CS230

HATRNet: Human Activity/Transition Recognition using Deep Neural Networks

Nicholas Gaudio, Akash Levy, and Jonas Messner
Department of Electrical Engineering, Stanford University
{nsgaudio,akashl,messnerj}@stanford.edu

Abstract

Human activity recognition based on sensor data is a topic with great potential for customized healthcare. Here, an end-to-end deep learning architecture for human activity/transition recognition is developed, achieving an error rate of 0.82 %. Various deep learning models are analyzed, and a hyperparameter search is conducted to optimize our chosen model. First, an LSTM architecture is examined, which has the advantage of allowing variable-length input sequences for both training and inference. However, our best architecture (HATRNet) is a deep convolutional neural network with late sensor fusion i.e. separate processing pipelines for subsets of the input channels. We feed in zero-padded time sequences to our network, and achieve accuracy exceeding the state-of-the-art reported in literature—all without the use of hand-extracted features.

1 Introduction and Related Work

In developed countries today, most people own a smartphone with all the necessary sensor elements to perform human activity recognition (HAR). This motivates the development of techniques to monitor human activity and encourage healthy lifestyle choices. We propose an end-to-end deep learning solution for categorizing accelerometer and gyroscope data into different activities/postural transitions. Our solution takes input sequences of gyroscope sensor data from a smartphone (6 channels: x,y,z components of accelerometer, and x,y,z components of gyroscope) and produces activity classifications, which are structured data suitable for health tracking and other purposes.

Implementations of human activity recognition have been demonstrated in prior studies. In [1], a support-vector machine (SVM) is used to classify activities and postural transitions. The approach taken does not employ an end-to-end strategy, and includes several signal processing and feature extraction steps. Frequency components of the data are obtained by performing a fast Fourier transform on the time-series data. The time and frequency data are then fed into a feature extractor which results in 561 features. These features are passed to the SVM model which predicts the activity with a feature-dependent error rate of 3.22 % on the SBHAR (smartphone-based human activity recognition) dataset. The weakness of the approach taken is the long development time required for hand-selection of features. In [2], activities are classified by means of a convolutional neural network (CNN) by an architecture titled PerceptionNet. The PerceptionNet architecture automatically extracts the temporal dependencies of the time-series data and leverages the idea of late sensor fusion employing 1D convolutional and max pool layers in the early hidden layers followed by a 2D convolutional layer and global average pooling layer, where the fusion of sensor channel processing occurs. PerpectionNet decreases the error rate to 2.75 % on the SBHAR test data compared to a CNN with early sensor fusion and LSTM model. In [3], another motion recognition algorithm based on feature extraction and SVMs is shown. [4] uses an approach based on the K-nearest neighbor method. [5] uses information fusion from multiple density maps for effectively assessing intensity patterns in health tracking.

The goal of this work is to implement an end-to-end deep learning architecture (HATRNet) without manual feature extraction. Our final solution is similar to PerceptionNet [2], however we improve upon this existing work with: (1) more advanced preprocessing including data augmentation, (2) frequency data from fast Fourier transform over the inputs, (3) high-accuracy classification of transitions ([1] lumps all transitions into a single category, and [2] does not

CS230: Deep Learning, Winter 2019, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

consider activity transitions at all), and (4) more thorough architectural optimization and hyperparameter search. Our code is made publicly available online under the MIT license[1]. As in PerceptionNet, we incorporate the highly effective idea of late sensor fusion. While the reported error rates in previous works are already low, we distinguish the activities "sitting" and "standing" more accurately, achieving an error rate of 0.82 %.

The results of [1] and [2] show the challenge of differentiating between the categories "sitting" and "standing" (see table 1 for details). This misclassification can be a problem, especially when using activity recognition in health or nutrition applications. As pointed out in [6], around 114 kcal per day are additionally expended if performing work at a standing desk instead of the usual sitting desks in offices and schools.

2 Dataset and Features

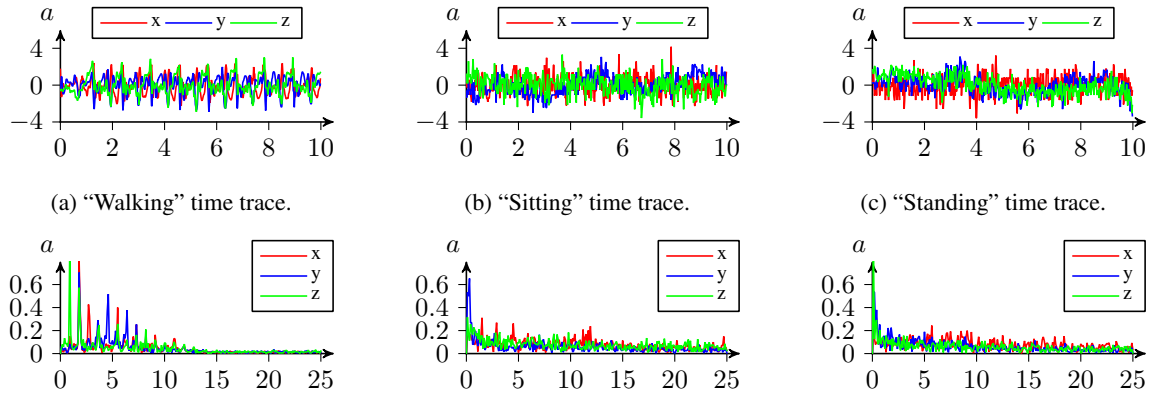
The SBHAR dataset is introduced in [7] and extended to include postural transitions in [1]. We use the SBHAR dataset with postural transitions from [1]. It contains 3-axial linear acceleration a and 3-axial angular velocity ω, recorded at a rate of 50 Hz. The dataset is labelled with six activities (walking, walking upstairs, walking downstairs, sitting, standing, laying) and six postural transitions (stand-to-sit, sit-to-stand, sit-to-lie, lie-to-sit, stand-to-lie, and lie-to-stand). There are 1214 time-traces captured from a group of 30 volunteers aged 19-48 using the Samsung Galaxy SII smartphone. The dataset was captured at a rate of 50 Hz with the longest time-trace being of length 2032 data points and the shortest of length 73 data points.

In contrast to [1], HATRNet involves no feature extraction beyond incorporation of frequency representations of the signals. As a result, our network determines relevant features automatically, even from a relatively small input dataset size of 728 training examples.

For preprocessing, we first normalized the data by subtracting the mean and dividing by the standard deviation for each input sequence. Next, we split our dataset into training, validation and test set with a 60 %/20 %/20 % split, resulting in 728/243/243 data traces in each set. We augment our training data by squeezing/stretching each training time trace with four different random squeeze/stretch factors between 0.75 to 1.25, which can be thought of as a person walking faster/slower. This augmentation step is based on the idea of [8] where a small window of a data trace is warped. Data augmentation increases the number of training examples to 3,640. Since our time traces are all of varying length, we zero-padded the time-series data to equal length, which is necessary for the CNN.

In order to make our input data of higher dimension, and thus enhance the information, we also incorporated frequency data. Each of the six axes (3D acceleration, 3D angular velocity) is transformed into the frequency domain by means of FFT before zero-padding. As the time data is recorded with a sample rate of 50 Hz (F_s) the FFT results in a frequency vector from 0 to 25 Hz ($F_s/2$).

Figure 1(a)-(c) show time-series data of three acceleration samples from the dataset. All data traces are normalized to zero mean and unit variance. While the "walking" activity can be easily distinguished from the other two activities, differentiating "sitting" and "standing" activities from each other is much harder. Figure 1(d)-(f) show the respective frequency spectrum obtained by performing a fast Fourier transform on the normalized time-series data.



(a) "Walking" time trace. (b) "Sitting" time trace. (c) "Standing" time trace.

(d) "Walking" frequency spectrum. (e) "Sitting" frequency spectrum. (f) "Standing" frequency spectrum.

Figure 1: Time-series and frequency data of three sample acceleration time traces.

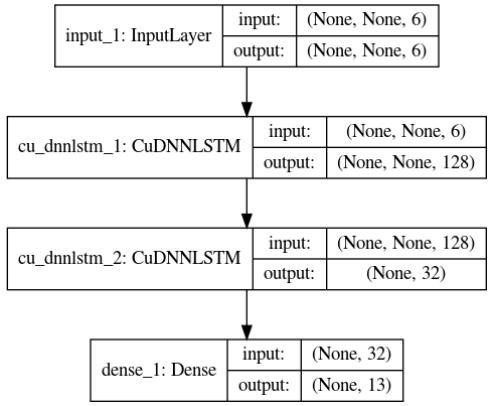
[1] https://github.com/akashlevy/HATRNet

3 Methods

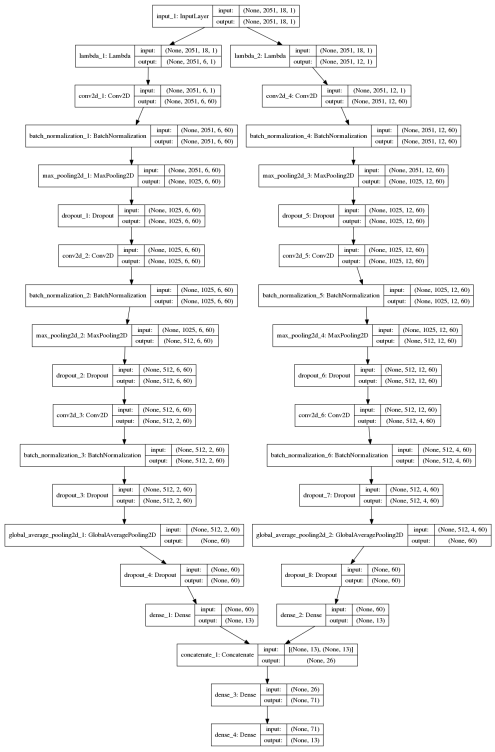
We used Keras [9], a popular deep learning front-end library, to implement our neural network. Google's TensorFlow framework [10] was used as the backend. Our preprocessing was done with MATLAB, SciPy [11], and the scikit-learn library [12]. We examined sequence networks as well as convolutional networks to classify the time series data. We found that convolutional networks produced higher performance so we iterated and focused our later efforts on this family of solutions. We used a *categorical cross-entropy loss function* for all our models: $\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{12} y \log(\hat{y})$

3.1 Sequence Model (LSTM)

Our hypothesis was that sequence models would perform well, given that we are dealing with sequential time-series data, potentially having long-term correlations between the data points. We tested a 2-layer LSTM model shown in Figure 2(a) using the CuDNN library [13], and obtained a maximum of 81.89% accuracy with our best hyperparameter set (we manually tweaked the hyperparameters over a small range). This approach produced significantly lower accuracy than our convolutional model, so we focused our efforts on refining that model instead of further developing our sequence model (following the philosophy of rapid iteration). Our chosen hyperparameters were $\alpha = 0.0005$ (learning rate), $\beta_1 = 0.9$ (decay factor 1), $\beta_2 = 0.999$ (decay factor 2), with $\epsilon = 0$ (no fuzz factor), and no learning rate decay.



(a) Our LSTM model. Our first LSTM layer takes sensor data as input, with 6 input channels and variable length sequences/batch sizes (hence the two *Nones* in input size), and produce an output sequence with 128 channels of the same length as the input. Our second LSTM layer takes an input sequence with 128 channels and produces an output of length 32. This is fed into a dense layer with a softmax activation function, which produces the output classification.



(b) Our deep convolutional model. The left subnetwork takes the time traces as input, with 6 zero-padded input channels. The right subnetwork takes the frequency and phase traces as input, with 12 interpolated input channels.

Figure 2: The LSTM model and deep convolutional model

3.2 Deep Convolutional Network

Convolutional neural networks are unique due to their translational invariance and ability to learn without prior knowledge or hand-engineered features. It was thought that the translationally invariant filters would not perform well on postural transitions due to their aperiodic time traces. It was found that filters of large width along the time dimension (Conv1D filter size: 1x14, Conv2D filter size 3x42), were able to successfully learn and classify all postural activities and transitions with an overall error rate of 3.29%.

The deep convolutional network employed two notable design choices: a Siamese (non-weight sharing) neural network architecture and late sensor fusion. The two subnetworks were leveraged to encode the (1) time and (2) frequency/phase representations of the signals. The non-weight sharing Siamese network was selected, since the time- and frequency-domain representations of the traces are in different domains and should not be convolved with the same filters. Late sensor fusion in the subnetworks, analogous to [2], was implemented with two Conv1D blocks (Conv{1,2}D block: Conv{1,2}D, ReLU, BatchNormalization, MaxPooling2D, Dropout) followed by a Conv2D block, treating the signals discretely and allowing the late Conv2D to operate on the encoded traces, leading to more efficient feature extraction.

It is imperative that the Conv2D block convolves across either the 3-axial linear acceleration traces or the 3-axial angular velocity traces due to independence of the two sets of traces. This was achieved using a vertical stride of 3. Lastly, instead of using a final dense layer in the subnetworks, it was empirically determined that a global average pooling layer resulted in increased accuracy before concatenating the time and frequency/phase subnetworks. Our final chosen hyperparameters were learning rate=0.0026, filter number=60, Conv1D filter size=1x14, Conv2D filter size=3x42, Dense layer size=71, Dropout=0.3660.

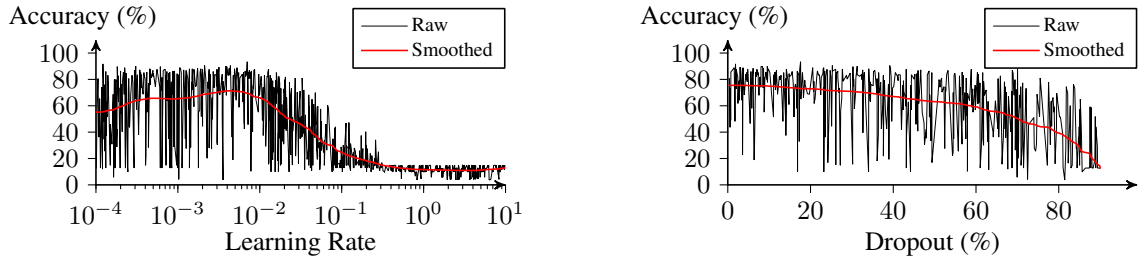
4 Experiments/Results/Discussion

4.1 Hyperparameter tuning

As described in the previous section the CNN with late sensor fusion outperforms the LSTM model. Hyperparameter search was performed over a defined search space in a random fashion. The following parameters were chosen to be variable: learning rate, batch size, number of Conv1D blocks, number of filters in first/last Conv1D block, kernel width in Conv1D blocks, kernel width in Conv2D block, dropout percentage and size of fully-connected layer.

The hyperparameter search was conducted in two steps: a coarse-grain and a fine-grain search. In the coarse-grain run all parameters were assigned randomly in a wide range of values. One coarse-grain run was performed on around 900 different random networks. After evaluating the coarse-grain run, the parameter range was decreased and a fine-grain run was performed with around 200 different random networks and five runs each in order to average over different random initializations.

Figure 3 shows the validation accuracy across the coarse-grain search space of two example hyperparameters. In (a), the validation accuracy is plotted against the learning rate, which nicely shows a learning rate optimum between 0.001 and 0.01. In (b), the validation accuracy is plotted against the dropout rate, which led to the insight that the convolutional architecture does not need a high dropout rate, because it inherently regularizes itself. The noisy nature of both graphs is due to the fact that in each run all parameters are random, so even if the learning rate is optimal other poor parameters might lead to a low validation accuracy.



(a) Learning Rate

(b) Dropout Percentage

Figure 3: Validation accuracy across different hyperparameters

4.2 Discussion

Table 1 shows the confusion matrices of [1], [2] and this work. In comparison to [1] and [2], we did not cut the data into smaller 2.5-second chunks. Instead, we used the whole data traces that were provided in the dataset, which included variable-size sequences. As a result, we had fewer data samples to work with (this can be clearly seen in our confusion matrix reported below). Additionally, HATRNet does not group postural transitions. As a result of the discrepancies between our datasets and the ones in literature, our results are not directly comparable. However, it is interesting to note how longer time traces resulted in increased accuracy.

Table 1: Top: Confusion matrices from [1] (left) and [2] (right). Bottom: This work. Sitting and standing confusion numbers are highlighted in bold.

	WA	WU	WD	SI	ST	LD	PT
WA	1834	64	5	3	2	0	1
WU	10	1743	51	5	5	0	16
WD	0	2	1671	1	7	0	1
SI	0	0	0	1875	**94**	6	3
ST	0	2	0	**109**	2049	0	1
LD	0	0	0	1	0	2148	2
PT	0	1	2	0	0	0	1036

	WA	WU	WD	SI	ST	LD
WA	487	0	9	0	0	0
WU	2	468	0	0	0	1
WD	0	0	420	0	0	0
SI	0	2	0	443	**46**	0
ST	0	0	0	**16**	516	0
LD	0	0	0	0	0	537

	WA	WU	WD	SI	ST	LD	ST-SI	SI-ST	SI-L	L-SI	ST-L	L-ST
WA	21	0	1	0	0	0	0	0	0	0	0	0
WU	0	36	0	0	0	0	1	0	0	0	0	0
WD	0	0	33	0	0	0	0	0	0	0	0	0
SI	0	0	0	20	**0**	0	0	0	0	0	0	0
ST	0	0	0	**2**	27	0	0	0	0	0	0	0
LD	0	0	0	0	0	26	0	0	0	0	0	0
ST-SI	0	0	0	0	0	0	19	0	0	0	0	0
SI-ST	0	0	0	0	0	0	0	10	0	0	0	0
SI-L	0	0	0	0	0	0	0	0	11	0	0	0
L-SI	0	0	0	0	0	0	0	0	0	15	0	2
ST-L	0	0	0	0	0	0	1	0	1	0	10	0
L-ST	0	0	0	0	0	0	0	0	0	0	0	7

WA: Walking, WU: Walking-Upstairs, WD: Walking-Downstairs, SI: Sitting, ST: Standing, LD: Laying-Down, PT: Postural Transition, ST-SI: Stand-to-Sit, SI-ST: Sit-to-Stand, SI-L: Sit-to-Lie, L-SI: Lie-to-Sit, ST-L: Stand-to-Lie, L-ST: Lie-to-Stand

4.3 Results with LSTM vs. Convolutional Network

Table 2 shows the achieved error rates of the different architectures of this work and of [1] and [2]. CNN1 represents the convolutional neural network trained to classify all 12 activities and postural transitions while the postural transitions are clustered in CNN2 leading to seven categories and a fair comparison to our baseline model of [1]. We achieved a minimum error rate of 0.82 %, a decrease of 3.9x compared to the baseline.

Table 2: Comparison of neural network accuracy.

	CNN1	CNN2	LSTM	SVM [1] (baseline)	Perceptionnet (CNN) [2]
Number of categories	12	7	12	7	6
Error Rate	3.29 %	0.82 %	18.11 %	3.22 %	2.75 %

5 Conclusion/Future Work

Here, we have demonstrated state-of-the-art performance on the SBHAR dataset using an end-to-end deep learning approach. Using several techniques, including preprocessing, incorporation of spectral information, late sensor fusion, data augmentation, and highly-tuned convolutional networks, we achieve state-of-the-art performance with an accuracy of 99.18% with our final model.

Somewhat surprisingly, LSTM, a sequence model designed for sequential data, performed less well—possibly due to short-term correlations in the sequences that were better captured with convolutional filters. Future work might include model ensembling using data representations extracted from sequence models (such as LSTM), or incorporation of hand-extracted features as in [1]. Overall, the techniques behind HATRNet should enable high-fidelity smartphone health tracking, advancing the field of customized healthcare.

6 Contributions

Nicholas wrote most of the Keras code used by the team as well as some of the preprocessing code. He developed the Siamese convolutional network architecture with late sensor fusion, and helped Jonas with hyperparameter tuning. He wrote the section on the convolutional architecture as well as lead the efforts on the poster.

Akash worked on the sequence models in Keras, and wrote the initial outline for the report. He also wrote the section on LSTMs as well as the introduction/conclusion. He set up the AWS GPU machine for training, formatted the LaTeX for the report, and helped produce some of the figures.

Jonas wrote most of the preprocessing code, including splitting, data augmentation, and spectral analysis. He also executed and evaluated the hyperparameter search of the convolutional architecture. He created the figures and wrote the sections on preprocessing and hyperparameter tuning.

References

[1] J.-L. Reyes-Ortiz, L. Oneto, A. Samà, X. Parra, and D. Anguita, "Transition-aware human activity recognition using smartphones," *Neurocomput.*, vol. 171, pp. 754–767, Jan. 2016.

[2] P. Kasnesis, C. Z. Patrikakis, and I. S. Venieris, "Perceptionnet: A deep convolutional neural network for late sensor fusion," *CoRR*, vol. abs/1811.00170, 2018.

[3] C. A. J. O. D. Fuentes, L. Gonzalez-Abril, "Online motion recognition using an accelerometer in a mobile device," *Expert systems with applications*, 2011.

[4] M. Kose, O. Incel, and C. Ersoy, "Online human activity recognition on smart phones," *Workshop on Mobile Sensing: From Smartphones and Wearables to Big Data*, 01 2012.

[5] J. Qi, P. Yang, M. Hanneghan, and S. Tang, "Multiple density maps information fusion for effectively assessing intensity pattern of lifelogging physical activity," *Neurocomputing*, vol. 220, pp. 199–209, 2017.

[6] C. Reiff, K. Marlatt, and D. Dengel, "Difference in caloric expenditure in sitting versus standing desks," *Journal of physical activity & health*, vol. 9, pp. 1009–11, 09 2012.

[7] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," *21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN*, 01 2013.

[8] A. L. Guennec, S. Malinowski, and R. Tavenard, "Data augmentation for time series classification using convolutional neural networks," *ECML/PKDD Workshop on Advanced Analytics andLearning on Temporal Data*, 2016.

[9] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[11] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," 2001.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python ," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[13] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.