN model achieves an accuracy of 0.96, LSTM is 0.97, CONVLSTM2D is 0.96, and CNN + LSTM model is 0.98. There is no significant difference in the performance of the models. Additionally, we present the recognition performance of each activity in terms of precision, recall, and F1-score in Table 3.2 for the WISDM dataset. Except for the walking upstairs and walking downstairs activities, all the activities are recognized with more than 0.95 scores by all the models. In Figure 3.4, we present confusion matrix of CNN+LSTM model for WISDM dataset. When the confusion matrix was examined, we observe that these two classes were mixed with each other. This is an expected results since we do not have sensor data to distinguish upstairs and downstairs. These two classes can be combined and evaluated as a single class which is

TABLE 3.2 – Recall, precision, f1 score and average accuracy for the WISDM dataset

WISDM							
		Downstairs	**Jogging**	**Sitting**	**Standing**	**Upstairs**	**Walking**
LSTM	**Accuracy**	0.9672					
	F1 Score-Weighted	0.9674					
	F1 Score-Micro	0.9672					
	F1 Score-Macro	0.9565					
	Recall	0.92	0.99	0.97	0.99	0.88	0.97
	Precision	0.84	0.99	1	0.98	0.91	0.98
	F1 Score	0.88	0.99	0.98	0.98	0.90	0.98
CNN	**Accuracy**	0.9574					
	F1 Score-Weighted	0.9563					
	F1 Score-Micro	0.9574					
	F1 Score-Macro	0.9327					
	Recall	0.79	0.99	0.95	0.94	0.82	0.99
	Precision	0.93	0.97	0.95	1	0.85	0.97
	F1 Score	0.85	0.98	0.95	0.97	0.84	0.98
CONVLSTM2D	**Accuracy**	0.9653					
	F1 Score-Weighted	0.9652					
	F1 Score-Micro	0.9653					
	F1 Score-Macro	0.9533					
	Recall	0.85	0.99	0.97	0.99	0.89	0.98
	Precision	0.90	0.98	0.99	1	0.87	0.97
	F1 Score	0.88	0.98	0.98	0.99	0.88	0.98
CNN + LSTM	**Accuracy**	0.9768					
	F1 Score-Weighted	0.9768					
	F1 Score-Micro	0.9768					
	F1 Score-Macro	0.9700					
	Recall	0.93	0.99	0.98	0.99	0.91	0.98
	Precision	0.92	0.98	1	0.99	0.92	0.98
	F1 Score	0.93	0.98	0.99	0.99	0.92	0.98

usually the case in the related studies and datasets.

These models were compared with previous accuracy results for WISDM dataset in the literature. Studies using deep learning were examined. In Figure 3.3, comparative results are presented.

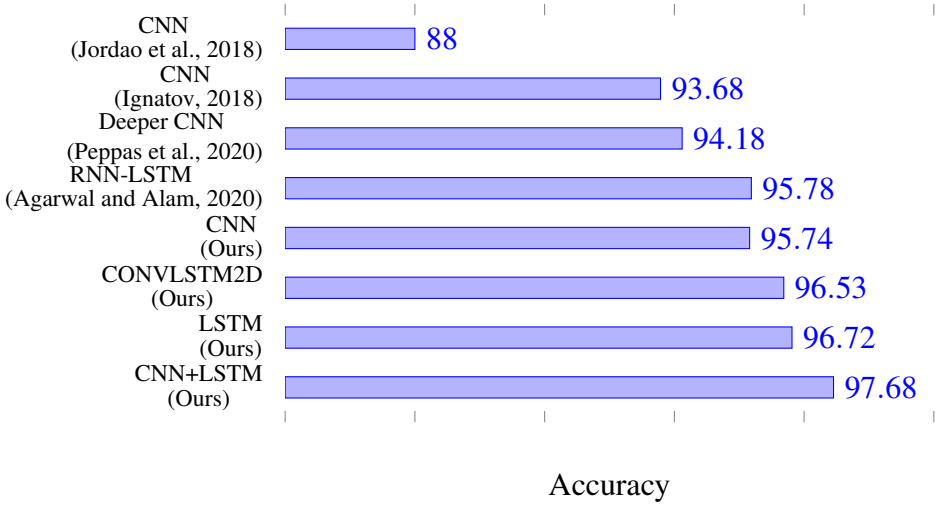


FIGURE 3.3 – Comparison of accuracy results with existing works for WISDM dataset

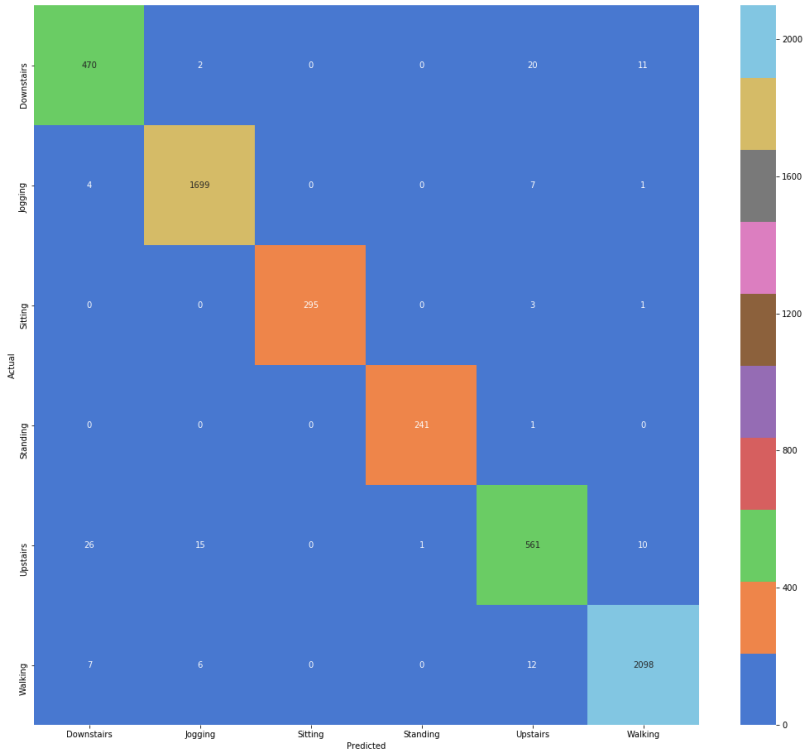


FIGURE 3.4 – Confusion Matrix of CNN+LSTM Model for the WISDM dataset

As presented in Table 3.3, for the MobiAct dataset the accuracy of the CNN model is 0.90, LSTM's accuracy is 0.92, ConvLSTM2D is 0.92 and the CNN + LSTM is 0.92. The results are lower than the WISDM data since there are more classes in this dataset, and the number of instances per activity is more unbalanced here. In Table 3.3 precision, recall, and f1 score values of all activities for MobiAct Data are presented for all four models. In Figure 3.5, we present a confusion matrix of CNN+LSTM model for MobiAct dataset.

TABLE 3.3 – Recall, precision, f1 score and average accuracy for the MobiAct dataset

MobiAct												
		Chair Up	**Car Step In**	**Car Step Out**	**Jogging**	**Jumping**	**Sit Chair**	**Sitting On Chair**	**Standing**	**Stairs Down**	**Stairs Up**	**Walking**
LSTM	**Accuracy**	0.9229										
	F1 Score-Weighted	0.9196										
	F1 Score-Micro	0.9229										
	F1 Score-Macro	0.7982										
	Recall	0.26	0.70	0.56	0.96	0.98	0.77	0.89	0.99	0.70	0.75	0.99
	Precision	0.55	0.71	0.64	0.98	0.98	0.78	0.85	0.91	0.83	0.80	0.98
	F1 Score	0.36	0.71	0.60	0.97	0.98	0.78	0.87	0.95	0.76	0.78	0.98
CNN	**Accuracy**	0.9015										
	F1 Score-Weighted	0.8935										
	F1 Score-Micro	0.9015										
	F1 Score-Macro	0.7305										
	Recall	0.11	0.52	0.48	0.96	0.97	0.83	0.56	0.99	0.66	0.62	0.99
	Precision	0.61	0.77	0.64	0.97	0.99	0.60	0.76	0.89	0.78	0.85	0.95
	F1 Score	0.19	0.62	0.55	0.97	0.98	0.69	0.64	0.94	0.72	0.71	0.97
CONVLSTM2D	**Accuracy**	0.9236										
	F1 Score-Weighted	0.9200										
	F1 Score-Micro	0.9236										
	F1 Score-Macro	0.7975										
	Recall	0.28	0.68	0.56	0.97	0.98	0.79	0.90	0.99	0.72	0.72	0.99
	Precision	0.58	0.75	0.63	0.97	0.98	0.77	0.77	0.92	0.80	0.87	0.98
	F1 Score	0.38	0.71	0.59	0.97	0.98	0.78	0.83	0.95	0.76	0.79	0.98
CNN + LSTM	**Accuracy**	0.9236										
	F1 Score-Weighted	0.9208										
	F1 Score-Micro	0.9236										
	F1 Score-Macro	0.796										
	Recall	0.31	0.59	0.63	0.97	0.98	0.78	0.89	0.99	0.74	0.73	0.99
	Precision	0.52	0.76	0.65	0.98	0.99	0.75	0.72	0.92	0.80	0.86	0.98
	F1 Score	0.39	0.66	0.64	0.97	0.98	0.76	0.80	0.95	0.77	0.79	0.98

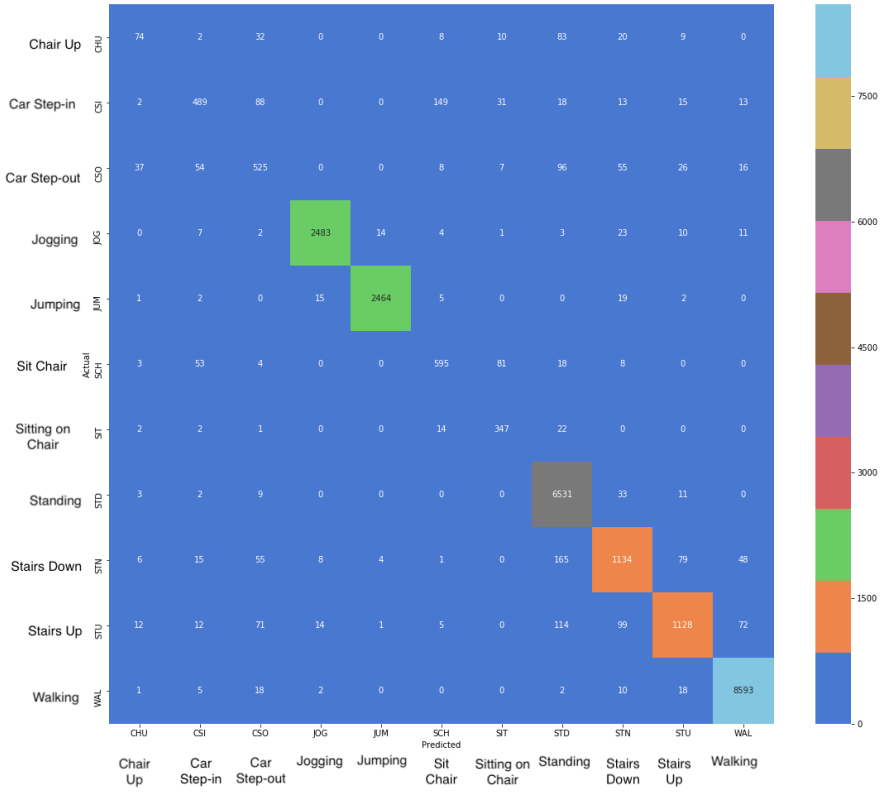


FIGURE 3.5 – Confusion Matrix of CNN+LSTM Model for the MobiAct dataset

3.4 Tensorflow Lite and Transfer of Models to Mobile Devices

TensorFlow-Lite [2] is a specialized tool-set for more efficient running of TensorFlow models on mobile, embedded and IoT devices. Other tools include CoreML [3], Create ML [4], and ARKit [5] from Apple, and FirebaseML [6], and ARCore [7] from Google. TensorFlow Lite is a DL framework for on-device deployment, and in this work, we utilize this tool. All the models explained in Section 3.2 were also created on Tensorflow Lite so that the optimized models can be run on a resource-constrained mobile device. After the model is converted, optimization methods are applied to the model. Quantification is one of the optimization methods that is run after training. Post-training quantization reduces model size, improves CPU and hardware accelerator latency, and it is a conversion technique with a slight reduction in model accuracy. For post-training quantization, models were created by dynamic range quantization and Float16 quantization. The simplest form of post-training quantization is dynamic range quantization, which merely statically quantifies weights from floating-point to integer. In Float16 quantization, the size of a floating-point model is reduced by measuring the weights as float16 for 16-bit floating-point numbers.

3.5 Monitoring Resource Consumption

Resource consumption measurements were logged while the models were making their predictions. We used three metrics: CPU, memory, and energy usage. 100000 test data were used for each value in the measurements. In smaller test data sizes, the measurement results were close to zero, especially in the resource consumption values of Tensorflow Lite models. Using 100000 test instances enables us to compare the performance of each model. The computer's specifications on which the experiments were performed are as follows: Processor is Intel(R) Core(TM) i7-8565U CPU @ 1.80 GHz 1.99 GHz, RAM is 16 GB, and OS is Windows 10 Enterprise. Since the results are not logged on a mobile device, one may argue that they may not reflect the exact values. This is correct. However, we aim to present comparative results with the original models, and it is challenging to run the full models on a mobile device. Hence, the results are taken on a laptop for comparison.

2. https://www.tensorflow.org/lite/
3. https://developer.apple.com/documentation/coreml
4. https://developer.apple.com/machine-learning/create-ml
5. https://developer.apple.com/augmented-reality/
6. https://firebase.google.com/
7. https://developers.google.com/ar

We used the *cProfile* library in Python in order to measure the CPU usage time. cProfile measures how long the function takes and how many times it is called (Wagner et al., 2017). We utilized the time spent on the CPU as the metric and *psutil* library for memory usage measurement. psutil has the memory_info function. The rss (resident set size) field returned by this function is the size of the physical memory used by the target process (Hansen, 2018). Windows Management Instrumentation (WMI) provides detailed information about a computer system (*Windows Management Instrumentation*, n.d.). The consumed energy was measured, with the remaining battery capacity information provided before and after the models ran.

3.6 Tensorflow Lite Performance Evaluation

In section 3.3, we present the accuracy results of the DL models. In this section, we present the model sizes and resource consumption results when optimized with TensorFlow Lite and compare them with the results of the original models. All models were evaluated for their accuracy values, model sizes, and resource consumption, including CPU, energy, and memory usage.

In Table 3.4, we present the accuracy results of the models after transforming with TensorFlow Lite. The column named TensorFlowLite+Dynamic shows the results with dynamic range quantization, whereas TensorFlowLite+Float16 includes the results of Float16 quantization. There is not a significant difference in the accuracy of the models in Tensorflow Lite versions for both datasets compared to the accuracy results of the original models presented in Table 3.2 and Table 3.3. Mostly, the accuracy values are the same. The reason for obtaining the same accuracy values could be that both data sets are small.

We present the model sizes in Figure 3.6. When Tensorflow models were converted to the optimized versions in Tensorflow Lite, we observed that the model size was reduced by more than two times. Tensorflow Lite models are about three times smaller in dynamic range quantization and around two times smaller in Float16 quantization. Finally, in Figure 3.7 and Table 3.5, CPU, energy, and memory usage measurements are presented for both datasets with four models and four different Tensorflow formats. In all models, CPU time, consumed memory, and memory usage are several times higher in Tensorflow formats than Tensorflow Lite models. Tensorflow Lite and float16 quantization of Tensorflow Lite show similar

TABLE 3.4 – Accuracy values for Tensorflow Lite versions of two datasets

Accuracy			
		WISDM	**MobiAct**
LSTM	**Tensorflow Lite**	0.9672	0.9229
	Tensorflow Lite + Dynamic	0.9668	0.9228
	Tensorflow Lite + Float16	0.9674	0.9230
CNN	**Tensorflow Lite**	0.9574	0.9015
	Tensorflow Lite + Dynamic	0.9574	0.9013
	Tensorflow Lite + Float16	0.9574	0.9015
CONVLSTM2D	**Tensorflow Lite**	0.9653	0.9236
	Tensorflow Lite + Dynamic	0.9653	0.9231
	Tensorflow Lite + Float16	0.9653	0.9235
CNN + LSTM	**Tensorflow Lite**	0.9768	0.9236
	Tensorflow Lite + Dynamic	0.9768	0.9231
	Tensorflow Lite + Float16	0.9768	0.9236

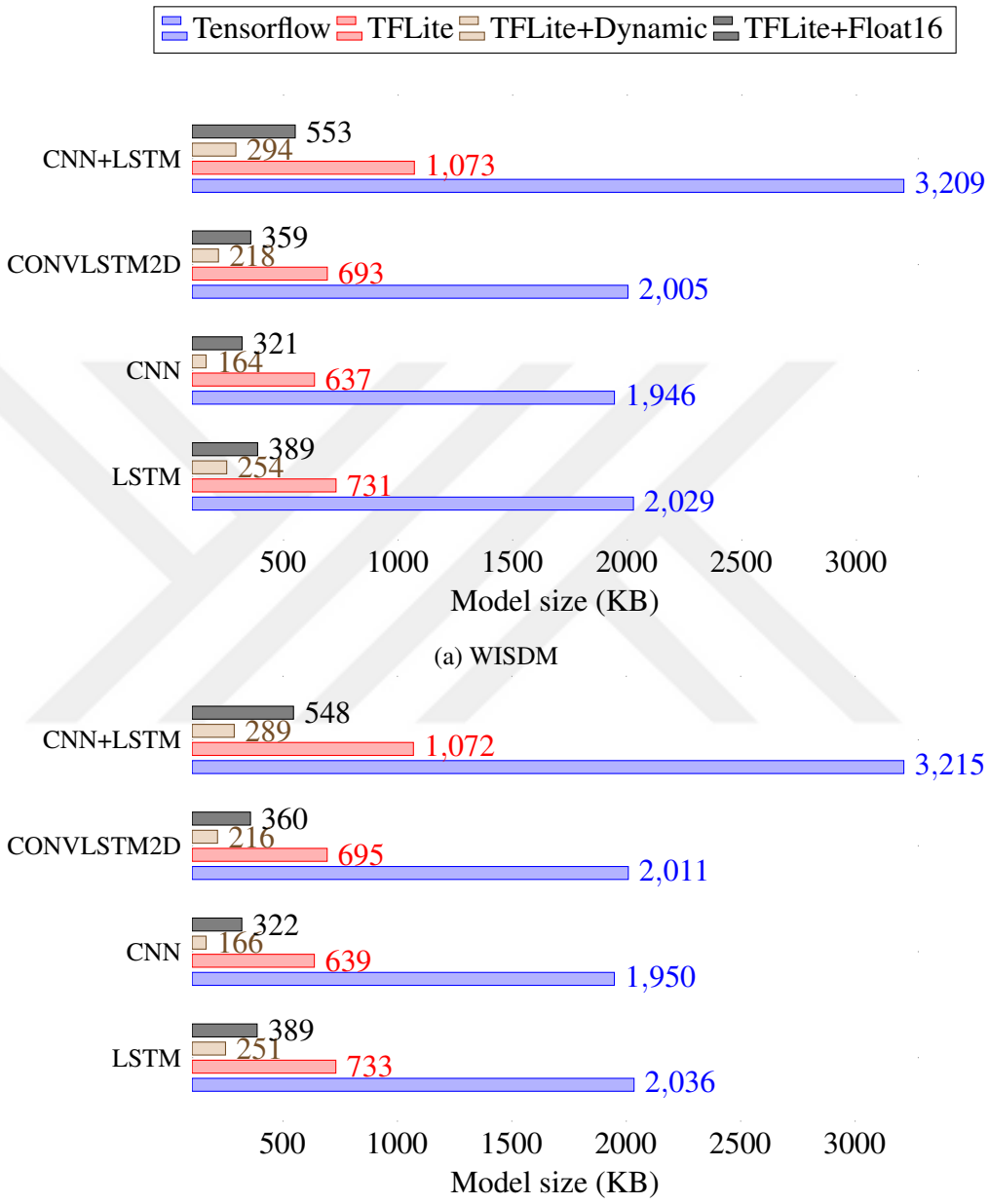
TABLE 3.5 – Memory and energy usage for WISDM and MobiAct data

		WISDM		**MobiAct**	
		Memory	**Energy(mWh)**	**Memory**	**Energy(mWh)**
LSTM	**Tensorflow**	160M	23300	80 M	22600
	TFLite	0.3 M	2980	0.4 M	3110
	TFLite + Dynamic	2.2 M	5100	2 M	4750
	TFLite + Float16	0.6 M	4320	60K	2540
CNN	**Tensorflow**	120M	15400	140 M	18500
	TFLite	4 K	24	4 K	29
	TFLite + Dynamic	16 K	580	16 K	880
	TFLie + Float16	4 K	21	4 K	25
CONVLSTM2D	**Tensorflow**	220 M	13300	100 M	15200
	TFLite	344 K	108	47 M	250
	TFLite + Dynamic	60 K	390	80 M	7800
	TFLite + Float16	52 K	500	64 K	490
CNN + LSTM	**Tensorflow**	120 M	17100	120 M	14600
	TFLite	12 K	160	4 K	100
	TFLite + Dynamic	392 K	3100	56 K	2000
	TFLite + Float16	32 K	130	12 K	220

behaviour for all models. Among the Tensorflow Lite models, for all three metrics, dynamic range quantization consumes the highest resources compared to the other two Tensorflow Lite models. For the WISDM dataset, which has a smaller size, the energy consumption of Tensorflow Lite with the LSTM model is approximately 124 times higher than the one with the CNN model. The exact ratio is about 107 for the MobiAct dataset. Tensorflow Lite with Float16 quantization gives the shortest CPU usage for the WISDM dataset, whereas Tensorflow Lite gives the shortest CPU usage for the MobiAct dataset.

For the WISDM dataset, Tensorflow Lite with dynamic range quantization reduces the

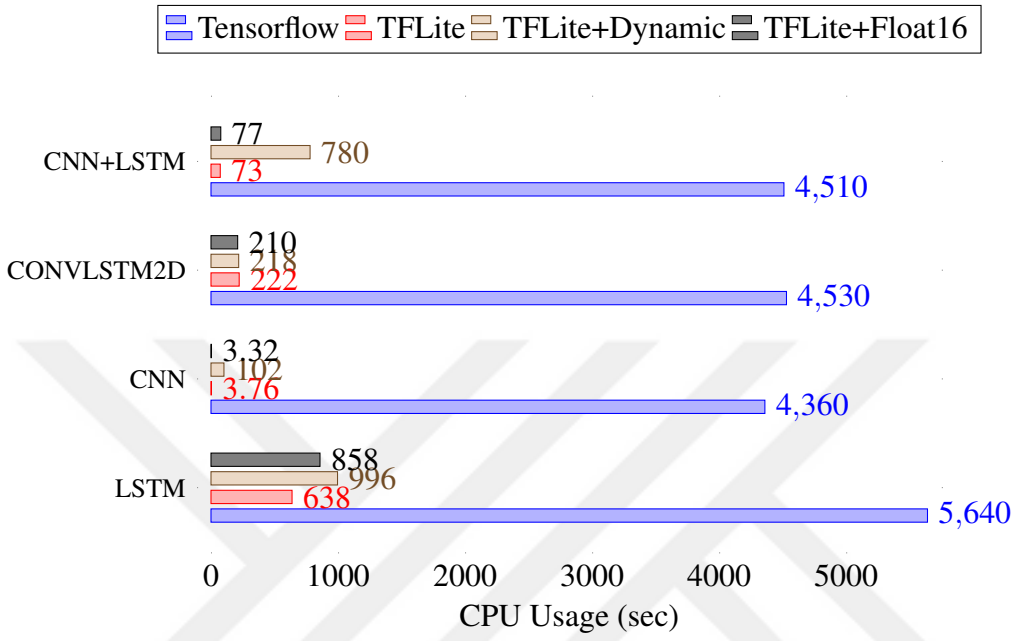
memory usage by approximately five times compared to Tensorflow Lite. However, the opposite case is noticed for the MobiAct dataset.



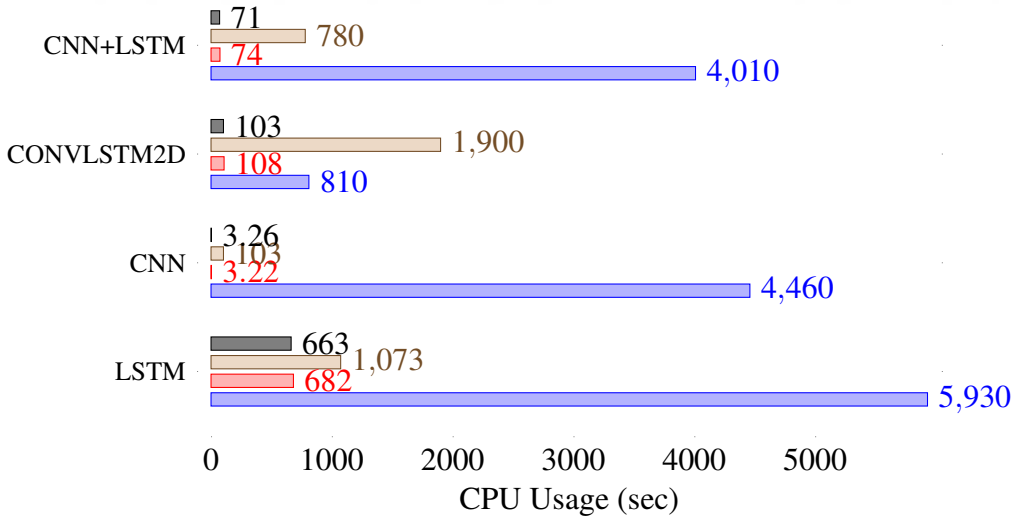
(a) WISDM

(b) MobiAct

FIGURE 3.6 – Model sizes for WISDM and MobiAct data



(a) WISDM



(b) MobiAct

FIGURE 3.7 – CPU usage for WISDM and MobiAct data

4 TRANSFER LEARNING

We use transfer learning to personalize deep learning models built for HAR tasks using MobiAct (*MobiAct*, n.d.) and OpenHAR (Siirtola et al., 2018) datasets. We analyzed the performance of deep learning models according to the accuracy rate and running time. The impact of unfreezing the last layer or other layers is investigated. The impact of the amount of user training data on the performance of transfer learning is analyzed. Transfer learning performance is examined with the new dataset containing new classes not available in the base model. And the models created with transfer learning were transformed with Tensorflow Lite and the performances of these models are also examined.

4.1 Transfer Learning and Transfer of Models to Mobile Devices

In Sections 2.3 and 2.4.3, we explained various methods for transfer learning. One of the most prevalent methods for upgrading a previously learned model is transfer learning. Transfer learning is a machine learning technique in which a model created for one task is reused for another second task.

Three issues are important in the transfer learning process: what to transfer, when to transfer and how to transfer. What part of the knowledge can be transferred between domains or activities is referred to as "what to transfer." Situations in which the data will be transferred are referred to as "when to transfer." Deciding which methods will be used is referred to as "how to transfer." Transfer learning techniques can be categorized according to domain and tasks. It can further be categorized according to whether the domains and tasks for source and target are the same, or if they are different but still related (Pan and Yang, 2010).

One of the reasons for using transfer learning instead of training a model from scratch is the faster training time. A lot of information can be used using the weights of the pre-trained models. With fine-tuning, the new model is trained faster. Another reason that smaller

datasets can be used. A large-scale dataset is needed to train a model from scratch. Better performance can be achieved with transfer learning. Better results can be achieved by adding new fully connected layers to the pre-trained models.

As mentioned in Chapter 1, we also concentrate on offline (Fig 1.1b) and end-to-end (Fig 1.1c) on-device learning methodologies. In offline learning, a general model is frequently trained with cloud data or, in some situations, simply with the data of the user. In the first scenario, a general model trained on data from other users may not perform well for some individuals who have a wide range of features. We require *personalized models* (Zhou et al., 2021) in such applications as human activity detection, where a general model can be fine-tuned/adapted or re-trained on the device with personal data to give tailored services and improve user experience (Zhang et al., 2020). When only the user's data is used to train the model, there is typically a limited quantity of training data available, particularly labeled data, and training DNNs with small data sets might lead to overfitting. In both cases, transfer learning (Pan and Yang, 2010) might be used to fine-tune a model learned on a bigger dataset for that particular user and, if possible, on the device (Buffelli and Vandin, 2021; Mairittha et al., 2021a).

4.2 Datasets and Methods

For transfer learning, MobiAct (explained in Section 3.1) and OpenHAR (Siirtola et al., 2018) datasets were used. We also experimented with the WISDM dataset presented in Section 3.1. But since it is a smaller dataset, we do not proide the results. OpenHAR is a free Matlab toolbox that allows to combine multiple open datasets. This dataset contains ten publicly available datasets on human activities. There are accelerometer signals. Sensor position is also included in the data. All datasets are provided in the same format. Sampling frequency is 10Hz. Over 65 million data samples are available in OpenHAR. These data is collected 211 participants who participated in 17 distinct everyday human activities in 14 different body positions. The distribution of the activities is shown in Figure 4.1.

Transfer learning takes features learned in a problem and uses them in a new similar problem. Usually, transfer learning is done when the dataset is too small to train from scratch. The most common workflow of transfer learning; 1) Layers are taken from a pre-trained model

2) These layers are frozen in order not to destroy the information they contain in the new training 3) New layers are added after the frozen layers 4) New layers are trained with old features and new dataset. Transfer learning models were created with this workflow.

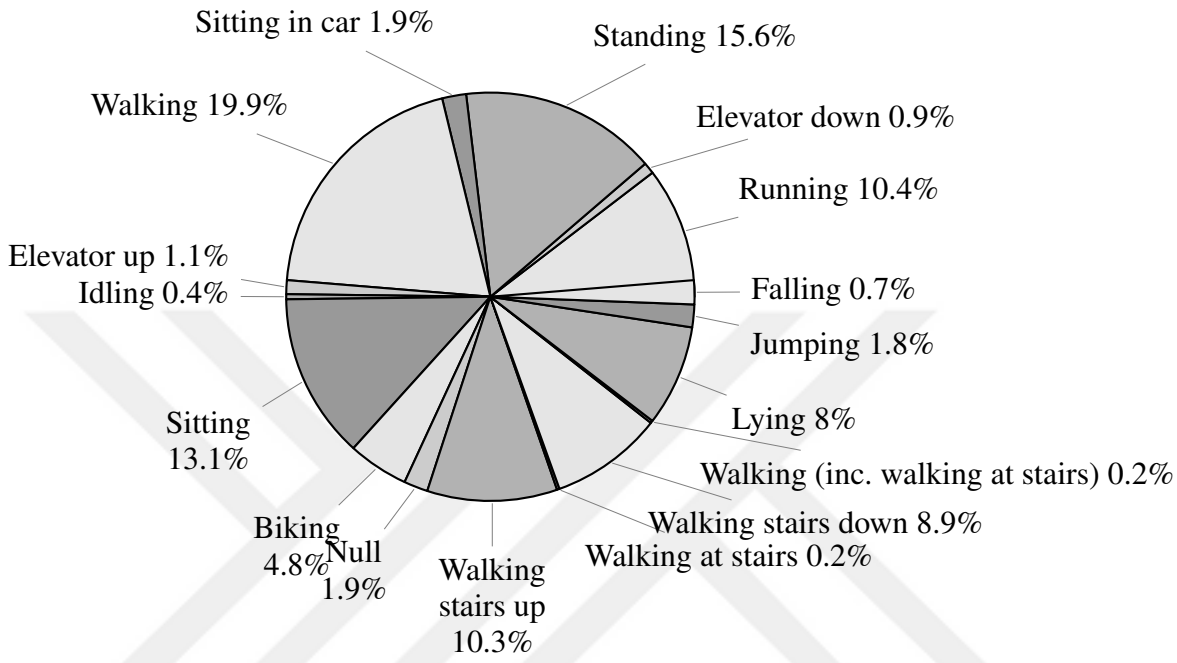


FIGURE 4.1 – Distribution of activities in OpenHAR (Siirtola et al., 2018) for all ten datasets

In Layer freezing, the weights of the layers of the trained model do not change when reused in transfer learning. Gradient calculation and backpropagation are disabled for the weights of these layers. Frozen layers will not be updated in backpropagation. Freezing the initial layers of the pre-trained model is important for learning the basic features of the base model. All learning in the base model will be lost if the first layers are not frozen. Fine-tuning to improve performance, unfreezing parts of the base model and retraining with lower learning rates can be performed in transfer learning. The last few layers of the pre-trained base model can be unfrozen for fine-tuning. Unfrozen layers will be trained with the new dataset, and the weights will be updated. With layer freezing, speed increase and less memory usage are provided during training.

The architectural structures used, the layers frozen in the base model, test rates, and performance evaluations are presented in each subsection in Section 4.3.

4.3 Performance Evaluation

Transfer learning experiments were carried out with 4 different deep learning models used in Section 3.2. In this section, we present the accuracy results of the base, transfer learning and user-specific models. All accuracy values contain weighted results based on the size of each test user. For comparison purposes; base and user-specific models were also created in addition to the transferred model. The base model is the model that includes the layer weights to be used in transfer learning. Base models are the same as the architectures described in Section 3.2. Base models were created with users other than test users. The base model in the accuracy values shown in the figures shows the accuracy values obtained from the base model with the test data of the test user. A user-specific model was created from scratch using only the split training data of the test user to be used in the transfer learning model. User-specific models are the same as the architectures described in Section 3.2. The user-specific model shown in the figures shows the average values obtained from this model using the remaining test data of the test user. The transferred learning model shown in the figures shows the average accuracy values obtained from the transfer learning model using the test data of the test user. In Section 4.3.1, we present the impact of unfreezing the last layer or other layers and the impact of added new layer architectures. In Section 4.3.2, we present the impact of the amount of transferred user training data. In Section 4.3.3, we evaluate transfer learning to new datasets.

Tests were made in transfer learning in two different ways using leave-one-subject-out method. Firstly, a single user was selected as the test in each of our datasets. The first model was created with the remaining users. Later, using this first model, transfer learning was applied with a certain percentage of the selected test user's data. In this test, the last transfer learning model was tested with the remaining data of the user. Secondly; the first model was created with a complete data set. A user of another data set was used in transfer learning. In these two different datasets, there are common human activities as well as different human activities. Again, the transfer learning model was tested with the test user's data other than the training data used in transfer learning. A summary of the 3 experiments is presented in Table 4.1. The data, architecture and period information used for the 3 experiments are presented. The y-axis in Figures 4.2 to 4.8 present the accuracy values in percentages.

TABLE 4.1 – Transfer Learning for Experiments 1-2-3

		Experiment 1	Experiment 2	Experiment 3
Data		MobiAct		
Base Model	**Layers**	4 different deep learning models presented in Section 3.2 are used with the same architecture.		
	Epochs	20		
	Data	The base model was created with all the remaining users except the user to be tested each time in the relevant data set.		
Transfer Learning	**Layers**	The layers were frozen until the last layer in the base model. For the transfer learning model; Dense layer was added to the base model.	The layers were frozen until the last two layers in the base model. For the transfer learning model; respectively Flatten, Dense, Dropout, and Dense layers were added to the base model.	The layers were frozen until the last two layers in the base model. For the transfer learning model; two times Flatten, Dense, Activation, BatchNormalization, and Dropout layers were added to the base model.
	Epochs	50		
	Data	40% as a train and 60% as a test of the data of the single user to be tested was used.		
User-specific Model	**Layers**	4 different deep learning models presented in Section 3.2 are used with the same architecture.		
	Epochs	20		
	Data	40% as a train and 60% as a test of the data of the single user to be tested was used.		
Test		Test data is tested in the basic model, the transfer learning model, and the user-specific model.		

4.3.1 Impact of Unfreezing the Layer vs Other Layers

In this section, 4 different deep learning models introduced in Section 3.2 were tested with 3 different experiments with the MobiAct dataset. The impact of unfreezing the last layer or other layers and the impact of added new layer architectures are presented in Figures 4.2, 4.3, 4.4, 4.5. For each deep learning model, 3 types of accuracy results are presented. In these experiments, only MobiAct data is used and the average results considering all participants is presented.

In Figure 4.2, we present the results of Experiment 1 where before adding new layers for transfer learning, the layers were frozen until the last layer in the base model. The last layer in the base model was unfrozen. Unfrozen layers were trained with the new dataset and the weights of this layer were updated. The Dense layer, including Softmax activation, was added to the base model by giving the number of activities for the transfer learning model. The

average accuracy of transfer learning is higher than the base model LSTM. LSTM's average accuracy rates are 79.20 in the base model, 82.87 in the transferred model and 88.63 in the user-specific model. In other deep learning architectures, the base models or use-specific models have achieved higher accuracy rates according to transferred models.

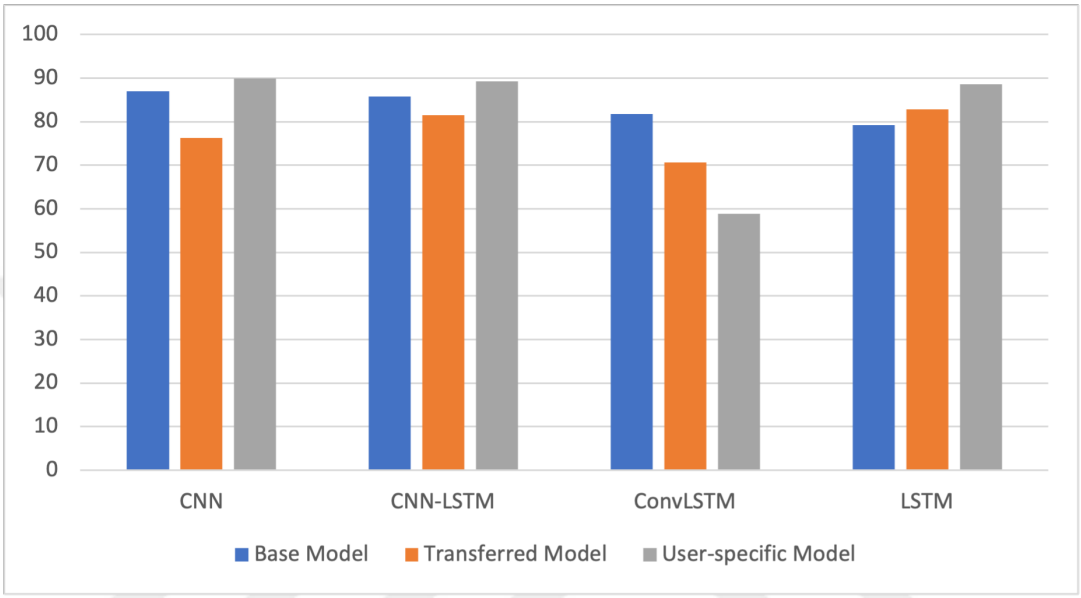


FIGURE 4.2 – Impact of Unfreezing The Last Layer in Experiment 1

In Figure 4.3, we are interested in observing the impact of retraining more layers in transfer learning. Before adding new layers for transfer learning, the layers were frozen until the last two layers in the base model. The two last layers in the base model were unfrozen. For the transfer learning model; respectively Flatten, Dense, Dropout, and Dense layers were added to the base model. The average accuracy of transfer learning for all models exceeded the accuracy of base models. For the LSTM model only, the accuracy of the user-specific model exceeded the accuracy of the transferred model. The highest accuracy rates is achieved for the transferred model in CNN-LSTM. CNN-LSTM's average accuracy rates are 94.83 in the transferred model, 85.76 in the base model, and 89.60 in the user-specific model.

In Figure 4.4, we see the results before adding new layers for transfer learning, the layers were frozen until the last two layers in the base model. The two last layers in the base model were unfrozen. For the transfer learning model; respectively two times Flatten, Dense, Activation, BatchNormalization, and Dropout layers were added to the base model. Finally,

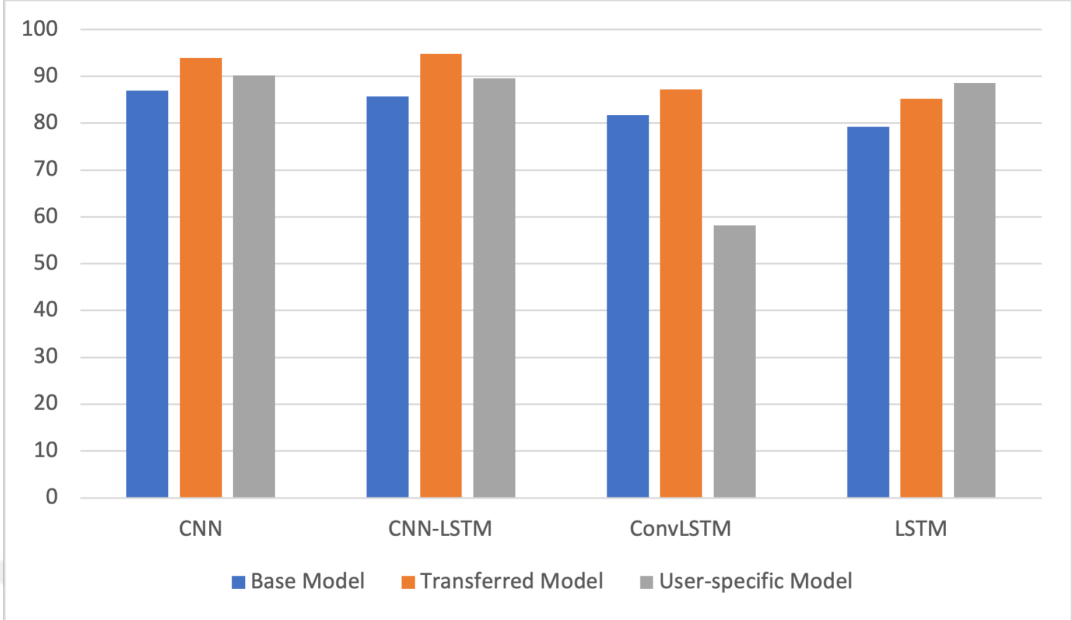


FIGURE 4.3 – Impact of Unfreezing The Two Layers and Added New Layers Architecture in Experiment 2

the Dense layer was added again. The average accuracy of transfer learning for all models exceeded the accuracy of base models. And also, the average accuracy of transfer learning for all models exceeded the accuracy of user-specific models. The highest accuracy rates is achieved for the transferred model in CNN-LSTM. CNN-LSTM's average accuracy rates are 93.47 in the transferred model, 85.38 in the base model, and 89.01 in the user-specific model.

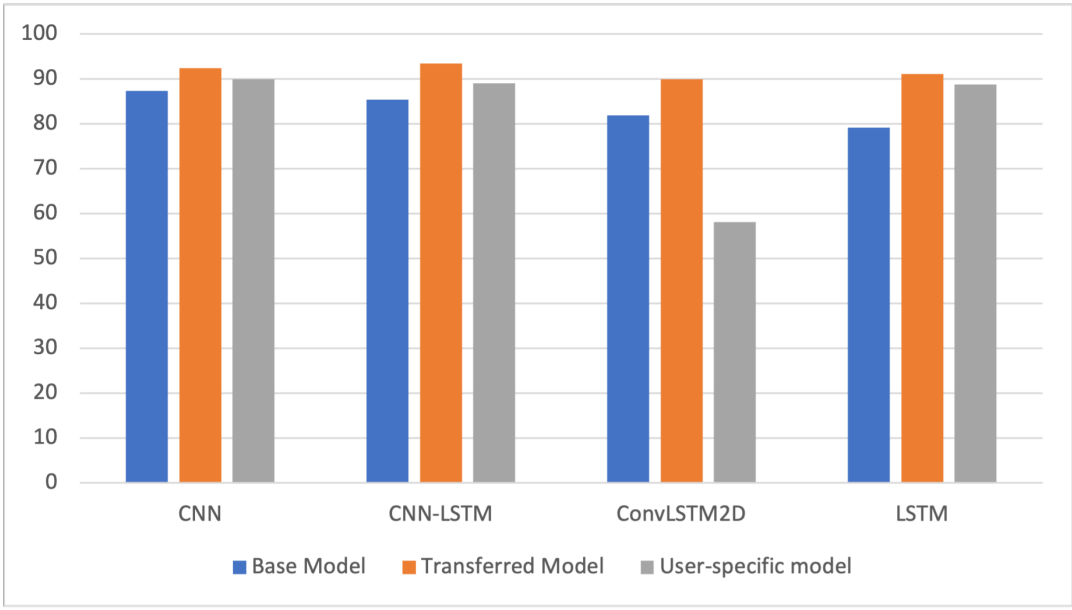


FIGURE 4.4 – Experiment 3, Impact of Unfreezing The Two Layers and Adding More New Layers Architecture than in Experiment 2

In Figure 4.5, comparison of three different transfer learning models is presented for each model.

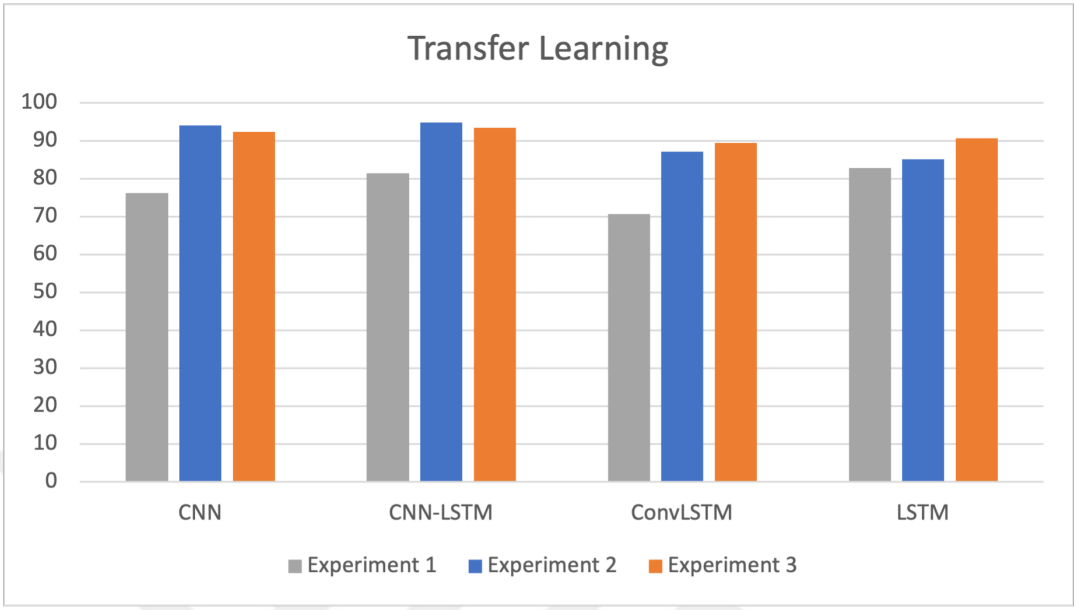


FIGURE 4.5 – Accuracy of Models with Transfer Learning

In Table 4.2, training times in seconds are presented for the base model and 3 different transfer learning experiments as an average for a single user. Transfer learning takes considerably less time than base model training. In these experiments, transfer learning training does not exceed 1 minute. The fact that transfer learning models take a much shorter time than these base models will facilitate the model update process.

TABLE 4.2 – Transfer Learning Training Time for Experiments 1-2-3 in Seconds

	Base Model	**Experiment 1 Transfer Learning**	**Experiment 2 Transfer Learning**	**Experiment 3 Transfer Learning**
CNN	882s	14s	12s	39s
CNN-LSTM	1968s	53s	44s	48s
ConvLSTM2D	695s	27s	21s	46s
LSTM	647s	46s	46s	37s

4.3.2 Impact of Amount of Transferred User Training Data

This section includes comparing the results obtained with the different training data ratios of the user to be tested in the transfer learning and user-specific models. In Figure 4.6, there are results for 4 different deep learning models. 3 different training data rates were used for each user; 20%, 40%, 80% of user's data is used in the training. To see the results in more detail, the results of the CNN-LSTM model are also shared in Figure 4.7. As the ratio of training data received from the user increases, both user-specific and transfer learning results increase. However, accuracy rates are above 90% in transfer learning results and better results are achieved compared to user-specific models, even if training data as little as 20% is used. Although we have little data from the user, better results can be obtained in transfer learning.

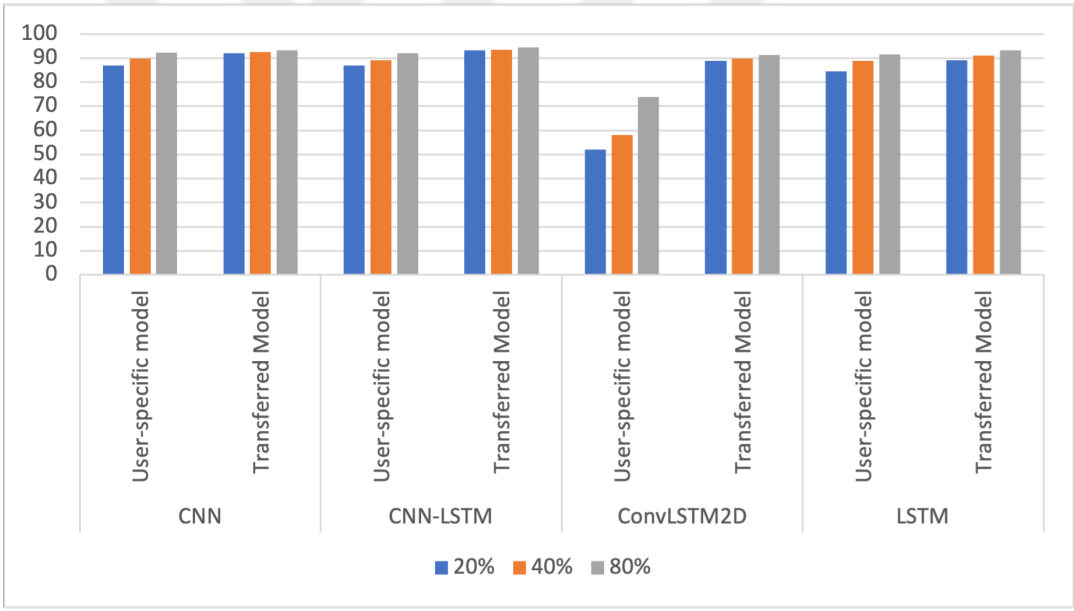


FIGURE 4.6 – Impact of Amount of Transferred User Training Data for the DL Models

4.3.3 Transfer to New Datasets

In transfer learning, an evaluation was performed by using a data set other than the data set used in the base model in these experiments. All MobiAct dataset was used in the base model and OpenHAR dataset was used in the transfer learning. One of the 10 datasets available in OpenHAR is the MobiAct data. In these experiments, we used the Mobiact version available in OpenHAR since the dataset was modified to have the same sampling

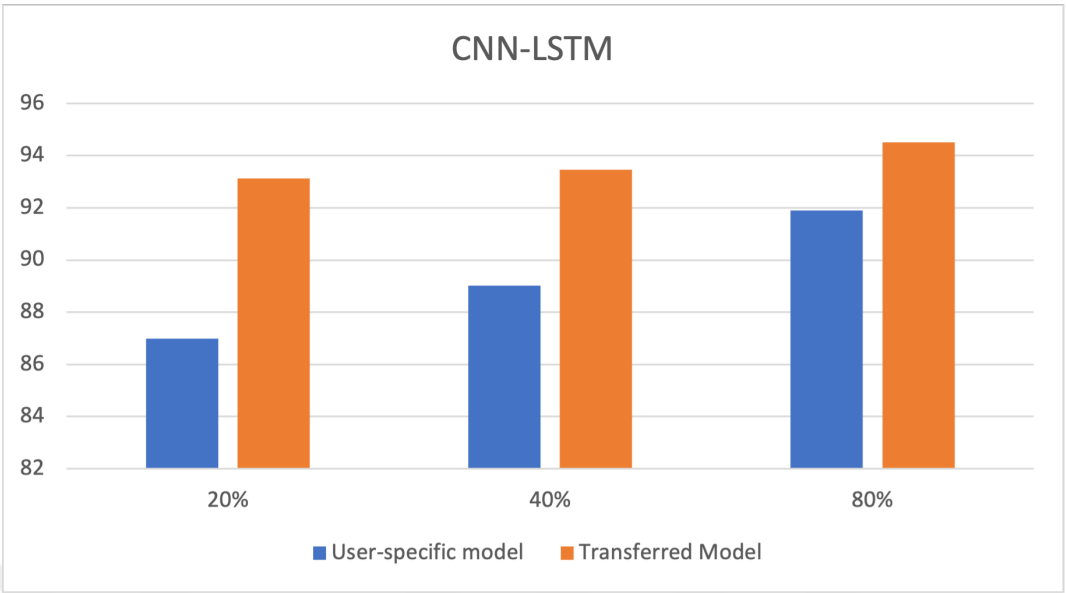


FIGURE 4.7 – Impact of Amount of Transferred User Training Data for CNN-LSTM Model

frequency in all datasets. Each data set was evaluated separately in transfer learning. The data was collected obtained from different positions. Some activities are not available in the MobiAct but are in other datasets. Since these are not in the base model, we define them as a new activity. It was desired to evaluate what kind of performance is achieved when new activities/classes are added with transfer learning. As some of the activities are very similar to each other, we saw in the predictions that they are confused with each other. The results were also evaluated by grouping the activities in the data set as another experiment. For example, there were 5 different activities related to walking; these are walking, walking (inc. walking at stairs), walking stairs up, walking stairs down and walking at stairs (inc. up and down). We re-labeled the activities as the only walking activity in the grouped data. Experiment 3, mentioned in Table 4.1, was used as both the base model and the transfer learning architecture in the experiment. Except for the data set used, the only difference was that 20% of the tested user data was used in training with transfer learning.

In Figure 4.8, user-specific and transfer learning results for the 9 data sets are shown for the default labels and merged labels. In 3rd, 4th and 8th data sets, transfer learning results achieved higher accuracy with both default labels and merged labels compared to user-specific models. Especially with the merged labels in the 8th dataset, the accuracy rate increased from 64% in transfer learning to 95%. In these data sets, "Biking" in transfer learning is the new label which was not in the MobiAct dataset. In the 8th dataset, there

are "Elevator up" and "Elevator down" labels as extra new labels. In the 6th dataset, transfer learning achieved higher accuracy than the user-specific model when only the labels are merged. In this dataset, there are "Walking stairs up" and "Walking stairs down","Elevator up" and "Elevator down" classes. Since they are very similar to each other, positive results were obtained in merging them. We should note that "Elevator" classes are used as a new label in transfer learning. In the 2nd data set, both user-specific and transfer learning accuracy increased with merged labels, and similar results were obtained with an 83% accuracy rate. In the 7th data set, a higher accuracy rate was obtained in transfer learning with default labels compared to user-specific. With merged labels, both user-specific and transfer learning accuracy have increased, and user-specific accuracy rates are higher than transfer learning. In the 1st, 5th and 9th data sets, transfer learning could not achieve a better accuracy rate compared to both label types. We think that MobiAct dataset was collected from the "Hip (inc. belt and waist)" position. The positions of the 1st dataset are "Ankle", "Chest" and "Wrist, any (inc. forearm)". These are quite different from the MobiAct default position. When MobiAct is the base model, we interpret that the data collected from different positions cannot be classified successfully when used in transfer learning, due to this reason. The positions in the 5th dataset are "Hip (inc. belt and waist)" and "Wrist, any (inc. forearm)". Here, there is a different position than the position in the MobiAct with the "Wrist" position. In the 9th data set, there are data collected from quite different positions. These are "Hip (inc. belt and waist)", "Trouser's pocket, any (inc. thigh)", "Chest", "Wrist, any (inc. forearm)", "Upper arm", "Head" and "Shin (inc. leg)". If similar positions and classes that are not confusing can be used in the base and transfer learning models, transfer learning can learn them and performs better than user-specific models, even if new labels are added.

4.3.4 Transfer Learning and Tensorflow Lite

After creating models with transfer learning, these models are transformed with Tensorflow Lite, which was used to optimize the deep learning models for mobile devices as we explained in Section 3.4. After the models were optimized with Tensorflow Lite, accuracy and running time values were measured and evaluated. As in Section 3.6, we present the results of models after transforming with TensorFlow Lite and the results of models created with dynamic range quantization and Float16 quantization for post-training quantization. 20% of each tested user's data was used for training. In Figure 4.9, for the CNN-LSTM

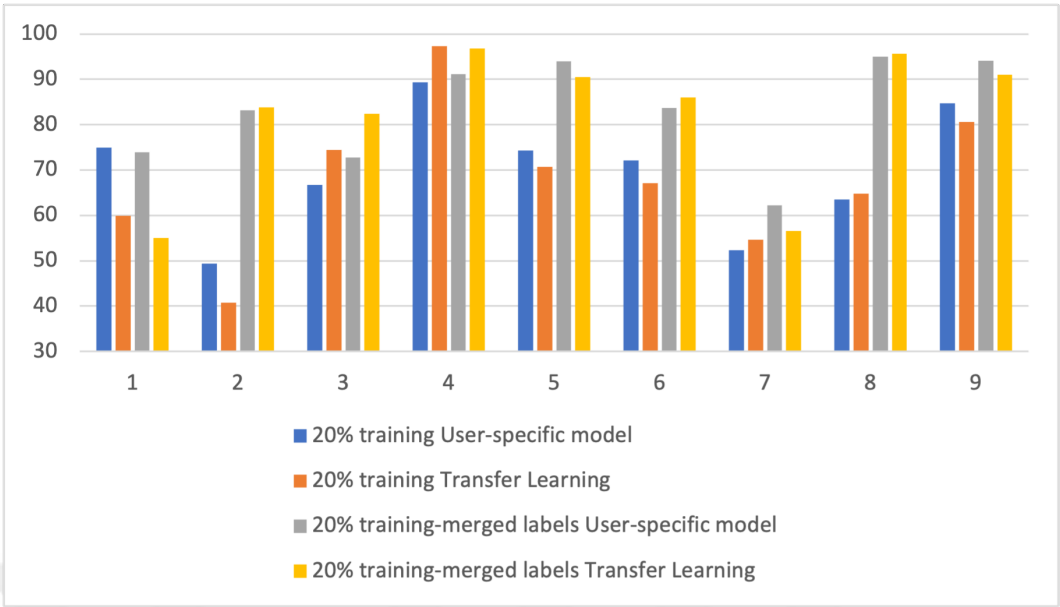


FIGURE 4.8 – User-specific and transfer learning results for 9 data sets with default and merged labels

model, the transfer learning accuracy values and the accuracy values obtained after the transfer learning model is translated into 3 different versions of TensorFlow Lite are presented. A success rate of 93% was achieved in all experiments. The decimal numbers of success rates vary. The results make little difference, the most different being the Tensorflow Lite + Dynamic model. Almost no compromise in accuracy is observed with Tensorflow Lite models.

In Table 4.3, the average time spent by TensorFlow and TensorFlow Lite versions obtained from the CNN-LSTM transfer learning model for a single prediction is presented in milliseconds. To predict a single value, the prediction time running for all users was averaged. In predictions, TensorFlow lite models take much less time than the Tensorflow model. When comparing Tensorflow and Tensorflow Lite + Float16, there is almost a 190 times difference. When both transfer learning and TensorFlow Lite are used, both the objectives of resource utilization and personalized model are obtained.

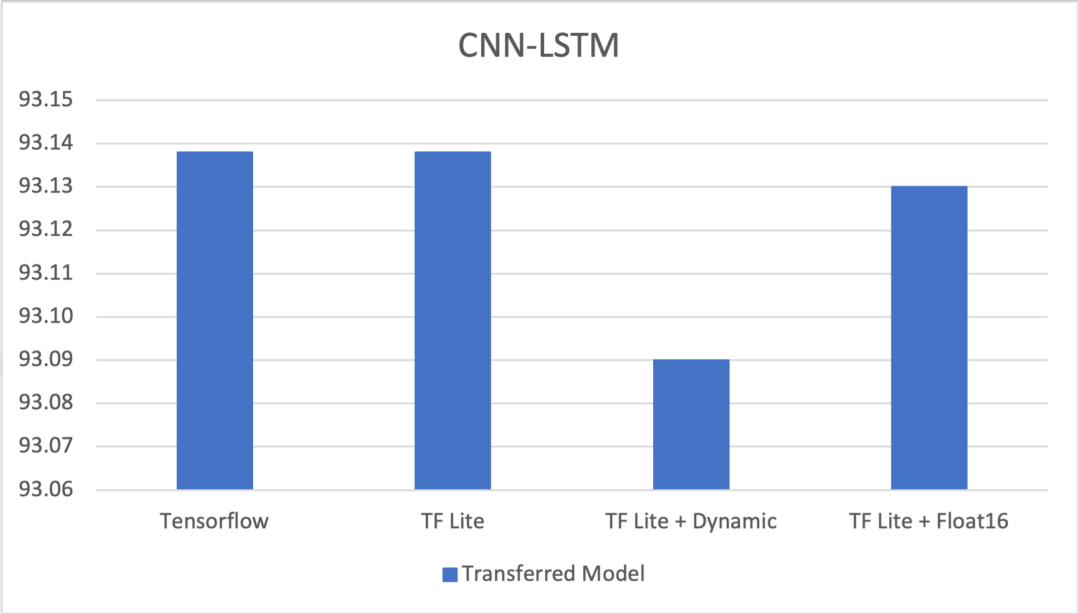


FIGURE 4.9 – Tensorflow and Tensorflow Lite Accuracies for CNN-LSTM Transfer Learning

TABLE 4.3 – Tensorflow and Tensorflow Lite Prediction Time for CNN-LSTM Transfer Learning

	Tensorflow	**TF Lite**	**TF Lite + Dynamic**	**TF Lite + Float16**
CNN-LSTM	379ms	6ms	3ms	2ms

5 CONCLUSION

The current state-of-the-art works on deep learning on mobile and wearable computing devices are examined as a comparison study. Different types of deep learning optimization approaches for mobile and wearable devices were applied on three datasets, then the results were compared. Meanwhile, the architectures and technical infrastructures of the developed platforms were explored.

In the on-device activity recognition experiments, an average of 97% accuracy was achieved with the WISDM data and an average of 92% accuracy with the MobiAct data. No significant changes in accuracy were observed when Tensorflow models were converted to TensorFlow Lite models. As a result of the experiments, it was observed that the model sizes were significantly reduced when optimized with Tensorflow Lite without sacrificing the accuracy of the models, particularly when quantization methods are used for optimizing the models. Similarly, we observe a significant reduction in resource consumption when the models are optimized compared to the original models. These results show that the optimized models can be ported to mobile and even wearable devices for real time activity recognition.

Higher accuracy rates were achieved with transfer learning models compared to the base or user-specific models. The accuracy rate can be increased with transfer learning. We have very little data about the user, better results can be obtained with an accuracy rate of over 90% in transfer learning. Even if we have a small amount of data from a test user, transfer learning performs well and this relieves the necessity of collecting a larger dataset from a user. Although there are new classes when updating the model with transfer learning, the model can learn and recognize them with good accuracy levels. New classes not in the base model can be integrated into transfer learning without changing the base model. When both transfer learning and TensorFlow Lite are used, both the efficiency of using resources and the accuracy of the personalized model can be improved. When Tensorflow Lite is applied after transfer learning, significant improvements are obtained mainly in the running time of prediction. It can provide about 190 times improvement.

These results show that the optimized and personalized models can be ported to mobile and even wearable devices for real-time activity recognition. Accordingly, we plan to perform the same experiments on a mobile device in future work.

REFERENCES

Agarwal, P. and Alam, M. (2020). A lightweight deep learning model for human activity recognition on edge devices, *Procedia Computer Science* **167**: 2364–2373.

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M. and Farhan, L. (2021). Review of deep learning: concepts, cnn architectures, challenges, applications, future directions, *Journal of Big Data* **8**.

Banbury, C. R., Reddi, V. J., Lam, M., Fu, W., Fazel, A., Holleman, J., Huang, X., Hurtado, R., Kanter, D., Lokhmotov, A. et al. (2020). Benchmarking tinyml systems: Challenges and direction, *arXiv preprint arXiv:2003.04821* .

Benmeziane, H., Maghraoui, K. E., Ouarnoughi, H., Niar, S., Wistuba, M. and Wang, N. (2021). A comprehensive survey on hardware-aware neural architecture search.

Buffelli, D. and Vandin, F. (2021). Attention-based deep learning framework for human activity recognition with user adaptation, *IEEE Sensors Journal* .

Cai, E., Juan, D.-C., Stamoulis, D. and Marculescu, D. (2017). Neuralpower: Predict and deploy energy-efficient convolutional neural networks, *Asian Conference on Machine Learning*, PMLR, pp. 622–637.

Chauhan, J., Kwon, Y. D., Hui, P. and Mascolo, C. (2020). Contauth: continual learning framework for behavioral-based user authentication, *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* **4**(4): 1–23.

Chen, C., Zhang, P., Zhang, H., Dai, J., Yi, Y., Zhang, H. and Zhang, Y. (2020). Deep learning on computational-resource-limited platforms: A survey, *Mobile Information Systems* **2020**: 1–19.

Chen, Y., Zheng, B., Zhang, Z., Wang, Q., Shen, C. and Zhang, Q. (2020). Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions, *ACM Computing Surveys (CSUR)* **53**(4): 1–37.

Chen, Z. and Liu, B. (2018). Lifelong machine learning, *Synthesis Lectures on Artificial Intelligence and Machine Learning* **12**(3): 1–207.

Dai, X., Spasić, I., Chapman, S. and Meyer, B. (2020). The state of the art in implementing machine learning for mobile apps: A survey, *2020 SoutheastCon*, pp. 1–8.

Darvish Rouhani, B., Mirhoseini, A. and Koushanfar, F. (2017). Tinydl: Just-in-time deep learning solution for constrained embedded systems, *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4.

Deng, Y. (2019). Deep learning on mobile devices-a review.

Dhar, S., Guo, J., Liu, J., Tripathi, S., Kurup, U. and Shah, M. (2019). On-device machine learning: An algorithms and learning theory perspective, *arXiv preprint arXiv:1911.00623* .

García-Martín, E., Rodrigues, C. F., Riley, G. and Grahn, H. (2019). Estimation of energy consumption in machine learning, *Journal of Parallel and Distributed Computing* **134**: 75–88.

Gou, J., Yu, B., Maybank, S. and Tao, D. (2021). Knowledge distillation: A survey, *International Journal of Computer Vision* **129**.

Grzymkowski, and Stefański, T. P. (2020). Performance analysis of convolutional neural networks on embedded systems, *2020 27th International Conference on Mixed Design of Integrated Circuits and System (MIXDES)*, pp. 266–271.

Gu, H., Wang, Y., Hong, S. and Gui, G. (2019). Blind channel identification aided generalized automatic modulation recognition based on deep learning, *IEEE Access* **7**: 110722–110729.

Hansen, R. (2018). *Metadata state and history service for datasets. enable extracting, storing and access to metadata about a dataset over time.*, Master's thesis, UiT Norges arktiske universitet.

Ignatov, A. (2018). Real-time human activity recognition from accelerometer data using convolutional neural networks, *Applied Soft Computing* **62**: 915–922.
URL: *https://www.sciencedirect.com/science/article/pii/S1568494617305665*

Jordao, A., Kloss, R. and Schwartz, W. R. (2018). Latent hypernet: Exploring the layers of convolutional neural networks, *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7.

Kwon, Y. D., Chauhan, J., Kumar, A., Hui, P. and Mascolo, C. (2021). Exploring system performance of continual learning for mobile and embedded sensing applications.

Lane, N. D., Bhattacharya, S., Georgiev, P., Forlivesi, C. and Kawsar, F. (2015). An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices, *Proceedings of the 2015 international workshop on internet of things towards applications*, pp. 7–12.

Liang, P. P., Liu, T., Ziyin, L., Allen, N. B., Auerbach, R. P., Brent, D., Salakhutdinov, R. and Morency, L.-P. (2020). Think locally, act globally: Federated learning with local and global representations.

Ma, X., Yao, T., Hu, M., Dong, Y., Liu, W., Wang, F. and Liu, J. (2019). A survey on deep learning empowered iot applications, *IEEE Access* **7**: 181721–181732.

Mairittha, N., Mairittha, T. and Inoue, S. (2021a). On-device deep personalization for robust activity data collection, *Sensors* **21**(1).

Mairittha, N., Mairittha, T. and Inoue, S. (2021b). On-device deep personalization for robust activity data collection, *Sensors* **21**(1).
URL: *https://www.mdpi.com/1424-8220/21/1/41*

Mathur, A., Zhang, T., Bhattacharya, S., Velickovic, P., Joffe, L., Lane, N. D., Kawsar, F. and Lio, P. (2018). Using deep data augmentation training to address software and hardware heterogeneities in wearable and smartphone sensing devices, *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 200–211.

MobiAct (n.d.). `https://bmi.hmu.gr/the-mobifall-and-mobiact-datasets-2/`. Accessed: 2022-03-30.

Palanisamy, K., Khimani, V., Moti, M. and Chatzopoulos, D. (2021). Spliteasy: A practical approach for training ml models on mobile devices.

Pan, S. J. and Yang, Q. (2010). A survey on transfer learning, *IEEE Transactions on Knowledge and Data Engineering* **22**(10): 1345–1359.

Peppas, K., Tsolakis, A. C., Krinidis, S. and Tzovaras, D. (2020). Real-time physical activity recognition on smart mobile devices using convolutional neural networks, *Applied Sciences* **10**(23).
URL: *https://www.mdpi.com/2076-3417/10/23/8482*

Radu, V., Tong, C., Bhattacharya, S., Lane, N. D., Mascolo, C., Marina, M. K. and Kawsar, F. (2018). Multimodal deep learning for activity and context recognition, *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, Vol. 1, ACM New York, NY, USA, pp. 1–27.

Ravi, D., Wong, C., Lo, B. and Yang, G.-Z. (2016). A deep learning approach to on-node sensor data analytics for mobile or wearable devices, *IEEE journal of biomedical and health informatics* **21**(1): 56–64.

Ren, H., Anicic, D. and Runkler, T. (2021). Tinyol: Tinyml with online-learning on microcontrollers, *arXiv preprint arXiv:2103.08295* .

Rouhani, B., Mirhoseini, A. and Koushanfar, F. (2016). Delight: Adding energy dimension to deep neural networks, pp. 112–117.

Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E. and Ramabhadran, B. (2013). Low-rank matrix factorization for deep neural network training with high-dimensional output targets, *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6655–6659.

Siirtola, P., Koskimäki, H. and Röning, J. (2018). Openhar: A matlab toolbox for easy access to publicly open human activity data sets, *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, UbiComp '18, ACM, New York, NY, USA, pp. 1396–1403.
URL: *http://doi.acm.org/10.1145/3267305.3267503*

Singh, S., Pandey, S. K., Pawar, U. and Janghel, R. R. (2018). Classification of ecg arrhythmia using recurrent neural networks, *Procedia Computer Science* **132**: 1290–1297. International Conference on Computational Intelligence and Data Science.
URL: *https://www.sciencedirect.com/science/article/pii/S1877050918307774*

SMOTE (n.d.). `https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html`. Accessed: 2022-03-30.

Stamoulis, D., Chin, T.-W., Prakash, A. K., Fang, H., Sajja, S., Bognar, M. and Marculescu, D. (2018). Designing adaptive neural networks for energy-constrained image classification, *Proceedings of the International Conference on Computer-Aided Design*, pp. 1–8.

Tai, L. and Liu, M. (2016). Deep-learning in mobile robotics - from perception to control systems: A survey on why and why not.

Vepakomma, P., Gupta, O., Swedish, T. and Raskar, R. (2018). Split learning for health: Distributed deep learning without sharing raw patient data, *arXiv preprint arXiv:1812.00564* .

Verhelst, M. and Moons, B. (2017). Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to iot and edge devices, *IEEE Solid-State Circuits Magazine* **9**: 55–65.

Wagner, M., Llort, G., Mercadal, E., Giménez, J. and Labarta, J. (2017). Performance analysis of parallel python applications, *Procedia Computer Science* **108**: 2171–2179.

Wang, C., Xiao, Y., Gao, X., Li, L. and Wang, J. (2019). Close the gap between deep learning and mobile intelligence by incorporating training in the loop, *Proceedings of the 27th ACM International Conference on Multimedia*, pp. 1419–1427.

Wang, E., Davis, J. J., Zhao, R., Ng, H.-C., Niu, X., Luk, W., Cheung, P. Y. and Constantinides, G. A. (2019). Deep neural network approximation for custom hardware: Where we've been, where we're going, *ACM Computing Surveys (CSUR)* **52**(2): 1–39.

Wang, F., Zhang, M., Wang, X., Ma, X. and Liu, J. (2020). Deep learning for edge computing applications: A state-of-the-art survey, *IEEE Access* **8**: 58322–58336.

Wang, J., Chen, Y., Hao, S., Peng, X. and Hu, L. (2019). Deep learning for sensor-based activity recognition: A survey, *Pattern Recognition Letters* **119**: 3–11.

Windows Management Instrumentation (n.d.). `http://timgolden.me.uk/python/wmi/index.html`. Accessed: 2021-11-01.

WISDM: WIreless Sensor Data Mining (n.d.). `https://www.cis.fordham.edu/wisdm/dataset.php`. Accessed: 2022-03-30.

Xu, M., Liu, J., Liu, Y., Lin, F. X., Liu, Y. and Liu, X. (2019). A first look at deep learning apps on smartphones, *The World Wide Web Conference*, pp. 2125–2136.

Xu, M., Qian, F., Zhu, M., Huang, F., Pushp, S. and Liu, X. (2020). Deepwear: Adaptive local offloading for on-wearable deep learning, *IEEE Transactions on Mobile Computing* **19**(2): 314–330.

Yamashita, R., Nishio, M., Do, R. and Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology, *Insights into Imaging* **9**.

Yang, H., Zhu, Y. and Liu, J. (2018). End-to-end learning of energy-constrained deep neural networks, *ArXiv* **abs/1806.04321**.

Yao, S., Hu, S., Zhao, Y., Zhang, A. and Abdelzaher, T. (2017). Deepsense: A unified deep learning framework for time-series mobile sensing data processing, *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, p. 351–360. **URL:** *https://doi.org/10.1145/3038912.3052577*

Yao, S., Zhao, Y., Shao, H., Liu, S., Liu, D., Su, L. and Abdelzaher, T. (2018). Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices, *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pp. 278–291.

Zhang, M., Zhang, F., Lane, N. D., Shu, Y., Zeng, X., Fang, B., Yan, S. and Xu, H. (2020). Deep learning in the era of edge computing: Challenges and opportunities.

Zhou, Q., Qu, Z., Guo, S., Luo, B., Guo, J., Xu, Z. and Akerkar, R. (2021). On-device learning systems for edge intelligence: A software and hardware synergy perspective, *IEEE Internet of Things Journal* pp. 1–1.

BIOGRAPHICAL SKETCH

She studied high school in Umraniye Anatolian High School. She received her BSc. degree in Mathematical Engineering from Istanbul Technical University, in 2016. In 2019, she started the master of science in Computer Engineering from Galatasaray University. Her current research interests are human activity recognition, deep learning and wearable devices.

PUBLICATIONS

— S. O. Bursa, O. D. Incel and G. I. Alptekin, "Transforming Deep Learning Models for Resource-Efficient Activity Recognition on Mobile Devices," 2022 5th Conference on Cloud and Internet of Things (CIoT), 2022, pp. 83-89, doi: 10.1109/CIoT53061.2022.9766512.