Human Activity Recognition using Deep Learning Models

Ankur Roychowdhury, Saagar Minocha, Joshua Crockett
Department of Computer Science and Engineering,
Texas A&M University, College Station
ankurrc@tamu.edu, saagar14@tamu.edu, jcroc@tamu.edu

***Abstract*-Human Activity Recognition (or HAR) is a field of study concerned with the recognition of specific human activity (which may include the posture or action) based on data acquired using sensors, cameras etc. The typical movements or activities recognized usually include walking, standing, lying, sitting, walking etc. Traditionally, acquiring sensor data for HAR was difficult due to hardware restrictions, cost, and inaccuracy of sensors. However, with the recent influx of smartphones with multiple sensors (like accelerometer, gyroscope etc.) as well as personalized fitness trackers, the sensor data can be acquired easily and in an inexpensive manner. The main objective of modern HAR is to predict the human activity from time-bound snapshots of sensor data collected using a single sensor or a combination of sensors. HAR systems are either supervised (need proper training with specific datasets) or unsupervised (configured with a set of rules). In this paper, we investigate the performance of different algorithms of varying complexity for HAR. We first establish a baseline for HAR using the Support Vector Machine (SVM) classifier across the hand-engineered feature dataset as well as the processed dataset. We, further, explore 5 deep learning architectures (based on spatial, spatio-temporal and structural features) - Convolutional Neural Networks (CNN) based WaveNet, Long Short Term Memory (LSTM), CNN-LSTM, ConvLSTM and LSTM-Conditional Random Fields (CRF). Finally, we export these models and deploy them on an Android mobile to perform real-time HAR.**

Keywords—Human activity recognition, machine learning, deep learning, mobile applications,

I. INTRODUCTION

In this project, we looked at the problem of HAR and tried to classify 6 human activities with SVMs, CNNs, and RNNs. These 6 activities were laying, sitting, standing, walking, walking upstairs, and walking downstairs. We used an SVM classifier to get a baseline for our other models. From this, we were able to see the benefits and disadvantages of using neural networks in HAR. The data we used from the UCI dataset included a hand-engineered feature set as well as the raw data. We analyzed the accuracy of an SVM using this engineered data compared to our neural networks using just the raw data to see if our neural networks could learn the processes used in the engineered data. If our neural networks performed as good or better than the SVM, we would know our neural networks learned these transforms or something similar to the transforms. With implementing the neural networks, we started with simple, shallow networks and tried to see how much improvement we would get from using deeper networks. Since deeper networks can learn more complicated functions, we expected our best results would come from the deepest networks, which we see is true in the results. For our CNN model, we tried a basic network with only convolution layers, dropout, and max pooling, as well as a WaveNet architecture. WaveNet models were expected to perform better than our normal CNN models since WaveNet is designed for time series data. We were able to see that WaveNet did indeed provide a slight advantage over the regular CNN. For the RNNs, we used LSTM, CNN-LSTM, and LSTM-CRF architectures. We expected the RNNs to perform better than the CNNs, since they are commonly used for temporal data, while CNNs are usually used for spatial data. While the RNNs did classify accurately, we were not able to get as good accuracies as the CNNs. While, it is possible we did not find the optimal hyper-parameters and architecture for our RNNs, we propose the discrepancy in our predictions and results are from the nature of our data. Local spikes in the data such as sudden acceleration seem likely for some of our

classes (walking upstairs, walking downstairs). Thus, the CNNs would be able to take advantage of these features to more accurately classify our data. Future work would include further optimizing our RNNs and looking into what features our CNNs built.

This report is organized as follows: Section II discusses Related Work, Section III discusses the methods used ranging from baseline models to structural models. Section IV discusses the Results and Section V provides insights into the discussions Section VI elaborates upon team task allocation and Section VII provides our concluding thoughts and future scope.

II. Related Work

HAR is an extensively studied problem. It can work both with vision based and sensor-based data. In this project we only consider the latter. There are mainly two approaches that deal with the HAR problem – *supervised* (deal with labeled data) and *semi-supervised* (deal with unlabeled data).

A. Supervised Learning

Some important supervised learning classifiers for HAR include *Decision trees, Bayesian methods, Support Vector Machines* (SVM), *Artificial Neural Networks* (ANN) and *Ensemble classifiers*.

The C4.5 decision tree classifier [1] selects attributes to be placed in the top nodes based on the concept of information gain. The generated model is simple, easy to understand and can be evaluated efficiently without consuming time. Bayesian methods like the *Bayesian Network* [2] and *Naïve Bayes* [3] compute class-wise posterior probabilities using estimated conditional probabilities from the dataset. However, the assumption of independence among the features required to maintain the topology in these methods does not hold in the case of HAR as the acceleration signals are usually highly correlated. SVMs [4] use kernel functions to project all instances to a higher dimensional space in order to find a linear decision boundary to separate the data. Neural networks [5] are considered universal function approximators but the set of rules learnt during training are hidden which restricts analysis and additional tuning. They also require large amounts of data and computational cost to achieve high accuracy. The outputs of several classifiers are sometimes combined to improve accuracy in models like *bagging, boosting* [6][7]etc. These ensemble classifiers are computationally expensive as multiple models need to be trained.

B. Semi-supervised learning

Semi-supervised learning classifiers for HAR are relatively few where some part of the data is unlabeled. As a result, there are is no standardization in terms of methods for semi-supervised HAR. Annotation of data for HAR may be cumbersome or difficult in some situations (high activity granularity, less cooperative subjects). Some important semi-supervised approaches to HAR include Multi-graphs, En-co-training and Multiple Eigenspaces (MES).

The Mutli-graph technique proposed in [8][9] propagates labels through a graph containing both labeled as well as unlabeled data. The graph nodes correspond to instances and edges encode similarities between a pair of nodes. After label propagation throughout the graph, classification is done with an SVM classifier using a Gaussian RBF kernel. The en-co-training method [10] is an extension of the co-training method in supervised learning which is free from the 2 sufficient and redundant attribute subset requirements of the later. This condition does not hold for HAR.

In [11], an MES technique based on Principal Component Analysis in combination with Hidden Markov Models was proposed to recognize finger gestures with a laparoscopic gripper tool. Similarly, in [12], MES was combined with SVM to recognize 8 ambulation and daily activities. This approach, however, was easily outperformed by the baseline of the Naïve Bayes.

C. *Deep Learning*

Recently, Deep Learning methods have become popular [13]. The deep learning approaches can be categorized into generative approaches, discriminate approaches and hybrid approaches.

In the case of generative deep learning models, joint statistical distribution with associated classes are identified to model dependent or independent data distributions and high order correlations. Some generative approaches include *Deep Belief Networks*, *Deep Boltzmann Machines*, *Deep Autoencoders* , *Sparse Coding* , *Stacked Deep Gaussian models* etc.

In the case of discriminative deep learning models, posterior distribution classes are used to aid for classification tasks in activity recognition and classification. Some discriminative approaches include *Convolutional Neural Networks*, *Long Short Term Memory*, *Bi-LSTM* , *Gated Recurrent Unit*, *Manifold Elastic Network* etc. CNNs and RNNs are the approaches that we have focused the base of the project on.

In the case of hybrid deep learning models, a combination of discriminative, generative approaches or both are used to capture robust features for HAR. Several approaches indicate the use of CNNs in combination with other generative or discriminative methods seem to be the most favorable for hybrid architectures. Some prominent hybrid approaches include *CNN and Restricted Boltzmann Machine*, *CNN and RNN* , *CNN and GRU* etc.

III. METHODS

A. *Datasets*

We selected a dataset specific to HAR from the UC Irvine Machine Learning Repository - Human Activity Recognition Using Smartphones Data Set [14]. The original dataset was constructed through experiments carried out with 30 participants (aged 19-48 years) where each participant wears a Samsung Galaxy S2 smartphone with an accelerometer and a gyroscope, while performing the 6 activities of walking, walking upstairs, walking down-stairs, sitting, standing or laying. The smartphone collected 6 signals - 3-axial linear acceleration and 3 axial angular velocity measurements, each at a constant rate of 50Hz. The experiments were video recorded, and the response variables were manually labeled. The sensor signals were filtered using noise filters and sampled in fixed width time windows. Each individual observation in the dataset is a construction of sensor signals received within a 2.56 second interval sliding window containing 128 timesteps, with consecutive observations overlapping by 50% in time. The acceleration signal was separated into gravitational and body motion components using a 3^{rd} order low-pass Butterworth filter with a cut-off frequency of 0.3 Hz. The filtered data is, then, processed to construct features using statistical metrics (mean, signal magnitude area, standard deviation, signal frequency, entropy, etc.) from the accelerometer signals in the time and frequency domain. The hand-engineered data is split into training and test in a 70/30 ratio. In this case, each line of the dataset is a 561-d vector representing the summary statistics across a period of 2.56 seconds. The filtered dataset has 9 (each for x, y and z-axis of the 3 sensor signals) splits, where each line in 1 out of these 9 files is a 128-d vector representing the low-pass filtered signal in a time window. Finally, each

record in the dataset has triaxial total acceleration and body acceleration measurements, triaxial angular velocity measurements, a 561-d feature vector with time and frequency domain variables, an activity label and a subject identifier.

We also make use of the updated version of this dataset [15] that provides the raw inertial signals from the sensors and includes labels of postural transitions between activities (making the number of labels from 6 to 12).

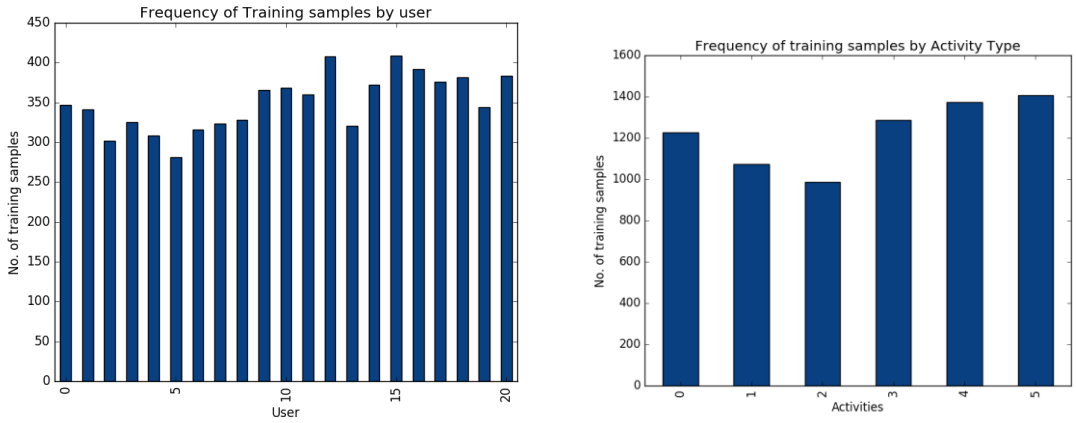

Figure 1: Dataset Statistics

B. Baseline

Since, the HAR problem can be formulated as a multiclass prediction problem, we started by establishing a baseline based on traditional statistical classifiers like Gaussian Naïve Bayes, k Nearest Neighbors, Decision Trees and SVM. As expected, SVM performed better than the rest of the classifiers as seen in Table 1.

Classifier	**Accuracy**
kNN	0.748558
Gaussian Naïve Bayes	0.724805
SVC (Linear)	**0.769596**
Decision Tree	0.722090

Figure 2: Classifier comparison

The SVM algorithm is a supervised learning model aids in classification and regression analysis. It was originally proposed as a binary classifier but has been adapted for multiclass problems using different schemes like One-Vs-All (OVA) and One-Vs-One (OVO). For this task, we chose the OVA method due to high accuracy and less memory consumption (advantageous when used in a resource constrained hardware device) compared to OVO method. Several support vector machines are used to assign labels to instances in a multiclass SVM, where the labels are drawn from a finite set of several elements. In case of non-separable data, kernels are used to map the input data into a high dimensionality feature space in order to find the linearly separable boundary.

The Kernel trick is used to avoid expensive computations in a high dimensional feature space by formulating an algorithm in terms of feature vectors dot products. Thus, kernel functions are used for feature vector computations instead of explicit computations. We tried several kernels like the polynomial kernel, the Radial Basis Function (RBF) kernel and the linear kernel with the linear kernel giving the best results out of the three. Linear kernel is a kernel represents the similarity of training samples in a feature space over linearly separable data.

The linear kernel function is given by:

$$k(x, y) = x^T . y$$

The polynomial kernel function is given by:

$$k(x_i, x_j) = (x_i . x_j + 1)$$

The RBF kernel function is given by:

$$k(x, y) = e^{-\frac{||x-y||^2}{2\sigma^2}}$$

We evaluated the SVM classifier over the hand-crafted feature dataset (with 561 features) and the accuracy came out to be ~95%. Thus, with handcrafted features an SVM can easily be used for HAR. Feature selection techniques like Principal Component Analysis and feature elimination methods like

C. Neural Networks

a. Spatial Model

Convolutional Neural Networks (CNNs)

For the CNN classifier, we first tested the data on a simple 2 hidden layer network to get a baseline for our CNN results. We used relu activation for each convolution layer and took the softmax for the output layer. Originally, we tried a shallow network with 50% dropout and a max-pooling layer, but optimizing the architecture with cross-validation, we found that batch normalizing the data before the first convolution and using convolutions with kernels of size 3 and output sizes of 32 gave the best result. We tested our models on 10 iterations to get average accuracy and the standard deviation in accuracy. The original architecture had an accuracy of 90.5% and 0.5% standard deviation, while our optimized architecture had an accuracy of 91.5% and 0.9% standard deviation.

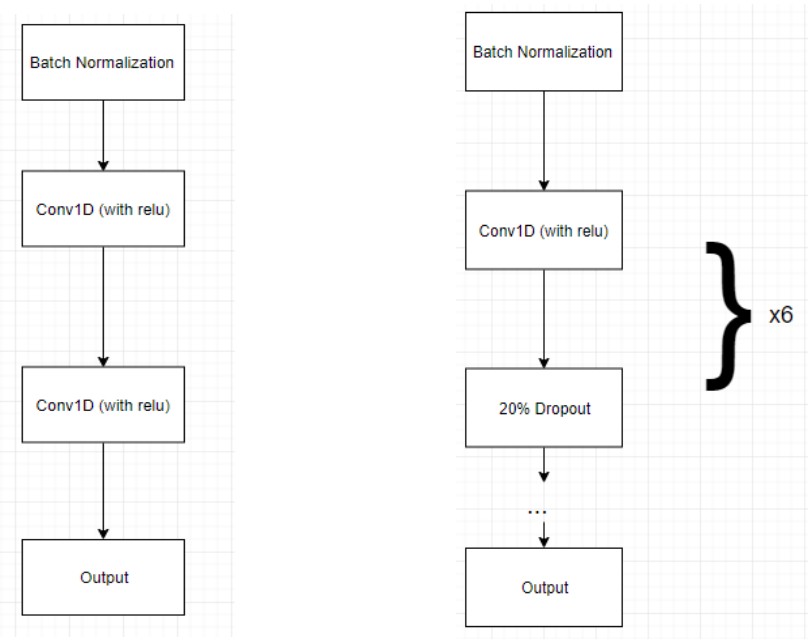

Figure 3: CNN Architecture: Shallow and Deep

We then tried to improve our classifier by making our network deeper and using an architecture better suited for time series data. One recently developed (2016) CNN architectures used for time series data is WaveNet [16]. The basic idea behind Wavenet is to look at more data points faster so that we can see all the input data at once instead of just looking at a local feature. WaveNet does this by exponential dilation which allows deep networks to take input from an exponentially wide receptive field. To implement WaveNet in keras, we simply increased the dilation rate exponentially at each layer (dilation rate = 2^(n-1), where n is the depth of the layer). Our final architecture had 6 convolution layers, all with relu activation, and 20% dropout between each convolution layer. Again, we batch normalized the data before the first convolution and took the softmax of the output. To keep more features early on, we increased the first 2 convolution outputs to size 128, and then halved the output size every 2 layers. The kernel size remained at 3 for each convolution to allow maximum depth while still taking meaningful convolutions. Testing our models on 10 iterations, we were able to improve on our baseline results, getting average accuracy of 93.2% and 0.7% standard deviation. Testing this network without the exponential dilation (ie not WaveNet architecture), we only got an average accuracy of 91.3% and 1.7% standard deviation.

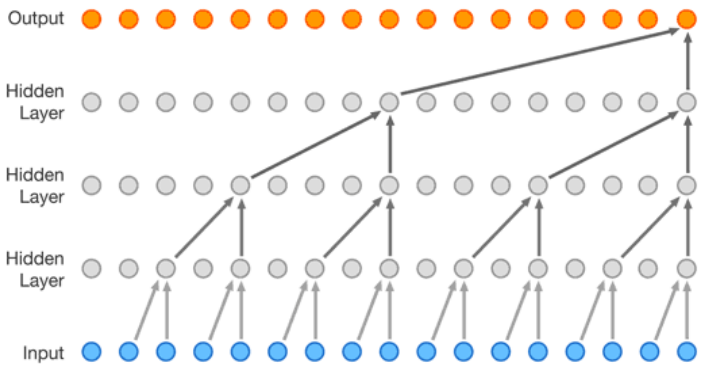

Figure 4: WaveNet

During the process of hyper-parameter optimization, we were getting accuracies around 92% for deep networks without WaveNet implementation, so the decreased accuracy from the shallow CNN to the deep CNN without WaveNet implementation was an interesting anomaly. It is likely that performance between WaveNet models and regular models are slightly negatively correlated, since WaveNet models will perform better when the hyper-parameters help look at as much data as possible, while the regular models perform better when we better refine the local features we are looking at. While we obviously saw an improvement using deeper networks and the WaveNet model, our normal shallow CNN performed much better than we expected and even outperformed our RNNs. A possible reason for this is the nature of our data. While our data is time series, we can see certain features such as a large positive z acceleration when walking up the stairs or a large negative z acceleration while walking down the stairs. These features will be local spikes in the data that convolutions can pick out very well. Thus normal CNNs will be able to classify reasonably well based on these local features. Another possible reason is that the time series data being small enough (128 samples) that the CNN can see enough of it at once to classify data. Even with the good performance from a simple shallow CNN, we were able to improve upon previous results to obtain an accuracy of 93.1% with our WaveNet CNN.

b. Spatio-temporal Model

Recurrent Neural Networks (RNNs)

RNNs is a family of neural networks used to model sequential data such as time series or raw sensor data. They use a temporal layer to capture sequential information and employs high dimensional hidden states units with non-linear dynamics to learn complex changes. These hidden unit cells can adapt according to the information available to the network which is updated regularly reflecting the current status of the network. The current hidden state is computed by using the activation of the previous hidden state to predict the next hidden state. However, RNNs suffer from training difficulties (memory bandwidth for computations) and vanishing or exploding gradients. This limits the use of vanilla RNNs for modelling long time activity sequences and sensor temporal dependencies. The LSTM variant of RNN allows to capture long temporal activity sequences using a variety of gates and memory cells (capable of storing contextual information). The use of memory cells in the form of input gates, function gates and

output gate with learnable weights allow LSTMs to capture temporal dependencies and global features in sequential data and increase the activity recognition accuracy.

LSTM

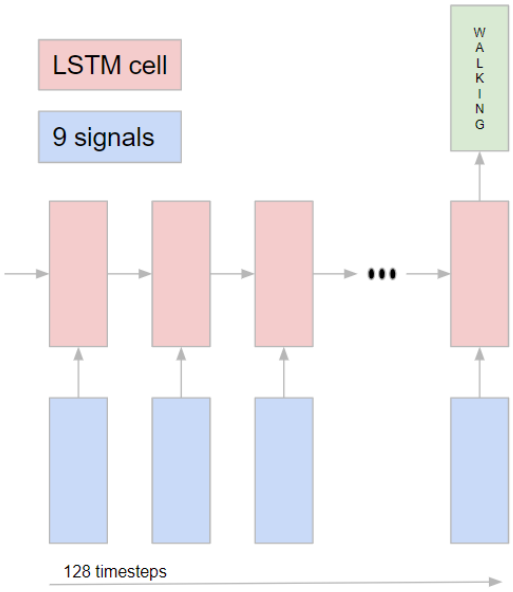

Figure 5: LSTM Model

An LSTM network is a type of RNN capable to learn and capture long input sequences (consisting of 200 to 400 timesteps). Multiple parallel sequences of input data (in our case signals from each axis of the accelerometer and gyroscope data) are fed into the LSTM and the model learns to extract features from sequences of observations mapping the internal features to activity types. LSTMs aid in sequence classification tasks by automatically learning from the raw time series data avoiding the need for domain expertise and hand-crafted input features. The model can therefore, learn an latent internal representation of the time series data and can even perform comparable to models fit on the dataset with engineered features. We used a many-to-one architecture. There are 9 variables for each time step and one row of data has 1152 (=128 * 9) elements which are reshaped to form an input layer. Since there are 561 hand-crafted features for the same row, there is a possibility of redundancy in these elements. The output layer produces a six-element vector containing the probability of belonging to an activity type for a given time window. The model is sequential in nature (does not share layers) and consists of a single LSTM hidden layer followed by a dropout layer (to reduce overfitting), a dense fully connected layer for feature extraction and finally, a Softmax layer to make predictions. The optimizer for the network is Adam and the loss function used is categorical cross entropy as the problem is a multi-class classification problem. Since, we want the LSTM to extract features across the timesteps within a window and not across the window, we only shuffle the windows during training.

The architecture of the model is given in the Figure above.

CNN-LSTM

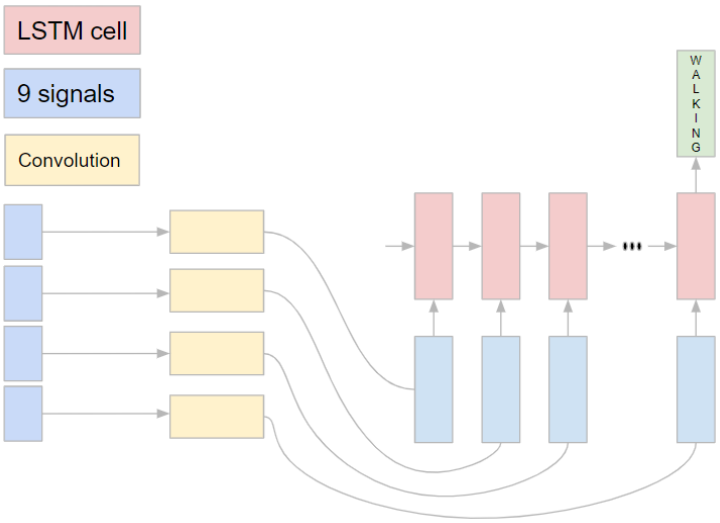

Figure 6: CNN-LSTM Model

Since the input data has a spatial structure (sensor data) in addition to the temporal structure, the Vanilla LSTM may not produce great results. An architecture like CNN-LSTM is specific to sequence problems with spatial data. They are both spatially and temporally deep in nature. The CNN-LSTM architecture uses 1D Convolutional layers for feature extraction on input data in combination with LSTMs for sequence prediction. CNN-LSTMs are usually employed for problems involving activity recognition (activity text description generation from a sequence of images), generating image descriptions and generating text description from videos. The CNNs in the model reduce the spectral variation of the input feature, passes this to LSTM layers for temporal modeling, and finally outputs this to Dense layers, producing an easily separable feature representation. Subsequences of the main sequence are read by the model as blocks and feature extraction is performed on each block. The extracted features from each block are then interpreted by the LSTM. Each window of 128 timesteps is split into 4 subsequences of 32 timesteps for the CNN model. The extracted features are flattened and provided to the LSTM model. The LSTM model then further extracts its own features and finally maps the sequence to an activity. Two consecutive CNN layers are used, followed by a dropout layer and a max pooling layer. This is followed by feature flattening, an LSTM network with a dropout and a dense layer. The final layer is again a Softmax layer.

The architecture of the model is given in the Figure above.

ConvLSTM

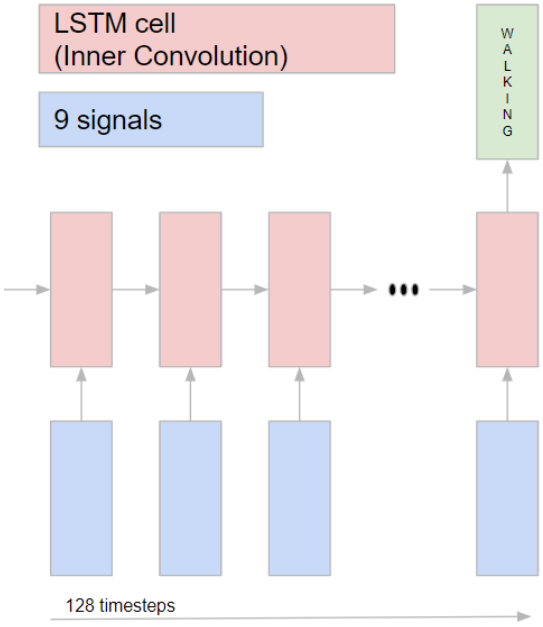

Figure 7: ConvLSTM

Another extension of the CNN-LSTM is to use convolutions inside LSTM cells instead of inner dot products. This architecture is called a ConvLSTM and is used for spatio-temporal data. An LSTM reads the data directly to calculate internal hidden state and state transitions and a CNN-LSTM interprets the output from CNN models which is, then, used by the LSTM. However, in a ConvLSTM model, the convolutions are directly used as part of reading inputs into the LSTM cells themselves. A sequence is again divided into 4 subsequences of 32 timesteps. The input to a ConvLSTM layer is shaped in the format of (samples, timesteps, rows, columns, channels). Here each timestep is defined in terms of (rows*columns) datapoints. In our case, the input was formulated as n samples (number of time windows), 4 timesteps (number of subsequences in a time window of 128 timesteps), 1 row (for 1D shape of the subsequence), 32 columns (for the 32 timesteps in each subsequence) and 9 channels (for the 9 features). Kernel size used is (1,3) and activation function used is ReLU. Using convolutional layers for state-to-state transition is essential to capture spatio-temporal patterns in the data instead of fully-connected layers. The architecture of the model is given in the Figure above.

Structured Model (LSTM-CRF)

We also implemented a hybrid structural model combining a bidirectional LSTM model and a Conditional Random Fields (CRF) model. This model is a state-of-the-art in sequence tagging tasks for NLP. In conditional random fields, we model a conditional probability $P(o_1, o_2, \dots, o_m | x_1, x_2, \dots x_m)$ of the output state o given an input sequence x by defining a feature map $\phi(x_1, .., x_m, o_1, \dots o_m) \in R^d$ which maps an entire input sequence to an entire state sequence to d-dimensional vector. The scoring of how well the state sequence fits the input sequence can be given as:

$$P(o|x; w) = \frac{e^{score_crf(x,o)}}{\sum_i e^{score(s,s^i)}}$$

where s^i ranges over all possible output sequences and $w \in R^d$ is the parameter vector. In LSTM-CRF we replace the linear scoring function with a non-linear neural network like LSTM. Following architecture shows the model layers in detail:

```
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        (None, 128, 6)            0
_________________________________________________________________
bidirectional_10 (Bidirectio (None, 128, 100)          22800
_________________________________________________________________
bidirectional_11 (Bidirectio (None, 128, 200)          160800
_________________________________________________________________
time_distributed_8 (TimeDist (None, 128, 50)           10050
_________________________________________________________________
crf_7 (CRF)                  (None, 128, 12)           780
=================================================================
Total params: 194,430
Trainable params: 194,430
Non-trainable params: 0
```

However, the LSTM-CRF model performed poorly in comparison to the previous models for HAR. A future area would be to look into GRUs instead of LSTMs with CRFs

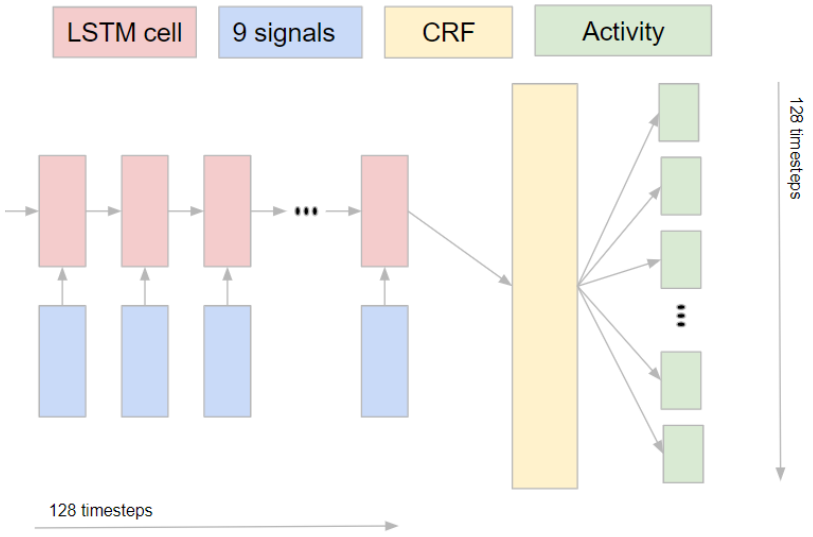

Figure 8: LSTM-CRF

D. Evaluation Metrics

As the HAR problem is formulated as a multiclass prediction problem, we use the values from a confusion matrix C_{nXn} where n is the number of classes to compare the performance of different models. An element C_{ij} of the matrix is the number of samples in class i that were classified as class j. The values obtained from the matrix include *True Positives* or *TP* (number of positive instances classified as positive), *True Negatives* or *TN* (number of negative instances classified as negative), *False Positives* or *FP* (number of negative instances classified as positive) and *False Negatives* or *FN* (number of positive instances classified as negative). The following metrics (usually used for binary classification) are generalized for n classes where an instance could be positive or negative for a particular class.

The accuracy of the model is given as:

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

The precision of the model is the ratio of correctly classified positive instances to the total number of instances classified as positive and is given by:

$$Precision = \frac{TP}{TP + FP}$$

The recall of the model is the ratio of correctly classified positive instances to the total number of positive instances and is given by:

$$Recall = \frac{TP}{TP + FN}$$

The F1-score of the model combines precision and recall in a single measure and is given by:

$$F1 - score = 2.\frac{Precision.Recall}{Precision + Recall}$$

E. Mobile Application

In order to perform real time HAR, we tested a few of our models on the Android platform by creating an Android application. Most modern Android smartphones have built-in Motion Sensors capable of monitoring the motion of a device (accelerometer, gravity sensor, and gyroscope) and providing raw sensor data. The Sensor framework allows us to access sensors of different types based on architecture – hardware-based and software-based sensors. Accelerometer and gyroscope are hardware-based sensors and they derive their data from direct measurements of environmental parameters (acceleration, angular change etc.). Software-based sensors are availability dependent and relay on one or more hardware sensors to derive data. The gravity, linear acceleration, rotation vector, step counter, step detector can be software-based or hardware-based depending upon the specific device.

All the motion sensors return multi-dimensional arrays of float sensor values for each *SensorEvent*. We make use of two sensor types namely – TYPE_GYROSCOPE and TYPE_LINEAR_ACCELERATION. The accelerometer measures the acceleration force applied to a device on the physical x, y, and z axes, including gravitational force. The raw data returned from the accelerometer is in the format of vectors ($Acc_i =< x_i, y_i, z_i >$). The gyroscope adds to the accelerometer information by tracking rotation and is used to measure angular velocity using earth's gravity to ascertain orientation. The raw data from the gyroscope is the rotation (around the 3 axes) rate in rad/s ($Rot_i =<$

$x_i, y_i, z_i >$. We used an IIR filtering library in Java, iirj [], to process the raw data from the sensors In order to match the signals used for training. A 3rd Order Low-pass Butterworth filter was with a cutoff frequency of 0.3 Hz and sampling frequency of 50Hz was used.

Google's TensorflowLite framework and the Google TensorFlowInferenceInterface library were used to create a prototype application capable of performing real time HAR. The Keras models created were first converted to a *protocol buffer* or *.pb* format from the *.h5* format (using Open source tools and scripts like keras2android [] and keras_to_tensorflow) and model was fed with real time sensor data from the device in a compatible format to get the predicted activities. Following are the wireframes and current progress of the Android application we hope to build as a prototype demonstrating HAR systems for personalized fitness guidance:

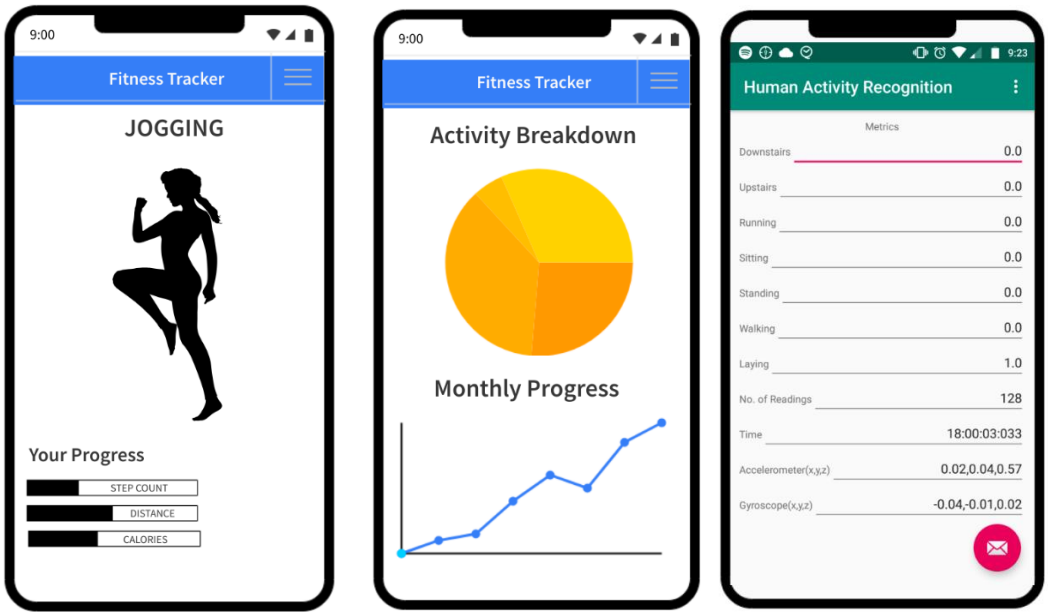

Figure 9: Mobile Application - Wireframes and App

IV. RESULTS

The following table indicates the evaluation metrics for the models we implemented for this project

Model	F1 Score	Precision	Recall
SVM	0.5834	0.5876	0.5917
SVM (feature engineered)	0.96	0.96	0.96
Simple CNN	0.91557 (+/-0.00804)	0.91689 (+/-0.00727)	0.91558 (+/-0.00831)
WaveNet CNN	0.93006 (+/-0.00593)	0.93129 (+/-0.00543)	0.93023 (+/-0.00609)
LSTM	89.45 (+/- 0.33)	89.01 (+/- 0.88)	89.66 (+/- 0.67)
CNN-LSTM	90.21% (+/- 0.34)	90.43% (+/-1.04)	89.65% (+/- 0.96)
ConvLSTM	90.65 (+/- 0.23)	90.81% (+/- 0.77)	90.26 (+/- 0.47)
LSTM-CRF	Accuracy-82%		

Figure 10: Evaluation scores for different models

The following plots give an idea of the accuracy vs epochs for different spatio-temporal models

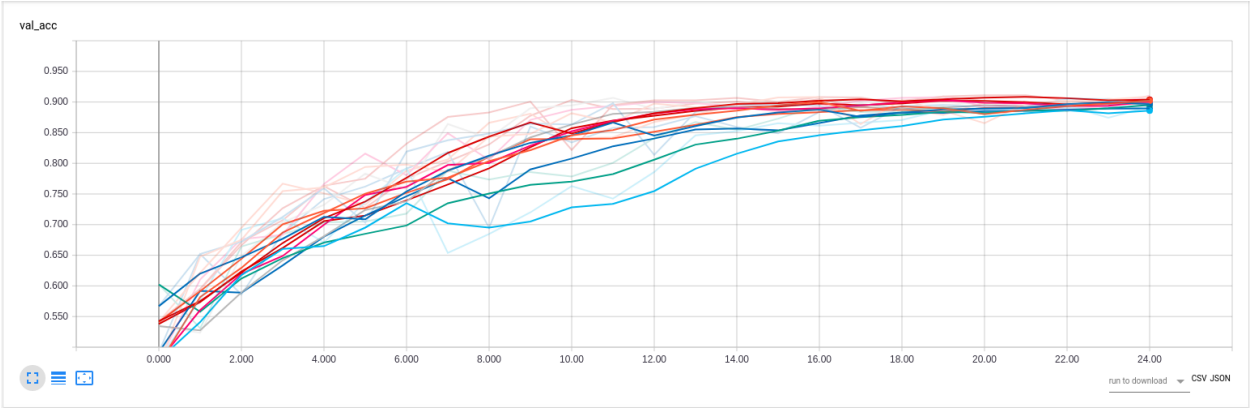

Figure 11: LSTM Accuracy vs Number of Epochs

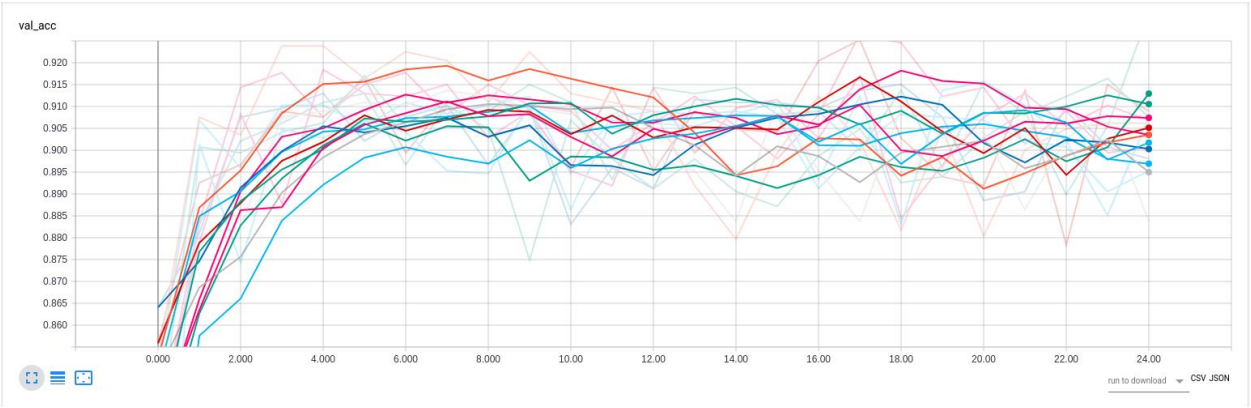

Figure 12: CNN-LSTM Accuracy vs Number of Epochs

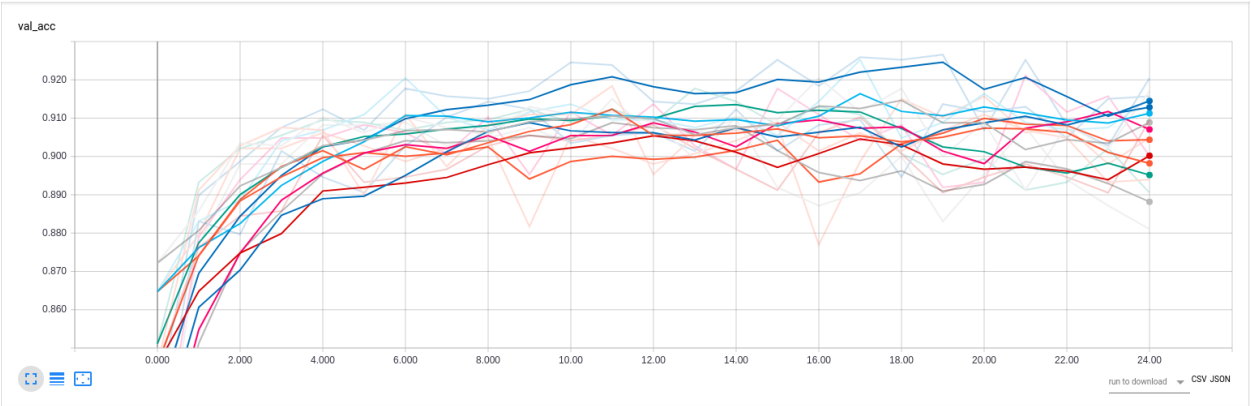

Figure 13: ConvLSTM Accuracy vs Number of Epochs

V. DISCUSSION

As seen from the above results, hybrid deep learning models like CNN-LSTM and ConvLSTM are well suited for the HAR task and achieve comparable results to CNN architectures. Applying SVM on the feature engineered dataset resulted in highly accurate models. All the deep learning models performed better than the baseline of SVM. LSTM-CRF performed poorly in comparison to the convolutional hybrids with RNNs.

The WaveNet architecture was able to improve on the baseline CNN results, getting average accuracy of 93.2% and 0.7% standard deviation. However, without the exponential dilation (ie not WaveNet architecture), we only got an average accuracy of 91.3% and 1.7% standard deviation. Overall, the temporal models performed better than the non-temporal models and the deep learning models outperformed their shallow counterparts.

VI. Team Roles

Team Member	Work
Ankur Roychowdhury	EDA, Implemented LSTM, CNN-LSTM, ConvLSTM,
Saagar Minocha	EDA, Implemented Baseline, LSTM-CRF, Android
Joshua Crockett	Implemented CNN, WaveNet

VII. Conclusions and Future Work

Many of the results we obtained were as expected, such as neural networks improving with deeper networks and doing better than SVM on raw data. While we did not quite reach the same accuracy as SVM on engineered data for the neural networks (NNs using only raw data), we were within 3% for the WaveNet model. Some unexpected results were the better performance of the CNNs over the RNNs. Since RNNs work better than CNNs on temporal data, we would expect our RNNs to do at least as well, if not better. As discussed before, this can likely be attributed to the nature of the problem. Several features in the data will be local spikes, like acceleration going up and down stairs. Thus the CNNs can learn these features better than the RNNs and therefore better classify them. Future work will be to better understand these features and use them to improve our RNN and CNN models. We would also like to continue the work on our mobile application to detect activity in real time and notify users when there are long stretches of sitting. We would also like to try to expand our model to include more activities that we can recognize such as running or jogging. We could then expand our application to more health applications and provide an useful monitor to users' well-being.

References

[1] L. C. Jatoba, U. Grossmann, C. Kunze, J. Ottenbacher, and W. Stork, "Context-aware mobile health monitoring: Evaluation of different pattern recognition methods for classification of physical activity," in *30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 5250–5253, 2008.

[2] E. M. Tapia, S. S. Intille, W. Haskell, K. Larson, J. Wright, A. King, and R. Friedman, "Real-time recognition of physical activities and their intensities using wireless accelerometers and a heart monitor," in *Proc. International Symposium on Wearable Computers*, 2007. [3] I.S. Jacobs and C.P. Bean, "Fine particles, thin films and exchange anisotropy," in *Magnetism*, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.

[3] L. Bao and S. S. Intille, "Activity recognition from user-annotated acceleration data," in Pervasive, pp. 1–17, 2004.

[4] Z.-Y. He and L.-W. Jin, "Activity recognition from acceleration data using ar model representation and svm," in International Conference on Machine Learning and Cybernetics, vol. 4, pp. 2245–2250, 2008.

[5] C. Randell and H. Muller, "Context awareness by analysing accelerometer data," in The Fourth International Symposium on Wearable Computers, pp. 175–176, 2000.

[6] O. D. Lara, A. J. Perez, M. A. Labrador, and J. D. Posada, "Centinela: A human activity recognition system based on acceleration and vital sign data," Journal on Pervasive and Mobile Computing, 2011.

[7] D. Minnen, T. Westeyn, D. Ashbrook, P. Presti, and T. Starner, "Recognizing soldier activities in the field," in 4th International Workshop on Wearable and Implantable Body Sensor Networks, vol. 13, pp. 236–241, Springer Berlin Heidelberg, 2007.

[8] M. Stikic, D. Larlus, and B. Schiele, "Multi-graph based semisupervised learning for activity recognition," in International Symposium on Wearable Computers, pp. 85–92, 2009.

[9] M. Stikic, D. Larlus, S. Ebert, and B. Schiele, "Weakly supervised recognition of daily life activities with wearable sensors," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, no. 12, pp. 2521–2537, 2011.

[10] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in Proc. eleventh annual conference on Computational learning theory, pp. 92–100, ACM, 1998. [6] O. D. Lara, A. J. Perez, M. A. Labrador, and J. D. Posada, "Centinela: A human activity recognition system based on acceleration and vital

sign data," Journal on Pervasive and Mobile Computing, 2011.
[11] A. Ali, R. King, and G.-Z. Yang, "Semi-supervised segmentation for activity recognition with multiple eigenspaces," in International Summer School and Symposium on Medical Devices and Biosensors, pp. 314–317, 2008.
[12] T. Huynh and B. Schiele, "Towards less supervision in activity recognition from wearable sensors," in 10th IEEE International Symposium on Wearable Computers, pp. 3–10, 2006.
[13] Henry Friday Nweke, Ying Wah Teh, Mohammed Ali Al-Garadi, and Uzoma Rita Alo. Deep learn-ing algorithms for human activity recognition using mobile and wearable sensor networks: Stateof the art and research challenges.Expert Systems with Applications, 2018.
[14] JL Reyes-Oritz, Davide Anguita, Alessandro Ghio, Luca Oneto, and Xavier Parra. Human activ-ity recognition using smartphones data set.UCI Machine Learning Repository; University ofCalifornia, Irvine, School of Information and Computer Sciences: Irvine, CA, USA, 2012.
[15] Jorge-L. Reyes-Ortiz, Luca Oneto, Albert SamÃ , Xavier Parra, Davide Anguita. Transition-Aware Human Activity Recognition Using Smartphones. Neurocomputing. Springer 2015.
[16] Van Den Oord, Aäron, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. "WaveNet: A generative model for raw audio." In SSW, p. 125. 2016.

RESOURCES

https://github.com/amir-abdi/keras_to_tensorflow
https://gist.github.com/fsausset/57b99a3db5e1a05569845894ec385eef
https://www.tensorflow.org/lite/
https://machinelearningmastery.com/how-to-develop-rnn-models-for-human-activity-recognition-time-series-classification/
https://github.com/Harshad-Pardeshi/Human-Activity-Recognition
https://www.depends-on-the-definition.com/sequence-tagging-lstm-crf/