Efficient Deep Learning Structure and Training Method for Human Activity Recognition

Dehua Bi[1], Anindya Paul[2] and Yijia Zhang[3]

[1]Department of Systems Engineering, Boston University
[2]Department of Electrical and Computer Engineering, Boston University
[3]Department of Computer Science, Boston University

Abstract

Human activity recognition is an integral task in developing smart human assistance machines. Existing studies in human activity recognition [1, 2, 3] have used MC-SVM with handcrafted features as the methodology. In this work, we proposed a deep neural network structure of CNN-LSTM and a feature extractor strategy to train a human activity recognition model that can achieve comparable accuracy with a much reduced training time when compared to existing state-of-the-art systems, and thereby lay the foundations for a more efficient system.

1 Introduction

In recent years, mobile devices, especially smartphones, has become a common commodity in today's society. With increasing amount of integrated sensors and computing powers, many user action related information may be extracted out from the mobile devices to better assist the daily activities of the users [4, 5]. Not only can these data reveal how many steps the user has walked during the day time for doctors to keep track of the users health situation, these data can further be used to advise injured people recover quickly, or to assist daily activities of the elderly people [6, 7], or to detect a fall and respond to such a situation immediately which may save the life of the person [8]. All of these systems are built around human activity detection to provide better healthcare.

In the past, many datasets have been provided for research on human activity recognition [1, 9], and several methods have been proposed to build human activity recognition systems[2, 3]. However, traditional machine learning methods utilizing handcrafted features is expensive as it requires expert knowledge to generate the features. Developments in neural network based deep learning methods in image recognition and language translation showed that these methods require almost no handcrafted features [10, 11, 12]. Interestingly, recurrent neural networks, a special branch of deep learning, is able to learn from sequential data, which makes it a suitable system to explore time-dependent relationships [11]. However, many of these deep learning machines are slow to train with, which makes it less applicable to real-life systems since users are in general unwilling to wait such a long time for the system to get trained.

In this work, we explore the different structures and training methods that can be used to learn the user activities with minimal preprocessed mobile signals. We assess these models and training methodologies to try to find a system that is both accurate and requires minimal time to train for. We put forward a CNN-LSTM model with a feature extractor method and show that it can match the accuracy of existing methodologies with substantially lower running time.

Preprint. Work in progress.

2 Related Work

Davide Anguita, et al. proposed a human activity recognition dataset and a MC-SVM system to classify the dataset in 2013 [1]. From the hand engineered features, the author is able to get 96% classification accuracy, which they claimed to be state-of-the-art results. One primary drawback from this method is that the system relies heavily on handcrafted features, which can be expensive due to the cost of hiring experts to find these features. Since then, many studies have either investigated the soundness of the dataset or explored better methods to train the dataset. For example, Quentin Mourcou et al [13] evaluated performance of smartphone inertia sensor measurements to determine if smartphones are reliable and accurate enough for clinical motion research, and found that smartphone measurements are Xsens gold standard for clinical research. Charissa Ann Ronao, et al [14]. proposed a two stage continuous hidden markov models as the classification algorithm, which achieved an accuracy of 91.76% (92%). Recently, Charissa Ann Ronao again proposed a convolutional neural network system, which achieved a testing result of 94.79% (95%) with the raw signal input [15]. Additionally, a blog posted by Guillaume Chevalier applied a special deep learning structure of stacked long short term memory (LSTM) recurrent neural network (RNN) on the mobile signal input, and achieved 91% accuracy [16]. However, these methods are not yet reaching the level of the handcrafted method (the MC-SVM proposed by Davide Anguita, et al. [1]), and running time of deep neural networks are longer. Recently, CNN-LSTM types of neural network has been applied in many areas such as caption generation of images or videos [17]. In this work, we are going to show that a special deep learning structure of CNN-LSTM type system with a novel training method to achieve comparable level of accuracy as the state-of-art result by Davide Anguita, et al. [1], while taking significantly lower time in training.

3 Approaches

We propose two training methods to train the CNN-LSTM neural networks structure in this work. In section 3.1, we are going to introduce the structure of CNN LSTM neural networks. Then, in section 3.2, we discuss two training methods: the end-to-end training method of this system for human activity recognition, and the feature extractor based training method that uses a pre-trained feature extractor to facilitate training of a personalized activity recognition system which will save training time on the client/user side.

3.1 CNN-LSTM Neural Networks

CNN-LSTM is a special recurrent neural network structure designed for sequence prediction problems with spatial inputs. Such a special type of neural networks has been applied in many existing systems, for example in image or video caption generation [17]. In this work, we applied this structure for human activity recognition with minimal hand-crafted (almost raw) signal inputs from mobile devices. The structure of the CNN-LSTM neural networks is shown in Figure 1A. As shown in the figure, there are three types of activation layers in this network: the 1 dimensional convolutional layer, the max pooling layer and a dynamic RNN layer that is constructed from a single LSTM unit.

3.1.1 1-Dimensional Convolution Layer

The input to the neural networks is a tensor of size M by N by W, where M is the number of samples, N is the time steps of the mobile signals, and W is the number of channels. For each training sample, the 1-dimensional convolution layer takes as input a matrix of the time steps of the mobile signal times the number of channels (number of different signals generated from the mobile). Similar to the well-known 2-d convolutional layer, the 1 dimensional convolution layer perform the convolution operation for each channel with kernel size h, but only in the x-direction (as there is no y-direction in the setting of image recognition to perform convolution with). There are K different filters that we are going to supply to the convolution layer to create K different feature-filters of size 1 by 5 to capture the features in the system. In our system, the kernel size h of both convolution layers is set to a window size of 5. The number of filters in the first convolution layer is set to 32, and in the second convolution layer, the filter number is 64. Finally, the activation function for this layer is the Relu function, and at the output of the last convolution layer, we are expecting the system to extract 64 different features from the input data.

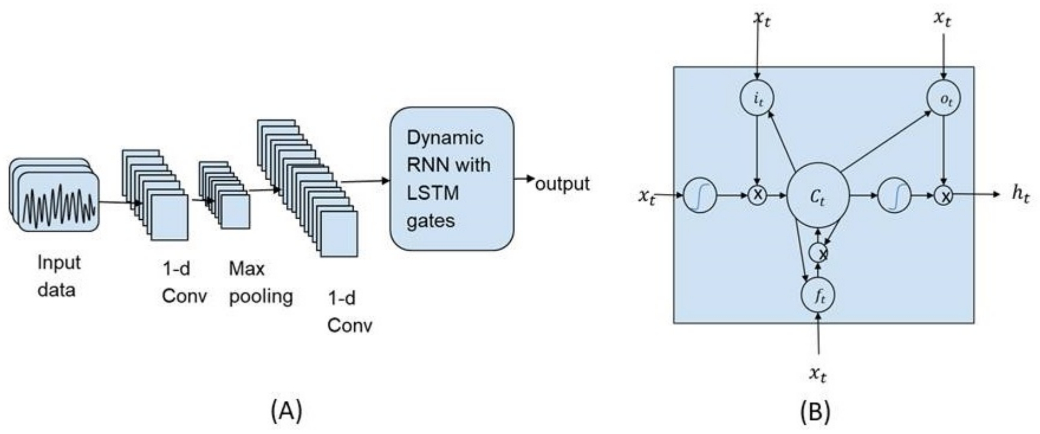

Figure 1: (A) The schematic diagram of the proposed CNN-LSTM system. (B) A single LSTM cell [18].

3.1.2 Max Pooling Layer

The max pooling layer is placed between the first and the second 1-d convolution layers in our system. This layer functions exactly the same way as in standard pooling layers of convolutional neural networks, where the operation in use is the Max function. In our work, the pool size is set to 20, and the strides of the pooling operation is set to 2.

3.1.3 RNN and the LSTM cells

RNN is known to be difficult to train because of vanishing gradient problem. When training RNN using backpropagation through time (BPTT) technique, and learning with long-range dependencies, the derivatives of the sigmoid or the tanh functions are going to be 0 on the -1 or the 1 side (saturated neurons), thus the gradient values will vanish after a few time steps. A soltion to this problem is the long short-term memory (LSTM) cells. A LSTM system is depicted in Figure 1B and equations of LSTM are provided below [18]:

$$f_t = \sigma(W_f[\dot{h}_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i[\dot{h}_{t-1}, x_t] + b_i)$$
$$\tilde{C_t} = \tanh(W_C[\dot{h}_{t-1}, x_t] + b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma(W_o[\dot{h}_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh C_t$$

Here f_t is the forget gate, i_t is the input gate, o_t is the output gate, and C_t is the recurrent cell.

In this work, the input data is first going through the two convolution layers with the max pooling layer in between to construct a feature tensor of M by 49 by 64, with 64 different features extracted out from 9 channels, and each feature has length 1 by 49. This data is transposed and fed directly to the dynamic LSTM layer. Finally, the output of the dynamic layer is multiplied with an output weights to form the prediction of the system.

3.2 Training Modes of the CNN LSTM system

There are two methods we can train this CNN-LSTM system. The first method is the commonly used end-to-end training method where we supply labeled input data with its true labels directly to the system. When it is fully trained, we can supply the system with new data for it to make predictions.

There is one primary drawback of such a system in the human activity recognition setting. Due to the fact that each person perform actions differently from others, a generic system is not going to predict well for individual users, and training a neural net for each user may take a long time. An alternative training method is to train a CNN or a CNN-LSTM as a generic feature extractor, and use this already trained system to preprocess the input data to form features so we can use these features to train a personalized classifier system for each individual users. For example, a generic system can be trained with data collected from numerous persons by the supplier of the system. Afterwards, this already-trained model can be distributed to each users of the system, and the user can extract the output of the second convolution layer in the system. This output would be the input to a single layer of dynamic RNN with LSTM cells for a personalized classifier to recognize activities of the user. Training a single layer of dynamic RNN with LSTM is going to be much faster than training an entire CNN-LSTM from end-to-end, which we will show in section 4.

4 Experiments

4.1 Setup

In sections 4.1.1, 4.1.2, and 4.1.3 we describe how we setup our environment for analyzing our CNN-LSTM system against other methods.

4.1.1 Dataset

The dataset we are using is the Human Activity Recognition Using Smartphone Dataset proposed by Davide Anguita et al [1] in their work to ESONN 2013. This dataset is collected from 30 volunteers within an age bracket of 19-48, where each volunteer preformed 6 different actions (walking, walking upstairs, walking downstairs, sitting, standing and laying) with a mobile device attached to the waist of each person. The x, y and z axis' position and angular velocity data are collected from the accelerator and the gyroscope embedded within the mobile device. Some pre-processing methods and hand engineered functions are applied to these data to produce a dataset composed of 7352 training action samples and 2947 testing action samples, where each action sample is sampled from fixed-width sliding windows of 2.56 seconds and 50% overlap (resulting in 128 readings/window). This online dataset is provided with both after-processed hand engineered features as well as the original accelerator, gyroscope and calculated gravitational accelerations signals from the sensors embedded within the mobile devices after denoising and Butterworth low-pass filtering. In this work, we are using the latter source (the original signal after denoising and low-pass filtering) as the training input, since it is generated with the least amount of feature engineering. For more detailed information of the dataset, please read the dataset description in the dataset repository: https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones

4.1.2 Baselines

Throughout the following sections, we compare our method with a MC-SVM classifier proposed in the original work of Davide Anguita, et al. [1], a stacked LSTM neural network proposed by Guillaume Chevalier [16], and a five layer convolutional neural network that is similar to LeNet5 [19] (1-d convolution-max pooling-1-d convolution-fully connected 1-fully connected 2-output). Notice that the MC-SVM classifier is training on handcrafted features and the stacked LSTMs and the CNN are training on the signal input used in our work.

4.1.3 Hardware and System Specification

All the systems were run on a Acer laptop with Intel core i5-8250U 1.6GHz with Turbo Boost up to 3.4GHz CPU. DDR memory is 12GB DDR4. The code is written in Python 3.0, with Tensorflow 1.0., and the SVM is implemented via Sklearn library in Python.

4.2 End-to-end Training Methods

We first test accuracy and running time of the end-to-end training systems. We refer to the stacked LSTMs neural network as Model 1, the convolutional neural network as Model 2, and the CNN-LSTM system proposed in this work as Model 3. The first experiment is to compare the accuracy

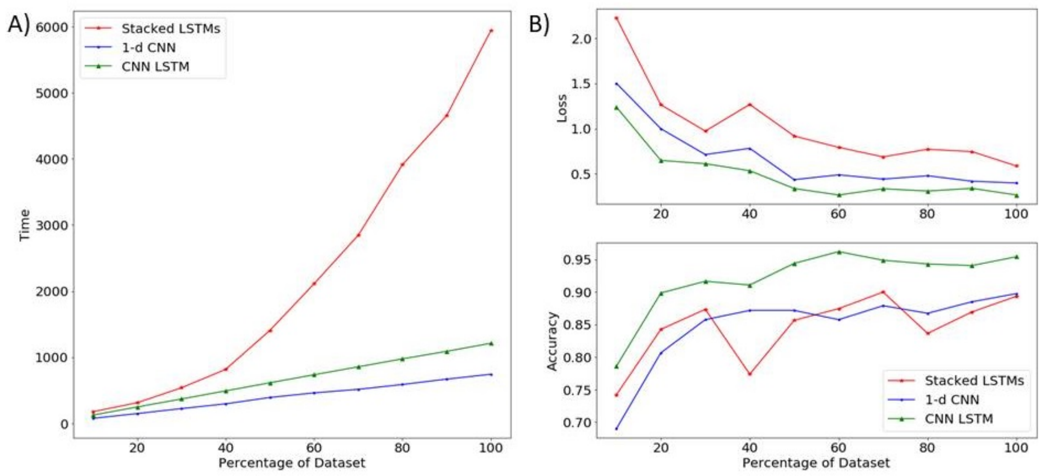

Figure 2: (A) Time taken for the three models to train, and (B) Loss and accuracy of the testing data for the three models when the dataset is visited 300 times for each model to train with. The x axis represents the size of dataset in terms of percentage, red line represents the stacked LSTM model (Model 1), the blue line represents the CNN model (Model 2) and the green line represents the proposed CNN-LSTM model (Model 3).

and running time of the three models when these models are all trained with the same number of iterations. The three models are all implemented with batch learning using the Adam optimizer. As the Adam optimizer is based on a variant of stochastic gradient descent algorithm, to properly train the models, we let the system train on the dataset by going through the training sets multiple times.

4.2.1 First Experiment

In the first experiment, we set the number of times of these models going through the dataset to be 300 times and we are running through different sizes of the dataset (from 10% to 100% of the dataset). The results of running time, testing loss and testing accuracy are plotted in Figure 2.

From Figure 2, we first noticed that the testing accuracy fluctuates for Model 1, but for Model 2 and Model 3, the accuracy is quite stable after training the systems with any size of the datasets that is greater than 30. In terms of accuracy, Model 3 provides the most accurate results, while in terms of the running time, Model 2 is the fastest among the three models, Model 3 runs slightly slower than Model 2, but Model 1 is the slowest among the three models. These results imply that the proposed CNN-LSTM model can greatly boost the testing accuracy while taking a lot less time than the model proposed by Guillaume Chevalier. Additionally, if accuracy is not the primary concern, CNN is also a good algorithm as it provides similar accuracy compared to the stacked LSTM model, but can greatly reduce running time.

4.2.2 Second Experiment

In some machine learning models, early stopping is a good mechanism to combat overfitting and thereby improve accuracy of classifiers. We wanted to determine whether the same phenomenon plays a role in these three models when training with the full sized dataset. Thus, we train these three models 400 times through the full set of the training set, and the results are illustrated in Figure 3.

From Figure3, we observed that in model 1, the testing loss and accuracy fluctuates a lot initially, and becomes stable after 1750 iterations (going through the data 350 times). Loss and accuracy in model 2 stabilizes after 1250 iterations (250 times going through the dataset). In the proposed CNN-LSTM model, model 3, loss and accuracy stabilizes after 1000 iterations (200 times going through the dataset). Based on this observation, we let model 1 run 350 times, model 2 run 250 times,

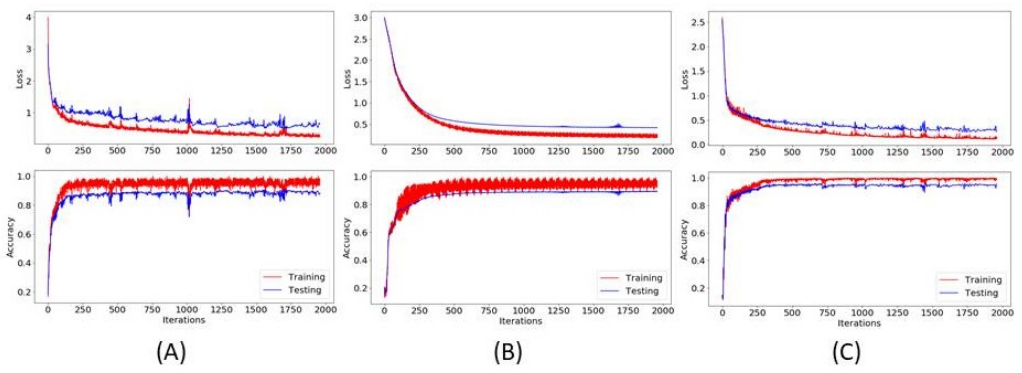

Figure 3: The loss and accuracy of (A) the stacked LSTM model (Model 1), (B) the CNN model (Model 2), and (C) the proposed CNN-LSTM model (Model 3). The red lines are the training loss and accuracy, the blue lines are the testing loss and accuracy.

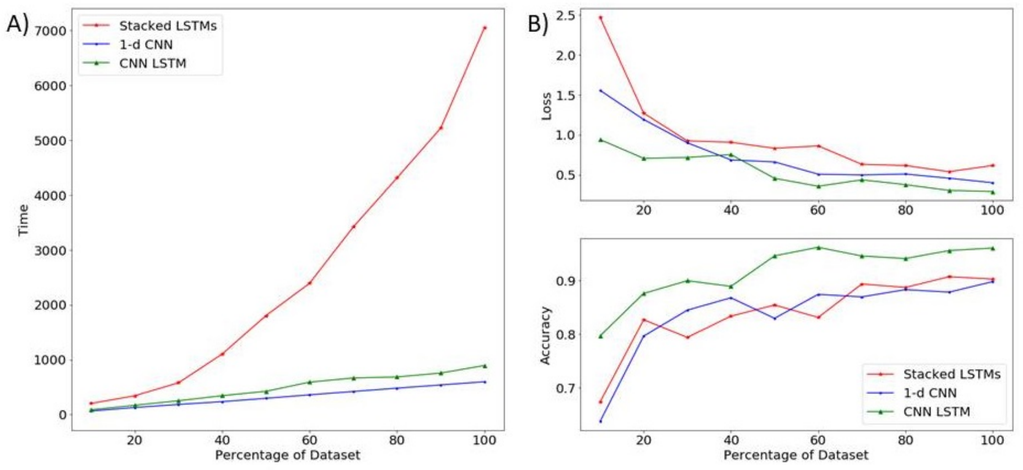

Figure 4: (A) The time taken for the three models to train, and (B) The loss and accuracy of testing data for the three models when the dataset is visited 350 times for the stacked LSTM model (Model 1), 250 times for the CNN model (Model 2) and 200 times for the proposed CNN LSTM model (Model 3). The x axis represents the size of dataset in terms of percentage, red line represents model 1, the blue line represents model 2 and the green line represents the proposed model 3.

and model 3 run 200 times once again through different sizes of the dataset (from 10% to 100% of the dataset). The running time, accuracy and loss are again recorded and depicted in Figure 4.

As illustrated in Figure 4, the accuracy of model 1 is again similar to model 2, and model 3 is again the best model in terms of testing accuracy. In addition, we noticed that the running time for model 2 and model 3 has reduced from experiment 1, and model 3 is only slightly slower than model 2. Especially, when training with 100% of the dataset, model 3 is capable of achieving 96% accuracy on the testing set, which is exactly the same as the state-of-the-art result reported in the work of Davide Anguita, et al., using MC-SVM algorithm with handcrafted features[1]. Moreover, it takes around 10 minutes for Model 2 and 15 minutes for Model 3 to go through the entire dataset, which means that Model 3 takes less than twice the time needed for Model 2. So we conclude that, Model 3 achieved state-of-the-art accuracy with minimal handcrafted features, and the running time is comparable with standard convolutional neural networks.

4.3 Feature extractor training methods

To further reduce the running time for individual users, the convolutional layers in both model 2 and model 3 can be used as feature extractors. Instead of training the system end-to-end for each user separately, a pre-trained system can be supplied to each user. So the user can use the pre-trained system with their own input signals, convert the input signal to features, and train a personalized classifier for each user individually. This mechanism would save time training the convolutional and pooling layers in both model 2 and model 3. In addition, users need to train only a single layer of LSTMs to make the system more accurate at predicting activities of the individual user. As a result this would save time from training a more involved deep neural network.

We performed two experiments in this section, first using the pre-trained CNN model (model 2) as the feature extractor, and then using the proposed CNN-LSTM model (model 3) as the feature extractor. We trained on the full dataset for these feature extractors. For individual classifiers, we used one-vs-one SVM, one-vs-rest SVM and a single layer of LSTM cells to predict the actions. In the first experiment, we extracted the features from pre-trained CNN model (model 2) with 50% and then 100% dataset, reformatted the features into correct format to be trained with a one-vs-one SVM, a one-vs-rest SVM and a single layer of LSTM cells RNN. As shown in Table 1 and Table 2, in terms of running time, both SVMs took 56 seconds, and 189 seconds for the 50% and 100% datasets, respectively. While, RNN took 756 seconds, and 763 seconds (about 12 minutes) for each size of the dataset, respectively. However, in terms of accuracy, the SVMs reached an accuracy of only 52% and 59% respectively, while the RNN is able to reach 95% for both (comparable with the convnet approach proposed by Charissa Ann Ronao, et al. [15]). In the second experiment, we used the proposed CNN-LSTM model (model 3) as the feature extractor. Both SVMs took 10 seconds and 31 seconds (less than a minute) for 50% and 100% datasets respectively. While, RNN took only 500 and 513 seconds (less than 9 minutes) to complete for these different sizes of the training set. In addition, for both 50% and 100% datasets, both SVMs achieved a 81% and 88% classification accuracy, while RNN achieved a 95% and 96% classification accuracy rate respectively.

Table 1: 50% Running time, Loss and Accuracy of the models

Feature Extraction Method	Personalized Training Method	Time	Accuracy
CNN	SVM OVO	56s	52%
CNN	SVM OVR	56s	52%
CNN	LSTM	756s	95%
CNN-LSTM	SVM OVO	10s	81%
CNN-LSTM	SVM OVR	10s	81%
CNN-LSTM	LSTM	500s	95%

Table 2: 100% Running time, Loss and Accuracy of the models

Feature Extraction Method	Personalized Training Method	Time	Accuracy
CNN	SVM OVO	189s	59%
CNN	SVM OVR	189s	59%
CNN	LSTM	763s	95%
CNN-LSTM	SVM OVO	31s	88%
CNN-LSTM	SVM OVR	31s	88%
CNN-LSTM	LSTM	513s	96%

We observed that in comparision with end-to-end training method, we are able to achieve the same accuracy as in the state-of-the-art system in the work of Davide Anguita, et al., using MC-SVM algorithm with handcrafted features[1]. Additinally, the running time is reduced from 15 minutes to less than 9 minutes, almost half the time. Finally, the confusion matrix of CNN and CNN-LSTM feature extractors with LSTM classifier layer is shown in Figure 5. From this figure, we observe that feature-extractor training method with CNN or the proposed CNN-LSTM model and a single layer of LSTM cell RNN classifier is able to accurately classify walking from walking up and down the stairs, and is also able to accurately predict sitting from standing, which could not be classified accurately in other studies.

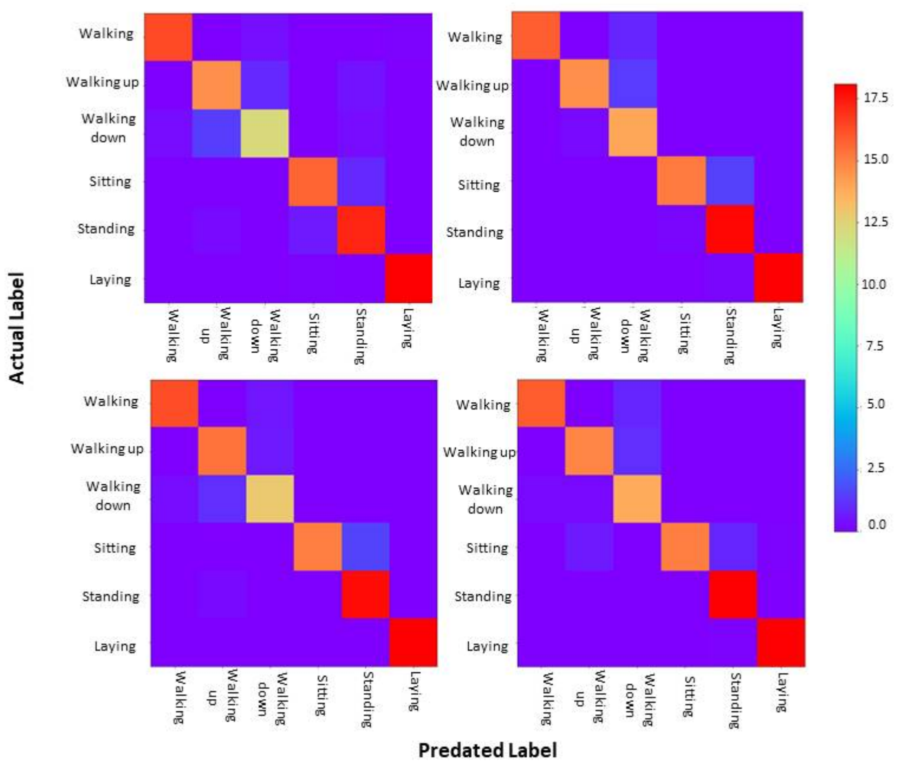

Figure 5: The confusion matrix of CNN(left) and the proposed CNN-LSTM model(right) trained using the feature extractor strategy on 50% (top) and 100% (bottom) training data.

5 Conclusion

In this work, we conclude that the proposed CNN-LSTM system can use almost raw features and is able to achieve the same accuracy as in the state-of-the-art system learning on handcrafted features[1]. Additionally, by using the pre-trained CNN-LSTM system as feature extractor with a personalized RNN (single layered LSTM) classifier, running time is reduced to less than 9 minutes.

In future, we intend to determine transitions from one human activity to another. Determining transitions in human activities like laying to sitting, or walking to laying, etc. can potentially be used to build smarter and efficient applications in future. Some areas where this can be used are child monitoring, monitoring health of patients, detecting a fall for an elderly person, etc.

Author Contribution

D.B., A.P., and Y.Z. contributed to the idea of the study; D.B. contributed to the design of the experiments in this study; D.B., A.P., and Y.Z. contributed to the coding and data analysis; D.B. and A.P. contributed to the writing of this manuscript; D.B., A.P., and Y.Z. contributed to the presentation.

Conflict of Interest

The authors declare that they have no conflict of interest. The authors had full access to all data and had the final responsibility for the decision to submit for publication.

Project Code and Related Documents

Our code, dataset, and relevant project related documents are available in https://github.com/edwardbi/CAS542FinalProject

Acknowledgments

We thank Dr. Sang (Peter) Chin and our TAs Yida Xin and Xiao Zhou at Boston University for guidance and technical assistance. We also thank Mr. Guillaume Chevalier for publishing some of the codes on Github under MIT License to provide us a baseline as well as a framework to better assist us on data processing. The study was a class project for CAS 542 Machine Learning and will be funded by grants from Boston University Computer Science department, if it is able to get to the next stage.

References

[1] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *ESANN*, 2013.

[2] Uwe Maurer, Asim Smailagic, Daniel P Siewiorek, and Michael Deisher. Activity recognition and monitoring using multiple sensors on different body positions. In *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on*, pages 4–pp. IEEE, 2006.

[3] Lin Sun, Daqing Zhang, Bin Li, Bin Guo, and Shijian Li. Activity recognition on an accelerometer embedded mobile phone with varying positions and orientations. In *International conference on ubiquitous intelligence and computing*, pages 548–562. Springer, 2010.

[4] Dairazalia Sanchez, Monica Tentori, and Jesus Favela. Activity recognition for the smart hospital. *IEEE intelligent systems*, 23(2), 2008.

[5] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011.

[6] Matthai Philipose, Kenneth P Fishkin, Mike Perkowitz, Donald J Patterson, Dieter Fox, Henry Kautz, and Dirk Hahnel. Inferring activities from interactions with objects. *IEEE pervasive computing*, 3(4):50–57, 2004.

[7] Diane J Cook and Maureen Schmitter-Edgecombe. Assessing the quality of activities in a smart environment. *Methods of information in medicine*, 48(5):480, 2009.

[8] Oscar D Lara, Miguel A Labrador, et al. A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys and Tutorials*, 15(3):1192–1209, 2013.

[9] Filippo Palumbo, Claudio Gallicchio, Rita Pucci, and Alessio Micheli. Human activity recognition using multisensor data fusion based on reservoir computing. *Journal of Ambient Intelligence and Smart Environments*, 8(2):87–107, 2016.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Martin Sundermeyer, Ralf Schluter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[12] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[13] Quentin Mourcou, Anthony Fleury, Celine Franco, Frederic Klopcic, and Nicolas Vuillerme. Performance evaluation of smartphone inertial sensors measurement for range of motion. *Sensors*, 15(9):23168–23187, 2015.

[14] Charissa Ann Ronao and Sung-Bae Cho. Human activity recognition using smartphone sensors with two-stage continuous hidden markov models. In *Natural computation (ICNC), 2014 10th international conference on*, pages 681–686. IEEE, 2014.

[15] Charissa Ann Ronao and Sung-Bae Cho. Human activity recognition with smartphone sensors using deep learning neural networks. *Expert Systems with Applications*, 59:235–244, 2016.

[16] Chevalier Guillaume. Lstms for human activity recognition. `https://github.com/guillaume-chevalier/LSTM-Human-Activity-Recognition/`, 2016.

[17] Subhashini Venugopalan, Lisa Anne Hendricks, Raymond Mooney, and Kate Saenko. Improving lstm-based video description with linguistic knowledge mined from text. *arXiv preprint arXiv:1604.01729*, 2016.

[18] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

[19] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, page 20, 2015.