Real-Time Human Activity Recognition using Convolutional Neural Network on Time Series from Mobile Sensors

Richard Sicoli, Zhe Lin
CSE 570 - Final Report
Stony Brook University
12/19/17

Abstract

Many health related applications such as Google Fit and Apple Health are becoming increasingly popular due to the large number of sensors found in mobile phones and smartwatches today. These embedded sensors allow mobile devices to track how many steps a user takes or even track a user's activity throughout the day.

However, detecting the actions a user is taking from mobile sensors can be a challenging task due to the large variability in motion when performing an activity. Traditional methods typically use manual feature design and heuristics. Our model, however, uses the power of deep convolutional neural networks to automatically learn important features directly from the raw time series signal captured from the mobile sensors. To demonstrate this, we have created a custom dataset from recording the motion of eight activities with a phone placed in a user's pocket. We show that our model achieves remarkable results on classifying each activity. Furthermore, we demonstrate the capability of real-time activity recognition by running the model locally on an iOS device.

1. Introduction

Convolutional neural networks have seen remarkable results in fields such as Computer Vision due to the recent surge in GPU performance [1]. We decided to apply them in a similar manner but to 1D time series data with multiple feature channels. This would allow us to capture a large amount of raw data and feed it directly into the model with very little preprocessing and no feature-engineering required. Instead, the convolutional neural network would learn the features that differentiate the various activities.

After training the model on our dataset of 8 activities, we evaluated the model and then loaded it on to an iOS device for local real-time activity recognition. In this paper, we will look at the data collection methods and network architecture. We will also look at the evaluation of the model on our test set as well as the performance of the model running in real-time on an iOS device.

1.1 Setup

Training was done on a desktop PC using Tensorflow/ Keras for python using a GTX 1070 GPU. Raw motion data was collected using Core Motion API on iPhone 6 using Objective-C. The model was loaded on the iOS device using Apple's hardware accelerated Core ML framework which was just introduced this year.

2. Dataset

The dataset was created from recording the motion of 3 users doing eight activities at a variety of locations. Multichannel time series data was recorded from the three-axis accelerometer of an iPhone 6. We experimented with including the rotational data from the gyroscope but found that the accelerometer alone was sufficient. So we opted to just use the accelerometer to reduce the dimensionality of the data during training.

Unlike some previous experiments which placed the iOS device in a custom harness [2], we decided to simply place the device in the pants pocket, which is a less controlled setting. The initial orientation of the device was assumed to be constant. Below is a list of all eight activities.

- Walking
- Jogging
- Sitting
- Standing
- Going Downstairs
- Going Upstairs
- Jumping
- Cycling

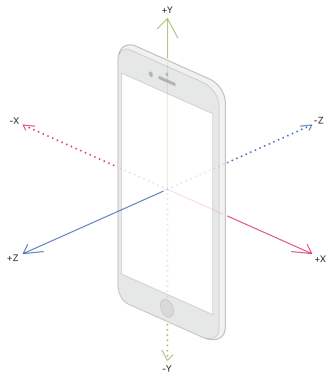


Figure 1: Three-axis accelerometer used for measuring change in velocity along each axis.

Each activity was recorded for a sum of 20 minutes at a sampling rate of 60 Hz for a total of 160 minutes and 576,000 3D sample points. Data was then normalized to zero mean and unit variance. Then a one second window was applied over the time series data to create 9600 unique 3D segments or 1200 3D segments per activity. Finally the segments were randomly shuffled and uniformly distributed into a 0.60, 0.20, 0.20 split for training, validation and testing respectively.

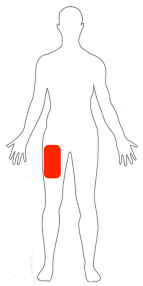


Figure 2: The mobile device was placed in the pants pocket with a constant initial orientation.

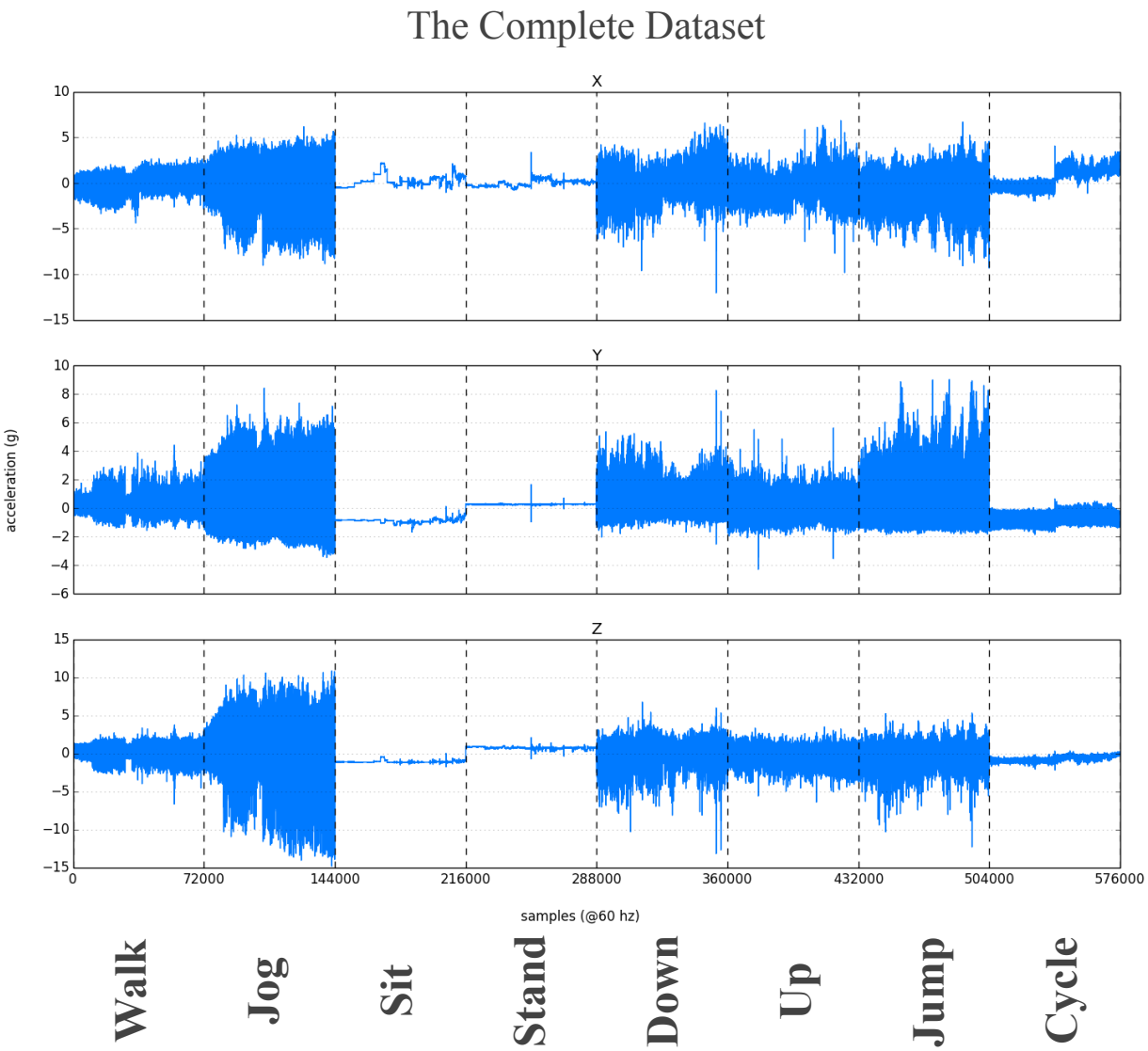


Figure 3: A plot of every sample point in the dataset for each channel. Each activity was recorded for a sum of 20 minutes at a sampling rate of 60 Hz for a total of 160 minutes and 576,000 3D sample points. The data is marked with separation lines to show where each activity is. Note the data in this plot is normalized to have zero mean and unit variance which is why the acceleration is centered at 0.

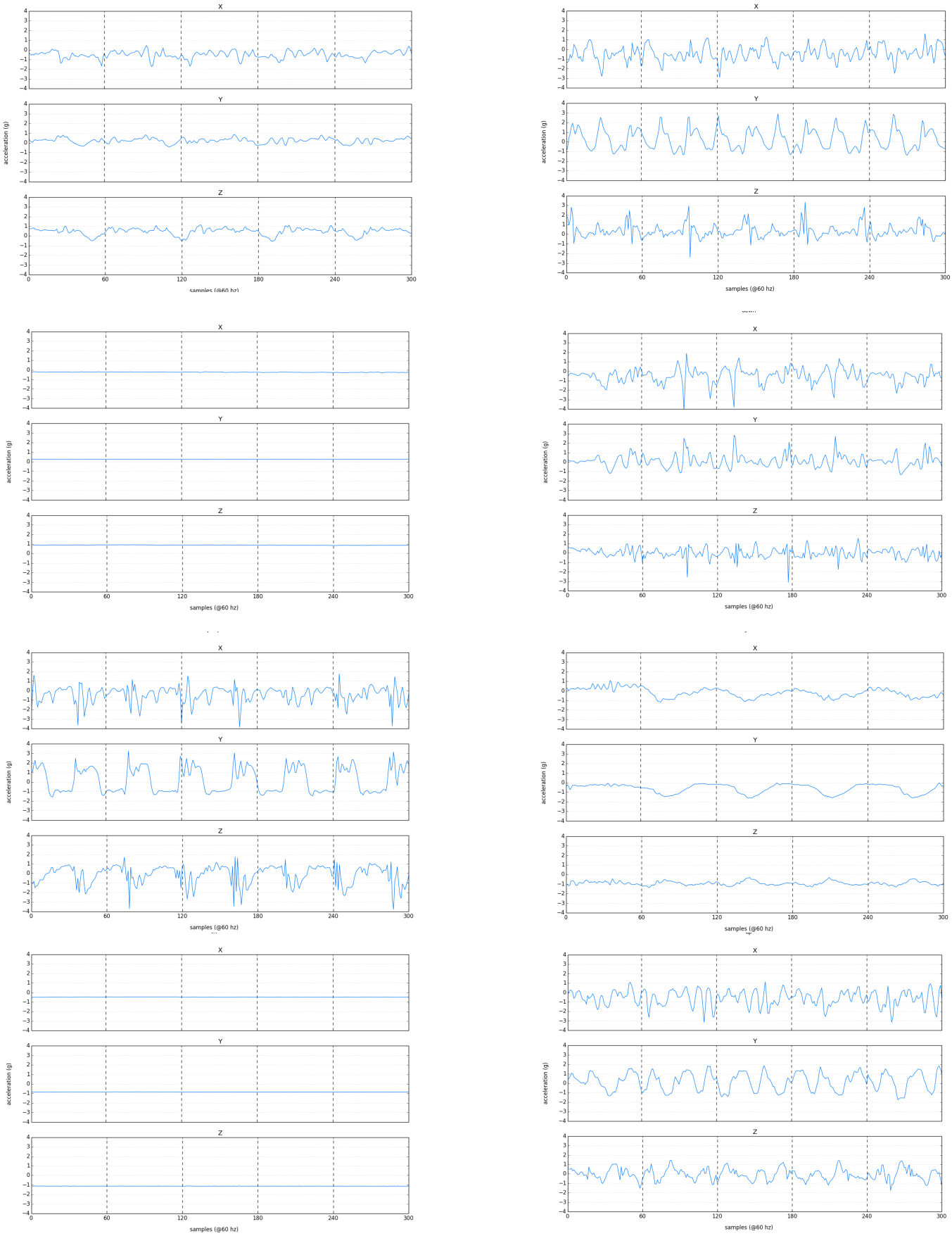


Figure 4: A close look at the time series for each activity in order to better visualize the patterns. Each dashed line corresponds to the one second window (60 Hz) we use to create segments. So the model only ever sees a small fraction of this signal. Note the ordering from left to right going down each row: Walk, Jog, Stand, Down-stairs, Jump, Cycle, Sit ,Up-stairs.

3. Network Architecture

Our architecture uses a standard deep convolutional network design. Convolutional neural networks are useful for (1) capturing local dependency of the time series signals and (2) being invariant to displacements in the signal. Our convolutional neural network layers use 64 filters with a window size of three with strides of one and a rectified linear unit (ReLU) for the activation function. We use multiple convolutional layers so the model can learn more complex signal pattern abstractions from the earlier feature map layers.

Furthermore, we downsample by appending a max pooling layer after each convolutional layer to further increase invariance to distortions. We then flatten the final feature maps to a 512 length vector which represents the high level features and connect them to a fully connected layer followed by a dropout layer with a rate of 0.5 to help reduce overfitting. Finally we connect the output layer which we use as a softmax classifier for the eight classes. See figure 5 for the exact details of the network architecture.

3.1 Training

Our model was built and trained using Keras/ TensorFlow. For the loss function we use categorical cross entropy with stochastic gradient descent (SGD). Furthermore we use a batch size of 32 and a learning rate of 0.01.

As discussed earlier, we randomly shuffled and uniformly distributed the dataset into a 0.60, 0.20, 0.20 split for training, validation and testing respectively. The model was trained on 5,760 60x3 segments. Each segment represents one second (60 Hz) of 3 channel time series data.

Our model began to converge fairly quickly after about 50 epochs. Once we were satisfied with our results on the validation set, we moved on to testing.

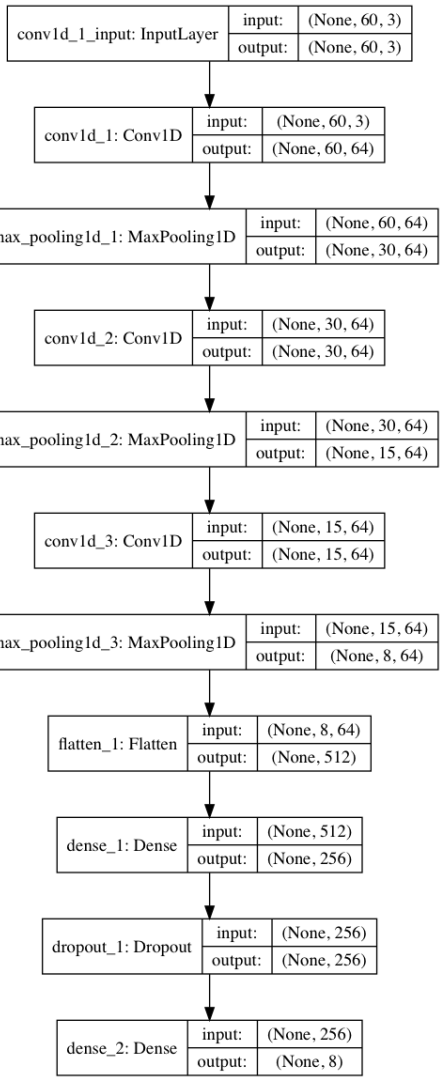
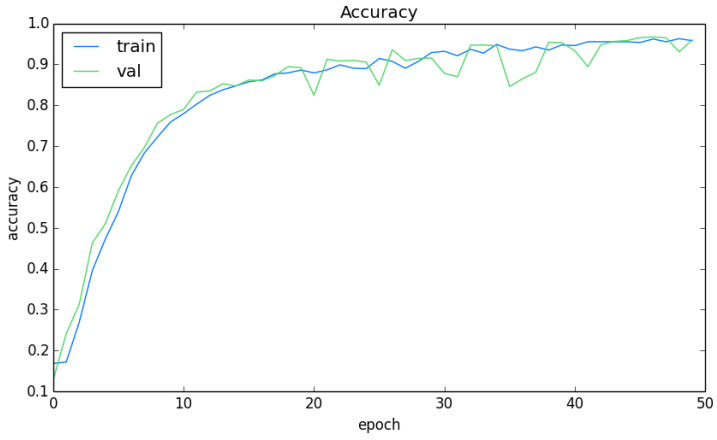


Figure 5: Details of the network architecture. Note that we used padding which is why the length of the signal remains the same after convolution.



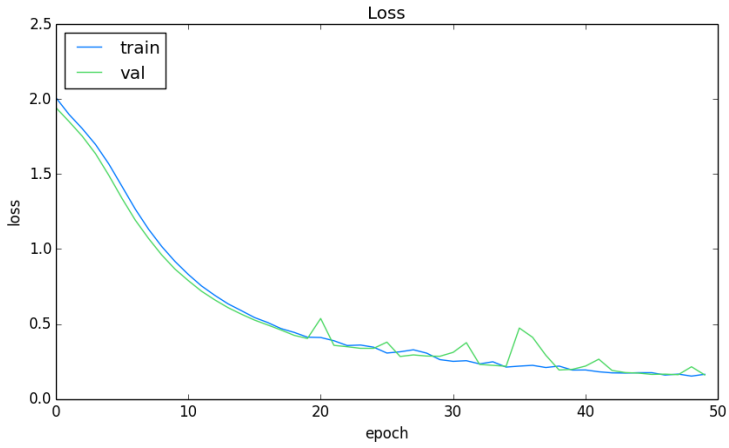


Figure 6: Accuracy and loss during training. As shown above, our network begins to converge after 50 epochs. The network also shows impressive results on the validation set.

4. Evaluation

After tuning our model on the validation set, we ran the model one time on the test set and report the results below. Note that our model was tested on 1920 3D segments which is 240 3D segments per class. We report an overall accuracy of 0.974. Overall we are very pleased with the results of our model. Below is a confusion matrix for specific details.

According to the data in the confusion matrix, upstairs and downstairs were the hardest for the network to classify. It's also interesting how most of the bad predictions were predicted as walking. The reason for most of the misclassifications are most likely due to noise when recording data. For example, when recording an activity there were times when an activity may have been interrupted by obstacles or other pedestrians. This can skew the data and create distortions that the model can't resolve. Furthermore, since the phone was placed in the pocket, it may have shifted too much if in a loose pocket.

Despite these sources of uncertainty, the model still managed to learn the features to differentiate the activities with remarkable accuracy.

It is important to note that our dataset was collected from a small number of users (only three). And while the data was collected at a variety of times and locations, the model may have trouble generalizing if used by new users with different body shapes and different movement behaviors. This could be resolved by creating an even larger dataset with more users.

5. Mobile Application

We have developed a mobile application for both logging data (to create the dataset) and real-time activity recognition. The app was developed for iOS using Objective-C. Data is recorded from the accelerometer using the Core Motion API. The model runs on the iOS device using Apple's Core ML which was introduced just this year. Core ML is a framework for integrating pre-trained models into an application and allows the model to take advantage of hardware acceleration for optimal performance.

It is important to note that the model runs locally on the iOS device and no data is transferred over a network to a server. This allows the iOS device to process data in real-time and prevent the user's private information from leaving the device. When profiling our app, we found that continuously recording from the accelerometer at a rate of 60 Hz and feeding the data into the model resulted in no significant energy impact. This makes our App very efficient for recording activity data throughout the day.

The reason why our overhead is so low is twofold. (1) Our model handles very small data. It only needs to receive 1D segments of size 60 with 3 channels, as compared to multidimensional image data in a typical Computer Vision setting; the model itself is only 119 KB. (2) Our model takes advantage of the hardware accelerated and efficient Core ML framework released just this year and optimized specifically for iOS devices.

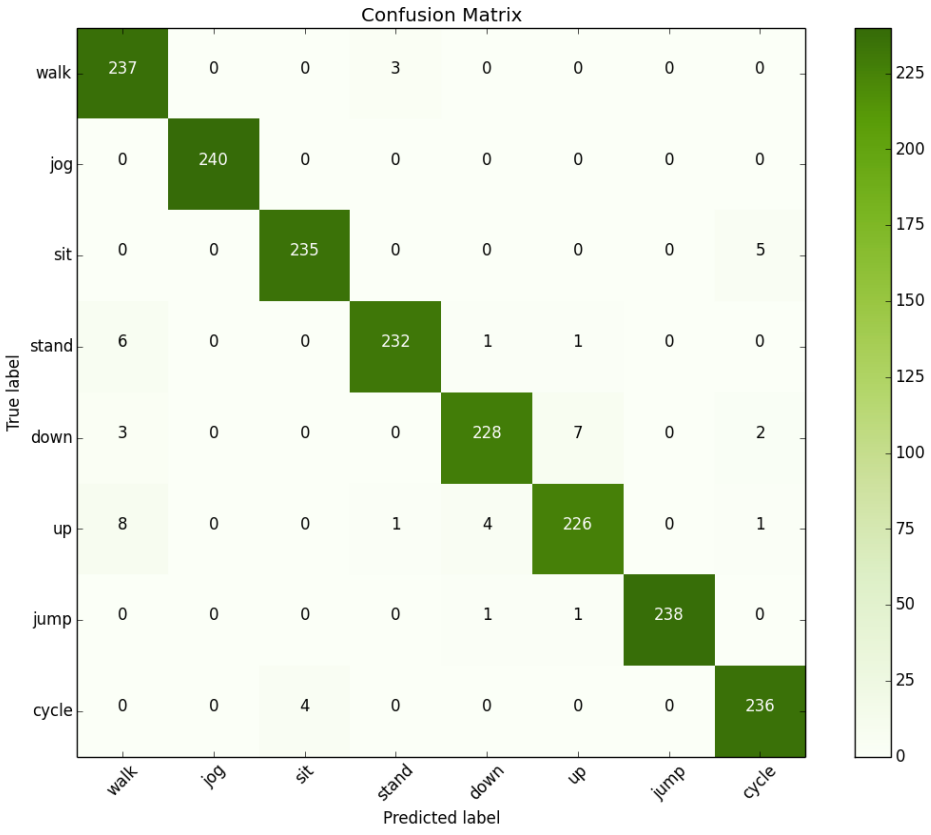


Figure 7: Confusion matrix of test results.

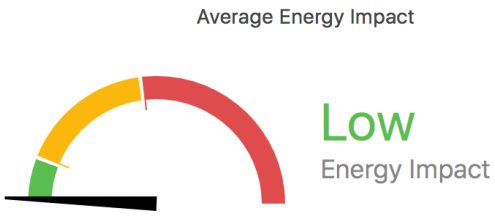


Figure 8: Low energy impact when continuously recording from the device sensors and feeding segments into model.

6. Discussion

In conclusion, we are very pleased with the results of our project. We accomplished exactly what we were hoping for when we started the project. Our model achieved remarkable results on our test set, and performed perfectly when demoed both at the poster session and in other real-world scenarios. As discussed earlier, the model was only trained by three users. And while we collected data from doing activities in a variety of locations and times, the model may still fail to generalize when tested on users with very different body shapes and motion behaviors. We hope to get the resources to run an even larger test with hundreds of users as this would drastically improve the model's ability to generalize. However, in reality this may not be necessary if we allow a user to train their own model on their own device. This would develop a personalized model specifically for the user, and the model would learn overtime from the user's actions.

We are also interested in looking into adding more activities and possibly gathering data from additional sensors like the gyroscope if needed. We also want to experiment with gathering motion data from smart watches as well which are becoming increasing popular. Human activity recognition on mobile devices can be immensely important to providing personal data to health and fitness based applications. We already see apps such as Google Fit and Apple Health integrating activity recognition into their applications. We believe having a real-time activity recognition model on a mobile device that can learn from a user overtime will be future of human activity recognition.

7. Contributions

We both worked together on the two main parts of the project: (1) the mobile application code (used for data logging and the realtime activity recognition) and (2) the deep learning code (used for training and evaluating the model in Tensorflow/Keras). We encountered numerous issues when implementing this project, so we were both actively assisting in debugging. We both spent time researching various aspects of the project as well such as Apple's iOS frameworks and deep learning concepts such as convolutions neural networks. We both contributed to recording data for the dataset. Lastly we both participated in the poster session.

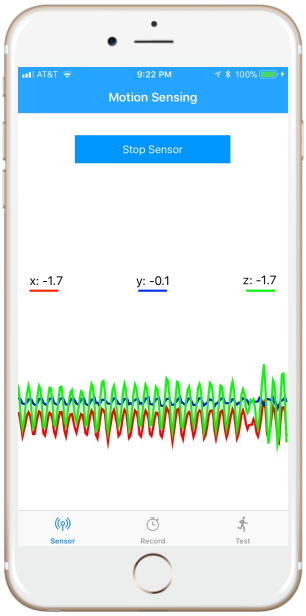


Figure 9: Mobile app developed for recording time series signals and real-time activity recognition.

8. References

[1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012..

[2] Anguita, Davide, et al. "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine." *International workshop on ambient assisted living*. Springer Berlin Heidelberg, 2012.