2021 43rd Annual International Conference of the
IEEE Engineering in Medicine & Biology Society (EMBC)
Oct 31 - Nov 4, 2021. Virtual Conference

Design and optimization of a TensorFlow Lite deep learning neural network for human activity recognition on a smartphone

Salah Eddin Adi and Alexander J. Casson, *Senior Member, IEEE*

***Abstract*— Human Activity Recognition (HAR), using machine learning to identify times spent (for example) walking, sitting, and standing, is widely used in health and wellness wearable devices, in ambient assistant living devices, and in rehabilitation. In this paper, a stacked Long Short-Term Memory (LSTM) structure is designed for HAR to be implemented on a smartphone. The use of an *edge* device for the processing means that the raw collected data does not need to be passed to the cloud for processing, mitigating potential bandwidth, power consumption, and privacy concerns. Our offline prototype model achieves 92.8% classification accuracy when classifying 6 activities using a public dataset. Quantization techniques are shown to reduce the model's weight representations to achieve a >30x model size reduction for improved use on a smartphone. The end result is an on-phone HAR model with accuracy of 92.7% and a memory footprint of 27 KB.**

I. Introduction

Human Activity Recognition (HAR), using machine learning to identify times spent (for example) walking, sitting, and standing, is widely used in health and wellness wearable devices, in ambient assistant living devices, and in rehabilitation [1]. As a result, there have been many papers published in recent years aiming to perform offline HAR using accelerometry or similar signals collected from a smartwatch or smartphone [2]. In addition to improving the accuracy of HAR algorithms, there is an ongoing need to allow HAR to be implemented in the *edge* device itself, particularly on smartphones. Current offline data analysis models typically rely on passing all of the raw collected data to a PC or the cloud for processing there. While this allows large scale data collection and analyses to be performed, transferring all of the raw data requires bandwidth which may be prohibitive to all users, and moreover potentially raises security and privacy concerns [3]. In contrast edge processing in a local smartphone means that the raw data never needs to leave the physical locality of the user.

A wide number of *edge algorithms* are thus currently being investigated [4] and recently many *deep learning* approaches have been investigated for on-phone data classification. For example [5], [6] proposed deep Cellular Neural Networks (CNNs) for HAR using smartphone sensor data. In our previous 2019 work, [7], [8], we presented CNN and Long Short-Term Memory (LSTM) deep networks for implementing HAR on a smartphone. These used the TensorFlow library [9] and with optimizations for on-phone use achieved accuracies of 93.6% and 93.5% respectively in a six class classification problem, with a memory footprint of 2.1 MB and 4.1 MB respectively.

The authors are with the Department of Electrical and Electronic Engineering, The University of Manchester, UK. Email: `alex.casson@manchester.ac.uk`.

Although CNNs have displayed promising performances for HAR classification, they use the spatial dependencies of local data to extract features, rather than using the temporal correlations between sequences of data expected during different movements. In contrast, LSTMs exploit the temporal relations by retaining information about previous samples in time to make contextual decisions. Theoretically, LSTMs are better-suited for classifying HAR time-series data, where an activity is the result of multiple sequences of motion related to each other, rather than an isolated snapshot. However LSTMs are under-explored in the literature compared to CNN models for edge-device implementation. LSTMs have previously struggled with edge implementation due to network complexity, heavy computation burden, high memory footprint, and a lack of library support. For example, [10] deployed an LSTM for HAR, by using TensorFlow on Raspberry Pi class hardware. [8] also used the full TensorFlow library, on a smartphone target.

TensorFlow Lite is the reduced subset of TensorFlow targeting low power, low memory, phone and embedded microprocessor implementations. As part of TensorFlow development in 2020, support was added for LSTM models to be converted from TensorFlow to TensorFlow Lite for use on edge devices. This requires TensorFlow versions 2.3 and later [11]. This paper presents a TensorFlow Lite implementation of a stacked LSTM structure designed to allow HAR to be run on a smartphone with minimum memory impact. We show the significant improvements that are now possible with minimal impact on classification performance, and indeed the memory footprint is now within the range required to allow the network to fit in the very limited memory size of a typical embedded device.

The remainder of this paper is structured as follows. Section II introduces our LSTM structure, the dataset used, and the implementation and optimization for use in Android. Section III presents the performance results, showing the classification accuracy and memory footprint. Finally, Section IV discusses the results and draws conclusions.

II. Methods

A. Dataset

We make use of the publicly available UCI-HAR dataset [12] as it is widely used and allows easy comparison to many previously reported HAR works. The UCI-HAR dataset consists of data recorded from 30 volunteers using a smartphone's (Samsung Galaxy S II) embedded accelerometer and gyroscope at a sampling rate of 50 Hz. There are a total of 9 sensor channels consisting of triaxial total acceleration, triaxial linear acceleration (obtained from subtracting the gravitational component from the total acceleration), and triaxial angular velocity from the gyroscope. Each data entry for each channel is composed of a fixed sliding-window of 2.56 seconds or 128 successive samples, with a 50% overlap.

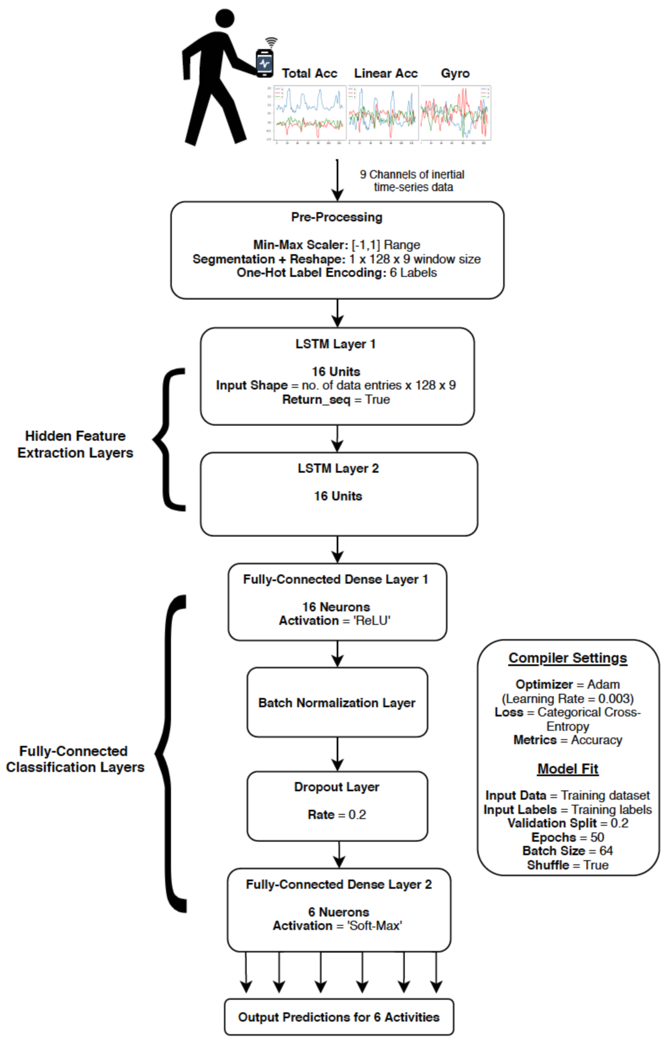
There are a total of 10,299 data entries for each sensor channel, which have already been split in the downloaded dataset into training and testing datasets, using a 70/30 train/test split. Each data entry has its own associated activity label from possible 6 activities, namely: *Walking*; *Walking upstairs*; *Walking downstairs*; *Sitting*; *Standing*; and *Lying*. The dataset also contains handcrafted features, but in this work we only make use of the accelerometer and gyroscope time series allowing the deep learning to generate its own feature representation of the signals.

B. *Offline prototype network design*

The design process for making an on-phone TensorFlow Lite model begins with making an offline prototype in TensorFlow which can run on a standard PC. We make use of a stacked LSTM structure as these have shown an increased robustness and performance for generalization and improved temporal pattern recognition [8], [10]. The LSTM model is designed, trained, and tested using TensorFlow 2, with our final network structure shown in Fig. 1. The stacked LSTM structure uses two hidden LSTM layers that are responsible for extracting temporal feature-maps from the input data. To ensure that the output of the first layer is reshaped into a 3D input to the second LSTM layer, the *return_sequence* parameter is set to *True*.

After the feature-map is extracted, it is passed into fully-connected dense layers, with batch normalization and dropout, for classification. Two dense layers are used as the feature-map should first be narrowed down into a larger classification set, before being passed to the final dense layer for activity classification. The first dense layer applies a ReLU activation function to add non-linearity to the feature-map. The second dense layer uses a soft-max activation function that calculates the probability of each of the six activity labels for the current time window. The LSTM is stateless, which means that the LSTM cell is reset after each batch of training data. A stateful LSTM would allow for variable batch sizes and more flexibility to manually reset the cell's state to retain data from previous batches. However, a stateful LSTM uses a lot of memory, is more complex, and is not supported for conversion using TensorFlow Lite at present. The network was training using the Adam optimizer with learning rate of 0.001 and a batch size of 64.

In Section III, in addition to using the prototype model of Fig. 1 to transfer to TensorFlow Lite, we also report the performance of using different numbers of stacked layers and different numbers of units per LSTM layer. For this optimization analysis, performance is reported for the training data set, and a 20% validation set split from this, with the test data held out for analysis only with the final model to avoid potential bias in the optimization. Classification performance is reported via the accuracy, defined as



Total Acc
Linear Acc
Gyro
9 Channels of inertial time-series data
Pre-Processing
Min-Max Scaler: [-1,1] Range
Segmentation + Reshape: 1 x 128 x 9 window size
One-Hot Label Encoding: 6 Labels
Hidden Feature Extraction Layers
LSTM Layer 1
16 Units
Input Shape = no. of data entries x 128 x 9
Return_seq = True
LSTM Layer 2
16 Units
Fully-Connected Classification Layers
Fully-Connected Dense Layer 1
16 Neurons
Activation = 'ReLU'
Batch Normalization Layer
Dropout Layer
Rate = 0.2
Fully-Connected Dense Layer 2
6 Nuerons
Activation = 'Soft-Max'
Output Predictions for 6 Activities
Compiler Settings
Optimizer = Adam (Learning Rate = 0.003)
Loss = Categorical Cross-Entropy
Metrics = Accuracy
Model Fit
Input Data = Training dataset
Input Labels = Training labels
Validation Split = 0.2
Epochs = 50
Batch Size = 64
Shuffle = True

Fig. 1. Prototype LSTM structure for conversion to a TensorFlow Lite model. The number of hidden layers (here shown as 2 as the final value used), and the number of units in each hidden layer (here shown as 16 as the final value used) are varied in Section III prior to TensorFlow Lite conversion to investigate the effect on performance.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

where symbols have their standard meanings for true positive, true negatives, false positives, and false negatives [4]. Recall, precision and F1 score are used with their standard definitions. Training and testing was performed on a MacBook Pro with a 2.2 GHz Intel i7 processor and 16 GB RAM. No GPU acceleration was used, and the reported training times indicative of this.

C. *TensorFlow Lite network design*

The pre-trained prototype model from Section II-B was taken as the starting point, and saved in the protocol buffer (.pb) format, which had a size of 1.9 MB. The TensorFlow Lite Python converter was then used to transform the TensorFlow operations by fusing them together to generate

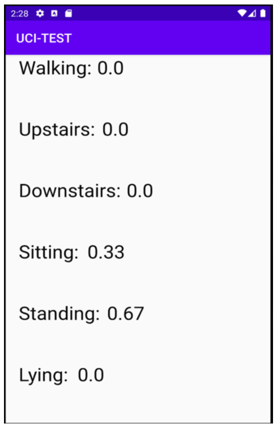

2:28
UCI-TEST
Walking: 0.0
Upstairs: 0.0
Downstairs: 0.0
Sitting: 0.33
Standing: 0.67
Lying: 0.0

Fig. 2. Screenshot of the TensorFlow Lite LSTM HAR app showing the probability of each activity class. Probabilities are updated every 2.56 s in response to a new window of data being received.

a TensorFlow Lite (.tflite) model with limited operations. This model was saved in a compressed FlatBuffer format, capable of performing cross-platform serialization without the need to parse the entire file. This has the effect of making the models more memory and speed efficient for running inference on edge devices.

In order to perform inference on a device using the TensorFlow Lite model, it must be run through an interpreter Application Programming Interface (API). There are interpreter APIs available for multiple platforms including Python, Java, C++, and Swift [11]. The TensorFlow Lite model's accuracy is first evaluated using the Python interpreter API on a PC. The model was then embedded into an Android application with the Java API. For this, an Android HAR app was designed in Android studio, compatible with Android smartphones with minimum API version 24 that are also equipped with a gyroscope and accelerometer for data collection. Here results are simulated on a Google Pixel 2 (API 30) smartphone using the Android Emulator on a PC in order to be as agnostic as possible to the chosen deployment device(s). A screenshot of the app is shown in Fig. 2. The application displays the output predictions for each activity class, and these probabilities are updated for every window of input data received, i.e. every 2.56 s. The Java interpreter API and TensorFlow Lite Android support library are used to run the model for inference on the simulated smartphone. The application is run using a single thread CPU, with no GPU or Neural-Networks API (NNAPI) acceleration. The Android studio profiler is then used to monitor the power consumption of the application between *light*, *medium*, and *high* states.

The default conversion converts the model's activations, weights, and biases to float-32 data types. We repeat the above steps also using a conversion to uint-8 integer representation for the weights using the *dynamic range* quantization approach [11]. In TensorFlow *full-integer* quantization is also supported, converting all the model parameters, including the activations and input and output tensors, to 8-bit integers. Unfortunately, this function is not currently supported by TensorFlow Lite for LSTM models. Nevertheless, the weight-only quantized model generates a reduced model size that is capable of performing inference on a smartphone.

The code for this work is available in GitHub at [13].

TABLE I
EFFECT OF INCREASING THE NUMBER OF STACKED LSTM LAYERS.

LSTM layers	Training accuracy	Validation accuracy	Training time / s	No. of parameters
1	86.4%	83.6%	117	2038
2	95.1%	91.9%	227	4150
3	95.4%	94.0%	403	6262
4	94.5%	91.4%	549	8374

TABLE II
EFFECT OF INCREASING THE NUMBER OF LSTM UNITS PER LAYER.

Units	Training accuracy	Validation accuracy	Training time / s	No. of parameters
8	89.0%	89.9%	201	1366
16	95.1%	90.8%	227	4150
32	96.1%	93.4%	328	14,326
64	95.5%	93.5%	501	53,110
128	96.2%	90.1%	1105	203,406

III. RESULTS

A. *Offline prototype network*

Table I shows the effect of increasing the number of stacked LSTM layers on network performance, and the number of parameters present within the network. Increasing the number of layers beyond two gives limited performance improvements, but gives increased numbers of parameters to be quantized which will impact the memory requirements. It is for this reason that the two layer prototype was chosen for conversion to TensorFlow Lite.

Table II shows the effect of increasing the number of units in each LSTM layer, for the two layer prototype. Increasing the number of units increases the model's accuracy as more temporal features can be extracted per layer, as there are more gating units used to extract discriminative features from the input sequences. However the number of trainable parameters also increases substantially. 32 units are chosen for further use as increasing beyond this number does not lead to a large increase in accuracy, and indeed validation accuracy decreases beyond 64 units due to over-learning with too many free parameters present.

For the final model configuration (Fig. 1) the performance metrics for each class are shown in Table III. These reveal an even distribution of accurate classification for each activity. The model is capable of detecting the temporal differences between *Walking*, *Walking upstairs*, and *Walking Downstairs*, with overall accuracies of over 98% for each. The *Lying* action was the most distinguishable activity. In-line with our previous work [7], the model is weakest at differentiating between the *Standing* and *Sitting* actions, as these activities are both static and do not have many temporal differences between them. The overall accuracy of the model is 92.8%

TABLE III

PERFORMANCE OF THE FINAL PROTOTYPE LSTM.

Activity	Accuracy	Recall	Precision	F1 score
Walking	99.1%	95.0%	99.4%	97.1%
Walking upstairs	99.0%	94.3%	99.1%	96.6%
Walking downstairs	98.7%	100%	91.5%	95.6%
Sitting	94.4%	83.7%	82.7%	83.2%
Standing	95.5%	85.2%	89.2%	87.1%
Lying	99.2%	100%	95.7%	97.8%
Overall	92.8%	93.0%	92.9%	92.9%

TABLE IV

PERFORMANCE COMPARISON TO PREVIOUS HAR ALGORITHMS.

Model	Dataset	Accuracy
CNN [14]	UCI-HAR	95.2%
CNN [5]	UCI-HAR	94.8%
CNN [6]	UCI-HAR	92.7%
LSTM [6]	UCI-HAR	89.0%
SVM [6]	UCI-HAR	90.5%
LSTM (this work)	UCI-HAR	92.8%

and mean F1-score 92.9%. This is compared to the state-of-the-art for on-phone HAR machine learning in Table IV.

B. TensorFlow Lite network

The performance of the TensorFlow Lite network is shown in Table V. The float-32 model achieves an overall accuracy of 92.7%. This is only slightly reduced from the full prototype model due to the fused operations. The output probabilities from the HAR Android app match the output probabilities from the Python interpreter implementation of the TensorFlow Lite model, so there is no performance loss from running the TensorFlow Lite model in Android rather than on a PC.

Table V also compares the TensorFlow Lite models with our previous work for HAR inference on a smartphone. The new models achieve comparable accuracies, but with a much smaller memory footprint. The size of the float-32 TensorFlow Lite model is 63 KB, which is a 30x reduction from the full prototype model. The uint-8 version requires only 27 KB with no further reduction in classification accuracy. The weight-only optimization was thus capable of achieving a 2.3x memory reduction. These memory footprints compare to 1.9 MB for our prototype network, and 4.1 MB for our LSTM network in 2019 [8]. Although the focus of the current work is for on-phone implementations, the memory requirements are now within the range where microcontroller implementations (which typically have a few hundred KB of memory) are now possible to explore in future work.

TABLE V

QUANTIZED MODEL PERFORMANCE.

Model	Accuracy	Memory footprint	Power consumption
Offline prototype	92.8%	1.9 MB	–
Float-32 model	92.7%	63 KB	*Light*
Uint-8 model	92.7%	27 KB	*Light*
CNN (Float-32) [8]	96.4%	16 MB	40 mW
CNN (Uint-8) [8]	93.5%	2.1 MB	–
LSTM (Float-32) [7]	92.2%	17.4 MB	–
LSTM (Uint-8) [7]	93.5%	4.1 MB	–

IV. CONCLUSIONS

This work has presented an optimized on-phone implementation of deep machine learning for human activity recognition for use in rehabilitation and similar applications. The memory footprint has been reduced by >30x, for a 0.1% reduction in classification accuracy. Weight quantization allowed a further 2.3x reduction with no impact on the accuracy. This lays the groundwork for further real-time machine learning implementations on-phones and embedded in wearable devices in future work.

REFERENCES

[1] O. D. Lara and M. A. Labrador, "A survey on human activity recognition using wearable sensors," *IEEE Commun. Surv. Tutor.*, vol. 15, no. 3, pp. 1192–1209, 2013.

[2] J. Wang, Y. Chen, S. Hao, *et al.*, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognit. Lett.*, vol. 119, no. 1, pp. 3–11, 2019.

[3] W. Shi, J. Cao, Q. Zhang, *et al.*, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, 2016.

[4] C. Beach, E. Balaban, and A. J. Casson, "Edge algorithms for wearables: An overview of a truly multi-disciplinary problem," in *Wearable Sensors*, E. Sazonov, Ed., Second Edition, Oxford: Academic Press, 2020, pp. 379–414.

[5] C. A. Ronao and S. B. Cho, "Human activity recognition with smartphone sensors using deep learning neural networks," *Expert Syst. Appl.*, vol. 59, no. 1, pp. 235–244, 2016.

[6] S. Wan, L. Qi, X. Xu, *et al.*, "Deep learning models for real-time human activity recognition with smartphones," *Mob. Netw. Appl.*, vol. 25, no. 2, pp. 743–755, 2020.

[7] T. Zebin, E. Balaban, K. B. Ozanyan, *et al.*, "Implementation of a batch normalized deep LSTM recurrent network on a smartphone for human activity recognition," in *IEEE BHI/BSN*, Chicago, 2019.

[8] T. Zebin, P. J. Scully, N. Peek, *et al.*, "Design and implementation of a convolutional neural network on an edge computing smartphone for human activity recognition," *IEEE Access*, vol. 6, no. 1, pp. 133 509–133 520, 2019.

[9] TensorFlow. (2021). "Home page," [Online]. Available: `https://www.tensorflow.org/`.

[10] P. Agarwal and M. Alam, "A lightweight deep learning model for human activity recognition on edge devices," *Procedia Comput. Sci.*, vol. 167, no. 2019, pp. 2364–2373, 2020.

[11] TensorFlow Lite. (2021). "Usage guide," [Online]. Available: `https://www.tensorflow.org/lite/guide`.

[12] J. Luis and R. Ortiz. (2012). "UCI machine learning repository: Human activity recognition using smartphones data set," [Online]. Available: `https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones`.

[13] LSTM-HAR. (2021). "Source code," [Online]. Available: `https://github.com/CASSON-LAB/LSTM-HAR`.

[14] W. Jiang and Z. Yin, "Human activity recognition using wearable sensors by deep convolutional neural networks," in *ACM Multimedia Conf.*, Brisbane, 2015.