Machine Learning for Human Activity Recognition

Using the WISDM Dataset

R. Cen, B. Coffman, P. Day, S. Gao, K. Gemalmaz, J. Goon, T. Lao, J. Ordaz, R. Tsang, J. Ugochukwu, Z. Xie

Final project report

ECS171 - Rafatirad
University of California, Davis
June 2, 2021

1 Introduction

Analyzing human activity using machine learning falls under the research topic of Human Activity Recognition (HAR). There are many potential applications of this field, such as biometric monitoring, sports rehabilitation, elderly care, and fitness tracking. Our project explores HAR for personal use, allowing users to input phone data and receive a description of the activity that data represents. We applied many models to this problem, including Random Forest, Logistic Regression, Artificial Neural Network, K-Nearest Neighbors, and Neural Network. After some trial and error we came to the conclusion that the neural network provides the most accurate predictions for our data. Our user-friendly website sends the input data to the web server, which then implements the model and provides, in a short amount of time, a very accurate prediction of the activity. In this paper, we will look over related literature, describe the dataset, analyze each of our models, and describe our web application.

2 Related Work

Since Human Activity Recognition is currently an active and thriving field of research, there have been many novel approaches and techniques that have already seen some application. Where we had initially thought that extracting spectral features would be a novel approach to the problem, further reading revealed that this has been attempted with some measure of success: several separate studies have explored the use of cepstral coefficients as features for HAR [8, 11, 4]. Moreover, many ML methods have been applied to the problem with many different strategies. Attal, et al. apply k-Nearest Neighbors (kNN), Support Vector Machines (SVM), Gaussian Mixture Models (GMM), Random Forest (RF), k-Means, unsupervised GMM, and Hidden Markov Model to public datasets and compare their results, which is similar to what we have done in our project [1]. Jiang and Yin apply a Deep Convolutional Neural Network model directly to time series data by conceiving of an "activity image" [5]. Jalal, Batool, and Kim apply notch filters and do significant feature extraction, including MFCCs, and finally apply a decision tree classifier with Binary Grey Wolf Optimization [4]. The approaches we take in our project are generally simpler than these and are an attempt to replicate the findings of past research.

Our primary model is an artificial neural network. This approach has been attempted by previous papers before. Jiang and Yin's Deep CNN approach using the gyroscope, total, and linear acceleration signals had an accuracy performance of 97.59% on a separate dataset [5]. Alsheikh et al. used wearable data in order to gather accelerometer data to train a deep belief network (An undirected neural network as opposed to a feed forward network) [7]. This method achieved a 98.23% accuracy using the optimum parameters. Ronao and Cho used both the accelerometer and the gyroscope data to perform a 1D Convolution Neural Network [10]. This approach is the approach that is the most similar to ours. This method achieved a 94.79% accuracy.

3 Dataset Description

3.1 WISDM Dataset

We selected the WISDM Smartphone and Smartwatch Activity and Biometrics Dataset to train and test our HAR models on for its broader range of activity types and use of common acquisition frameworks like smartphone and smartwatch sensors [13]. The dataset was collected from 51 subjects performing 18 different activities each. For a given subject and activity, 3 minutes of gyroscope and accelerometer data is collected from a Samsung Galaxy S5 and LG G Watch at a sampling rate of 20Hz. The types of activities vary in purpose and intensity, from sitting and eating, to jogging and playing sports.

The dataset provided contains data in its raw form, as well as a set of pre-extracted features. Each of these features are ways of transforming the time-series data into a form that a ML model can handle, as most of the supervised learning methods we have learned in this course cannot handle time-series data without first preprocessing it. There are a total of 88 precomputed features, which are described in Table 1. Some additional time will be spent discussing Mel Frequency Cepstral Coefficients (MFCC) as we attempted to implement them to confirm our ability to reconstruct the pre-computed features.

3.1.1 Mel Frequency Cepstral Coefficients

Mel Frequency Cepstral Coefficients capture information about the spectral shape of the signal. They are often used in voice recognition algorithms and their calculation is as follows (detailed in [3]). First, a pre-emphasis

Feature	**Description**
X[0-9], Y[0-9], Z[0-9]	Distribution of values over x, y, and z axes, separated into 10 equally sized bins.
[X,Y,Z] AVG	Average sensor value over the window per axis
[X,Y,Z] PEAK	Time in milliseconds between the peaks associated with most activity (determined by heuristic specified in [6])
[X,Y,Z] ABSOLDEV	Average Absolute difference between each sensor value and the mean of the window over each axis.
[X,Y,Z] STANDDEV	Standard Deviation of values in the window per axis
[X,Y,Z] VAR	Variance of the values in the window per axis
XMFCC[0-12] YMFCC[0-12] ZMFCC[0-12]	Mel Frequency Cepstral Coefficients for each axis (linear cosine transform of mel-scale power spectrum). 13 MFCCs are kept.
[XY,XZ,YZ] COS	Cosine distances between sensor values for pairs of axes
[XY,XZ,YZ] COR	Correlation between sensor values for pairs of axes
RESULTANT	Euclidean magnitude of (x,y,z) readings averaged over window.

Table 1: Precomputed Feature Descriptions [6]

filter is applied to amplify high frequencies to balance the frequency spectrum, potentially improve SNR, and prepare for Fourier transform. Next, the signal is split up into a number of windows/frames and a window function is applied to reduce spectral leakage (FFT expects infinite data, which it isn't). Then the Fast Fourier Transform (FFT) is computed for N points (arbitrarily, empirically chosen) and the power spectrum is calculated by taking the square of the magnitude of each resulting frequency bin and dividing it by N. Then triangle filter banks, evenly spaced in the mel scale, are transformed to the linear scale and applied to the log power spectrum. This groups the spectrum into frequency bands with corresponding filter-bank coefficients that capture information about how much power is in each band. We finally obtain cepstral coefficients (named so because it's a spectrum of a spectrum) by applying the Discrete Fourier Transform to the filter bank coefficients, which are changing over time based on the window they are computed from.

3.2 Errata

There is a noticeable issue with the data in that the sampling rate is not constant across measurements. In fact, although the authors note that there are about 1M additional measurements for phone accelerometer data, they do not specify a cause [12]. It turns out that it is because the sampling rate for several subjects and several activities is substantially higher than 20Hz. Moreover, some measurements have significant variance in sampling interval, which can potentially have a severe impact on any spectral analysis of the signal. Indeed, this turned out to be the case when we trained a neural network on MFCCs manually computed from raw data. The distribution of sampling period means can be seen in Fig 1. Note the number of measurements whose average sampling period is less than 45 ms, which can significantly affect spectral and cepstral feature extraction.

In the future, some care should be taken to address and correct these issues, but as they are not directly related to the application of machine learning, they were not addressed in this project and most of our models are applied to the pre-extracted data with minimal corrections aside from the pre-processing described in section 4.1.

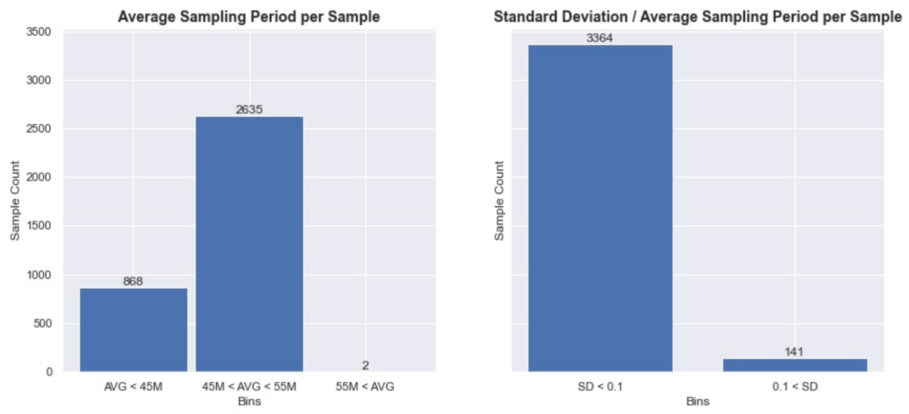

Figure 1: Distribution of Average Sampling Period across all Measurements and Standard Deviations greater than 10% of the measurement average.

4 Methods

4.1 Pre-processing

Before implementing our models, we had to preprocess our data. Even though the raw time series data was already processed into arff files, as described in section 3, further processing is necessary before the data can be funnelled through a machine learning model. In some cases, we one-hot-encoded our dependent variable (the activity label) because it was categorical. We normalized our independent variables using a min-max normalizer because they were numerical. We selected all of the attributes suggested by the dataset description [13] and dropped the rest. Among the remaining attributes, we see some multicollinearity but decided to keep the correlated attributes because they might still add prediction value to our model. At this point, it would be possible to find and remove outliers. Using sklearn's IsolationForest and LocalOutlierFactor classes [9], we found 41 outliers (0.18% of the dataset). It was difficult to tell why these samples were outliers, so we chose to not remove them.

4.2 Artificial Neural Network

4.2.1 Framework

Neural networks provide good solutions for complex problems like human activity recognition, as mentioned in section 2. As we will see, neural networks provide some of the best prediction accuracies for human activity recognition with the WISDM dataset.

To implement a neural network, we used the Keras framework [2]. With Keras, we quickly created a neural network model. First, we created our network layers. The input layer has the same number of nodes as there are attributes. The input layer feeds forward into three dense hidden layers. Lastly, the third hidden layer feeds to the output layer, which has the same number of nodes as there are classes of activities - 18.

Each hidden layer uses Keras' 'relu' activation function. The output layer uses Keras' softmax activation function, since we are investigating a multiclass classification problem. Ideally, we could have investigated multiple activation functions for both the hidden and output layers. Given the computational cost of optimizing many hyperparameters (which we will see later), we chose to keep the activation function constant and investigate other hyperparameters instead.

For simplicity, all of the hidden layers are uniform. They all have the same number of nodes, n. Early on in our model development, it became clear that many nodes resulted in higher accuracy. As such, we investigated a range of n values starting at 128. Each hidden layer has dropout regularization applied to it, with the hyperparameter p controlling the probability that any given node is dropped out. From early experiments, we found that dropout probabilities of 0.2 and higher were not as successful. We also found that dropout regularization was superior to the unregularized model (p=0). Therefore, we chose to evaluate p values between 0.025 and 0.1.

Our model is optimized with Keras' Adam optimizer [2]. Similar to our activation functions, we did not have the time and computational resources to investigate a variety of optimizers. However, we thought it was important to investigate several different learning rates. Keras uses a default learning rate (lr) of 0.0001 for the Adam optimizer. To optimize learning rate, we considered lr values at, above, and below this default value. Learning rates too high might make the model less accurate, while learning rates too low might make the model too slow to train.

4.2.2 Hyperparameter optimization

In order to thoroughly optimize our hyperparameters, we performed a 3-fold cross-validation grid search using sklearn's GridSearchCV [9]. While optimizing n, p, and lr, the grid search evaluated 27 combinations of hyperparameters. In order to increase search speed, this grid search was run for 500 epochs with a batch size of 1000 (as opposed to our final model, which was run for 750 epochs with a batch size of 500).

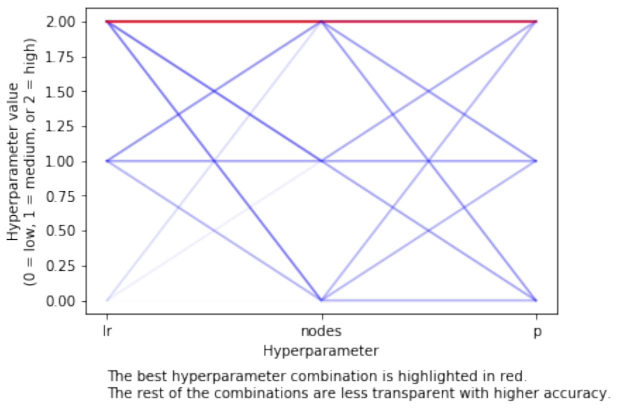

Figure 2: A grid search was conducted on 3 hyperparameters, across 27 total values combinations, for the neural network model. The best hyperparameter combination is highlighted in red. The hyperparameter values are: lr [0.00001, 0.0001, 0.001], n [128, 256, 512], and p [0.025, 0.05, 0.1].

Through this exhaustive search, we concluded that n=512, p=0.1, and lr=0.001 resulted in the highest model accuracy, as seen in figure 2. Since the grid search only ran for 500 epochs, it may favor a higher learning rate, which would train faster than a lower learning rate. In order to assure that a higher learning rate is ideal, we would need to run another grid search with the same number of epochs as our final model. We must also note that our apparently-optimal parameters values are at the edge of our grid search. In order to get more comprehensive results, we would need to expand our grid search to include parameter values above our apparently-optimal values. We did not run these additional grid searches because we did not have the computational resources to do so.

4.2.3 Evaluation

Now that we knew which hyperparameter values to use, we were able to train our final model. We randomized and split the dataset 70:30 using sklearn's `train_test_split` function [9]. Then, we trained our model using the larger fraction of the dataset and tested our model using the smaller fraction. Overall, the model was up to 86% accurate in recognizing human activity, with precision and recall scores that also reached up to 86%. Depending on the train-test split, these numbers could fall to 85%.

While this is a good prediction accuracy, it is not fully reflective of the model's performance. The dataset includes classes like "eating chips" and "eating a sandwich," which should be more difficult to tell apart because they do not require much leg movement. We hypothesize that activities with more lower-body movement will be identified with higher accuracy because the data is collected from the user's pocket [13]. The model confirms this hypothesis: the best-predicted activities are kicking a soccer ball, walking, going up stairs, and jogging, as seen in figure 3.

Overall, the neural network model performs well. It performs better than other models trained on the same dataset, as we will see next. In creating, training, and evaluating this neural network, we ran into a

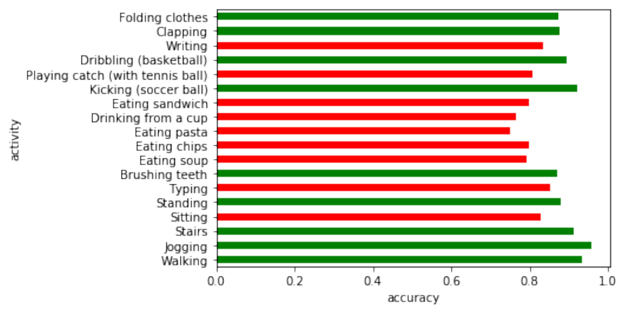

Figure 3: Neural network accuracies for each activity in the dataset. Green activities have greater accuracy and red activities have worse accuracy than the model tested on all activities.

primary obstacle: time. Since we only had access to our home computers, we were unable to sufficiently evaluate all of the hyperparameter values in our grid search. Regardless, the model performs well, especially for activities that involve lower-body movement.

4.3 K-Nearest Neighbors

K-Nearest Neighbors is a model for machine learning where computation is not done until classification. The algorithm looks for the k-nearest neighbors and classifies the input data based on what the majority of those k neighbors are. This method can be useful for many different applications, but fails when it comes to more complicated ones. Human activity recognition is one of those more complicated use cases as our testing will show.

To implement a K-Nearest Neighbors model, we used sklearn's KNeighborsClassifier. This made is simple to make a K-Nearest Neighbors model because it was simple like every other model in sklearn and would be compatible with the hyperparameter searches that sklearn offered. To optimize the hyperparameters we used GridSearchCV from sklearn with various different values on the parameters: n_neighbors, weights, and metric. The values that we tested for each of these was n_neighbors from 1 to 10, weights of manhattan and distance, and metric of euclidean, manhattan, and minkowski. We used GridSearchCV on these parameters scoring based on accuracy with 5 fold cross-validation. The parameters that came up as the best were metric as manhattan, n_neighbors as 1, and weights as uniform with a score of 0.845707.

Testing a fitted model with these parameters on the test set gives us a recall score of 0.850281 and a precision score of 0.850786. On paper, these scores sound very good and would be very good if it was another model type. However, with a KNN where n_neighbors = 1, the model is only finding the nearest existing data point and classifying based on that. This results in recall and precision scores of 1 when we evaluate the model with the training set, which indicates over-fitting. This is by the nature of KNN. Since the data in our use case is so complex, 1 neighbor is optimal because there are not enough dense clusters in which more than one neighbor would classify better.

Having a KNN model only using 1 nearest neighbor makes the model not generalized enough. While it is accurate in testing, that prediction value might not apply to other cases. In addition, the complexity of this dataset may not be represented well enough in KNN if novel data were to be used to predict because there may be some undocumented values that correspond to a different activity. There are also situations in which the data for two different classifications may be very similar because there would not be as much of a difference in movement in the activities, as explained in section 4.2. Given the context of our data and the factors just mentioned, KNN is at best only a decent model because of its downfalls in complexity expression while maintaining generalization, despite its very good accuracy.

4.4 Support Vector Classification

For SVM, the hyperplane (line) is found through the maximum margin. In multiclass classification tasks, the same principle is utilized after breaking down the multiclass classification problem into multiple binary classification problems. We used one vs rest strategy here. For each classifier, the class is fitted against all the other classes.

We can see that there are 46 features. PCA could reduce the dimension and improve the speed. We'd like to see the effect of the PCA. We chose 15 principle components which could explain over 0.95 of the total variance. Also, to optimize the hyperparameters, we used GridSearchCV from sklearn [9] to tune them: kernel, value of gamma and the choice of C. The best choices for hyperparameters are: kernel is 'rbf', gamma is 10, C is 500 for model without PCA and 1000 for model with PCA.

	duration	accuracy
without PCA	0:08:50.184540	0.8524
with PCA	0:05:39.457769	0.8160

Table 2: PCA comparison

First we trained the model without PCA. Then we applied PCA on the dataset. From the Table 2, we can see there is a trade-off between running time and accuracy rate. After applying PCA, we could save 3 minutes but the accuracy decreased by 0.0368. The accuracy, precision score and recall scores for SVM with PCA are 0.8114, 0.8107 and 0.8108, respectively.

4.5 Logistic Regression

We have also conducted a logistic regression model that will classify the types of activities that a user has conducted based on the dataset that has information regarding human activity. We separated the features as independent and dependent variables where the dependent variable is predicting which human activity the user was doing and the independent variables are the factors that lead to the classification of the type of activity. We used the MinMaxScaler technique from Scikit-Learn [9] on the independent variables where it could give an equal variance such that these variables won't lead towards a high level of bias in the classification process. We split the dataset into 70% training data and 30% testing data so that the model can prioritize what it's actually familiar with before conducting the classification. We have also calculated the accuracy score of the model overall and it led to having an overall score of approximately 40%, which means that the logistic regression model is not accurate in terms of predicting what specific activity that a user has done. We have also calculated the precision and recall scores for testing and training data to measure the ratio of true positives over total positives (precision) and the ratio of true positives over true positives and false negatives (recall). The scores for these calculations ranged from approximately 38% to 40%.

4.6 Random Forest

Next, we constructed a Random Forest model for dataset classification. The purpose, similar to other models, is to distinguish between the types of data in different human activity situations. Random forest consists of a large number of decision trees. The biggest advantage of random forests over decision trees is that they correct the habit of overfitting decision trees to their training sets. The specific number of trees can be controlled by the n_estimators variable of the RandomForestClassifier in sklearn.ensemble. Theoretically speaking, more trees can make the model fits better. However, we found that after a certain number, the accuracy of the model may not improve and may even decrease. Choosing a reasonable number of trees is the biggest obstacle to using random forests. We chose the most appropriate number of trees by adding 100 trees at a time, all the way to 2000 trees. In figure 4, a graph was drawn to express the different accuracy of the model under different numbers of selection trees.

After training the model, we used the test group to check the fit of the model. We can see the results of using this model to make predictions. The vast majority of them are correct. There are a few data predictions that are wrong but they are small in percentage. We got an average recall of 0.8509 and an average accuracy of 0.8509. If look at a particular class, such as "Jogging", an accuracy of 0.95 is achieved. Overall, we can see that this model gives us very good results comparable to those found earlier.

5 User Interface

We decided to go with a simple two tone background for the website to make it easy for the user to navigate. It is all contained on a single page with a prompt for the user to select and upload a file that will be processed. The machine then will take that information in and determine the users activity.

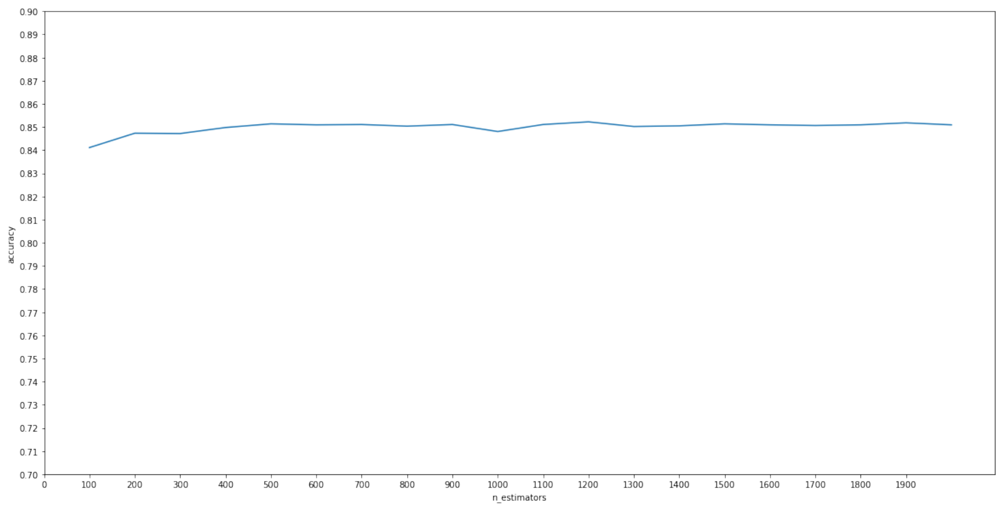

Figure 4: We can see that when the number of trees increases, the accuracy is improving but not dramatic. Sometimes it even goes down. So we choose 2000 trees to train our model. On the one hand, the accuracy of 2000 trees looks really good. On the other hand, the model of 2000 trees can be used directly without retraining the model, saving computing power.

We used HTML/CSS/js as our frontend, and Node.js as our backend. Keeping it simple, we have a single button to upload a datafile. We have prepared five sample datafiles, each representing a different action performed for three minutes. When a file is uploaded, our javascript sends an HTTP POST request to the Node server, which then invokes our neural network. The trained neural network is loaded from disk to save time. Then, we make a prediction based on the input data. The server then responds to the POST request with the name of the activity that most resembles the input data.

We ran into a lot of struggles building this web app. At first, figuring out how to use python in the browser was a struggle, as we only had experience with js backends. We initially planned to use pyodide, which is a package that allows python to be run in the browser, supporting many data libraries such as pandas and numpy. Unfortunately, Keras is unsupported, so we had to start from scratch and figure something else out. After figuring out how to save and load models, and how to run python processes using Node.js, things started coming together. We eventually arrived at a fully working version.

6 Conclusion

Throughout this project we have seen the complexities of Human Activity Recognition. We see that a variety of models perform well. The only model with low accuracy, predicting with below 80% accuracy, is the logistic regression model. The neural network performs best with 86% accuracy. The Random Forest and K-Nearest Neighbors models perform similarly well with 85% accuracy.

For future research, we could apply our models to phone gyroscope, smart watch accelerometer, and smart watch gyroscope data; all of which is included in the WISDM dataset [13]. We could also improve the accuracy of our model by grouping together similar activities. For example, we could group together all of the activities that involve eating. This would make sense in the context of the phone data, which is collected from the user's pocket.

This project is a proof of concept for a future in which every person with a smartphone could have a full profile of their activity simply using their smartphone's accelerometer and gyroscope data. We hope that our project shows how accessible Human Activity Recognition could be in the future.

Our code can be found in a GitHub repository: `https://github.com/rchtsang/ecs171-HAR`.

References

[1] Ferhat Attal et al. "Physical Human Activity Recognition Using Wearable Sensors." In: *Sensors* 15.12 (2015), pp. 31314–31338. ISSN: 1424-8220. DOI: `10.3390/s151229858`. URL: `https://www.mdpi.com/1424-8220/15/12/29858`.

[2] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[3] Haytham M. Fayek. *Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between.* 2016. URL: `https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html`.

[4] Ahmad Jalal, Mouazma Batool, and Kibum Kim. "Stochastic Recognition of Physical Activity and Healthcare Using Tri-Axial Inertial Wearable Sensors." In: *Applied Sciences* 10.20 (2020). ISSN: 2076-3417. DOI: `10.3390/app10207122`. URL: `https://www.mdpi.com/2076-3417/10/20/7122`.

[5] Wenchao Jiang and Zhaozheng Yin. "Human Activity Recognition Using Wearable Sensors by Deep Convolutional Neural Networks." In: *Proceedings of the 23rd ACM International Conference on Multimedia*. MM '15. Brisbane, Australia: Association for Computing Machinery, 2015, 1307–1310. ISBN: 9781450334594. DOI: `10.1145/2733373.2806333`. URL: `https://doi.org/10.1145/2733373.2806333`.

[6] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. "Activity Recognition Using Cell Phone Accelerometers." In: *SIGKDD Explor. Newsl.* 12.2 (Mar. 2011), 74–82. ISSN: 1931-0145. DOI: `10.1145/1964897.1964918`. URL: `https://doi.org/10.1145/1964897.1964918`.

[7] D. Niyato L. Doyle S. Lin H.-P. Tan M.A. Alsheikh A. Selim. "Deep activity recognition models with triaxial accelerometers." In: *Proc. of the AAAI Workshops* (Feb. 2016).

[8] Alfredo Madrid García. *Human activity recognition by inertial signals obtained from a smartphone.* Available at `http://oa.upm.es/41512/`. 2016.

[9] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[10] C.A. Ronao and S.-B. Cho. "Human activity recognition with smartphone sensors using deep learning neural networks." In: *Expert Syst. Appl.* 59 (Oct. 2016), pp. 235–244.

[11] Sebastián Vanrell, Diego Milone, and Hugo Rufiner. "Assessment of Homomorphic Analysis for Human Activity Recognition From Acceleration Signals." In: *IEEE Journal of Biomedical and Health Informatics* PP (July 2017), pp. 1–1. DOI: `10.1109/JBHI.2017.2722870`.

[12] Gary M. Weiss. *WISDM Smartphone and Smartwatch Activity and Biometrics Dataseet.*

[13] Gary M. Weiss, Kenichi Yoneda, and Thaier Hayajneh. "Smartphone and Smartwatch-Based Biometrics Using Activities of Daily Living." In: *IEEE Access* 7 (2019), pp. 133190–133202. DOI: `10.1109/ACCESS.2019.2940729`.