Human Activity Recognition Modeling

Ethan Fang, Jackson Schuetzle

January 9th, 2025

1 Introduction

The topic we will be trying to solve is related to **Human Activity Recognition**. HAR models oftentimes take in sensor input from accelerometers and gyroscopes (rotational movement) on smartphones or wearable devices, and return a physical activity they predict the user to be doing. Since each device detects changes in three dimensions, it is typical to see datasets with readings taken at some frequency in 6 dimensions. Additionally, readings are grouped together into "windows" containing many readings (typically 50-100), and oftentimes these windows have some overlap.

Generalizing to new user's behavior or to existing user's unforeseen behavior are crucial aspects of successful HAR models–the main goal of the project being the former. **We achieved this by properly managing data for cross validation, and by using a long short-term memory RNN for classification.**

1.1 Dataset

We developed the model on a dataset from the UCI ML Repository which consisted of accelerometer and gyroscope readings from a Samsung Galaxy S II smartphone [1]. The readings were captured every 20 milliseconds from all three dimensions on each device (six dimensions total). Typical of many HAR collections, this dataset describes samples by windows of readings; specifically, each window consists of 128 readings with 50% overlap (i.e. adjacent windows share 64 of the same readings). Each window has a subject identifier–representing which of the 30 human subjects completed the activity. Furthermore, each window has an activity label–representing one of the six activities measured in the study: walking, walking upstairs, walking downstairs, sitting, standing, and laying. Finally, the data was split 70-30 into training and test sets, respectively.

The train and test data came in the form of text files. Labels came in a simple list, while inputs came in two different types: 1) raw sensor measurements, and 2) extracted feature vectors. The files with extracted features were crafted by experts in the field, and for the sake of personal learning we decided not to use them. On the other hand, each axis of raw measurements had its own file, so we decided to load in each file and merge them together into a single `numpy` matrix of dimensions $(7500, 128, 6)$ for the training set (code for which can be found under `Data Importing` in submitted notebook).

2 Preprocessing

2.1 Assumptions

A common difficulty in HAR models that are trained on sliding window data is **maintaining the Independent and Identically Distributed assumption during cross validation**. During any type of train/test splitting, it's important that samples from both sets not be too similar to each other, otherwise cross validation results may over evaluate the effectiveness of the model. In the context of our data, there's three ways that the IID assumption can be broken.

1. Samples in different sets can contain up to 50% of the same data if they were initially adjacent.
2. Sets may contain data from the same subject, known as **subject-dependent CV.**
3. Samples drawn in a short time interval will most likely be more similar than that of samples drawn further apart in time.

Item 1 above is a trivial case of test set leakage. Subject-dependent CV references the fact that two samples drawn from the same subject are much more likely to be similar for a given activity than that of two randomly drawn samples [2]. Finally, based on the fact that activities normally last longer than 2.56 seconds (the length of one window), long contiguous chunks of the same activity label appear in the dataset; e.g. if samples 10 and 12 are labeled as walking, then it's almost certain that sample 11 will also be labeled as walking, which violates the IID assumption.

2.2 Addressing IID Violations

This section describes how the three items listed above were mitigated.

2.2.1 Item 1

Since the exact percentage of overlap between adjacent samples was given, it's then possible to transform the data in such a way that none of the samples overlap. We first decided to prove that all adjacent samples contain 50% of overlapping data (code provided in `Testing Overlap Percentage` section in notebook).

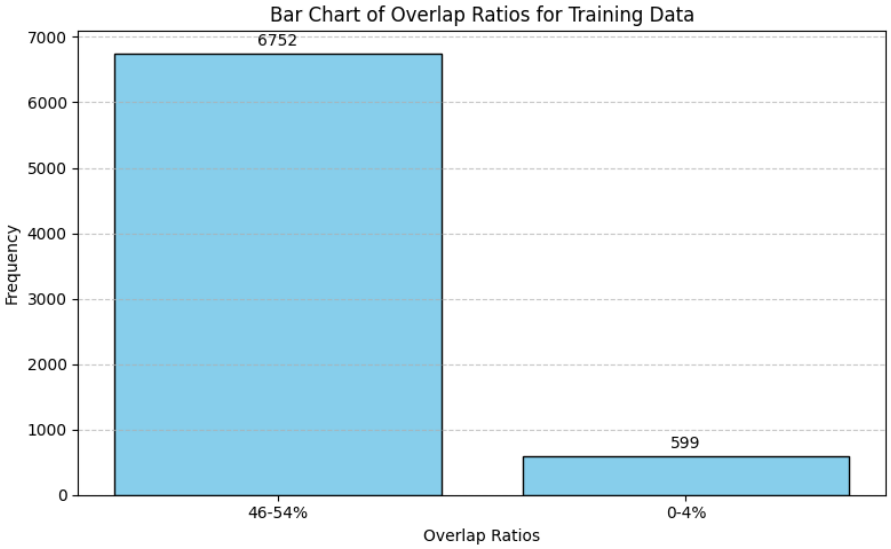

Figure 1: Frequency of Overlap Ratios Between Adjacent Samples

In Figure 1, we found most adjacent pairs had the 50% overlap, however some had approximately zero overlap. Most of these instances are due to adjacent samples having different activity labels, which intuitively makes sense since it's likely sensors don't take readings in between activities. We include a range of percentages in Figure 1 in order to take into consideration that readings could be identical by random chance. This is especially applicable to sedentary activities which may result in readings that are relatively static. In order to properly transform the data into non-overlapping windows, we 1) took only the first half of samples that had 50% overlap with their successor, and 2) split samples in half that had 0% overlap with their successor (code provided in `Transform Data into Non-Overlapping Windows` section in notebook).

2.2.2 Items 2 & 3

We used the provided subject labels in order to properly handle items 2 and 3 above. Conveniently, all readings for each individual subject are provided contiguously and in sorted order. During our custom k-fold cross validation, we ensure that none of the subject chunks are split while creating the validation set. In doing so, we automatically ensure that samples from a single instance of an activity aren't split either (code provided in `Create Subject Chunks` section in notebook).

2.2.3 Cross Validation

The largest benefit of grouping samples into subject chunks is that these chunks can be placed in any order without violating IID. Therefore, we can **easily move chunks around to perform k-fold cross validation**. There are 21 subjects in the training set, so we decided that $k = 5$ would result in folds being relatively close in size.

One risk of our methodology is that folds aren't uniform in size. If subjects with relatively more samples are grouped together, then it could be the case where one fold has hundreds of more samples than the rest. After creating a histogram of the subject chunk sizes, we realize that the standard deviation of fold sizes (assuming 4 chunks per fold are chosen) ends around 70 samples, which is reasonable variance.

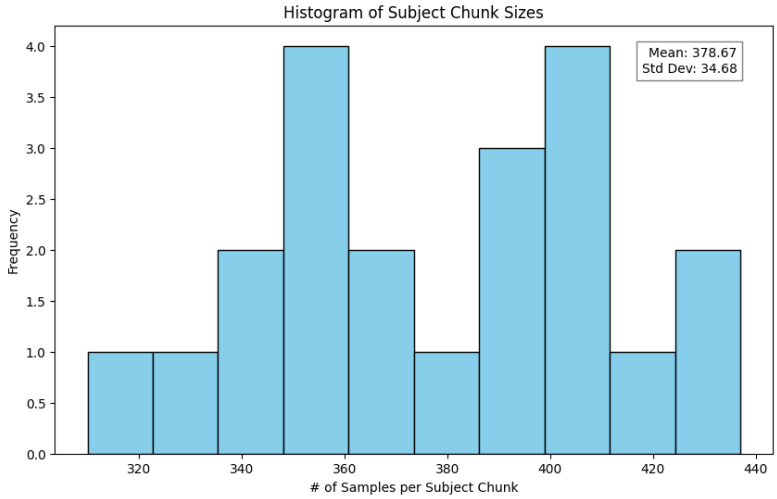

Figure 2: Distribution of Subject Chunk Sizes

Since the number of folds isn't exactly a factor of the number of subjects, there will always exist one fold that has an extra subject. Our implementation of k-fold is provided in `cross_val_score` function in the notebook.

3 Model

3.1 Architecture

Our final choice of model involved **deep learning using an a Long Short-Term Memory RNN model (LSTM)**. LSTM models are designed to process time-series data by learning both short-term and long-term dependencies. Our model is composed of a sequence of three LSTM layers, each with a hidden size of 128. The hidden size refers to the number of neurons in the hidden state vector of each LSTM layer, which determines the size of the output at each time step. We found a hidden size of 128 to pass the most valuable information from layer to layer.

To avoid over-fitting, we added dropout layers to our model between each LSTM cell. Dropout is a regularization technique where a fraction of the layer's outputs (in our case, 0.5) are randomly set to zero during training. This ensures that the model doesn't rely too heavily on specific neurons, promoting robustness across layers.

Our final layer in the LSTM takes the last time step's output as a summary of the entire input window. After passing through all the LSTM and dropout layers, we map the summary vector into a fully connected linear layer to 1 of 6 output classes (code can be found in the `Model Architecture` section in the notebook).

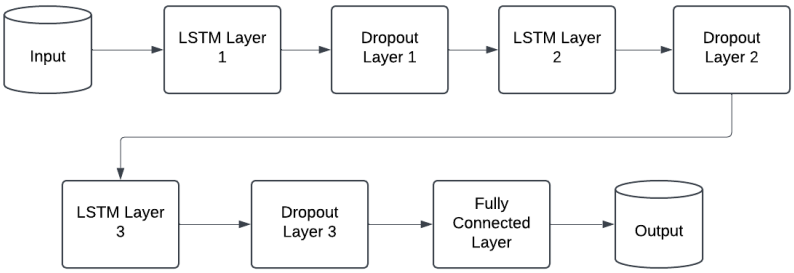

Figure 3: Architecture Visual

3.2 Training

When training on large datasets, it is inefficient–and sometimes infeasible–to process the entire training set in a single forward and backward pass per epoch. To address this, we used **mini-batch training**; instead of presenting the model with all samples at once, we split the dataset into smaller chunks of a chosen size. For each mini-batch, we perform a forward pass to compute predictions, calculate the loss, and back-propagate to update the model parameters. This means each epoch will contain multiple "update steps" rather than a single large update step.

Mini-batch training has a few benefits. Loading only a fraction of the dataset is more memory efficient and reduces the risk of running out of GPU RAM. Additionally, updating the model multiple times per epoch on smaller datasets often leads to faster convergence since each mini-batch contains a different "view" of the dataset. This generally should help the model learn better by introducing beneficial noise in the gradient.

Figure 4 shows the graph of our training loss vs. our validation loss, which shows a steady decrease for both until we reach around 80 epochs. After this, our losses begin to flatten out, showing that training with epochs in this range is optimal for our model. We found that training using 100 epochs gave us the best trade-off between performance and stability during mini-batching (code can be found in the `Training` section in the notebook).

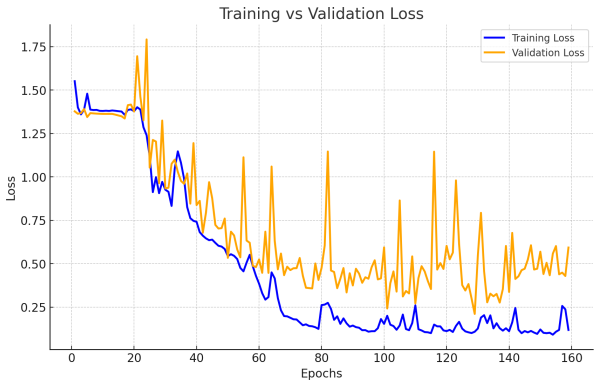

Figure 4: Training vs. Validation loss

3.3 Results

Choosing the the right accuracy metric for machine learning models is of great importance. Simply choosing the pure accuracy (ratio of correct to incorrect classifications) does not provide a proper evaluation of the model because it doesn't take into account 1) class imbalances and 2) different types of errors.

Proper metrics often use a confusion matrix in order to obtain a more complete evaluation of the model. A general structure for a confusion matrix is given in Figure 5, where p and n are classes in the dataset.

With this tool, we can more closely see classification accuracy on individual classes and can describe the errors in an informative manner.

Two of the most common metrics used are **Precision** and **Recall**. Precision describes the how well your predictions for a certain class were, whereas recall describes how well the class actually got predicted. In the diagram above, the squares involved in the calculation of each metric are circled. It's very common in HAR to ignore true negative rates, as there isn't much value in being able to definitively tell a user what they *aren't* doing. Therefore, a metric which ignores the true negatives may be a better fit for HAR models. Luckily, there does exist a popular metric which does this! It's known as the $\boldsymbol{F_1}$ **score**, which is just the harmonic mean of the Precision and Recall.

$$F_1 = 2 \cdot \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

There are two flavors of this metric, known as the $\boldsymbol{F_1}$ **micro** and $\boldsymbol{F_1}$ **macro** scores. Simply put, the micro average gives each sample the same amount importance, whereas the macro average gives each class the same amount of importance. We chose to consider both metrics in order to account for any class imbalances in the testing data. Figure 6 contains the results; the model achieved performance with a F_1 macro $= 0.901$ and F_1 micro $= 0.916$ against the test set. Our model struggled the most with differentiating between standing and walking activities.

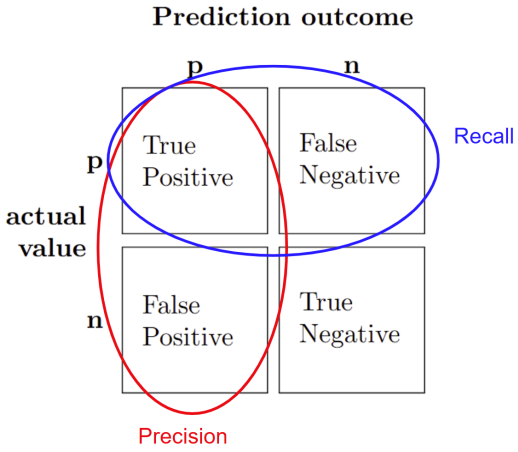

Figure 5: Confusion Matrix for Binary Classification

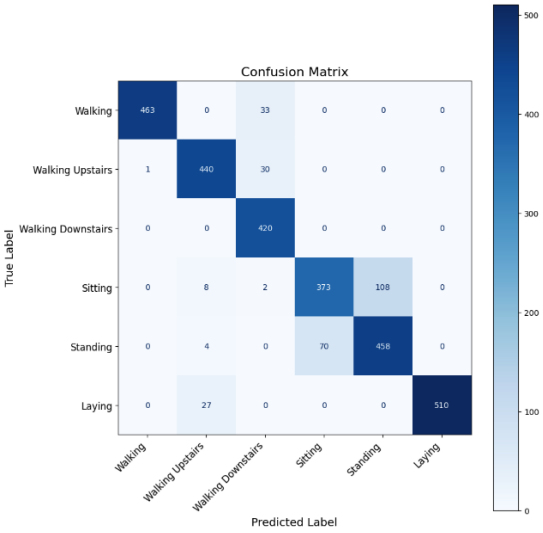

Figure 6: Resulting Confusion Matrix from LSTM Model

4 Conclusion and Reflection

We learned a lot about building neural network architectures from this project, from choosing the right model (traditional vs. deep) to editing the internal architecture of the "most optimal" model. Something we struggled with before was getting too attached to a single model for a problem, and not keeping an open mind to trying new things.

Another thing we learned from the project was to start small and keep building upwards when training neural networks. It is common to try and combine models together as a "hybrid" architecture, but doing this randomly without any thought is wasted effort. Starting small, seeing incremental changes in performance, and adding on top of that is key to success.

Next time, we would like to try an ensemble combination of traditional and deep learning models to increase accuracy. After examining our confusion matrix, it seems that we could build different models to target certain errors, and try to fix them.

5 Works Cited

[1] J. Reyes-Ortiz, D. Anguita, A. Ghio, L. Oneto, and X. Parra. "Human Activity Recognition Using Smartphones," UCI Machine Learning Repository, 2013. [Online]. Available: link.

[2] A. Dehghani, O. Sarbishei, T. Glatard, and E. Shihab, "A Quantitative Comparison of Overlapping and Non-Overlapping Sliding Windows for Human Activity Recognition Using Inertial Sensors," Sensors, vol. 19, no. 22, p. 5026, Nov. 2019. Available: link .