Machine Learning and Pedometers: An Integration-Based Convolutional Neural Network for Step Counting and Detection

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

by
Basil Lin
December 2020

Accepted by:
Dr. Adam Hoover, Committee Chair
Dr. Jon Calhoun
Dr. Yingjie Lao

Abstract

This thesis explores a machine learning-based approach to step detection and counting for a pedometer. Our novelty is to analyze a window of time containing an arbitrary number of steps, and integrate the detected count using a sliding window technique. We compare the effectiveness of this approach against classic deterministic algorithms. While classic algorithms perform well during regular gait (e.g. walking or running), they can perform significantly worse during semi-regular and irregular gaits that still contribute to a person's overall step count. These non-regular gaits can make up a significant portion of the daily step count for people, and an improvement to measuring these gaits can drastically improve the performance of the overall pedometer.

Using data collected by 30 participants performing 3 different activities to simulate regular, semi-regular, and irregular gaits, a training and testing strategy was implemented using a sliding window algorithm of pedometer accelerometer data. Data was cut in rows representative of the sliding window, normalized according to the minimum and maximum values of the corresponding sensor-axis combination, and finally collated in specific training and holdout groups for validation purposes.

Nine models were trained to predict a continuous count of steps within a given window, for each fold of our five-fold validation process. These nine models correspond to each gait and sensor combination from the collected data set. Once models are trained, they are evaluated against the holdout validation set to test for both run count accuracy (RCA), a measure of the pedometers detected step to actual step count, and step detection accuracy (SDA), a measure of how well the algorithm can predict the time of an actual step. These are obtained through an additional post-processing step that integrates the predicted steps per window over time in order to find the total count of steps within a given training data set. Additionally, an algorithm estimates the times when predicted steps occur by using the running count of total steps.

Once testing is performed on all nine models, the process is repeated across all five folds to verify model architecture consistency throughout the entire data set. A window size test was implemented to vary the window size of the sliding window algorithm between 1 and 10 seconds to discover the effect of the sliding window size on the convolutional neural network's step count and detection performance. Again, these tests were run across five different folds to ensure an accurate average measure of each model's performance.

By comparing the metrics of RCA and SDA between the machine-learning approach and other algorithms, we see that the method introduced in this thesis performs similarly or better than both a consumer pedometer device, as well as the three classic algorithms of peak detection, thresholding, and autocorrelation. It was found that with a window size of two seconds, this novel approach can detect steps with an overall average RCA of 0.99 and SDA of 0.88, better than any individual classic algorithm.

Acknowledgments

Firstly, I would like to thank Dr. Adam Hoover for the countless hours he has committed to helping me, and the belief he has shown in me. He has been integral in both my personal and professional development during my graduate education, and a major impetus in my decision to pursue a graduate degree. Without him, I could not have completed this thesis.

I would also like to thank Dr. Jon Calhoun and Dr. Yingjie Lao for taking the time out of their busy schedules to serve on my defense committee. Additionally, I would like to thank all the professors that I have had the pleasure of learning from, and Clemson University for providing me an incredible environment to study and grow in.

Special thanks go to Dr. Ryan Mattfeld, whose previous publications and help have gone a long way in making this work possible. I also want to thank both past and present members of our research group, who have all been eager to share their knowledge with me.

Lastly, I want to thank my friends and family for their unconditional support throughout my educational career. Their encouragement has been inspirational in keeping me motivated and determined to further my academic goals.

Table of Contents

List of Tables

List of Figures

Chapter 1

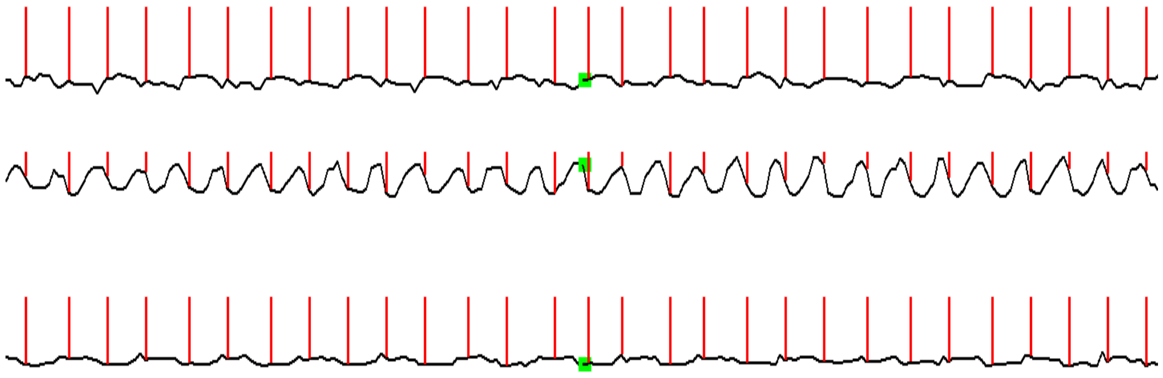
Introduction

1.1 Overview

This thesis considers the problem of detecting and counting steps by analyzing accelerometer sensor data from a body-worn device. Such devices, commonly known as pedometers, have seen widespread use as personal health instruments designed to give a general idea of the overall activity level of an individual. In particular, the work in this thesis aims to apply a neural network to the pedometer in an attempt to count and detect steps through machine learning techniques.

Traditionally, pedometer algorithms for detecting and counting steps have been based on heuristic strategies. These algorithms are often developed by researchers manually reviewing data collected from pedometer sensors and developing deterministic methods that identify specific patterns distinguishable to the researcher. Sensor signals associated with individual steps are analyzed and deconstructed in an attempt to discover the data composition unique to a step, so that an algorithm can be devised to find these unique patterns by filtering sensor data based on an established set of criteria. Common methods for this type of algorithmic design include threshold detection, where signals that clip above or below a certain threshold are counted as steps, peak-detection, where local maximums in signals are compared to known step signals, and autocorrelation, where sequences of signal data known to mark steps are compared continuously with new signal data. However, these methods can face impediments when signal patterns become varied or more difficult to immediately discern, especially when confronted with irregular step gaits and sub-optimal sensor data.

The research in this thesis focuses on applying machine learning to the pedometer. Specifically, this research applies a convolutional neural network (CNN) to a sliding window of data in an attempt to predict the number of steps within each window. By continually sliding the window through new sensor data, we can approximate the total number of steps taken. This further differs from traditional step detection approaches due to the use of a longer window of data that may contain more than just a single step. Such window lengths may prove useful in pedometer accuracy by providing additional context during analysis. For example, Figure 1.1 shows a comparison between two sensor signals from a regular gait of steps (top) and an irregular gait of steps (bottom). Steps are marked by red lines. It can be seen that the signals that denote steps do not follow the same pattern for the two shown gaits, so a larger window size that can capture this context could prove useful. By predicting the number of steps in each window, we effectively create a distribution that describes number of steps taken per window, for every slide of the window. Accordingly, the total count of steps can be predicted by integrating this function over time.


(a) Accelerometer sensor readings from regular gait, wrist-based sensor

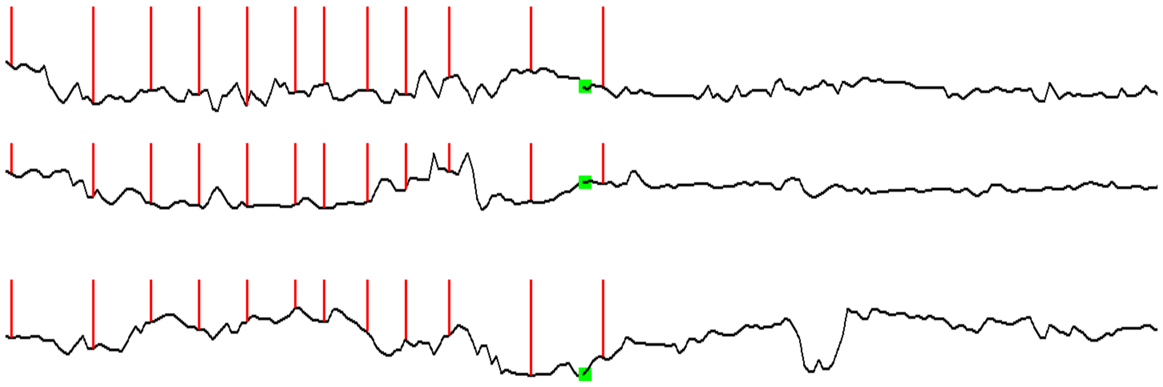

(b) Accelerometer sensor readings from irregular gait, wrist-based sensor

Figure 1.1: Signals gathered from an accelerometer tracking a regular gait compared to an irregular gait. Times that steps are taken are indicated by red lines. The X, Y, and Z axes of acceleration are displayed from top down in each figure.

By predicting the steps that occur within a window, we can also roughly estimate the time in which a step was taken. However, as our neural network is designed for a prediction of a continuous variable, we do not predict the specific time at which a step occurs with the neural network itself. Instead, the time when a step occurs is an estimation based on the predicted number of steps within the window. That predicted number is distributed throughout the window to give an approximation of the times when steps were predicted. While this somewhat limits our accuracy in predicting steps, we believe that the degree of accuracy in the number of steps predicted outweighs the usefulness in precision of when a particular step is detected. The number of steps is often a reliable predictor of the overall calorie burn of an individual when walking [1], and while the detection of when a step is taken may be important to knowing when physical activity occurs, the tolerance on such a window can be much higher without impacting user activity statistics compared to something as critical as the accuracy of the total step count. For example, during exercise such as jogging, it is more critical to know information such as the distance run and calories burned, rather than the exact times in which steps were taken.

Why use a machine learning approach, though? Traditional pedometer measurements perform reasonably well when walking with a normal gait, but accuracy can fall drastically with non-standard gait types, as evidenced by the inaccuracy of Android smartphone pedometer applications in free-living situations [2]. We refer to non-standard gait types as steps that do not follow a roughly uniform interval. An example of this is shifting back and forth while standing, as shifts back and forth can be considered steps, but are not as explicitly delineated as steps taken during activities such as walking or jogging. For instance, an individual working at the checkout counter of a grocery store is typically standing at the register. Though they remain standing while at work, intermittent shifts back and forth still count as steps, despite the lack of what is considered walking. Much like how a shift can be visually ambiguous, accuracy in step counts of traditional pedometer algorithms suffer due to less well-defined signals measured by the sensors of the pedometer. In spite of the variety of algorithms behind these heuristic methods, the essence of these rule sets remains constant; by adhering to pre-determined criteria when analyzing sensor data, an algorithm can determine when a step is taken. Our hypothesis is that by applying machine learning to the same data, instead of manually tuning thresholds and other values to analyze sensor data, we can train a neural network to automatically do the same.

The data set used in this thesis was previously collected in another work [3]. 30 participants were recorded with sensors fastened at the wrist, hip, and ankle of each participant. Each participant performed the same set of activities, designed to simulate a variety of conditions that mirror a regular interval of steps, a somewhat regular or semi-regular interval, and an irregular interval. These sensor readings were then analyzed in conjunction with video recordings of each subject, and the times at which steps occurred were manually found and marked. This thesis was made possible through a variety of tools, including a combination of programs developed by our own group, Keras and TensorFlow for the machine learning framework, C and Python programming languages for developing data pre-processing, training, and post-processing programs, and an assortment of Unix shell scripts for easy scaling of the aforementioned tools. The majority of the computational workload from this research was handled by a variety of servers hosted on the Clemson University Palmetto Cluster, the university's high-performance computing resource.

1.2 Background

1.2.1 Mobile Health

Healthcare is a basic human need, yet it remains a difficult challenge to modern society. Seeking treatment for ailments can be a long, complicated, and even expensive process; yet this pattern of seeking care only once affected by a condition is the dominant pattern in current healthcare systems. Unfortunately, the cyclic process of diagnoses and treatments continues to consolidate this existing phenomenon, strengthening the status quo of healthcare as an impossible obstacle. If there were a mobile health solution that could allow individuals to access deeper, more complete statistics of their own health without sacrificing convenience, it could act as an inspiration for individuals to actively maintain their own health before afflictions occur and drive them towards a diagnosis-treatment pattern. Fortunately, with the explosion in wearable technology and smartphone adoption, mobile health tools and technologies are more accessible now than ever before. Personal health tracking can be an extremely helpful tool in combating the reactive cycle of healthcare, as it can be a motivating factor in staying healthy before health issues have the opportunity to arise.

Examples of the effectiveness of self-monitoring health tools is evident in the battle against obesity. Obesity has always been heavily correlated with a wide variety of health complications, but recently, the problem of obesity has been magnified by the increasing sedentary lifestyles of many due to the COVID-19 pandemic [4]. While prevalent, obesity is largely preventable by dieting and living an active lifestyle [5], but despite such a seemingly simple solution, as evidenced by the growing concern of obesity in the current pandemic, the issue still persists. However, it has been shown that providing self-monitoring tools to individuals often act as a catalyst for breaking out of sedentary habits [6]. In fact, using self-monitoring devices to actively track caloric intake and movement, especially when using common mobile technology such as smartphones and smartwatches, has been associated with a positive effect on losing or maintaining weight [7]. While a suite of tools designed to monitor specific health metrics, such as professional-grade medical equipment, would collect health data with much higher fidelity and accuracy, it is unnecessary, over-complicated, and too expensive for the average individual to invest in. The convenience of mobile health devices is imperative to the adoption of self-monitoring health tools, as individuals may be much more willing to use additional tools on the devices they already own. Two of the most common wearables nowadays are shown in Figure 1.2, and both tout numerous health monitoring features. Increasing the adoption and accuracy of mobile intervention devices should be a priority to help wearers become more active, healthier individuals, hopefully resulting in a more proactive healthcare system where individuals are keener on maintaining good health before illnesses transpire.

Figure 1.2: Examples of industry standard smart wearables. The Apple Watch (left) and the Fitbit (right) are two of the most popular fitness tracking devices on the market at the time of writing. The two companies dominate the market for smart wearables, with a combined market share of nearly 57% [8].

Accordingly, our research group has developed technologies that can leverage the sensors in commonplace wearable technology. Using sensor data gathered from wrist-worn devices, we are capable of finding the number of bites taken from food, as well as measuring activity such as step counts, all from the same set of sensors [9, 10, 3]. As we move towards less and less intrusive wearable devices, the algorithms for these can be ported and tuned for other sensor positions as well [11]. While obesity can be a complex combination of factors, at its very core, it can be simplified down to a measure of caloric surplus. Hence, a single body sensor that contains these two combined abilities of measuring bites and steps allows us to not only notify wearers of increased food consumption, but also of decreased physical activity. By combining the two functions, we can effectively design systems with warnings for the intake of too many calories, and conversely the exertion of too few calories.

In addition to early prevention of ailments, wearable health sensors also provide an additional layer of data for physicians to apply to diagnoses and check-ups. While many healthcare professionals already have access to accurate and precise equipment to make objective, quantitative health measurements, it can be much more helpful for healthcare professionals to already have rudimentary health data, or even complementary data to on-premise measurements. For example, while a physician may be able to easily take a patient's heart rate during an appointment, a single measurement of a patient's heart rate may not provide a comprehensive view of a patient's cardiac health history. Had the patient worn a wearable device that could continually measure heart rate, the physician could gain additional insights into heart rate variability that cannot be found through a single measurement, such as the patient's typical resting heart rate, or irregular rhythms over time [12].

Another example lies in improving the accuracy of prior patient activity reporting. Patients often do not accurately self-report metrics, especially when concerned with more sensitive subjects such as diet, height, and weight. Patients tend to significantly under-report caloric intake and over-report exercise [13], and even when self-monitoring day to day with manual tracking, will still under-report compared to objectively quantitative bio-markers, such as doubly-labeled water [14]. This presents a problem for health care professionals, as they are given incomplete or inaccurate information. Data reporting based on wearable devices could help mitigate this issue, as it provides an objective overview of the patient's prior health metrics that will not be influenced by inherent human biases.

As such, it is clearly obvious that the adoption of health-monitoring tools could significantly shift the diagnosis-treatment paradigm established in current health care systems, potentially evolving in a more proactive healthcare approach. A proactive approach in preventative health may result in an overall healthier population, less pressure on the healthcare system due to treatment needs, and more accurate diagnoses with a more objective, inclusive range of health metrics to supplement data obtained from clinical visits.

1.2.2 The Pedometer

1.2.2.1 History

One of the earliest and simplest trackers of overall fitness and personal health has been the pedometer. A pedometer is defined as an instrument used to measure the number of steps the wearer takes. The idea of a pedometer has been around since the 15th century, when Leonardo Da Vinci proposed a similar device with military applications. However, the first pedometer was not physically created until 1780, when Swiss horologist Abraham-Louis Perrelet designed a device that "specifically measured steps and distance while walking" [15]. Early pedometers were mechanical devices that functioned much like pendulum clocks, with a mechanical system that would rotate gears to increase step count, much like the one seen in Figure 1.3. However, as micro-electromechanical system (MEMS) sensors became more widely adopted in the late 20th century, pedometers became electronic devices. Prior to the rise of smartphones and wearable technology, pedometers were often sold as individual electronic devices that individuals could purchase and carry around, but as MEMS sensors have grown smaller and more integrated with the smartphone, an independent pedometer device is no longer widely distributed, as most pedometers are currently software implementations that take advantage of the accelerometers within modern devices that individuals are likely to carry anyways, such as smartphones.

Despite changes in pedometer technology over time, the core objective of a pedometer has remained the same: to accurately and precisely count steps and distance. The step count of an individual is often an indicator of overall health and activity, and an increased daily step count is even associated with decreased mortality [16]. By using pedometers, individuals have an excellent resource to keep track of their own health, and with the drastic increase and adoption in smartphones and smart wearable technology, a pedometer is a helpful tool that does not require any additional hardware to use. Simple, unobtrusive instruments such as the pedometer are critical to the increase in adoption of self-monitoring health tools.

Even with the widespread use and simplicity of the pedometer, it is imperfect. As this thesis

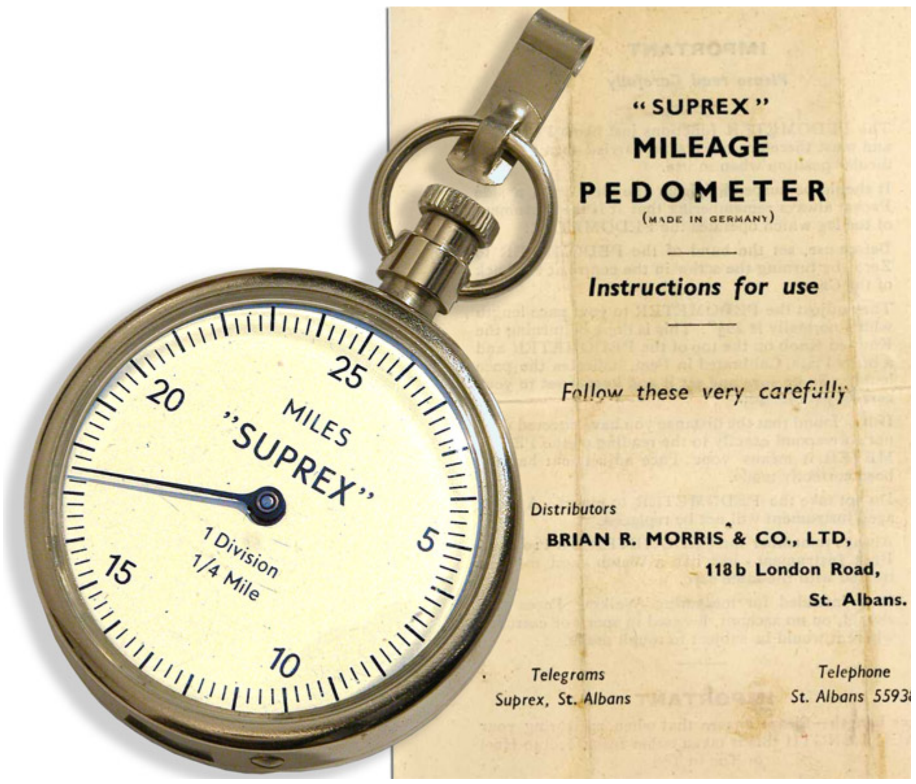


Figure 1.3: Example of a common early pedometer. The Suprex was a German pedometer manufactured in the 1970s and functioned with a mechanical lever that activated internal gearing, in turn increasing the displayed mileage count.

is primarily motivated by the improvement of the healthcare system and self-health through mobile health monitoring tools, the pedometer represents a significant area in which an improvement would benefit the entire motive as a whole. While the general trend in industry seems to suggest that the pedometer is a mature instrument, with focus in mobile health instrumentation shifting more towards more novel features such as electrocardiograms and blood oxygen sensors, the pedometer remains a major and fundamental part of self-monitoring health, and improvements to the pedometer may have positive implications for the entire mobile health industry. For example, many fitness trackers now have the ability to perform exercise detection. These methods use a combination of sensor readings from the fitness tracker to come to a conclusion that a person is exercising. By improving the fundamental algorithms that act as intermediates to a larger system, not only can we improve basic step count tracking and detection, but also improve more advanced features that leverage step detection, such as activity recognition and exercise detection algorithms.

1.2.2.2 Pedometer Accuracy

Consumer pedometers are typically evaluated on how accurately they determine the total step count within a chosen time period, by comparing the number of actual steps taken to the number of steps the pedometer shows, also called run count accuracy (RCA). While this metric is an important determinant of the efficacy of a pedometer, another important metric is how well the pedometer detects the actual times at which steps are taken, also called step detection accuracy (SDA). Unfortunately, SDA is far less common in pedometer evaluations, but just as crucial of a metric to determining overall performance. We believe that by analyzing these two metrics in tandem, we can provide a more complete idea of our work's performance relative to other studies.

As RCA is the most common metric used in pedometer evaluations, this work will continue to use this metric. RCA is calculated as a proportion of detected steps by the pedometer to actual steps taken, as shown in Equation 1.1.

$$RCA = \frac{\textit{Detected Step Count}}{\textit{Actual Step Count}} \tag{1.1}$$

An RCA of 1 in Equation 1.1 is ideal, indicating that the number of detected steps by the pedometer exactly equates the number of actual steps taken by the user. RCA metrics do not limit accuracies between a standard $0 - 1$ scale, but instead allow for higher than 1 accuracies, should the number of detected steps be more than the number of actual steps, and lower than 1 accuracies should the detected step count drop below the actual step count.

Whereas RCA intends to gauge the accuracy in total steps, SDA aims to quantify the accuracy of the pedometer in predicting the times in which unique steps occur. In accordance with previous work [17], this work uses an SDA based off of the classification of true positives (TP), false positives (FP) and false negatives (FN). A TP indicates a step predicted within $\frac{1}{2}$ a second of an actual step. Once a true positive is identified, the predicted step and the actual step in which it is paired with is eliminated from further TP consideration. FN and FP counts are calculated once this pairing process is complete, and is determined by Equation 1.2 for FN:

$$FN = \textit{Actual Step Count - TP} \tag{1.2}$$

and Equation 1.3 for FP:

$$FP = \textit{Predicted Step Count - TP} \tag{1.3}$$

Practically speaking, the FN count refers to the number of actual steps that were not detected by the pedometer algorithm, or the number of steps outside of the $\frac{1}{2}$ second range in which an actual and predicted step was paired. Similarly, the FP count refers to the number of predicted steps outside of the $\frac{1}{2}$ second range of an actual step.

In order to condense metrics into a single SDA, we use a precision measurement known as the positive predictive value (PPV), given in Equation 1.4.

$$PPV = \frac{TP}{TP + FP} \tag{1.4}$$

The PPV assesses how likely a prediction made by a pedometer will be an actual step, or in this work within $\frac{1}{2}$ a second of an actual step. Next, a measure called sensitivity is calculated, which measures how likely a pedometer algorithm will detect a step when an actual step is taken. This is defined by Equation 1.5.

$$\textit{Sensitivity} = \frac{TP}{TP + FN} \tag{1.5}$$

Lastly, the F1 score is calculated as a single standalone metric to combine both PPV and sensitivity into one single measurable value that can accurately provide insight into the SDA. The F1 score is a weighted average of the PPV and sensitivity, but biased to decrease significantly more when the PPV and sensitivity become increasingly inconsistent relative to each other, as shown in Equation 1.6.

$$\textit{F1} = \frac{2 \cdot TP \cdot \textit{Sensitivity}}{PPV + \textit{Sensitivity}} \tag{1.6}$$

As the F1 score encapsulates the PPV and sensitivity metrics that are reflective of a pedometer algorithm's overall step detection performance, the SDA is simply a measure of the defined F1 score in Equation 1.6 - that is,

$$SDA = \textit{F1} \tag{1.7}$$

The defined RCA in Equation 1.1 and SDA in Equation 1.7 are the two metrics used in this work to evaluate both the pedometer algorithm introduced in this thesis, as well as to compare to previous algorithms, discussed more in Section 1.3.

1.2.3 Sensors

The sensor used to collect the data set analyzed in this thesis is the Shimmer3, which is described in further detail in Section 2.1.1. These sensors contain multiple components, with the accelerometer specifically pertaining to most to this research. Accelerometers are sensors designed to measure the forces of acceleration of the sensor. Accelerometers are based on the concept of Hooke's Law, visualized in Figure 1.4, and can be described as shown in Equation 1.8:

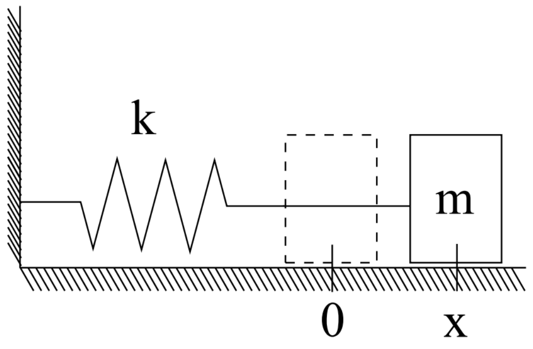


Figure 1.4: The mass spring system that describes Hooke's Law. k describes the spring constant, m is the mass, and x describes the displacement of the mass m from the labeled 0 equilibrium position [18].

$$F = -kx \tag{1.8}$$

where F is the force exerted on the spring, k is the spring constant, and x is the displacement of the spring from its initial equilibrium position to its final position. By taking Newton's second law

$$F = ma \tag{1.9}$$

where F is the force, m is the mass of the object attached to the spring, and a is the acceleration of that mass, we can combine Equation 1.8 and Equation 1.9 in such a way that allows us to calculate the acceleration a of the mass m, given the measured displacement of the spring from its neutral position x and the spring constant k:

$$a = -\frac{kx}{m} \tag{1.10}$$

With Equation 1.10, the acceleration of a mass spring system a can be calculated by simply measuring the displacement of the spring x, given constants m for the mass and k for the spring constant, as m and k remain unchanged for the purposes of an accelerometer.

As briefly described in Section 1.2.2.1, MEMS is an acronym for a micro-electromechanical system. These devices combine electronic and mechanical components through a circuit board on a single chip. MEMS sensors are often used to digitize physical measurements, such acceleration and rotation. As such, tools such as accelerometers and gyroscopes utilize MEMS sensors to make measurements. Pedometer algorithms are typically applied to accelerometer data, and sometimes gyroscope data as well, meaning that the core foundation of the data in the application of any pedometer algorithm consists of measurements collected from a MEMS sensor. Modern accelerometers use a combination of three separate MEMS sensors to measure acceleration along each axis, while gyroscopes also use a combination of MEMS sensors to measure rotational velocity. Both accelerometers and gyroscopes can be combined into a single inertial measurement unit (IMU), an electronic device for gathering acceleration and rotational velocity measurements. Nearly all modern smart wearables and smartphones contain an IMU, and although the components for which an IMU can measure can differ, nearly all IMUs contain at the very least an accelerometer. As most human activity recognition algorithms are implemented with accelerometer data and rarely gyroscope data [18], we have limited the scope of this research to contain only data gathered from the x, y, and z axes of acceleration on our study participants.

1.2.4 Neural Network Approach

In this thesis, we attempt to solve our problem with a deep learning approach. This section will provide a high level overview of fundamental machine learning concepts and terminology that will be prevalent in the description of our methods in Chapter 2. For purposes of this thesis, these descriptions will largely be based off of neural network models, as the overall field of machine learning is a very complex and broad subject that is beyond the scope of this paper.

1.2.4.1 Machine Learning Basics

There are four major categories of machine learning: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. The first of these, supervised learning, is a method in which labeled data is provided to a machine learning algorithm. Labeled data is data that has been identified with the ground truth. The ground truth is labeled values of data defined as the ”correct” values of the item that the machine learning algorithm attempts to predict. For example, if we were to try and predict the temperature at a given date in a year, the ground truth of the data set would be the recorded temperatures, where the recorded temperature data could be acquired from a trusted weather service. Supervised learning is the method used for this thesis, and its process is further described in Section 1.2.4.2.

Unsupervised learning can be considered the opposite of supervised learning. Instead of giving labeled data to an algorithm to train on, the algorithm is given unstructured, unlabeled data, and is instead tasked with drawing conclusions from unlabeled data. This method is often used when researchers are unsure of the patterns in the data. By tackling the problem through unsupervised learning techniques, patterns and relationships not immediately discernible by researchers can potentially be uncovered. This approach is also widely employed in clustering problems to discover groups of similar data that is related in difficult to determine manners.

If supervised and unsupervised learning are opposites, semi-supervised learning is a middle ground. Semi-supervised learning is applied to the same problems unsupervised learning can be used to solve but combines elements of supervised learning to enhance the process. A relatively small proportion of labeled data is introduced to the large proportion of unlabeled data to help the model perform better. For example, in image recognition algorithms, introducing labeled data can drastically improve performance of image classification on images corrupted by digital noise [19].

Lastly, reinforcement learning is also a popular method of artificial intelligence. Reinforcement learning focuses on training an algorithm to optimize a sequential instruction set to perform a task. As an example, suppose we want to use machine learning to train a remote-controlled car to race around a pre-determined track, optimizing for the best lap time. The algorithm and the car have no knowledge of the track to begin with, but through experience, the algorithm will eventually learn the turns of the track. Through multiple tries, the algorithm will eventually optimize turns and speed for the best lap time.

Machine learning, regardless of the method, is an iterative, optimization-based approach in which a computer can gradually improve the task it has been given by repeatedly attempting the given task. This process is not unlike the procedure in which humans learn a new activity. For example, if an inexperienced, yet able-bodied participant is shown how to properly kick a soccer ball for the first time, it is doubtful that the participant will be able to take the kick with perfect technique. However, through the iterative process of practice, they will eventually experience many different kicks and learn different techniques that result in better kicks, leading to general improvement and eventually proficiency in properly kicking a soccer ball.

Machine learning can be thought of as a similar cyclic process of progressive improvements. The field of machine learning is essentially the study of such techniques in computers; that is, algorithms that have the ability to dynamically change in order to better accomplish its task. Before proceeding to the specific machine learning techniques used for this research, it is important to explain some of the basic ideas. Firstly, a machine learning algorithm refers to the set of rules in which the computer can modify and adapt parameters to best produce a prediction based on the data set it has been trained on. The prediction refers to some final value that the machine learning algorithm forecasts based on a given query. This prediction is defined by the type of algorithm and architecture used to train a machine learning model, as well as the data set given. The training data set refers to the collection of information provided to the machine learning algorithm that is pertinent to helping the algorithm generate better predictions. Referring back to the example of using machine learning to predict weather, the prediction could be the temperature at a certain time of year, whereas the training data set could consist of the temperatures and dates for the past five years. The data set given to a machine learning algorithm to train on is one of the most essential elements of producing a well-trained machine learning model. As machine learning as an experiential process, the broader the data given to a machine learning algorithm is, the better it will generalize. Referring back to our soccer example in the paragraph above, this phenomenon can be likened to a participant kicking a soccer ball in multiple ways. The more ways the participant has trained with kicking the ball, the better they will be when given a target to kick at. With a large sample of different kicks, the participant will eventually learn the minute deviations from an ideal technique that result in a weaker kick, and the strategies that work best in a stronger kick. A machine learning algorithm behaves much in the same way. The more data it has to train on, the better the algorithm performs when generalized to new data that has not been fed to train the model.

Training a machine learning model refers to the iterative process in which a machine learning algorithm uses the training data set given to it to improve its predictions. When a model is training, it is undergoing a process of learning the relationship between the given data set and the correct value of its prediction. Taking our weather example again, the training operation would refer to the process in which the machine learning algorithm defines a mathematical relationship between the dates and the temperatures given to it from the training data set. Training is done in iterations, where every iteration the algorithm makes a set of predictions and compares its predictions to the ground truth in the training data set. Eventually, these iterations may make up an epoch, or an entire pass through the training data set. Depending on how close it is to the ground truth, the algorithm can dynamically adjust the values of mathematical relations between the data and the prediction, called weights. This variation between the ground truth and the prediction is determined by a value known as the loss, calculated by the loss function. While the loss function is up to the implementation of the algorithm, the goal is to minimize the loss, and hence the better the predictions, the smaller the loss. This process is repeated until the algorithm is determined to stop, commonly set at a designated number of epochs, or once the loss has reached a certain acceptable value. Once this process of iterating over epochs is finished, training is complete, and the weights and mathematical relations determined from the training process can be saved to a model.

Once a model has been trained, it can be evaluated. The majority of machine learning models are not trained on the entire available data set; rather, a smaller subset of the entire data set is reserved for validation purposes. This is done to prevent a phenomenon called overfitting. Overfitting occurs when a machine learning algorithm has learned the patterns in a given data set *too* well and creates predictions that are too dependent on patterns specific to the training data set. This becomes a problem as it means that the model has learned to predict labels for the training data set very well but may do poorly when generalizing to a larger data set it has not seen before. For example, suppose a model has been trained on a weather data set containing temperatures and dates from the state of Hawaii (HI). Aside from the obvious error of a poor choice of the training data set due to its limited location, the biggest impediment in the trained model becomes overfitting. A model trained on data from HI may make temperature predictions well for Hawaii and other geographic areas with similar climates but would fail at making predictions for an area with a different climate. For example, suppose a user in Honolulu, HI queried the model to make a prediction for the weather on November 1st. The model may make a good prediction in line with typical historical weather for the location at the time of year, but if a user in New York, New York (NY) had queried the same thing, the model would make a poor prediction, since it has only been trained with a restricted data set. For reference, the average high temperature in Honolulu, HI in November is 84°Fahrenheit (F), whereas the average high in NY, NY is 55°F, as per NOAA data.

1.2.4.2 Deep Learning

In order to understand deep learning and the novelty behind this thesis, we must first explain the concept of neural networks. A neural network is a system of interconnected virtual nodes designed to compute some intermediate value. These values are then modified as they pass through nodes until a final prediction is made, which changes depending on the usage and type of neural network. As evident in its naming, neural networks are inspired by biological cognitive processes, specifically the nervous system. In biology, neurons fire when certain triggers activate them, which successively triggers other neurons until the process is complete. For example, a thought to raise your arm could act as the original trigger, and a series of neurons firing will eventually result in activation of the muscles required to raise your arm.

While neural networks are completely virtual computing nodes, they act in a very similar way. Despite not being as complex as a biological nervous system, the mechanisms and architectures are analogous. Neural networks contain at least two layers of nodes; and input and output layer. As a single layer of a neural network consists of a vector of these computing nodes, the input layer must be the same size as the given data, while the output layer may be mapped to a different size. This is useful as it can constrain or expand the nodes needed to perform different computations, which can vary widely depending on the specific problem being solved by the neural network. When multiple neural network layers are combined together, a deep neural network is formed. Deep learning is the use of these multi-layer neural network architectures to produce some prediction. In addition to the basic input and output layers, deep neural networks contain at least one and often multiple hidden layers. These hidden layers are essentially layers between the input and the output layers designed to extract certain features that may not be readily apparent through a single input and output layer. An example of the architecture behind a simple neural network is shown in Figure 1.5.

A critical piece of neural network architecture is the activation function. As likened to a biological stimulus that triggers the first neuron in a nervous system, an activation function in a neural network layer defines when the nodes in that layer may be triggered. Activation functions

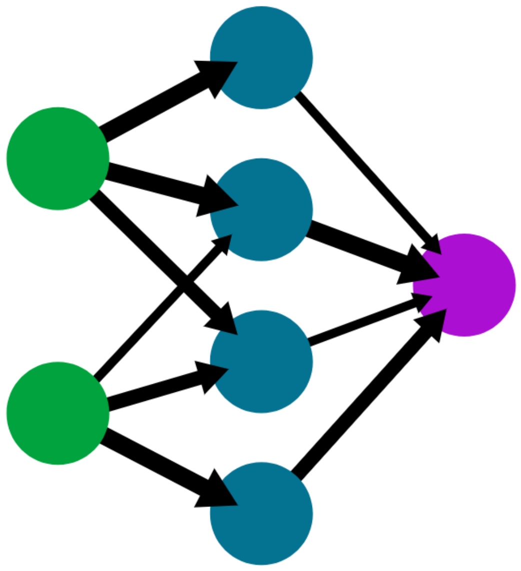


Figure 1.5: Simple example of a neural network architecture. The layer to the left is the input layer, the layer in the middle is the hidden layer, and the layer to the right is the output layer.

are applied to the output of a neural network layer, and defines how the next layer of a neural network is triggered. While a variety of activation functions exist for different purposes, the goal of the activation function is to both constrain the output domain of a neural network layers, as well as introduce non-linearity to the output. This ensures that problems that may not follow linear patterns can still be solved using deep neural networks. As linear relationships are often simple enough patterns that they do not require neural networks to solve, introducing non-linearity is critical to allowing neural networks to learn complex connections between the data and the labels.

Although the process of training was briefly mentioned in Section 1.2.4.1, training in neural networks takes on a more specific goal of performing weight optimization between neuron connections through gradient descent. Gradient descent is a process of finding the local minimum within a differentiable function. In neural networks, this function is also the loss function mentioned in Section 1.2.4.1. One of the most common loss functions for gradient descent in neural networks is mean-squared error (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2. \tag{1.11}$$

In Equation 1.11, n is the total number of samples, Y is the ground truth label, and $\hat{Y}$ is the predicted label. The MSE compares all predicted results with the ground truth, squaring the difference to eliminate over and under-predictions cancelling out the difference. The algorithm then adjusts the weights of parameters, determined from the training data set, and repeats an epoch, calculating the loss once again. In neural networks, weights are adjusted in an attempt to minimize the loss through a process called back propagation, where gradient descent optimization is performed after every epoch to iteratively reduce loss across the entire model.

Neural networks have been steadily on the rise in industrial applications [20, 21, 22, 23] due to their great flexibility in adapting to numerous problems, as well as their robustness in using even the same architecture for different data sets. For example, our research group currently studies eating detection using neural networks in addition to the step detection covered in this thesis [24, 25]. The same neural networks applied to detecting bites can be applied to detecting steps, with just minor modifications, and vice versa. However, it is important to note that the data collected for our group's bite counting research was collected using the same Shimmer3 IMUs and organized in a similar manner. This is affirmation of one of the primary drawbacks of deep learning neural networks: despite the relative adaptability of neural networks compared to other machine learning approaches, the data set required for a neural network approach is critical in obtaining relevant results. Building a data set comprehensive enough to satisfy a neural network solution is often more time consuming and difficult than simply duplicating the network architecture from other works, and thus the usefulness of a neural network method must be carefully reviewed before applying the concept to just any data set.

1.2.4.3 Convolutional Neural Networks

Convolutional neural networks, or CNNs, are neural networks that utilize the abilities of convolution to extract patterns from the given data set. Convolution is a mathematical process that describes how two functions are related; that is function f and function g are evaluated to see how one is affected by the other. This can be expressed by Equation 1.12, where t is the real number variable of functions f and g.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{1.12}$$

Convolution is often used in computer vision to detect features in images. As images have both rows and columns, the convolution is in two dimensions. This process occurs when a template image, called a kernel, is convolved pixel-by-pixel across the main image. The mathematical convolution will produce a matrix of values, which can then be aggregated and centered on the kernel. Common aggregations of the convolutional result include mean filtering, which simply takes the average of all values in the convolution and centers the mean on the pixel convolved in the middle of the kernel. An example of two-dimensional convolution across an image with a kernel can be seen in Figure 1.6. It is also worth noting that for both computer vision and CNNs, the convolution described is typically an equivalent term for cross-correlation.

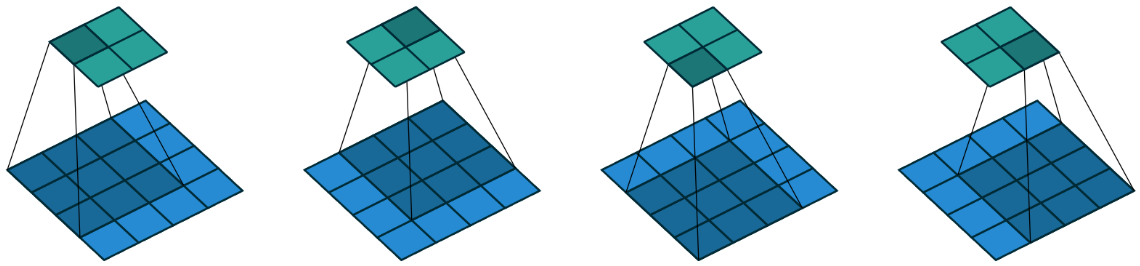


Figure 1.6: A visual example of two-dimensional convolution. The square on top represents the 2x2 area produced by convolving a 3x3 kernel across a 4x4 image on the bottom. As borders cannot be captured by convolution without padding, the 4x4 image loses its borders during convolution and results in a 2x2 area [26].

After convolving the kernel across every pixel in the image, the values are then normalized, or scaled within a defined window. In general, scalar normalization follows Equation 1.13:

$$I_N = (I - Min)\frac{newMax - newMin}{Max - Min} + newMin \tag{1.13}$$

where I_N is the normalized value of a given datum, I is the original value of the datum, Min is the minimum value of any data in the original image, Max is the maximum value of any data in the original image, $newMin$ is the minimum bound of the normalization, and $newMax$ is the maximum bound of the normalization.

In traditional computer vision, this overall technique is called template matching. While many different template matching algorithms exist, for this example we will use what is known as normalized cross correlation (NCC). This example of classic template matching is from [27], as part of another study under our research group. The equation for the value of the NCC at location (x, y)

is shown in Equation 1.14 [27].

$$NCC(x,y) = \frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b}(T(s,t) * I(x+s,y+t))}{\sqrt{\sum_{s=-a}^{a}\sum_{t=-b}^{b}T(s,t)^2 * \sum_{s=-a}^{a}\sum_{t=-b}^{b}I(x+s,y+t)^2}} \tag{1.14}$$

A template image, often called a kernel, is convolved pixel-by-pixel across the main image. The mathematical convolution will produce a matrix of values, which can then be averaged and centered on the kernel. After convolving the kernel across every pixel in the image, the values are then normalized, or scaled within a defined window. Higher values in the matrix represent areas that much more closely match the template image than lower values. In Equation 1.14, T is the template image, I is the original image, a is half the width of the template (rounded down), and b is half the height of the template (rounded down) [27]. The term in the denominator is a normalizing term that ensures all scores range from 0-1, and is an extension of Equation 1.13. To compute the entire NCC, this equation must be computed at each pixel in the original image. Additionally, by changing the normalization of the NCC to 0-255 to output an 8-bit grayscale image, we produce what is known as the matched-spatial filter, or MSF. The better matches in the MSF image can be visualized as the brighter spots. An example of the NCC template matching process is shown in Figures 1.7 and 1.8.

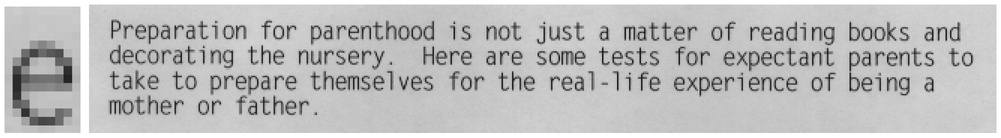

Preparation for parenthood is not just a matter of reading books and
decorating the nursery. Here are some tests for expectant parents to
take to prepare themselves for the real-life experience of being a
mother or father.

Figure 1.7: The template image, or kernel (left, enlarged) and the original image (right). The NCC matches the template inside the original image.

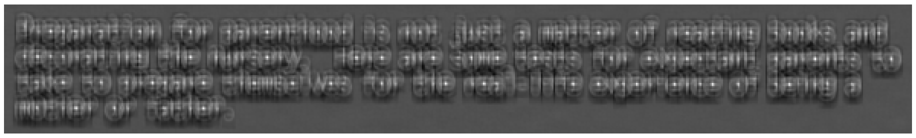


Figure 1.8: The MSF image for the NCC example. Notice how bright spots occur at the locations corresponding to the center of all letters that resemble an 'e' in the original image.

As illustrated, classic template matching is a powerful use for convolution. However, template matching suffers from many pitfalls. For example, take the Modified National Institute of Standards and Technology (MNIST) database of handwritten Hindu-Arabic numerals, illustrated in Figure 1.9. While most of these digits are distinguishable to humans, by using classic computer vision template matching techniques, one digit may not be classified as the correct digit. Suppose the leftmost '9' is clipped and used as the kernel image for template matching. A '9' rotated in a different direction, such as the '9' to the right of the clipped numeral, does not follow the same template, and would yield a significantly poorer NCC value than a '9' that more closely resembles the chosen template, with the same rotation. Similarly, imagine attempting to find dogs in an image by using template matching. The dog in the image would have to very closely match the orientation, posture, and even shape of the template dog image relative to the camera in order to yield a high NCC template matching score. Fortunately, CNNs address many of the weaknesses found in traditional computer vision.

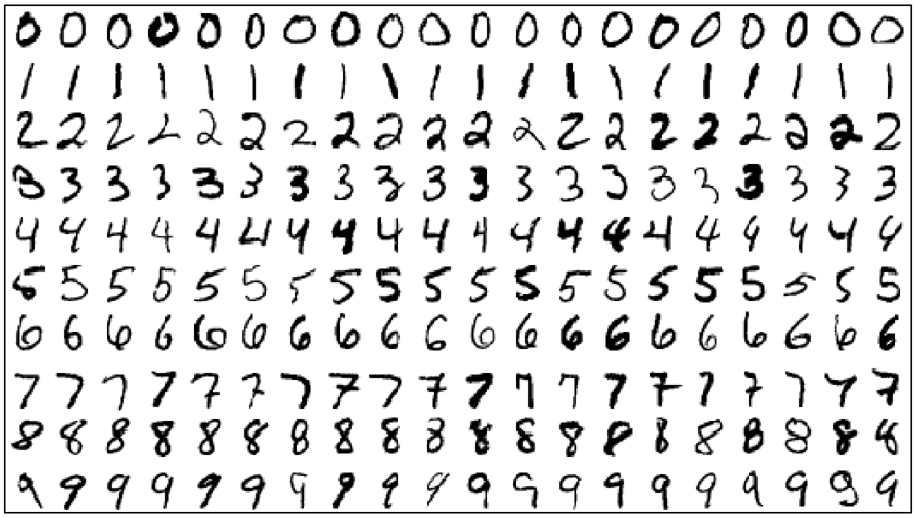


Figure 1.9: Example of the MNIST handwritten digit database. Notice how the leftmost '9' differs from many of the other '9's due to its orientation.

A CNN operates in a similar fashion as traditional template matching algorithms, but the key difference is that CNNs can generate their own filters (kernels or template images in traditional template matching) to use against the original image. These generated filters can pick up independent features of an item in an image that details what the item is. By using multiple of these filters, a CNN can pick up individual features of an item. For example, in the MNIST digit database, multiple filters could be generated by a CNN. The result of a convolution between a '9' and one of the filters would be different from the result of a '0' convolved with the same filter. Across all the filters, the CNN will then adjust weights for each neuron connection until it learns the filters that create strong outputs matching the ground truth.

While two-dimensional convolution is common for images and one of the most frequently applied neural networks in practice, for this thesis, we examine time-series sensor data, a one-dimensional data type. As such, instead of the 2D convolution described above, our data is convolved in a single dimension, exemplified by Figure 1.10.

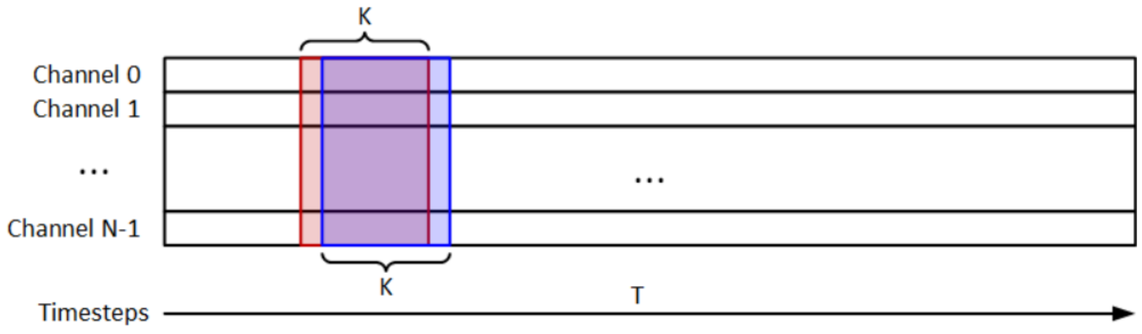


Figure 1.10: A visual example of one-dimensional convolution. Each channel represents some given time-series data for timesteps T. The kernel size K is the length of the kernel vector used in the convolution.

One-dimensional convolution is essentially the same process as a two-dimensional convolution, but the convolutional kernel only slides in a single dimension. Whereas we see the 2D convolution slide the kernel in both the horizontal and vertical orientation across the image, with our 1D convolution, we only see our kernel slide across the time domain. Again, the intrinsic value of a 1D convolution is the same as a 2D convolution: we are looking for a pattern of features through a kernel that can be mapped to a value. 1D convolution is often applied to a wide variety of objectives, including cardiac arrhythmia detection through electrocardiogram (ECG) signals [28] and building damage monitoring through shock signals [29]. The data utilized in training a 1D CNN for both these examples, as well as in this research, share a commonality in sensor readings across the time domain. Evidently, time-series data typically lends itself well to implementations of 1D CNNs.

1.3 Related Work

As described in Section 1.2.2.1, early pedometers were mechanical systems tuned with physical components and gearing. This gearing would turn an analog dial, which displayed either steps or distance, not unlike an analog clock. With the rise of MEMS-based pedometers, related work in this area has shifted towards heuristic algorithm design in filtering MEMS sensor data associated with a step. Previous work in pedometer accuracy has concentrated on improving and tuning these algorithms to produce the best deterministic rule set of what constitutes a step.

These algorithms consist of three major categories: peak detection, thresholding, and autocorrelation. The peak detection algorithm, broadly speaking, attempts to detect a step by counting the local maximums reflected in the sensor signals [30]. Thresholding algorithms such as [31] attempt to detect steps by determining the number of times sensor signals drop below or increase above set thresholds, again analyzing the periodic peaks and troughs in sensor activity. Lastly, autocorrelation algorithms such as [32] compare the pattern of sensor signals when a step is taken with a given window of data. Autocorrelation spikes when the two patterns are similar, thereby detecting a step. It is important to note that while the three implementations behind these papers are specific, the particular methodology behind peak detection, thresholding, and autocorrelation can vary, as evidenced by the differing designs published in both peak detection [33] and autocorrelation [34].

Due to the diversity in devices available to consumers that can implement step counting and detection, many related studies also emphasize the evaluation of pedometer algorithms from sensors mounted on a range of body positions. While the most common pedometers by far exist in the forms of smartwatches and smartphones, a cursory search on Amazon.com shows that a variety of dedicated pedometer devices meant for use with different body positions still exist. Additionally, end-users do not always heed ideal operating instructions, so pedometers meant for use in one body position may still be mounted to other body parts. A variety of pedometer use locations are evaluated in [17, Tab. 3.1], including the head, chest, arm, wrist, hand, hip, leg, ankle, foot, and on non-body positions such as pockets and handbags as well. With the increasing use of pedometers on the smartphone, the effect of smartphone orientation on pedometer performance has also been tested [35, 36, 37].

Previously in our group, studies have been taken to reproduce results from [30, 31, 32] with our own data set [3, 17]. Additionally, work was done to improve upon these surveyed step detection algorithms by automatically switching between algorithms through gait detection techniques [17]. Despite the myriad of methods and the quantity of papers in improving pedometer accuracy, the use of neural networks to improve upon existing algorithms has not seen significant uptake, contrary to the increasing application of CNNs to other human body sensor measurements. As written in [38], "deep learning methods [are receiving increased] attention within the field of human activity recognition due to their success in other machine learning domains," but very few instances of deep learning have been applied to the pedometer itself. In fact, in a survey of over 300 papers [39], only one was found to study the application of machine learning towards specifically counting steps [40]. Additionally, it is important to note that while some literature may apply deep learning towards detecting steps, these detections are often used as a means to categorize some other factor, such as activity or gait recognition, in which other methods such as Fourier analysis become popular tools.

While [40] does apply a deep neural network to the pedometer, their CNN algorithm focuses on a classification problem between three different labels: a left step, a right step, and what they consider a negative step, which is essentially the collection of accelerometer signals from usage of a smartphone without actually taking a step. When the CNN is applied to this labeled data, it struggled with distinguishing between a left and right step, as expected, so both left and right steps were combined into what they call a positive step. While this algorithm is directly used to detect steps, it primarily focuses on mitigating the effects of negative step noise in sensor readings used in training the CNN to classify steps.

In addition to [40], others have also applied neural networks in a related manner, such as [41] on using double autocorrelation for pedometer algorithms. While this paper also applies a deep learning model directly towards the pedometer, the model is not used to detect steps in itself. In fact, the neural network used for this study instead relies on classification techniques to detect and segment periods of true step activity, rather than to detect individual steps. The neural net will classify between continuous activity and no activity, before running a double autocorrelation algorithm to individually determine the steps within the classified activity period. This is similar to [17], where a gait analysis is performed before one of the classic step detection methods is applied.

1.4 Novelty

The main novelty of this work is that it is the first to use an integration-based convolutional neural network to count and detect steps in accelerometer data from a body-worn sensor. This work attempts to leverage the power of CNNs to extract step patterns, thereby predicting the number of steps within a given window. This also allows estimates of the times at which steps were taken by distributing the number predicted steps throughout the prediction window. Moreover, by calculating an integral across all of our prediction windows, we can determine the step count across many windows in a larger time frame, similar in function to a fitness application's running step counter for daily step target tracking.

Windows of steps that the CNN operates on are agnostic to the step count within the window. As such, these windows contain an arbitrary number of steps, including zero. We therefore take a CNN approach in predicting the number of steps within the window, which will allow us to both count and detect the number of steps within any given time frame. Specifically, the goal of this thesis is to evaluate the following three questions:

1. Can a neural network learn to detect and count steps by analyzing a sliding window of accelerometer data?

2. Does the length of the sliding window affect its prediction accuracy?

3. How does the accuracy of this approach compare to the classic peak detection algorithm [30], threshold-based algorithm [31], and autocorrelation algorithm [32]?

Chapter 2

Methods

This chapter first describes the data used in our experiments. We then describe the three main steps in our method: pre-processing, step count prediction within a window using a neural network, and post-processing in which a sliding window count is integrated in order to output individual step detections.

2.1 Data Set

2.1.1 Shimmer3 IMU

All sensor data used in this study was collected using the Shimmer3 IMU unit. The Shimmer3 IMU unit is a sensor produced by Shimmer Sensing that features two accelerometers, a gyroscope, a magnetometer, and an altimeter, with optional additional accessories that allow for deeper sensor expansion. Pairing the two accelerometers in tandem allow for the sensor to make precise, low-noise accelerometer measurements at the cost of its measurable range. Due to the relatively small-scale accelerations of human step activity, the low-noise low-range option was used to collect this data set. The orientation of the Shimmer3's accelerometers is given in Figure 2.1. Data collected from the Shimmer3 sensor is stored on an SD card, and can be transferred to a computer through the included Shimmer docking station. Shimmer additionally includes a software suite that allows for sensor management, where the data can be viewed and exported to a CSV file for further processing and analysis. The sensor also offers multiple units in which data is recorded in. For this data set, the accelerometer readings were recorded in gravities, where 1 gravity $\approx 9.81 m/s^2$. In addition, the sensor could also be set to record at different frequencies. The data set used in this thesis was collected a sample rate of 15 Hz, allowing us to easily interpolate our sensor samples with our ground truth video, described in Section 2.1.2.

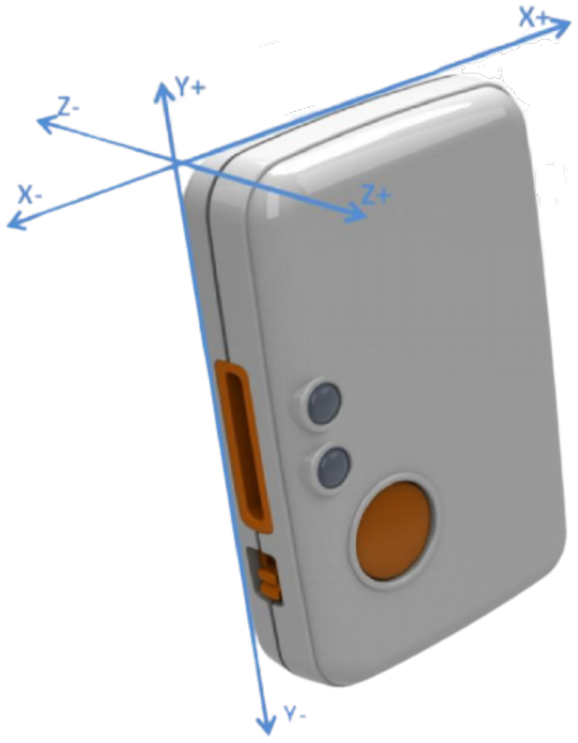


Figure 2.1: The Shimmer3 IMU. This figure shows the sensor along with axes showing the orientation of the sensor's measurements. For example, if the negative Y axis points towards the ground, the accelerometer reading would be negative, due to the effect of gravity on the IMU's readings.

2.1.2 Data Collection

The data set used in this thesis was collected by our group in a previous work [3]. Using the Shimmer3 sensor described in Section 2.1.1, three Shimmer3 units were attached to each subject; one on the wrist, one on the ankle, and one on the hip, as seen in Figure 2.2. Additionally, participants wore a Fitbit Charge 2 directly adjacent to the wrist Shimmer IMU in order to gather consumer pedometer metrics for comparison.

A total of 30 participants were analyzed for this data set, with an even split of 15 males and 15 females in order to provide a more comprehensive range of step gaits. Each participant was

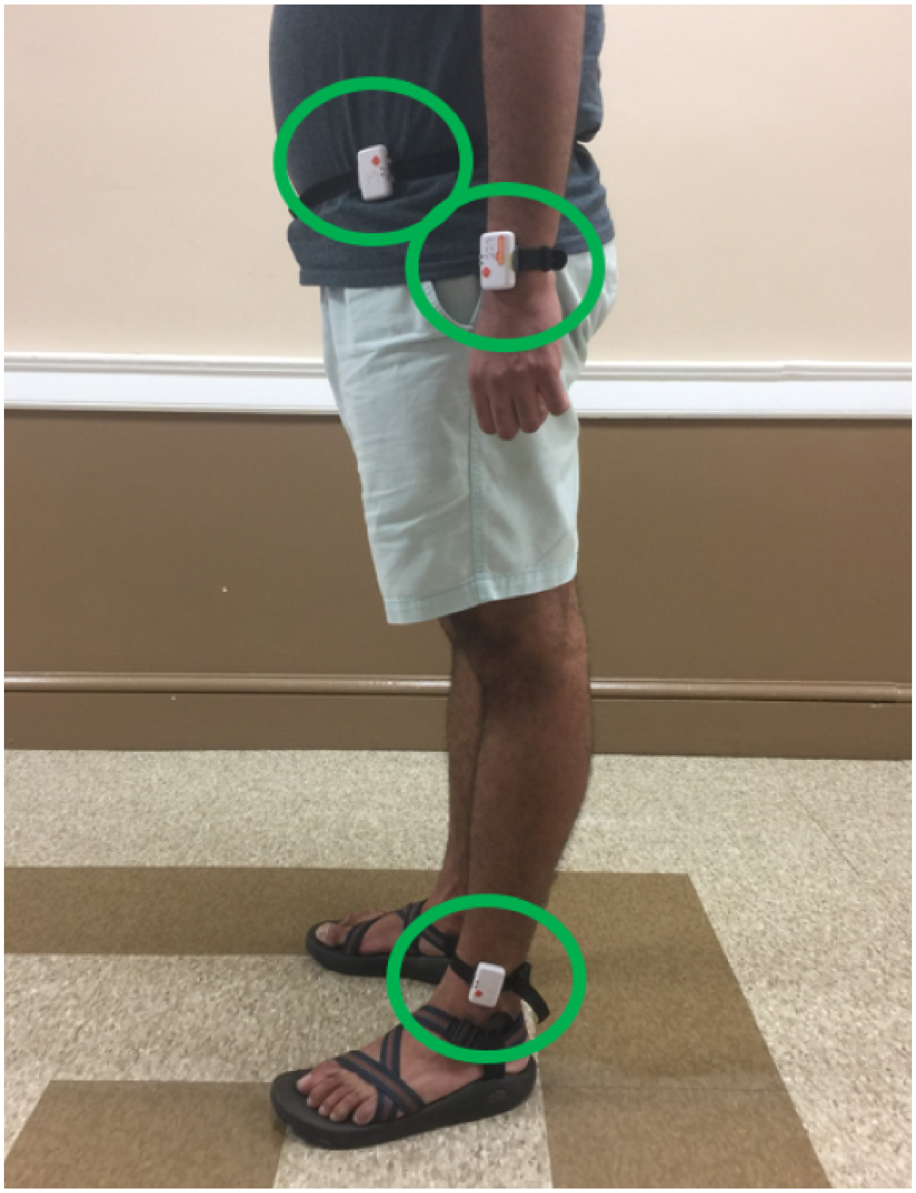


Figure 2.2: The three Shimmer3 devices worn at the wrist, hip, and ankle by a participant.

surveyed on their overall health and fitness in order to allow subjects to self-assess their ability to complete the physical requirements of the experiment. Additionally, the data collection was approved by the Clemson University Institutional Review Board for the protection of human subjects (IRB Number: IRB2017-048). Each participant in this study was instructed to perform three different activities: walk normally around a track, perform an indoors scavenger hunt, and build a Lego kit with groups of pieces distributed around a room. These activities were designed to simulate regular, semi-regular, and irregular gait types, respectively.

For the regular gait, participants were instructed to walk two laps around the Watt Family Innovation Center at Clemson University. At roughly 350 meters of perimeter, walking around the Watt Center allowed for an approximate recording length of 10 minutes for normal gait activity for each participant, for around 700 meters of total walking length. The walking track is shown in Figure 2.3.

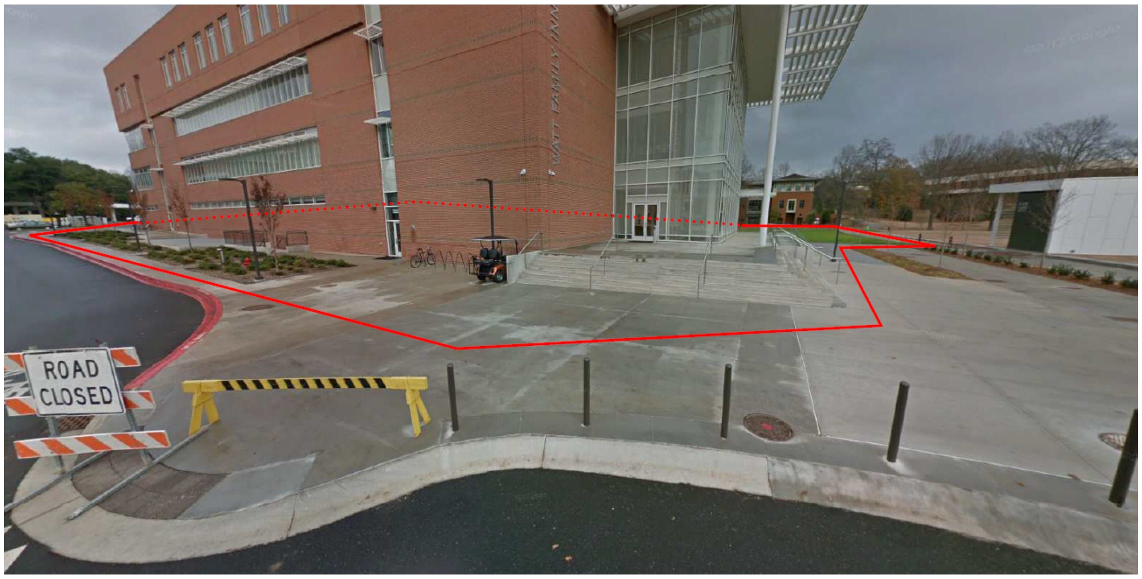


Figure 2.3: The walking track simulating a regular gait is outlined in red, surrounding the Watt Family Innovation Center at Clemson University.

To simulate semi-regular gaits, participants were instructed to perform a scavenger hunt in the classrooms of Riggs Hall at Clemson University. Four small toy footballs were hidden in four different classrooms in Riggs, each on a different floor. The classrooms used in this experiment typically contain multiple obstacles (exemplified in Figure 2.4), making it more difficult for the participant to locate the football, thereby extending the length of the experiment and increasing the amount of data collected. Participants were given the rooms in which footballs were located and were instructed to find each football in the room with limited help from the researcher. After each item was found, participants would be led to the next room, until all footballs were found. The scavenger hunt promotes many motions not typically found in a normal walking gait, such as stopping and starting, shuffling, and pivoting.

Lastly, irregular gaits were simulated by leading the participants into a classroom with

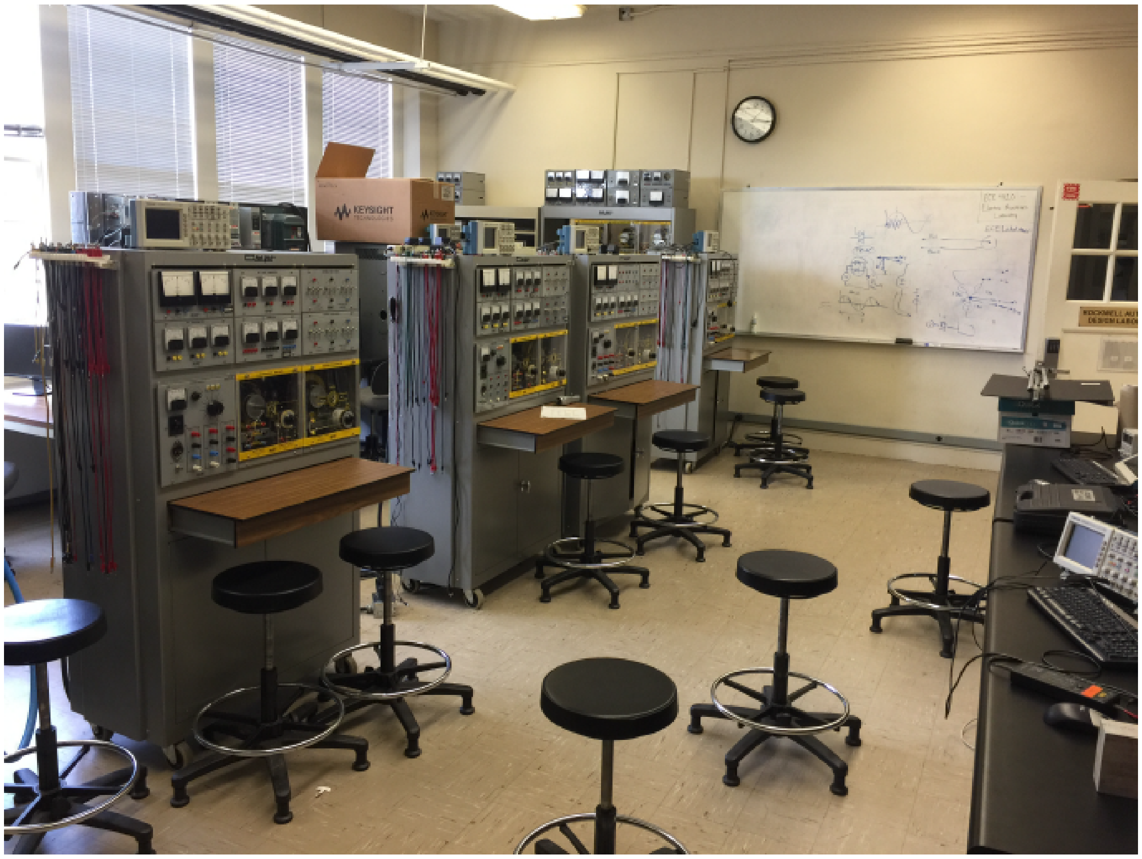


Figure 2.4: One of the classrooms used in the semi-regular gait experiment. A variety of obstacles promotes a wide range of stepping motions.

various Lego pieces spread out into multiple bins across the room. Participants were instructed to use a central table at the front of the room to build a Lego toy using an instruction manual. Participants were required to retrieve certain pieces of the toy in the different bins distributed across the room when needed, but were only allowed to build at the central table. An illustration of this layout can be seen in Figure 2.5. This activity simulates conditions in which subjects are stationary for the majority of the time, but still frequently perform wrist movements.

Prior to starting each experiment, researchers began recording video of the participants' feet using an iPhone 4, shot at a resolution of 1280x720 at 30 frames per second. As briefly mentioned in Section 2.1.1, shooting at 30 frames per second allows us to easily interpolate our 15 Hz sensor recording, as every other frame in our video will correspond with a sensor reading from the Shimmer3 IMU. Participants were then fitted with Shimmer3 sensors as seen in Figure 2.2, and each sensor was set to record by holding the main orange button on the sensor down for three seconds. The

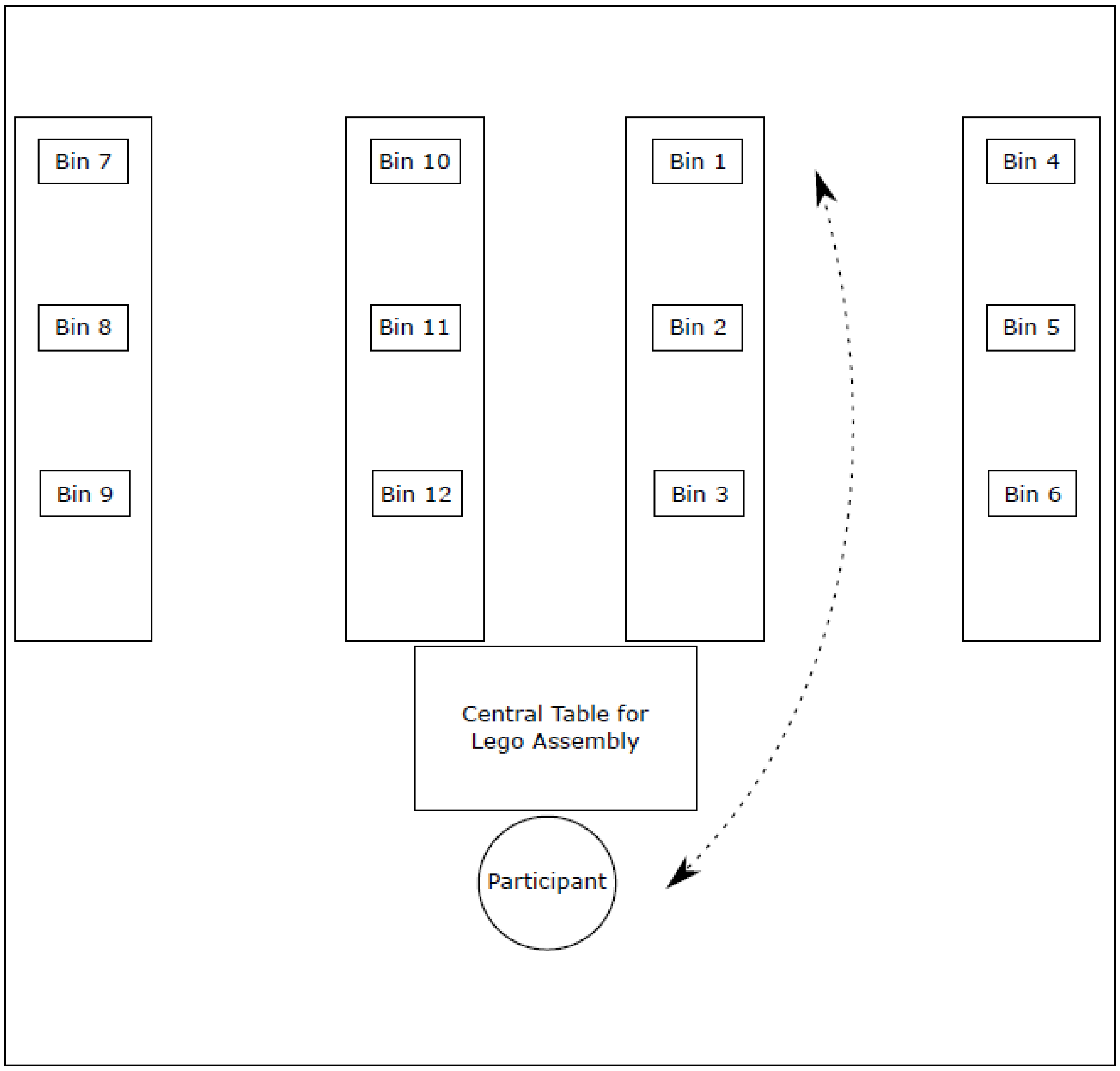
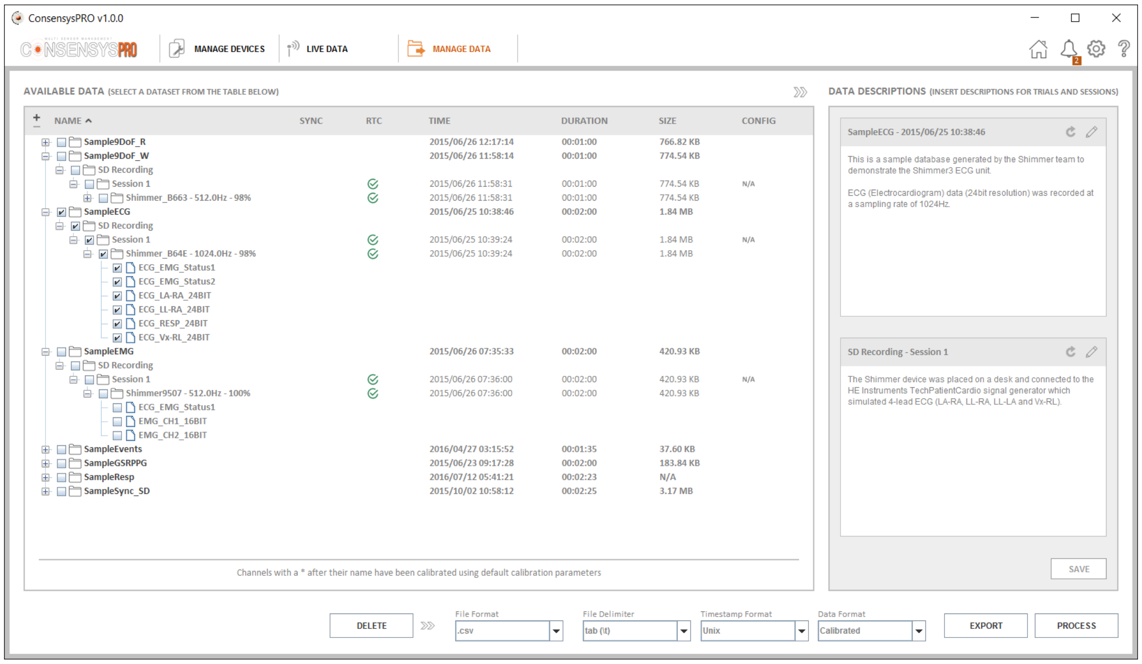


Figure 2.5: An illustration of the room layout used in the irregular gait experiment. Lego pieces are distributed throughout multiple bins located across the room, but the participant may only build at the central table. This promotes periods of steps, followed by longer periods of standing and shifting.

video recording focus shifts from the participants' feet to the Shimmer3 during sensor activation, as this will be used to determine the time offset between the video recording and the sensor recordings. After verifying that all three sensors were recording data by visually confirming the blinking LED on the sensor, the experiment begins.

Following the completion of the experiment, the data recorded on the IMU is exported into CSV format through the Shimmer's software, ConsensysPro, shown in Figure 2.6. The CSV files are exported as Sensor01.csv, Sensor02.csv, and Sensor03.csv, corresponding to the wrist, hip, and ankle mounted sensors, respectively. These CSV files are imported by our group's Stepcounter VIEW program, which can be used to manually ground truth the times at which steps are taken, determined roughly by the frame at which a participant's foot can be seen striking the ground in the recorded ground truth video. The raw CSV data is imported and processed by the program along with the ground truth video recording of the participant's steps, as well as a manually entered offset file indicating the offset in time between the beginning of the ground truth video recording and the beginning of the last-activated sensor recording. Offsets between each sensor recording are automatically calculated within Stepcounter VIEW, as each sensor's exported CSV file contains a synchronized time measurement. An index of 0 in Stepcounter VIEW corresponds with the first sensor reading obtained from the last device set to record, as well as equivalent timestamps in the other two sensors. To explain further, since sensors are not all activated at the same time, the first shown recording in our GUI viewer corresponds with the first recorded timestamp of the IMU activated last.



AVAILABLE DATA (SELECT A DATASET FROM THE TABLE BELOW)

NAME	SYNC	RTC	TIME	DURATION	SIZE	CONFIG
Sample9DoF_R			2015/06/26 12:17:14	00:01:00	766.82 KB	
Sample9DoF_W			2015/06/26 11:58:14	00:01:00	774.54 KB	
SD Recording						
Session 1		✓	2015/06/26 11:58:31	00:01:00	774.54 KB	N/A
Shimmer_B663 - 512.0Hz - 98%		✓	2015/06/26 11:58:31	00:01:00	774.54 KB	
SampleECG			2015/06/25 10:38:46	00:02:00	1.84 MB	
SD Recording						
Session 1		✓	2015/06/25 10:39:24	00:02:00	1.84 MB	N/A
Shimmer_B64E - 1024.0Hz - 98%		✓	2015/06/25 10:39:24	00:02:00	1.84 MB	
ECG_EMG_Status1						
ECG_EMG_Status2						
ECG_LA-RA_24BIT						
ECG_LL-RA_24BIT						
ECG_RESP_24BIT						
ECG_Vx-RL_24BIT						
SampleEMG			2015/06/26 07:35:33	00:02:00	420.93 KB	
SD Recording						
Session 1		✓	2015/06/26 07:36:00	00:02:00	420.93 KB	N/A
Shimmer9507 - 512.0Hz - 100%		✓	2015/06/26 07:36:00	00:02:00	420.93 KB	
ECG_EMG_Status1						
EMG_CH1_16BIT						
EMG_CH2_16BIT						
SampleEvents			2016/04/27 03:15:52	00:01:35	37.60 KB	
SampleGSRPPG			2015/06/23 09:17:28	00:02:00	183.84 KB	
SampleResp			2016/07/12 05:41:21	00:02:23	N/A	
SampleSync_SD			2015/10/02 10:58:12	00:02:25	3.17 MB	

Channels with a * after their name have been calibrated using default calibration parameters

DATA DESCRIPTIONS (INSERT DESCRIPTIONS FOR TRIALS AND SESSIONS)

SampleECG - 2015/06/25 10:38:46

This is a sample database generated by the Shimmer team to demonstrate the Shimmer3 ECG unit.

ECG (Electrocardiogram) data (24bit resolution) was recorded at a sampling rate of 1024Hz.

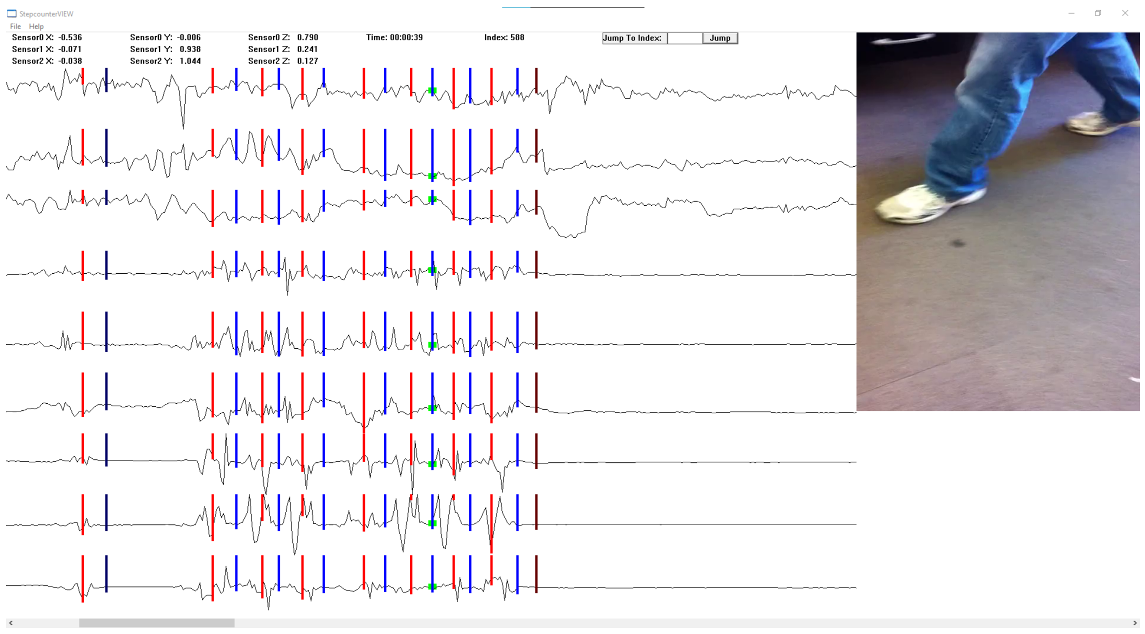
SD Recording - Session 1

The Shimmer device was placed on a desk and connected to the HE Instruments TechPatientCardio signal generator which simulated 4-lead ECG (LA-RA, LL-RA, LL-LA and Vx-RL).

File Format: .csv
File Delimiter: tab (\t)
Timestamp Format: Unix
Data Format: Calibrated

Figure 2.6: A screenshot of Shimmer's ConsensysPro software. This program allows for GUI-based management of Shimmer devices, as well as exportation of recorded on-device data.

An example of the usage of this GUI ground truth tool is found in Figure 2.7. As shown, there are nine separate sensor readings displayed on the screen, characterizing the X, Y, and Z acceleration of the wrist, hip, and ankle mounted IMUs from top to bottom. Stepcounter VIEW allows the operator to play, pause, rewind, scrub quickly, or scrub frame by frame through every sensor reading (every two video frames). By slowly scrubbing through the video and indexing through the sensor readings, the operator can effectively use the video as evidence to mark the ground truth indices of every step. Once the ball of a participant's foot strikes the ground, a step or shift was marked through this program. While the majority of marked indices are considered steps, we also allowed distinction between a ”step” and a ”shift,” where we define a ”shift” as an action where the ball of a participant's foot strikes the ground, but did not follow the typical motion of a normal step. In semi-regular gaits, simulated by our scavenger hunt experiment, these shifts occurred the most frequently, followed by the irregular gait, simulated by the Lego build experiment.



Sensor0 X: -0.536 Sensor0 Y: -0.006 Sensor0 Z: 0.790 Time: 00:00:39 Index: 588
Sensor1 X: -0.071 Sensor1 Y: 0.938 Sensor1 Z: 0.241
Sensor2 X: -0.038 Sensor2 Y: 1.044 Sensor2 Z: 0.127

Figure 2.7: A screenshot of our group's Stepcounter VIEW GUI. This tool enables us to create ground truth files marked with the indices of steps. Blue marks indicate a left step, red marks indicate a right step, and other colors indicate left and right shifts.

Once steps are marked in Stepcounter VIEW, the program will automatically generate a text file containing the indices of the ground truth, as well as the type of step taken (left, right, left shift, or right shift). This ground truth text file, called steps.txt, is necessary to labeling our training data later on, and necessary in our testing metric calculations. While Stepcounter VIEW allows us to individually input the foot on which a step was taken on, as well as distinguish between a full step and what we consider a shift, these factors were not considered for this thesis, as only the indices of each step were required.

While video proofing of the ground truth may seem to be an certain method to accurately determine the factual time in which a step occurred, human error is still a factor. While the majority of the ground truth data was compiled by a single operator in [3], in order to examine inter-rater reliability, or the variability in ratings between operators determining the ground truth, three additional reviewers were enlisted to help provide more context. Each reviewer labeled three chosen participants' data across all three gaits, providing an average of 6,187 additional labeled steps each, with an average variance of 0.2% in total labeled steps. Despite the contention in the total number of steps, we do not expect this result to be perfect, as differences in reviewer attentiveness can vary as a result of such a tedious, monotonous task. Additionally, there is also potentially room for dispute on what constitutes a shift. For example, one reviewer may mark a minor foot re-positioning as a shift, while another may not consider it any form of a step or shift at all. The average difference in shifts labeled was 0.1% for regular gaits, 3.5% for semi-regular gaits, and 3.3% for irregular gaits. While these are important distinctions, when taken in context of this work, where only the labeled indices matter, the variance in the definitions of shifts do not play a large role. As such, we can safely assume that any given reviewer's ground truth file will not significantly differ from others and conclude that the ground truth files used in this thesis are valid and accurate.

2.1.3 Summary of Collected Data

In total, sensor data for 30 participants, each performing three experiments simulating regular, semi-regular, and irregular gaits, and each wearing three Shimmer3 IMUs at the wrist, hip, and ankle, was collected in previous works [3, 17]. Data files are organized by subject, gait, then sensor, where sensor data files consist of CSV file types containing raw, unprocessed data directly from the sensors used in the experiments. The data collection is organized by 30 subject folders, each containing 3 gait folders, each containing 3 sensor files and a single ground truth file. The collection is comprised of 90 total ground truth text files, and 270 total raw data CSV files. Each of these raw data CSV files contains 17 different fields, but for this research, we will only use 4: the synchronized time between sensors, and the 3 accelerometer measurements made in low-noise mode, as described in Section 2.1.1. In total, 60,820 steps were marked in the ground truth files, with $31,538$ steps from regular gaits, $22,221$ steps from semi-regular gaits, and $7,061$ steps from irregular gaits. While the Shimmer3 sensors can collect data in a variety of units, the sensors in this experiment were configured to collect data in units of gravities, roughly equating to the gravitational pull of Earth near the surface at $9.81m/s^2$. It is important to note that the this data set is publicly available at `http://cecas.clemson.edu/∼ahoover/pedometer/`.

2.2 Pre-Processing

There are three major steps to pre-processing our data: cutting the sliding windows, normalizing the cut windows, and finally collating data for cross validation.

2.2.1 Sliding Window

Once all data has been gathered in the format described in Section 2.1.3, we begin pre-processing of the sensor CSV files by cutting our sensor data into a format representative of a sliding window, which consists of multiple overlapping windows of data containing a selected number of sensor readings. Using a set slide s and a window size w, we can iterate our sliding window to capture the entire data set by moving the window s sensor readings every time we cut a new window of data. An illustration of this process is shown in Figure 2.8. The goal of slicing accelerometer data into consecutive windows is to provide our CNN with subsets of data in which it can make a prediction on the number of steps within the designated subset. Our models are designed to predict step count within a given window of data, and as such, will make continual predictions on the number of steps in the specified window of data as the window slides across our entire data set. The program for cutting the sliding windows was written in the C programming language. Further details can be found in Appendix A.

2.2.1.1 Smoothing

Accelerometers are unfortunately prone to noise from a variety of uncontrollable variables such as mechanical noise, like vibrations and other environmental factors, as well as electrical noise, often from the circuitry of the accelerometer itself. As such, raw accelerometer sensor measurements are often not an ideal data set to use, and smoothing should be implemented across accelerometer data to limit the effects of noise and signal variation in the raw measurements. Our implementation for smoothing our accelerometer measurements relies on using a two-sided moving average $x_{average,t}$ of j sensor readings on either side, centered around the current sensor reading x_t, with $i_{max}+1$ total

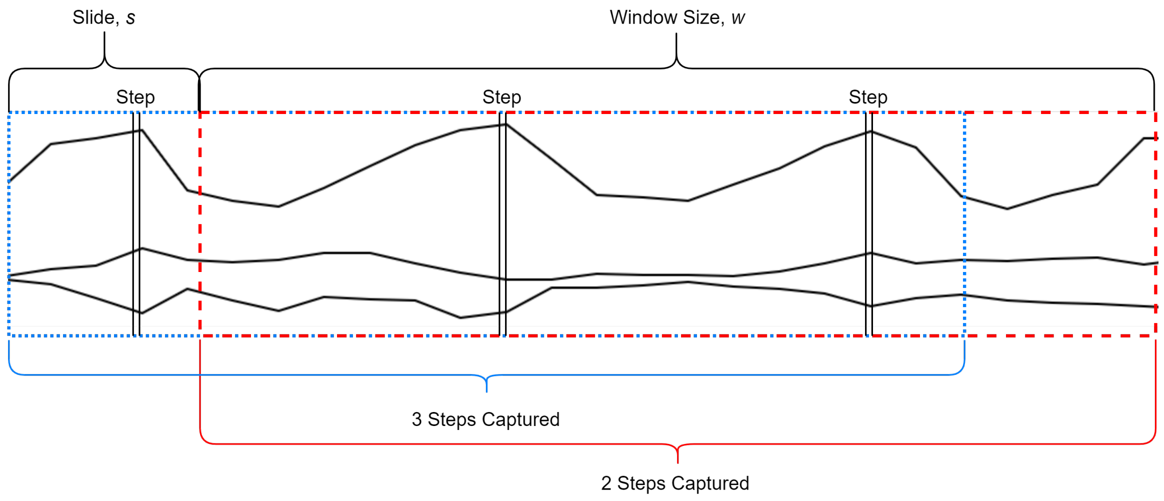


Figure 2.8: An overview of the sliding window procedure. A window with size w slides with a slide of s across accelerometer data. The number of steps contained in that window is captured and recorded, as well as the actual accelerometer data in that window.

sensor readings. While live pedometer sensor readings may require a one-sided unweighted mean of the previous j data, our moving average is centered with $j = 7$ data on either side in order to best smooth our given data set. As the first and last seven sensor readings $[x_0, x_6], [x_{i_{max}-6}, x_{i_{max}}]$ do not have enough data on both sides of the sensor reading to perform a two-sided moving average calculation, these values remain without any smoothing or padding. Mathematically, this process is described in Equation 2.1.

$$x_{average,t} = \begin{cases} \frac{1}{2j+1} \sum_{i=-j}^{j} x_{t+i} & \text{if } 7 \leq t \leq i_{max} - 7, \\ x_t & \text{otherwise} \end{cases} \tag{2.1}$$

When Equation 2.1 is applied $\forall (t \in x)$, we obtain a smoothed moving average of nearly all accelerometer data, save for the first and last 7 sensor readings that we cannot calculate a moving average with our current two-sided window of $j = 7$. The smoothed two-sided moving average of the raw accelerometer data mitigates the effect of noise and outlier sensor measurements, as can be seen in figure 2.9.

2.2.1.2 Index Matching

To understand the algorithm behind our sliding window, the process in cutting a single instance of any particular sensor file occurs is as described: Suppose we choose to cut a CSV file

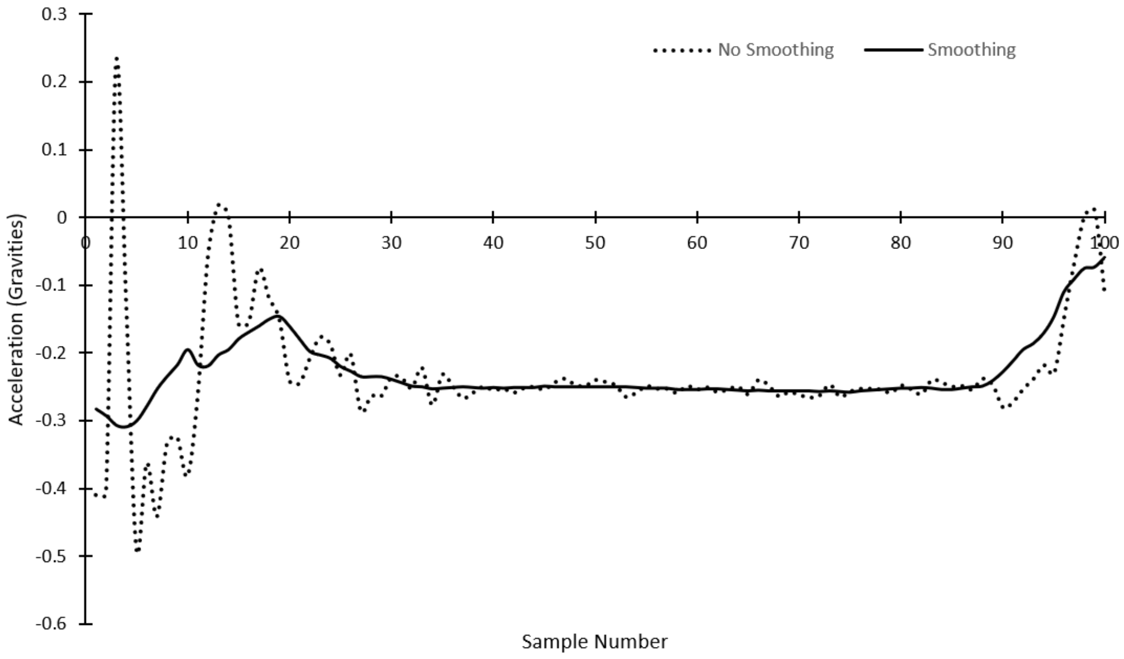


Figure 2.9: This example shows the effect of smoothing on accelerometer data. As shown, the dotted line represents raw, unsmoothed accelerometer data. This data is noisy and fluctuates significantly. By smoothing with a two-sided moving average of $j = 7$ sensor readings, the effect of noise can be diminished, as seen in the solid line.

from sensor 1, the wrist attached IMU. The CSV file containing raw sensor data, Sensor01.csv, is imported by a C program that will first store necessary data into an array, discarding extraneous fields. In this research, since we are primarily focused on the accelerometer data, we store just four fields from the raw data in the CSV file: The synchronized time between sensors, the X axis acceleration, the Y axis acceleration, and the Z axis acceleration. The array of synchronized times for these sensors can be represented by t_{csv1}, t_{csv2}, and t_{csv3}, denoting the wrist, hip, and ankle sensor synchronized time data, respectively. In addition to the CSV file, the ground truth file containing the reviewer-marked step locations is also imported, and step locations are stored in an array. Once all data has been scanned in, the synchronized sensor time t_{match} corresponding to index 0 in the ground truth file is calculated. This time is determined by the time at which the last sensor recording started. For example, if the wrist, hip, and ankle sensor were activated in that order, then t_{match} would equate to the time at which the ankle sensor records its first sensor reading, or $t_{csv3,0}$, since the ankle sensor was activated last. This can be represented by:

$$t_{match} = \max\{t_{csv1,0}, t_{csv2,0}, t_{csv3,0}\} \tag{2.2}$$

This calculation is necessary to correctly index through the data array to the stored ground truth step locations. An offset index, i_{offset}, is found between the corresponding time t_{match} and the chosen CSV file's first sensor reading's time, $t_{csv1,0}$, by iterating through t_{csv1} until a certain $t_{csv,n}$ comes within 66.6 milliseconds of t_{match}. This value of 66.6 milliseconds is obtained from our sample rate of 15 Hz as described:

$$66.6ms = \frac{1s}{15} \tag{2.3}$$

The process to find the offset is described in Equation 2.4:

$$i_{offset} = \max_{n}\{n \mid (t_{match} - 66.6) < t_{csv,n} < (t_{match} + 66.6)\} \tag{2.4}$$

where we take the last matching $t_{csv,n}$ within 66.6 milliseconds of t_{match}. With i_{offset} calculated, we can now begin cutting data to fit our sliding window.

2.2.1.3 Window Slicing

To cut data that will fit the sliding window, the ground truth file is sorted to find the minimum and maximum values of ground truth steps. This is done in order to remove extraneous data from our sensor readings. The data for the sliced windows is cut from a single window prior to the first marked ground truth step, to a single window after the last ground truth step. The starting index i_{start} is thus defined by Equation 2.5:

$$i_{start} = g_{min} - k + i_{offset} \tag{2.5}$$

where g_{min} is the smallest index of the ground truth steps, w is the determined window size to use in our slicing, and i_{offset} is the offset index calculated in Equation 2.4. Similarly, the ending index i_{end} is defined by Equation 2.6:

$$i_{start} = g_{max} + w + s + i_{offset} \tag{2.6}$$

where g_{max} is the largest index of the ground truth steps and s is the slide at which our window slides across the accelerometer data. The window slide s must be included in this calculation as it requires us to cut our last window exactly one window after the last ground truth step is captured.

Now that i_{start} and i_{end} have been determined, the cutting process can begin. Starting at i_{start}, the array of smoothed accelerometer data is iterated over and cut into windows of w length. Each window of cut data is saved and checked against the ground truth file to establish the number of ground truth steps taken within that window. This process is continually repeated until i_{end}. Each window of acceleration values, along with the number of ground truth steps within that window, is then output to a file. Should $i_{start} < 0$ or $i_{end} > i_{max}$, where i_{max} is the last index of gathered sensor data, then we pad the output by writing value of 0 as our acceleration values. While this does not accurately represent accelerometer data collected in a motionless sensor due to noise and the pull of gravity, the number of these data points is negligible relative to our total data set.

The output from cutting windows is organized as follows:

$$\begin{array}{lllllllllll}
gt_0 & x_{0,0} & y_{0,0} & z_{0,0} & x_{1,0} & y_{1,0} & z_{1,0} & \cdots & x_{w-1,0} & y_{w-1,0} & z_{w-1,0} \\
gt_1 & x_{0,1} & y_{0,1} & z_{0,1} & x_{1,1} & y_{1,1} & z_{1,1} & \cdots & x_{w-1,1} & y_{w-1,1} & z_{w-1,1} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
gt_{n-1} & x_{0,n-1} & y_{0,n-1} & z_{0,n-1} & x_{1,n-1} & y_{1,n-1} & z_{1,n-1} & \cdots & x_{w-1,n-1} & y_{w-1,n-1} & z_{w-1,n-1}
\end{array}$$

where n represents the total number of windows cut, w represents the window or kernel size, gt_b represents the number of ground truth indices for window number b, $x_{a,b}$, $y_{a,b}$, and $z_{a,b}$ represents the smoothed x, y, and z acceleration value at index a within window b, respectively.

The cutting program is administered to every single CSV file in the collected data set. This is a total of 270 files per window size w, as detailed in Section 2.1.3. For more information on the organization of the data set and software implementation of the window cutting program, refer to Appendix A.

2.2.2 Normalization

Normalization is an important concept in machine learning as it allows for differently distributed data within a training data set to be used jointly in training a model without biases associated with different data ranges. As described in Equation 1.13, the process basically redistributes an existing data set to newly defined conditions. While multiple normalization techniques exist, including normalizing based on variance, for this work, we will focus on a simple scaling normalization, which scales an existing data set to a newly defined minimum and maximum value, keeping the values of data relative to each other the same. By examining the distributions of the cut window data, it is evident that the vast majority of data lies between -1 and 1 gravities prior to normalization. This becomes apparent when reviewing a histogram of the entire step distribution, as seen in Figure 2.10.

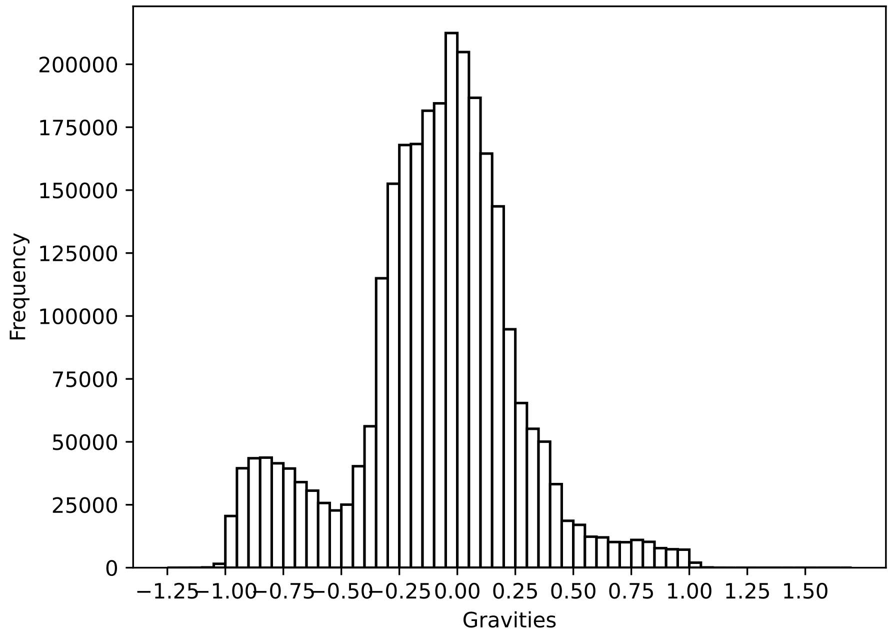


Figure 2.10: Histogram containing all smoothed accelerometer readings. The majority of the data lies between -1 and 1 gravities.

The data follows a roughly Gaussian distribution, save for the second local mode found near a gravity of -1. This is an anticipated distribution, as the effect of gravity on our sensors is likely to increase the number of measurements centered around a gravity of -1 due to the designed orientation of our sensors in the data collection experiment. Despite a visually well-defined range of data, a small number of outliers still remain outside of the -1 to 1 range. As such, two normalization techniques were attempted:

1. Scalar normalization between -1.5 to 1.5 gravities, to capture nearly all data

2. Scalar normalization using nine minimum maximum pairs of data from each sensor, axis combination, to capture all data

As expected, the two techniques differ very little, with the second method offering marginally better training accuracy, which we will consider an anecdote and within range of error. As such, the second method of normalization was chosen moving forward with our experimentation simply due to its more adaptive structure. Using the sliced window data described by the process in Section 2.2.1, we can normalize by scaling the data between 0-1, using independent wrist, hip, and ankle minimum and maximum accelerations of x, y, and z. Nine minimum and nine maximum values were computed for each sensor-axis combination in the data set to use as the new minimums and maximums when normalizing. As such, Equation 2.7 can be derived from Equation 1.13.

$$\begin{aligned} \forall t \in a, \quad a_{N,t} &= (a_t - Min_{sensor,axis})\frac{1-0}{Max_{sensor,axis} - Min_{sensor,axis}} + 0 \\ &= \frac{a_t}{Max_{sensor,axis} - Min_{sensor,axis}} \end{aligned} \tag{2.7}$$

where $a_{N,t}$ is the normalized acceleration at time t, a_t is the unnormalized acceleration at time t, $Min_{sensor,axis}$ is the minimum unnormalized value of a for the specific sensor-axis pair, and $Max_{sensor,axis}$ is the maximum unnormalized value of a for the specific sensor-axis pair. A full table of the min-max values for each sensor-axis pair is found in Table 2.1. Code for the normalization process was implemented using Python 3.8.3.

Sensor Position	**Axis**	**Minimum (gravities)**	**Maximum (gravities)**
Wrist	X	-1.186	1.118
Wrist	Y	-1.438	1.561
Wrist	Z	-1.121	1.079
Hip	X	-1.010	1.033
Hip	Y	-1.166	1.283
Hip	X	-1.077	1.001
Ankle	X	-1.016	0.954
Ankle	Y	-0.540	1.546
Ankle	Z	-1.093	0.948

Table 2.1: The minimum and maximum values in each sensor, axis pair. These min-max pairs are used in Equation 2.7 to normalize our data set.

2.2.3 Cross Validation Collation

With both cut and normalized data available, data is next aggregated for k-fold cross validation. K-fold cross validation is a technique to check the performance of a machine learning algorithm by creating k folds of training and testing data. Folds are defined as a split subset of data withheld for testing purposes - for example, with 10 data files and 5 folds, we can split a subset of the last 2 data files for testing. When determining the amount of data to withhold for validation in k-fold cross validation, the amount of validation data v is simply a ratio of the given amount of data n to the given number of folds k:

$$v = \frac{n}{k} \tag{2.8}$$

By splitting the data into k distinct folds that encompass the entire data set, we are able to verify the authenticity of the results produced by our CNN by testing on each fold and training on the rest, meaning that by completing cross validation, every part of our data will have been trained and tested against the others. An example of this procedure is shown in figure 2.11. Typically, k is set at either 5 or 10 folds, and for this work, we will use $k = 5$. In addition, while typical data sets include a nearly even split of data in each fold, due to the structure of our data set and our SDA algorithm in Equation 1.7, our folds are split by participant, rather than by total data. To elaborate, our data is organized by participant (30), then by gait (3), then by sensor (3), as explained in Section 2.1.3. Ground truth files are defined for each gait type for each participant, as each of the three sensors will share the same ground truth file. However, as these ground truth files contain indices specific to the sensor data files in each gait and participant, we are unable to aggregate all of our data into a single file before splitting into five even folds. Because each ground truth file contains indices relative to the three sensor files contained within the same participant and gait, we cannot evaluate SDA should a fold split a file part way through.

As such, the data was split into five folds by participant. With the 30 total participants, this split denotes that 6 participants will be consigned to each validation fold. A Unix shell script was written to perform a concatenation operation on all data belonging to each fold, as well as concatenating all of the remaining data. This concatenation script will produce 10 files in total: five folds of data for validation, and five remaining data files for training. While our data is not split completely evenly due to the varying amount of data collected per participant, by splitting data

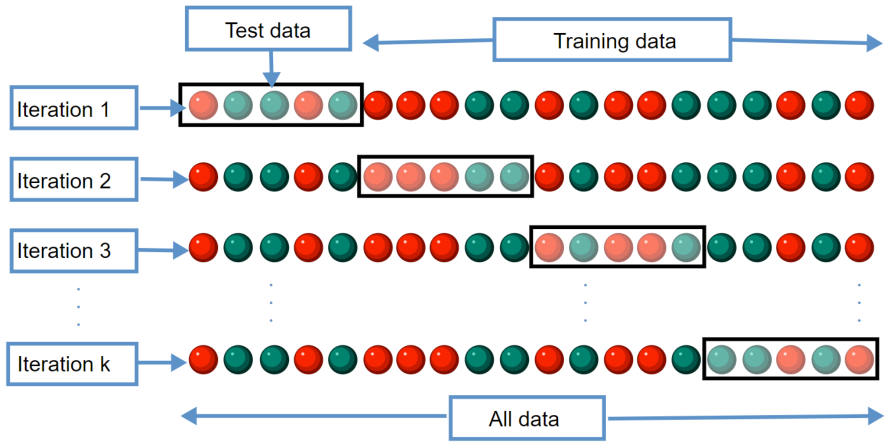


Figure 2.11: An example of how k-fold cross validation works. The entire data set is split into folds of testing and training data for k different iterations [42].

by participant, we still retain the goal of cross validation by successfully creating five training and testing splits across our entire data set.

2.3 Step Count Prediction

This subsection describes the CNN architecture used to extract step predictions within each window of our data set. The CNN architecture used for this work contains three major layers - two one-dimensional convolutional layers, and a fully connected dense layer to map our convolutional outputs to one-dimensional float predictions. Additionally, a number of hyperparameters allow for modulation of the performance of the model training process.

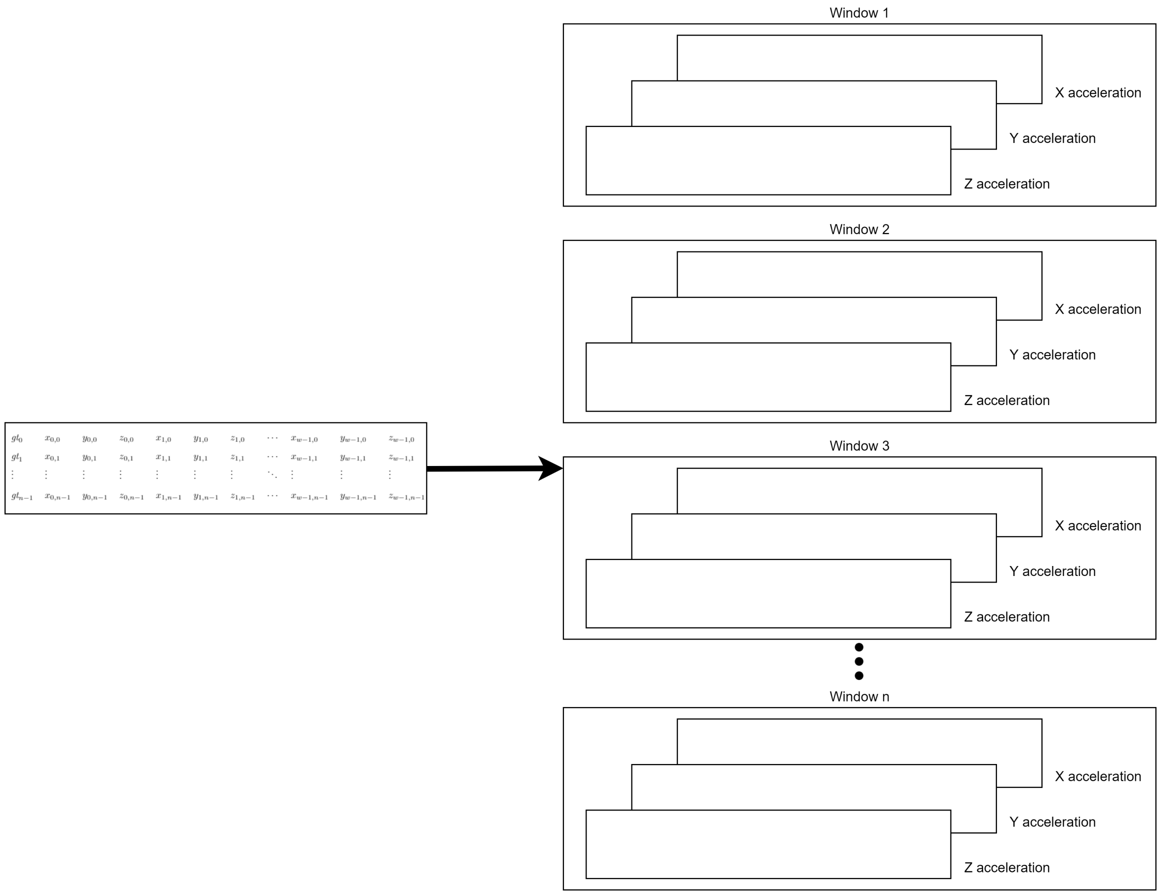
2.3.1 Convolutional Neural Network Layers

2.3.1.1 Reshaping

In Section 2.2.1.3, our normalized accelerometer data was stored in the following format:

$$\begin{array}{ccccccccccc}
gt_0 & x_{0,0} & y_{0,0} & z_{0,0} & x_{1,0} & y_{1,0} & z_{1,0} & \cdots & x_{w-1,0} & y_{w-1,0} & z_{w-1,0} \\
gt_1 & x_{0,1} & y_{0,1} & z_{0,1} & x_{1,1} & y_{1,1} & z_{1,1} & \cdots & x_{w-1,1} & y_{w-1,1} & z_{w-1,1} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
gt_{n-1} & x_{0,n-1} & y_{0,n-1} & z_{0,n-1} & x_{1,n-1} & y_{1,n-1} & z_{1,n-1} & \cdots & x_{w-1,n-1} & y_{w-1,n-1} & z_{w-1,n-1}
\end{array}$$

However, this data must be transformed into two separate data structures for the CNN: the labels and the features. The first column of the above data containing gt_0 through gt_{n-1} is separated into a labels structure, which contain the ground truth labels for the CNN classifier to properly compare the cost function with the correct value. The rest of the data is separated into the features, which constitutes the data in which the CNN uses to optimize the cost function for. The features must be manipulated into a three-dimensional structure in order to properly feed the CNN architecture. The features are reconstructed into multiple channels of data, each representing some specific measure that is part of the training data set. For this thesis, our data set consists of three channels: The X, Y, and Z smoothed and normalized accelerometer data. As each of these channels contain time-series data, the second dimension consists of the channel measurements for each time step within the window, for a total number of time steps equating the size of the window w for the cut and normalized data files. These two dimensions are much like the representation in Figure 1.10. Lastly, the two dimensions of channel and time are then placed in a third dimension containing the entire window, for n number of windows. The described reshaping process is illustrated in figure 2.12.



gt_0 $x_{0,0}$ $y_{0,0}$ $z_{0,0}$ $x_{1,0}$ $y_{1,0}$ $z_{1,0}$ $\cdots$ $x_{w-1,0}$ $y_{w-1,0}$ $z_{w-1,0}$
gt_1 $x_{0,1}$ $y_{0,1}$ $z_{0,1}$ $x_{1,1}$ $y_{1,1}$ $z_{1,1}$ $\cdots$ $x_{w-1,1}$ $y_{w-1,1}$ $z_{w-1,1}$
gt_{n-1} $x_{0,n-1}$ $y_{0,n-1}$ $z_{0,n-1}$ $x_{1,n-1}$ $y_{1,n-1}$ $z_{1,n-1}$ $\cdots$ $x_{w-1,n-1}$ $y_{w-1,n-1}$ $z_{w-1,n-1}$
Window 1
X acceleration
Y acceleration
Z acceleration
Window 2
X acceleration
Y acceleration
Z acceleration
Window 3
X acceleration
Y acceleration
Z acceleration
Window n
X acceleration
Y acceleration
Z acceleration

Figure 2.12: Illustration of how cut and normalized windows of accelerometer data are reshaped prior to training.

2.3.1.2 Convolutional Layers

The first layer of the network takes the 3D shape described above and performs convolution with a depth of 10 filters, using a kernel size of $k = 10$ and a stride of $s = 1$. With no padding involved, our convolution produces a feature map of a smaller dimensionality than the input shape due to the process of convolution itself, as well as data loss on the borders of the data from the lack of padding. For each filter, the convolutional layer produces an output vector of size $size_v$ for every window it convolves across depending on the window size of the input, generalized in Equation 2.9:

$$size_v = \frac{w - k}{s} + 1 \tag{2.9}$$

where w is the cut window length, k is the kernel size of the convolutional layer, and s is the stride. The output shape of the convolutional layer is then the size of the vector by the depth of the layer, or $(size_v, depth)$. If we use a window size w to cut and normalize the raw accelerometer data, after reshaping and feeding in the data to the first layer of the CNN with filters *depth*, kernel size k, and stride s, we receive an output shape shown in Equation 2.10. This output shape from the convolution is also known as a feature map.

$$(\frac{w-k}{s}+1, depth) \tag{2.10}$$

For example, assume we use a window size of $w = 75$ for the first layer of the CNN described above $(depth = 10, k = 10, s = 1)$. Then the feature map's shape can be computed in Equation 2.11:

$$(\frac{w-k}{s}+1, depth) = (\frac{75-10}{1}+1, 10) = (66, 10) \tag{2.11}$$

Additionally, the number of trainable parameters *params* can also be calculated with just one additional parameter, the number of features $features$ in our training data set. This is the same as the channel dimension mentioned in Section 2.3.1.1. Equation 2.12 provides the formula for the number of parameters.

$$params = depth \cdot (k \cdot features) + depth \tag{2.12}$$

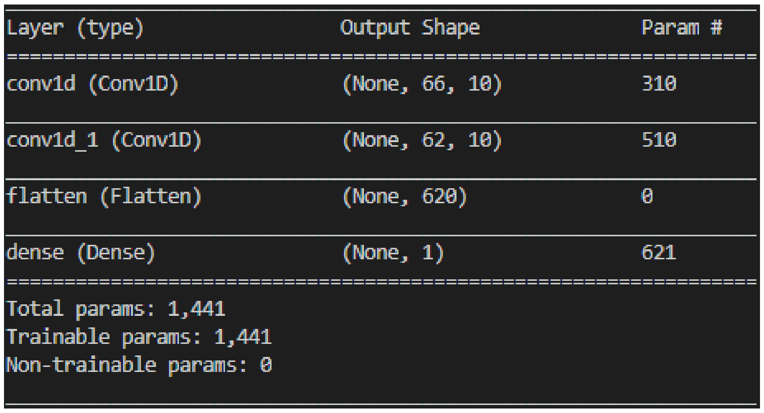
The number of filters *depth* added to the end of Equation 2.12 represents an additional bias term for every filter that must also be trained. As such, the number of trainable parameters *params* can be represented as a sum of the number of weights and biases:

$$params = weights + biases \tag{2.13}$$

Again, this calculation is performed for this specific first layer of the CNN $(depth = 10, features = 10)$:

$$params = depth \cdot (k \cdot features) + depth = 10 \cdot (10 \cdot 3) + 10 = 310 \tag{2.14}$$

Equations 2.10 and 2.12 can be corroborated by simply printing a model summary within Keras, as shown in Figure 2.13.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 66, 10)            310

conv1d_1 (Conv1D)            (None, 62, 10)            510

flatten (Flatten)            (None, 620)               0

dense (Dense)                (None, 1)                 621
=================================================================
Total params: 1,441
Trainable params: 1,441
Non-trainable params: 0
```

Figure 2.13: Screenshot of the Keras model summary using the CNN architecture described in this section.

A second one-dimensional convolutional layer is then applied to the result of the first convolutional layer. A second convolutional layer allows for extraction of higher level features not possible in a single convolution, and is a widely used method for activity recognition based on accelerometer sensor data [43, 44, 45]. For the second convolution, we choose a smaller kernel size of $k = 5$ and keep filters and stride the same ($depth = 10, s = 1$). The output shape and number of parameters can be calculated again according to equations 2.10 and 2.12, respectively. It is important to note that for the second convolution, the new number of features becomes the depth of the feature map generated from the first convolution. In general, for any convolutional layer past the first, the features equates the depth of the feature map of the convolution prior to it.

For both convolutional layers, a rectified linear unit (ReLU) activation function was used. The ReLU activation function is defined as follows:

$$f(x) = x^+ = \max(0, x) \tag{2.15}$$

Introduced in [46], the ReLU activation function replaces the nonlinear logistic sigmoid and hyperbolic tangent activation functions used previously. ReLUs are one of the most commonly implemented activation functions for CNNs now due to their relative simplicity and apparent efficacy in CNNs.

2.3.1.3 Flattening and Dense Layer

The last layer of our CNN architecture features a fully connected dense layer. However, prior to its implementation, a flattening layer is included to increase the trainable parameters given to the dense layer. A flattening layer reshapes our 2D feature map from the second one-dimensional convolution layer to a 1D vector. The size of this 1D vector is determined by the output shape of the previous layer (a, b) by simply multiplying the two dimensions a and b for a vector length $c = a \cdot b$.

By reshaping our multidimensional feature map to a single flattened vector, we can then apply a fully connected dense layer with a trainable weight associated for each element of the flattened vector. The number of parameters found in a dense layer can be calculated as a function of the last dimension of the input layer c and the dimensionality dim of the dense layer itself, as shown in Equation 2.16.

$$params = (c + 1) \cdot dim \tag{2.16}$$

$c+1$ is required for the bias associated with each dimension of the dense layer. Additionally, the output of a dense layer is simply the dimensionality dim given.

In this work, our architecture uses a dense layer dimensionality of $dim = 1$ to receive a single floating-point prediction of the number of steps within a window.

2.3.1.4 CNN Summary and Use

In summary, our CNN architecture contains three main layers: two separate one-dimensional convolution layers, and a dense layer to produce the prediction. However, two additional layers are added in order to properly use the first and last layers in our network: first, a reshape that modifies the input data into a 3D format suitable for the first convolution, and second, a flattening layer that reduces the shape of the feature map output from the second convolutional layer to a single dimension for the fully connected dense layer to operate on. The architecture is outlined in Figure 2.14. It is worth noting that the total number of trainable parameters is simply a summation of the number of trainable parameters in each layer. These will differ depending on the window size of our sliding window pre-processing.

As briefly discussed in Section 1.1, these experiments were conducted using the Clemson University Palmetto Cluster, a high-performance computing server. Code for training our CNN

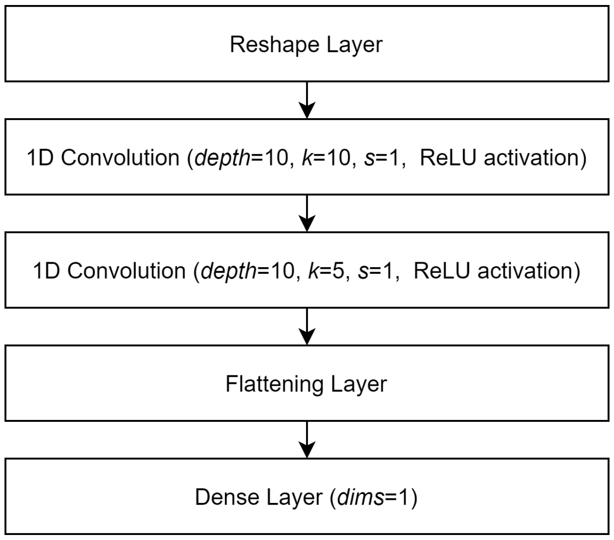

Reshape Layer
1D Convolution (*depth*=10, *k*=10, *s*=1, ReLU activation)
1D Convolution (*depth*=10, *k*=5, *s*=1, ReLU activation)
Flattening Layer
Dense Layer (*dims*=1)

Figure 2.14: Architecture of the CNN

was developed in Python 3.8.3, utilizing Keras 2.4.3 with a back-end of TensorFlow 2.3.1. Using the Portable Batch System (PBS) scheduler, jobs are submitted to nodes on Palmetto using PBS scripts. Each of these scripts requests 16 CPU cores, 125GB of RAM, and the Nvidia Tesla K20 GPU, with a wall time limit of 72 hours. On average, training the nine gait-sensor pair models takes $\mu = 8.95$ hours, with a standard deviation of $\sigma = 0.348$ hours. This was tested using a window size of two seconds, but larger window sizes require longer training times - for reference, a five second window resulted in $\mu = 9.57$ hours and $\sigma = 0.623$ hours. The PBS script used to train the nine models is parallelized across every fold and window size we test. As such, while training the nine models may take an average time of under 10 hours, training all models required for testing may take longer due to queuing and hardware limitations of Palmetto. This is the primary reason why an older GPU model was selected for training.

2.3.2 Network Hyperparameters

Network hyperparameters are tunable options not captured by the layers inside the architecture of the neural network itself. Specifications such as the number of epochs for training and the loss function are examples of hyperparameters. The hyperparameters relevant to this CNN architecture are defined below:

1. **Loss Function:** The loss function defines how well the model is making predictions compared

to the labeled data. We use a standard mean-squared error loss function defined in Equation 1.11.

2. **Number of Training Epochs:** An epoch is an entire pass through the data set when training. For this CNN architecture, we use a training epoch limit of 200. This value was chosen as the decrease in loss began to see little improvement around this limit. As such, the maximum number of epochs our CNN will train for is 200.

3. **Learning Rate:** Another common hyperparameter in deep neural networks is the learning rate. The learning rate controls how drastic weight changes are. A higher learning rate results in greater changes that may allow the network to converge faster upon a small loss but could also result in sub-optimal weights and training instability, including divergence. A small learning rate allows more fine-grain tuning of trainable parameters, but takes longer for the network to converge. This hyperparameter was left at its default value of 0.001.

4. **Optimizer:** The optimizer describes the gradient descent algorithm used to calculate the weights after each batch. We use the 'adam' optimizer for this CNN, an extension of stochastic gradient descent designed specifically for deep neural networks. Moment constants for the exponential decay rate in the adam algorithm are left default.

5. **Batch Size:** Batch size (also called mini-batch size when less than the total number of training samples) affects training by determining the number of samples propagated through the network before weights are re-optimized. This value is left at the TensorFlow default of 32.

6. **Early Stopping:** Early stopping is a technique introduced to prevent overfitting. The early stopping algorithm used in this CNN monitors the loss function and stops training once the loss stops decreasing. The patience of early stopping is set at 50 epochs, meaning that once loss stops decreasing, the model will continue to train for another 50 epochs before stopping. It should be noted that the early stopping implementation used was primarily to check for errors in loss, and in practice does not typically trigger.

7. **Training Weights:** Training weights allows for models being trained to account for the availability in different labels. For example, if a model were trained with data containing 1,000 samples with a label of 0 and just 10 samples with a label of 1, the model is very likely to

bias towards predicting a 0 label for everything. By weighting the training data, the model is given additional context that there is an imbalance in the available training data. Weighting was found to produce less desirable results, and was discarded.

2.4 Post-Processing

The goal of this step is to output individual step detections calculated from a series of step counts per sliding window. The idea resembles calculating area using a Riemann sum, but there are two key differences between a true Riemann sum and our approach. First, the relationship between the width and height of each individual bar does not represent the total step count, or the area of a Riemann sum. The height of the bar represents the estimated step count c in a window of size w, while the width of the bar is the amount of slide s. Second, the goal of our method is to output individual step detections rather than calculating the area. The area is analogous to the total step count, which can be used to calculate the metric RCA. However, individual step detections are needed to calculate the SDA metric. We therefore monitor the total of the sum as increments are added at each time step s. Whenever the total passes a new integer value, an individual step is detected. In contrast, a Riemann sum composed of multiple areas of rectangles is defined by Equation 2.17.

$$Area = \sum_{i=0}^{n-1} f(x_i)\Delta x \tag{2.17}$$

where $Area$ is the approximate area under the function, $f(x_i)$ is the height of rectangle i, n is the total number of rectangles used to approximate the area, and Δx is the width of each rectangle. Our procedure is shown in Figure 2.15, where windows of width w and height marked in steps counted are shown across time. The red line denotes the count of steps per window over time.

Mathematically, the sum of steps per window defined by the area under the red line is found through Equation 2.18 by substituting $f(x_i)$ for c_i, the number of predicted steps in window i, and substituting Δx for s, the slide in each window.

$$Area = \sum_{i=0}^{n-1} c_i \cdot s \tag{2.18}$$

However, as the area under the red line provides the total number of predicted steps in

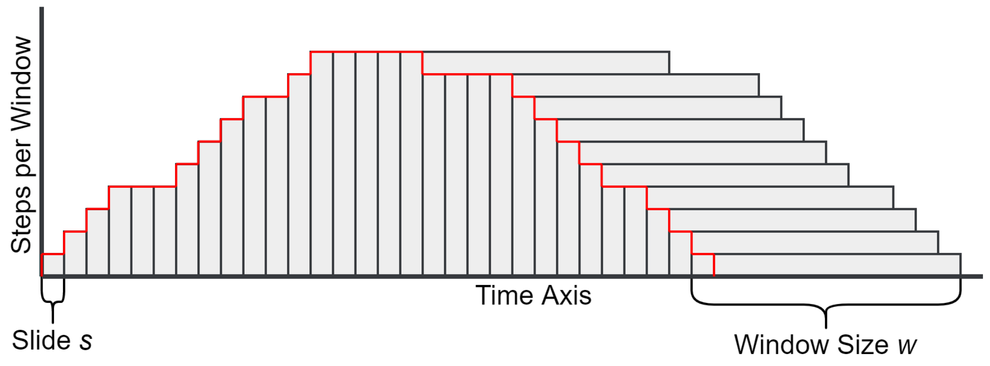


Figure 2.15: Visualization of step count per window. This visualization shows that our CNN predicts the number of steps for every sliding window. The area under the red line forms the sum of predicted steps per window across each slide s.

various windows of data and is not representative of a true Riemann sum, we must account for the overlap in predicted steps and only predict the number of steps in a single slide s by dividing the number of steps in the window by the size w. This yields the total predicted step count across a given range of time, as denoted in Equation 2.19.

$$predicted\ steps = \sum_{i=0}^{n-1} \frac{c_i}{w} \cdot s \tag{2.19}$$

Next, to find the model RCA when testing a file, Equation 2.19 can then be combined with Equation 1.1 to form Equation 2.20:

$$RCA = \frac{\sum_{i=0}^{n-1}(\frac{c_i}{w} \cdot s)}{Actual\ Step\ Count} \tag{2.20}$$

where *Actual Step Count* is computed from the ground truth file, and n is the total number of windows cut.

In addition to RCA, the SDA must also be calculated in testing for a complete picture of the model performance. Unfortunately, SDA cannot simply be determined by a sum of the total predicted steps. Since SDA requires indices of predicted steps to compare to the indices of the ground truth, we need create predicted indices for every detected step, as described above. As such, another algorithm was designed to predict the index at which predicted steps occurred. Based off the running step count algorithm in equation 2.19, by continually tracking the running sum as the model makes predictions for successive windows, we can estimate when a step was taken. Every time the running sum increases over the next integer value of steps, a predicted step is output at the index in the middle of the window where the running sum increase occurred. The method for predicted step indices is detailed in Equation 2.21.

$$i_{predicted} = \min(g) - \lfloor \frac{w}{2} \rfloor + s \cdot n \tag{2.21}$$

where $i_{predicted}$ is the index of the predicted step, $\min(g)$ is the first step in the ground truth, w is the window size, s is the slide, and n is the count of the current window where the running step count increased to the next positive integer. This algorithm is repeated until all windows in the test file have been predicted, and an SDA metric can be calculated based on the equations given in Section 1.2.2.2.

Chapter 3

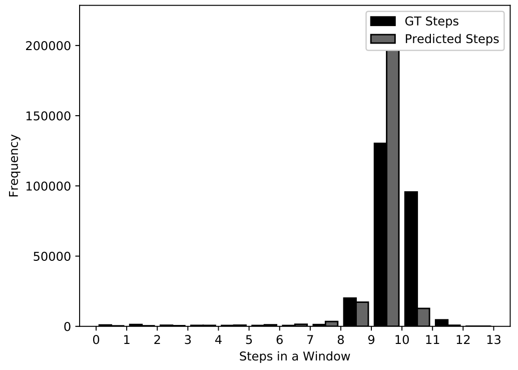
Results

Several different experiments were analyzed in this work. First, the general accuracy of the CNN is evaluated. This is done by comparing step count per window between the ground truth and CNN output, as well as viewing our algorithm's step outputs. Second, the complete pedometer method across multiple window sizes is evaluated. This is done using both the RCA and SDA metrics. Third, our pedometer results are evaluated against the three classic algorithms. This is done across gait type and sensor position using the RCA and SDA metrics. Fourth, we compare our pedometer algorithm against a commercial pedometer. This is done across gait type using the RCA metric.

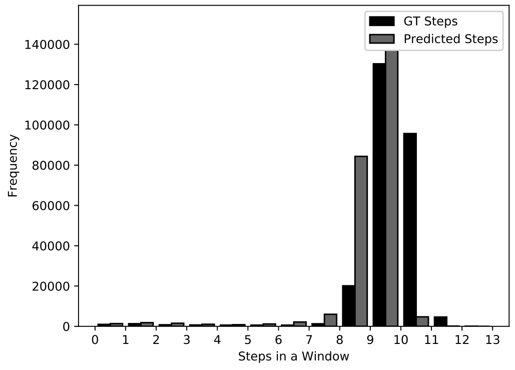
3.1 CNN Accuracy

For our testing, nine different models were generated for each gait-sensor pair. This was chosen in an attempt to increase model accuracy by providing independent models trained on the relevant data sets for the gait and sensor pairing. As mentioned in Section 1.3, many studies in human activity recognition have attempted to improve pedometer accuracy by separating gaits, including [17]. Additionally, sensors can typically be designed for use in specific positions, such as the commonly worn smart watch on the wrist. As such, we distinguish models by training each model with its respective gait and sensor data. For this initial test of CNN accuracy, we arbitrarily chose a window size of 5 seconds, or 75 data.

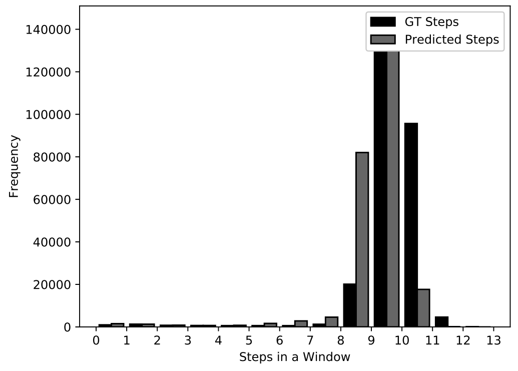
The general accuracy of the CNN can be evaluated by comparing the predicted step distribution to the ground truth step distribution. To do this, we show a histogram of the original step distribution for each gait-sensor combination, juxtaposed with the predicted step distribution of the same, as displayed in Figure 3.1.



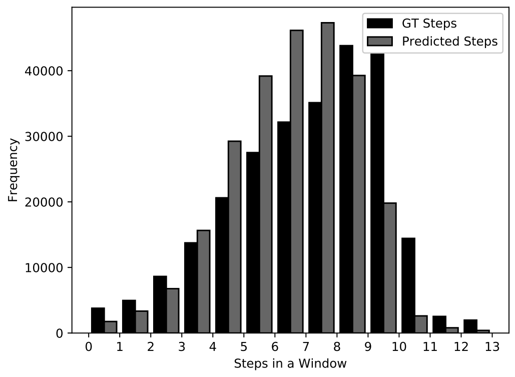
(a) Regular Wrist



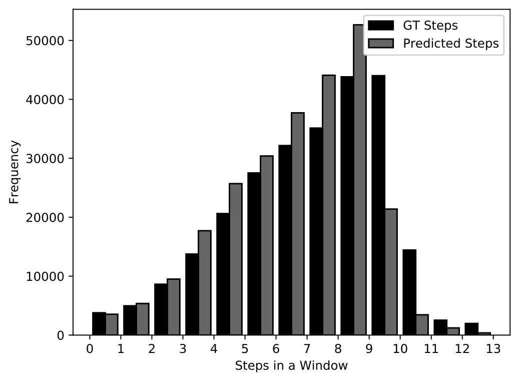
(b) Regular Hip



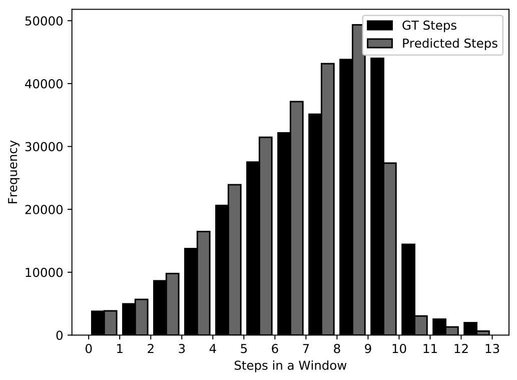
(c) Regular Ankle



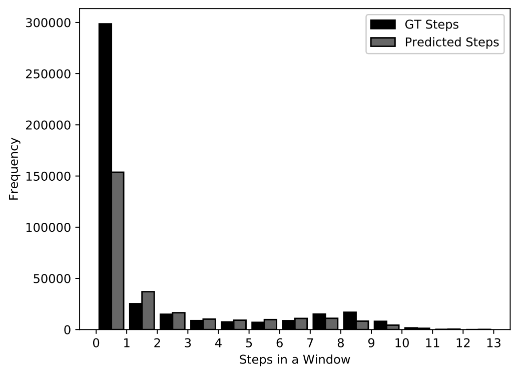
(d) SemiRegular Wrist



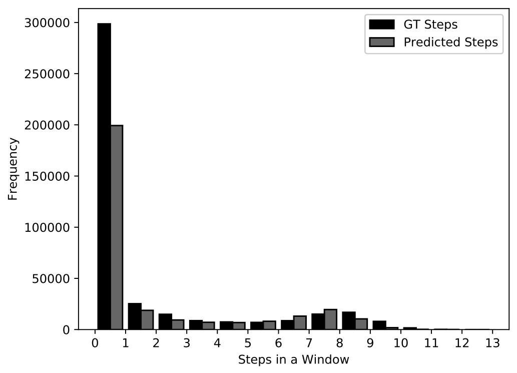
(e) SemiRegular Hip



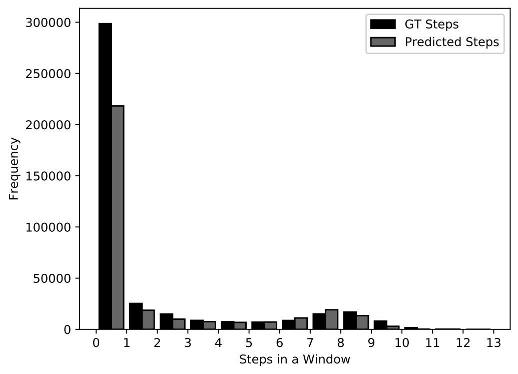
(f) SemiRegular Ankle



(g) Irregular Wrist



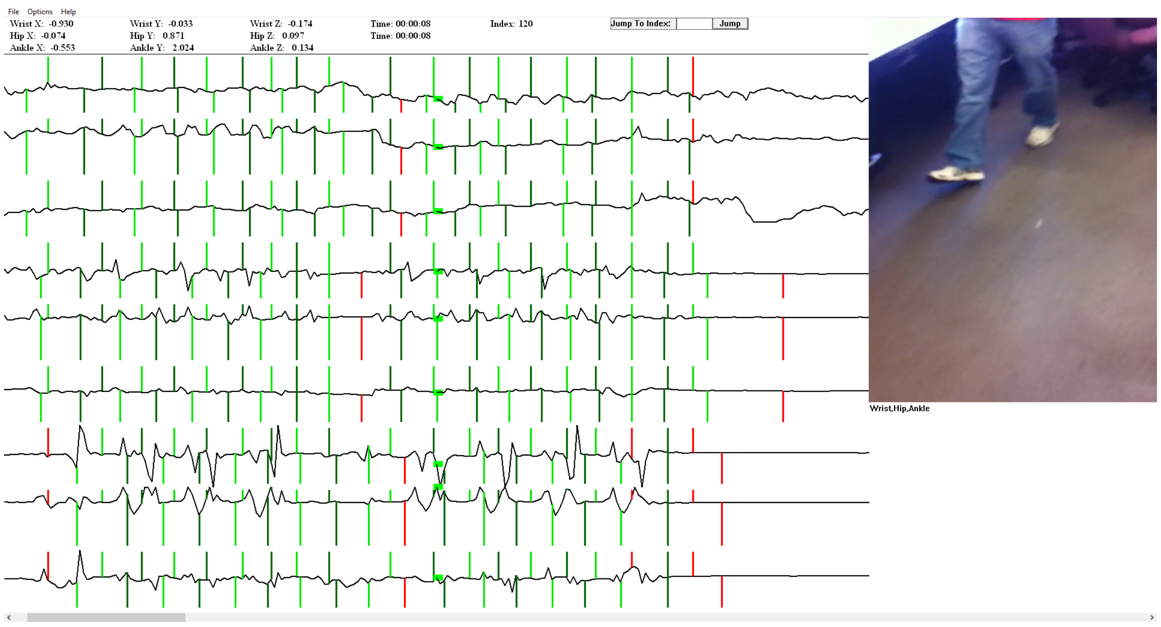
(h) Irregular Hip



(i) Irregular Ankle

Figure 3.1: Comparison between ground truth and predicted step distributions at a window size of five seconds.

While the two histograms are not identical, it is evident that the general shape of the ground truth distribution holds in the predicted step distribution. These results indicate that our models predict the step counts in each window reasonably well, as the predicted steps per window distributions follow the same patterns as the ground truth distributions. Furthermore, we can gain insight into the SDA of our model by creating a tool that allows us to check the locations of our predicted steps. An extension of Stepcounter VIEW, the Stepcounter PAIR program can load the raw sensor data, the ground truth indices, and three predicted step indices (each one corresponding to either the wrist, hip, or ankle sensor within a participant and gait), and display paired groups of predicted and ground truth steps, as seen in Figure 3.2. Vertical lines above the sensor data are ground truth steps, vertical lines below the sensor data are detected steps. Green lines are true positives, red lines indicate either a false positive (detection) or a false negative (missed). Different shades of green indicate alternating true positives. It can be seen that most predicted and ground truth steps match within a period of steps, and our model does well to stop predicting steps when they are no longer taken, despite the variation in accelerometer signals, especially as shown on the top sensor reading.



Wrist X: -0.930 Wrist Y: -0.033 Wrist Z: -0.174 Time: 00:00:08 Index: 120
Hip X: -0.074 Hip Y: 0.871 Hip Z: 0.097 Time: 00:00:08
Ankle X: -0.553 Ankle Y: 2.024 Ankle Z: 0.134

Figure 3.2: Screenshot of the Stepcounter PAIR GUI tool to visualize SDA. Vertical lines above the sensor data are ground truth steps, vertical lines below the sensor data are detected steps. Green lines are true positives, red lines indicate either a false positive (detection) or a false negative (missed). Different shades of green indicate alternating true positives.

The positive indicators of good model accuracy from Figures 3.1 and 3.2 are supported by our averaged five-fold validation test results, found in Table 3.1. It can be seen that both RCA and SDA accuracies remain high across all nine gait-sensor combinations.

Despite performing well, the CNN struggles the most with the RCA of regular gaits. It is

Gait	Sensor	RCA	SDA
Regular	Wrist	0.92	0.94
Regular	Hip	0.92	0.93
Regular	Ankle	0.98	0.98
SemiRegular	Wrist	0.97	0.87
SemiRegular	Hip	0.98	0.90
SemiRegular	Ankle	0.99	0.91
Irregular	Wrist	0.98	0.63
Irregular	Hip	0.98	0.79
Irregular	Ankle	0.98	0.82
Overall:		0.97	0.86

Table 3.1: Five-fold testing averages for each gait-sensor pair, with a window size of five seconds.

unexpected to see the CNN performing significantly worse in regular gaits relative to the predictions it made for semi-regular and irregular gaits. This is contrary to intuition, as regular gaits should include signals easily distinguishable as steps, and could indicate that the regular gait data may not be homogeneous. A complete table of the five-fold validation results for a window size of five seconds can be found in Appendix B, Table B.1.

3.2 Window Size Test

A window size test was built to determine the effect of window size w on our architecture accuracy. The procedure to cut data listed in Section 2.2.1.3 was repeated 10 times for window sizes from 1 to 10 seconds (inclusive) in intervals of 1 second. Next, models were trained for each fold of each window size, resulting in a total of 450 models. This number is comprised of nine models for each gait-sensor pair, multiplied by 5 folds for each window size tested, multiplied by 10 different window sizes, for a total of $9 \cdot 5 \cdot 10 = 450$ models. The appropriate gait-sensor model is then tested on every single participant, sensor, and gait for every fold and every window size. This means that for each of the 270 participant, sensor, and gait files tested, 50 tests were performed - 10 window sizes with 5 folds each, for a total of $270 \cdot 50 = 13,500$ tests. Finally, results of these tests are redirected to CSV files, which are then manually aggregated into a concise table of results. This is detailed by both RCA and SDA in Table 3.2 and Figures 3.3 and 3.4. These results are an average of the five-fold validation strategy we implemented. Full details on every fold can be found in Appendix B, Tables B.3, B.4, and B.5.

By evaluating both RCA and SDA across 5 folds of data and 10 different window sizes, we

Window Size (seconds)	RCA	SDA
1	0.99	0.85
2	0.99	0.88
3	0.98	0.87
4	0.97	0.86
5	0.97	0.86
6	0.99	0.86
7	0.98	0.85
8	0.98	0.84
9	0.97	0.84
10	0.96	0.83

Table 3.2: 5-fold testing averages across all gait types and sensors. Window sizes of 1-10 seconds were tested.

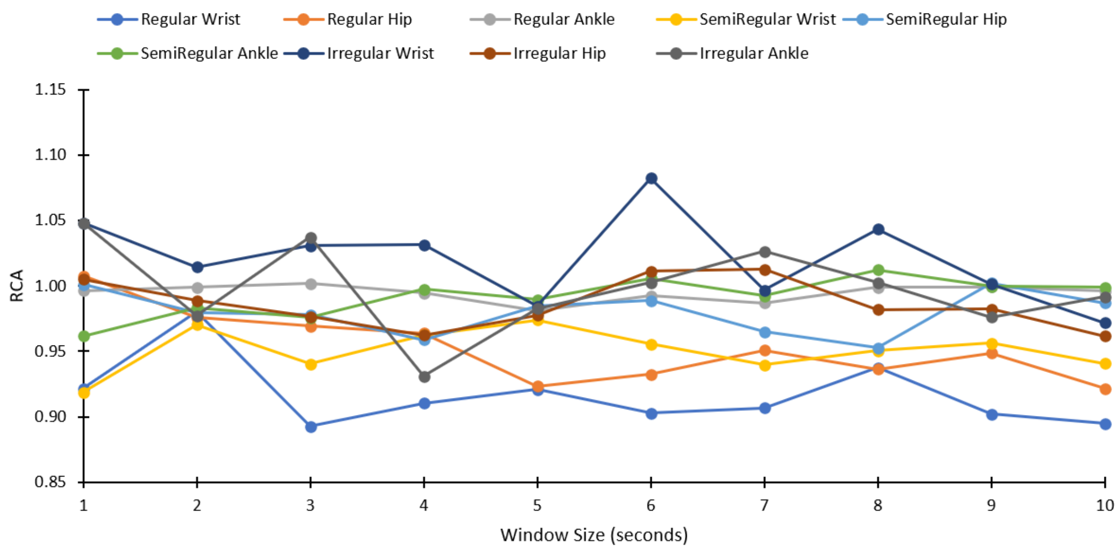


Figure 3.3: The testing RCA results as window size is varied.

can see that the average RCA for all gaits and sensors does not fall below 0.96, and the SDA does not fall below 0.83. While this is promising, this data is more suitably displayed in graph form. We individually display testing results of the average RCA and SDA from the five folds in each window.

As shown in Figure 3.3, it seems that window size does not affect the RCA of our models. As these results average the RCA from testing each of the five folds, variations in RCA between folds have been accounted for. Figure 3.4 displays a similar pattern with the SDA as we vary the window size. This effect is further analyzed in Figures 3.5 and 3.6, where average RCAs and SDAs are taken across all nine gait-sensor pairs.

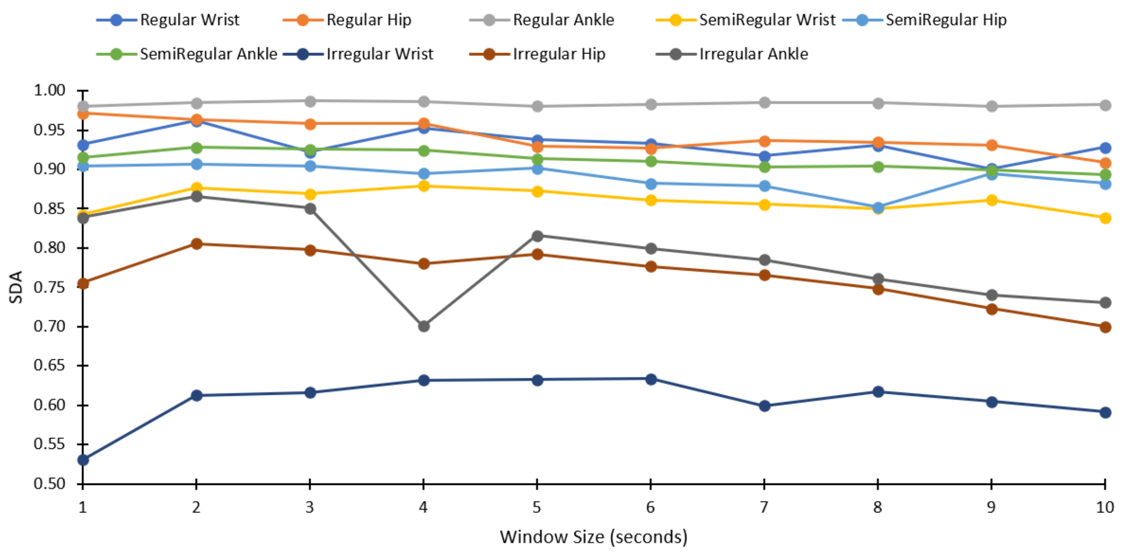


Figure 3.4: The testing SDA results as window size is varied.

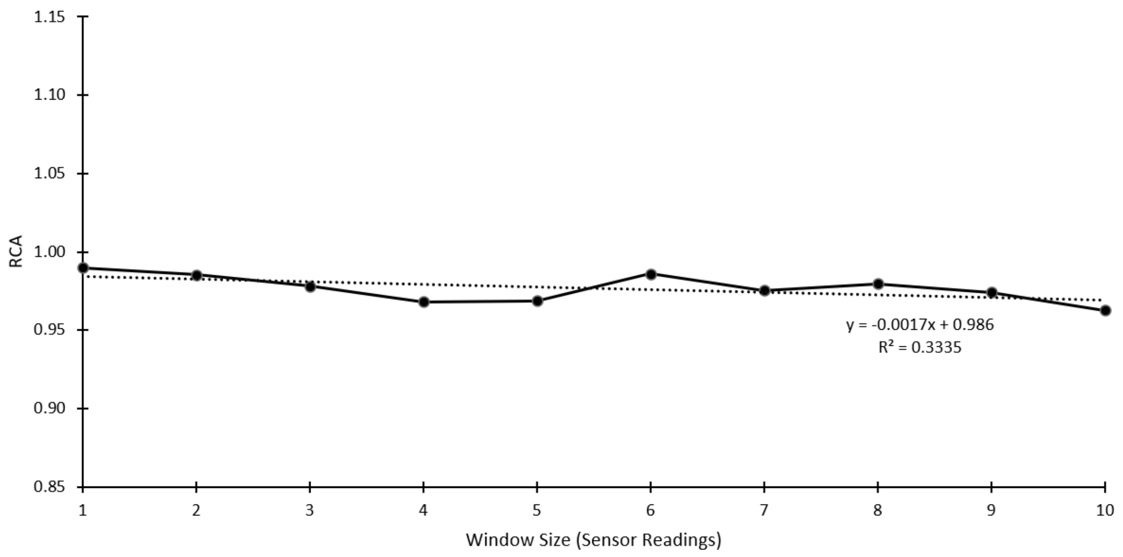


Figure 3.5: The average testing RCA result as window size is varied.

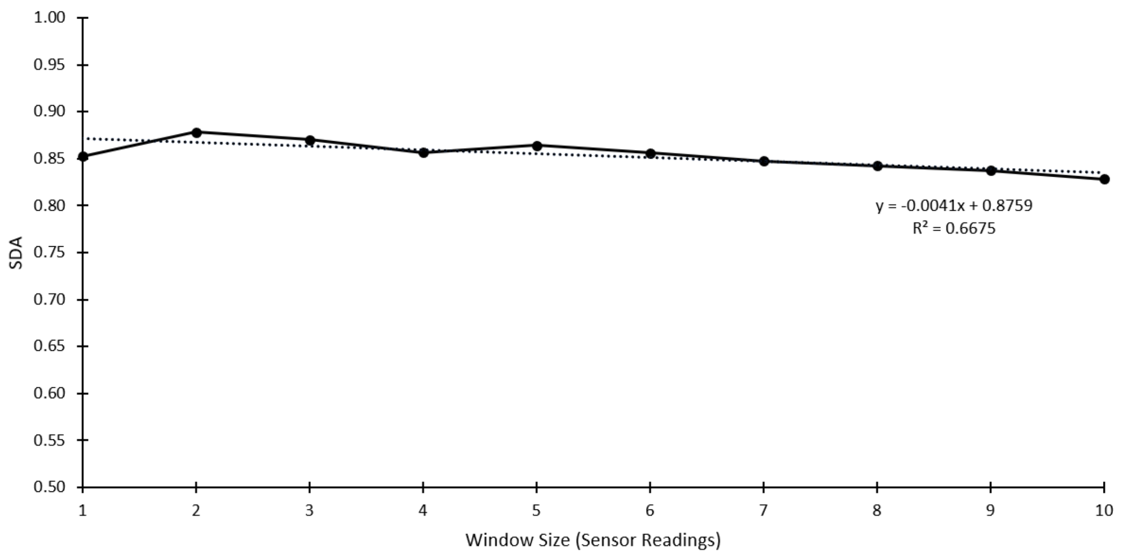


Figure 3.6: The average testing SDA result as window size is varied.

When the two figures above are fit with linear trend lines, they verify the pattern we see in Tables 3.3 and 3.4: with slopes of -0.0002 and -0.0004 for the RCA and SDA, respectively, there is a near negligible downward trend in accuracy as window sizes increase from 1 to 10 seconds. However, it is important to mention the subtle decrease in SDA towards larger window sizes, particularly in the irregular gait. As shown, it seems that all three irregular gaits decrease in SDA the larger the window size becomes. This can be explained by an expected decrease in SDA as the window size becomes larger. Since the step index prediction algorithm used in this work can only estimate the index of a predicted step somewhere in the window, we expect SDA to fall as our window size grows larger. This phenomenon is exaggerated when viewing irregular gaits, as steps are taken in more closely clustered groups than in regular intervals. As such, larger window sizes cannot capture the clustering of steps in smaller time frames, hence making worse predictions of times at which steps are taken within the window. While this is something to consider as window sizes are further increased, in general, the results shown here do not strongly link the two.

3.3 Comparison to Classic Algorithms

While the CNN accuracy determined in 3.1 may seem favorable, the results in Table 3.2 must be evaluated within the context of previous algorithms. In order to assess this, we compare

Algorithm	Overall RCA	Overall SDA
Peak Detector	1.64	0.81
Threshold	2.46	0.61
Autocorrelation	0.94	0.77
CNN	**0.99**	**0.88**

Table 3.3: Overall CNN accuracies compared to classic pedometer algorithm accuracies for RCA and SDA [17, Tab. 3.7]. CNN accuracies reported with a window size of two seconds. Better scores are bolded.

the results in Table 3.2 to results in the three listed classic algorithms [30, 31, 32]. To do this, we choose a single window size of 2 seconds, or 30 sensor samples, in comparison. This window size was chosen because across all gait-sensor combinations, a window size of two seconds produces the joint highest RCA and the highest SDA compared to other window sizes tested. This window size produces an overall 5-fold testing RCA of 0.99 and an SDA of 0.88. Full five-fold validation details of this window size can be found in Appendix B. When comparing these two metrics to the metrics in Table 3.3, the CNN approach used in this work performs better than the given overall score of any single classic algorithm. While the peak detector algorithm performs the best out of the classic algorithms in detecting steps, it generates a high number of false positives, as seen by the RCA greater than 1. Likewise, while the autocorrelation algorithm performs the best in step count out of the classic techniques, it does not detect individual steps as well as the peak detector.

However, classic algorithms differ in performance on unique gaits, shown in [17, Tab. 3.5]. As mentioned in Section 1.3, much research in the field of pedometers has shifted towards gait detection techniques. As such, if gait detection techniques can distinguish between gaits effectively, we must compare our algorithm with the best RCA and SDA performance among all three classic algorithms when dynamically chosen depending on gait type. A summary of [17, Tab. 3.5] for the best classic algorithms relative to our CNN approach is shown in Tables 3.4 and 3.5. The best RCA and SDA from the three evaluated traditional algorithms for each gait-sensor pair are used for comparison. From the results in Table 3.4, it can be seen that the CNN either performs better or comparably to the best classic algorithm in RCA.

In a similar vein, we can see in Table 3.5 that the CNN SDA also performs comparably or better than the best classic algorithm. This verifies that the estimation of step locations within the window that steps are predicted for is relatively accurate. Again, we expect this accuracy to decrease with longer windows, as well as worst-case scenarios where short bursts of steps are followed

Gait, Sensor	Classic	CNN
Regular, Wrist	**1.00**	0.98
Regular, Hip	0.98	0.98
Regular, Ankle	0.99	**1.00**
Semi-Regular, Wrist	0.94	**0.97**
Semi-Regular, Hip	0.78	**0.98**
Semi-Regular, Ankle	1.03	**0.98**
Irregular, Wrist	1.36	**1.01**
Irregular, Hip	1.29	**0.99**
Irregular, Ankle	**0.99**	0.98

Table 3.4: RCA Comparison between the CNN (window size of two seconds) and the best classic pedometer algorithm by gait-sensor pairing [17, Tab. 3.10]. Better scores are bolded.

Gait, Sensor	Classic	CNN
Regular, Wrist	**0.97**	0.96
Regular, Hip	**0.98**	0.96
Regular, Ankle	0.91	**0.98**
Semi-Regular, Wrist	0.81	**0.88**
Semi-Regular, Hip	0.84	**0.91**
Semi-Regular, Ankle	0.81	**0.93**
Irregular, Wrist	0.60	**0.61**
Irregular, Hip	0.81	0.81
Irregular, Ankle	0.86	**0.87**

Table 3.5: SDA Comparison between the CNN (window size of two seconds) and the best classic pedometer algorithm by gait-sensor pairing [17, Tab. 3.11]. Better scores are bolded.

by long periods of rest, but from our tested window sizes and gaits, it appears that the estimations of step times is relatively accurate compared to classic methods. The positive results found in Tables 3.4 and 3.5 demonstrate that the CNN has the potential to perform better than the best classic algorithm for any gait-sensor pair, and that even when the CNN performs worse, it does not do so by a significant margin.

While these results indicate a successful implementation of our methods, the accuracies across subjects and their variations must be examined. For each of the five tested folds and each of the nine gait-sensor pairs per participant, RCA and SDA were averaged. Again, these averages are taken with a two-second window size, consistent with our test results above. The results of these averages can be found in Table B.6 in Appendix B. Figures 3.7 and 3.8 demonstrate the average value of the accuracies for each participant, along with standard deviation error bars. It can be seen that there are low variations in both mean and standard deviation accuracies between participants.

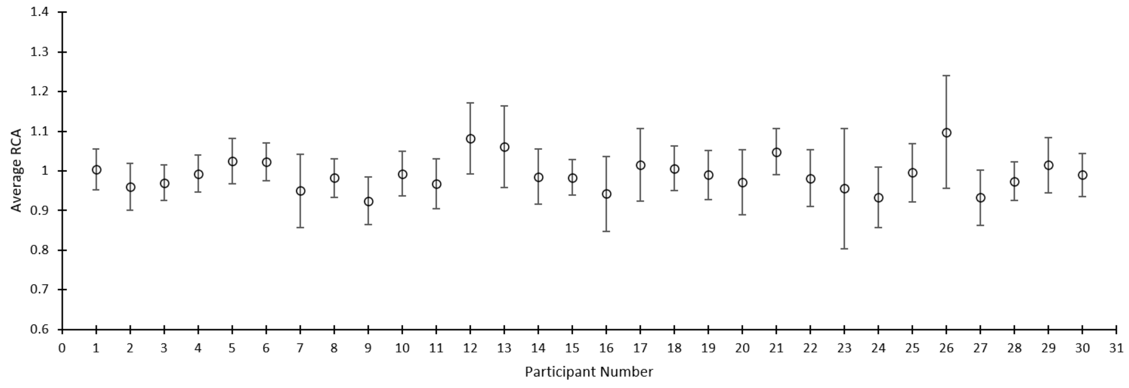


Figure 3.7: Average RCA and standard deviation per participant. Average RCA is marked by a circle, and the standard deviation is represented by the error bars.

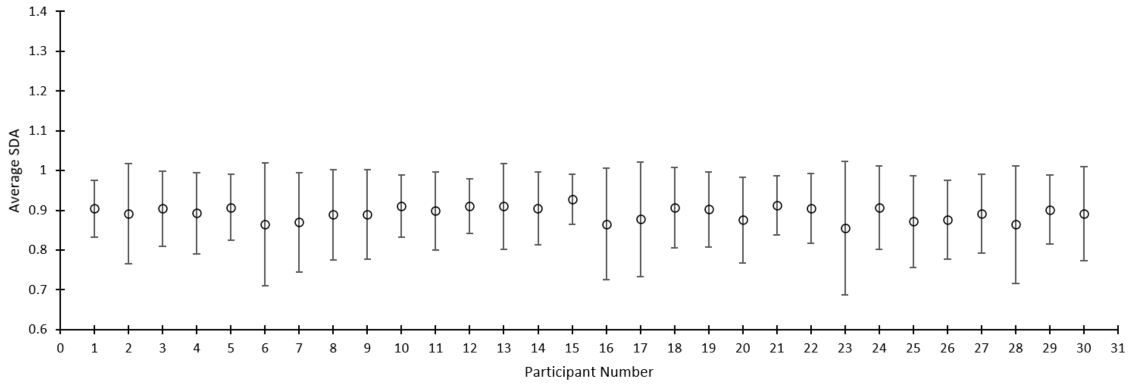


Figure 3.8: Average SDA and standard deviation per participant. Average SDA is marked by a circle, and the standard deviation is represented by the error bars.

3.4 Comparison to Consumer Device

While evaluations against classic algorithms provide a deeper look into the relative performance of our approach, it is also important to compare this work's performance against typical consumer pedometers. This section compares results from a window size of two seconds with a consumer pedometer, the Fitbit Charge 2. Note that due to the proprietary nature of consumer pedometer algorithms, we can only evaluate the RCA and not the SDA.

Results used in Table 3.6 are derived from wrist mounted sensors only, as the Fitbit data gathered was only worn on the wrist, adjacent to the wrist-based Shimmer3 IMU for our own data set. When comparing the RCA to the metrics in this table, the CNN approach used in this work

Gait	Actual Steps	Fitbit Detected Steps (RCA)	CNN Detected Steps (RCA)
Regular	31,528	30,318 (96.2%)	**30,897 (98.0%)**
Semi-regular	22,225	19,679 (89.7%)	**21,565 (97.0%)**
Irregular	7,081	4,552 (64.3%)	**7,185 (101.5%**
Overall	60,834	54,815 (90.1%)	**59,647 (98.0%)**

Table 3.6: Step count and RCA gathered from the Fitbit Charge 2 [17, Tab. 2.1], compared to the CNN. All results were obtained from a wrist-based sensor location. Better scores are bolded.

performs better in every category, with an overall RCA of 98%. This is significantly better than the Fitbit's overall of 90.1%. The margin in accuracy increases between the Fitbit and our approach when counting non-regular gaits. While our algorithm performs exceptionally well for the irregular gait, the Fitbit's RCA falls to just 64.3% in comparison. Although we are unable to verify the SDA of the Fitbit, based on results in Section 3.3 and the Fitbit's lower RCA, we have reason to believe that our algorithm's SDA performance can improve upon the Fitbit's as well.

Chapter 4

Conclusion

4.1 General Discussion

In this thesis, a novel integration-based CNN for step counting and detection was introduced in an endeavor to not just analyze its effectiveness as a pedometer, but to determine its performance relative to classic heuristic pedometer algorithms as well. In general, the CNN architecture in this thesis performs well, and can be considered a viable solution that may be applied towards both counting and detecting steps in a pedometer. The experiments in this work answer our three questions:

Can a neural network learn to detect and count steps by analyzing a sliding window of accelerometer data?

The CNN was able to learn a model that can accurately predict a varying number of steps in a given window. Figure 3.1 shows that even though this distribution differs depending on gait type and sensor position, the CNN was able to predict the step counts for the tested five second windows. Since experiments varying our window size did not significantly change accuracies, we can infer that our CNN can predict step counts occurring in a 1-10 second range of windows.

Does the length of the sliding window affect its prediction accuracy?

As seen in Figures 3.3 and 3.4, despite our initial hypothesis that a longer window would lend more context to the CNN and allow it to more accurately determine a step, we see little effect on RCA. While we do see a subtle effect on SDA, presumably owing to the distribution of irregular gait steps and the decreasing step estimation accuracies with an increasing window size, the evidence is not strong enough to assert that window size has an effect.

> How does the accuracy of this approach compare to the classic peak detection algorithm [30], threshold-based algorithm [31], and autocorrelation algorithm [32]?

Our CNN approach results in a more accurate overall RCA and SDA relative to any single given heuristic algorithm. However, as recent related work concerns gait recognition as an intermediate in step detection, the results of this work were also compared to the best performing classic algorithm for each gait-sensor pair. Out of nine different gait-sensor combinations, the CNN performed better on six, equally on one, and marginally worse on two, for both RCA and SDA, albeit for different combinations. Additionally, compared to a widely used consumer pedometer such as the Fitbit, the CNN model provides a better overall estimation of the RCA of steps taken, seen in Table 3.6. This establishes that our integration-based CNN is a viable approach for counting and detecting steps.

4.2 Limitations

Despite generally good performance from the CNN, caution must be taken with its application. As seen in Figure 3.1a, the model is not perfect and can perform worse than even a consumer grade pedometer such as the Fitbit in specific scenarios. More testing must be performed to individually tune models to perform better. Our data set is limited to what we collected in [3], and thus our algorithms can only be evaluated against this data. Despite varying the gait to simulate real-life step activity, it is impossible in a designed experiment to capture the full range of step movements found in a free-living situation. For example, we do not include data of athletic activity such as running, which is an important use of the pedometer. As such, we do not know how well our CNN architecture will perform with data sets not previously collected in [3].

Another limitation of this study is the unverified consistency of our models. As our 'adam' weight optimization function is a modified stochastic gradient descent, two models trained on the same data set will never have the same weights, practically speaking. As such, in order to verify model accuracy, the same CNN architecture should be used to train multiple models, and the average testing accuracy across all models will give a better general idea of the variance in test results of models trained on the same data set. While our cross validation strategy touches on this, a deeper consistency test is required. Unfortunately, due to both time and computational constraints, this was not tested as a part of this study.

Lastly, while this approach to step counting and detection is novel, it is currently impractical for end-users. This approach requires the use of pre-trained CNN models to effectively evaluate data, meaning that data must either be transferred to a computing server, or processed on-device. While smartphone and wearable technology has made great strides recently in implementing powerful hardware for machine learning purposes, the feasibility of consumer electronics in applying such algorithms is unknown. While current generation hardware could potentially support this technique, we do not know if power usage, thermal limitations, or other mobile technology limitations could make our approach inapplicable to current mobile hardware.

4.3 Future Work

There are a variety of paths this project may take in the future. The most apparent and perhaps straightforward of these is the experimentation with the convolutional neural network architecture itself. This work contains a relatively simple CNN architecture with three main layers. More advanced architectures have been introduced in the field of human activity recognition, so applying relevant architectures is a core starting point in improving our method's accuracy.

Likewise, other algorithms in this work may also be adjusted and optimized. The current step time estimation algorithm relies on a simple estimation of the step in the middle of the window. Perhaps more advanced techniques that consider the variance in steps of adjacent windows can be implemented to produce more exact predictions. Normalization is another area where further experimentation is needed. This work conducts a scalar normalization using minimums and maximums of the appropriate sensor and axis combination. However, these values are calculated across all subjects and gaits. Normalization techniques should be tested by using different min-max pairs, such as by subject, and also by using non-scalar means, such as Gaussian normalization.

Additionally, as described in Section 4.2, as our architecture has not been evaluated for practical use, much more research must be done in order for this method to be end-user friendly. Primarily, our data set should include moderate to intense physical activity and other exercises, as pedometers are frequently used to measure exercise activity. Once this method has been vetted against exercise data, this approach must also be tested with consumer electronics.

Finally, supplementary research should be initiated in developing a single algorithm that can accurately predict steps. As briefly mentioned in Section 1.3, users do not often follow ideal operating conditions. The nine models tested for this work requires knowledge of the specific gait and sensor of the user. Thus, by combining current relevant research in gait segmentation in addition to a sensor position detection algorithm with this work, a comprehensive algorithm should be created that can be sensor and gait agnostic. Furthermore, segmentation may not even be a necessary procedure - future studies should evaluate the viability in training a single CNN model that can possibly provide an all-encompassing scope for pedometer measurements.

Appendices

Appendix A File Organization and Software Details

This appendix details the file structure and organization of the data set used in this thesis, as well as details of the programs developed to run our experiments.

A.1 File Structure

The file structure of both the data and the code organization is critical to understanding the training and testing pipeline.

A.1.1 Pedometer Data Set Structure

```
PedometerData/                  // base
    P001/                       // first participant data files
        Regular/                // Regular gait type
            Sensor01.csv        // wrist-based raw sensor data
            Sensor02.csv        // hip-based raw sensor data
            Sensor03.csv        // ankle-based raw sensor data
            steps.txt           // ground truth indices
        SemiRegular/            // SemiRegular gait type
            Sensor01.csv
            Sensor02.csv
            Sensor03.csv
            steps.txt
        Irregular/              //Irregular gait type
            Sensor01.csv
            Sensor02.csv
            Sensor03.csv
            steps.txt
    P002/                       //second participant data files
        Regular/
            ...
        SemiRegular/
            ...
        Irregular/
            ...
    ...                         // other participant files
    P030/                       // last participant files
        Regular/
            ...
        SemiRegular/
            ...
        Irregular/
            ...
```

Within the directory `PedometerData`, each of the 30 participants of this data set are separated into their own directory, denoted by `P0[Participant #]`. Each participant directory contains three gaits, `Regular`, `SemiRegular`, and `Irregular`. Each of these gait directories contain `Sensor01.csv`, `Sensor02.csv`, and `Sensor03.csv`, containing raw sensor data from the wrist, hip, and ankle based Shimmer3 IMUs, respectively. The fourth file, `steps.txt`, contain indices of the ground truth steps, relative to the last activated Shimmer3 sensor.

A.1.2 Palmetto File Structure

```
pedometer/                      // base
    cut/                        // directory to cut windows
        create_trainandtest.sh  // shell script to collate K-fold
                                //      validation data
        cutnorm.sh              // script to cut and normalize all sensor data
        cutsteps.c              // C program that cuts sliding window data
        launch_createtrainandtest.sh // K-fold data for multiple window sizes
        launch_cutnorm.sh       // cut and normalize multiple window sizes
        makefile                // makefile for cutsteps.c
        normalize.py            // program that normalizes
                                //      accelerometer data
    data/                       // contains normalized and K-fold
                                //      validation data
        cutnorm_15/             // data for window size of 15 sensor readings
            ...                 // data files
        cutnorm_30/             // data for window size of 30 sensor readings
            ...
        ...                     // different window sizes
        cutnorm_150/
            ...
    palmetto/                   // contains scripts to run jobs on Palmetto
        job_output/             // output files for Palmetto jobs
        pbs/                    // PBS scripts for Palmetto
            ...                 // various PBS scripts to launch jobs
        launch_5fold_train.sh   // trains all models for all window sizes
        launch_5fold_test.sh    // tests all models against all data
        launch_hist.sh          // creates histograms
    PedometerData/              // data set described above
        ...
    training/                   // code to train and test models
        compare_histograms.sh   // script that creates histograms with
                                //      predicted and actual step distribution
        generate_histogram_debug.py // program that generates histograms based
                                //      on debug data from test_model.py
        predict_steps.sh        // generates predicted step indices for all 3
                                //      sensors of a given participant/gait
        train_mode.py           // program that trains a model with given data
        test_model.py           // program that tests a trained model
    window_test/                // directory containing window size test files
```

```
results/                 // directory containing Excel files of various
                         //      test results
kfold_train.sh           // trains all models for all window sizes for
                         //      all 5 folds
kfold_test.sh            // tests all models for all window sizes for
                         //      all 5 folds
```

All experiments were run on the Clemson University Palmetto Cluster. This file structure details how code and data was stored on Palmetto. It should be noted that data directories such as `PedometerData/` and `data/` were copied over to the `/scratch1/` directory on Palmetto to allow for high performance I/O operations.

A.2 Code Description

For full code documentation, please view header information in each file. Code can be found at `https://github.com/acmilandroid/pedometer`. This section will briefly describe the function of each code file.

1. **create_trainandtest.sh:** This shell script collates normalized and cut data files into 10 different files - 5 folds of validation data and 5 files of training data, comprised of everything not in the corresponding validation fold.

2. **cutnorm.sh:** This shell script uses `cutsteps.c` and `normalize.py` to cut each sensor file in `PedometerData/` into windows and normalize it.

3. **cutsteps.c:** This C program reads in raw sensor files, extracts relevant sensor information, and uses the ground truth indices to cut windows of data representative of the sliding window algorithm described in Section 2.2.1.

4. **launch_createtrainandtest.sh:** This shell script is an extension of `create_trainandtest.sh` that collates training and validation data for each of the window sizes chosen in the window size test.

5. **launch_cutnorm.sh:** This shell script is an extension of `cutnorm.sh` that cuts and normalizes data for each of the window sizes chosen in the window size test.

6. **normalize.py:** This Python program normalizes data that has already been cut into windows.

7. **launch_5fold_train.sh:** This shell script launches a PBS script on Palmetto to train all 450 models required for our 5-fold cross validated window size test.

8. **launch_5fold_test.sh:** This shell script launches a PBS script on Palmetto to test all 450 models required for our 5-fold cross validated window size test on every file in `PedometerData/`.

9. **launch_hist.sh:** This shell script launches a PBS script on Palmetto that uses `compare_histograms.sh` to generate predicted vs actual step count histograms.

10. **compare_histograms.sh:** This shell script uses `test_model.py` to create a concatenated debug file of all participants. It then uses `generate_histogram_debug.py` to create histograms comparing the predicted and actual step distributions.

11. **generate_histogram_debug.py:** This Python file a histogram comparing predicted and actual step distributions within the given window size. It uses debug data from `test_model.py` to create this.

12. **predict_steps.sh:** This shell script uses `test_model.py` to generate predicted step indices of the three sensor files in a given participant and gait. This file can then be used in Stepcounter PAIR to visualize SDA.

13. **train_model.py:** This Python file will train a model using given cut and normalized input data. Models are trained using Keras 2.4.3 with a back-end of TensorFlow 2.3.1.

14. **test_model.py:** This Python file will test a cut normalized input file with a pre-trained model. This will provide RCA and SDA statistics.

15. **kfold_train.sh:** This shell script trains nine models (one for each gait-sensor pair) for a given fold and a given window size. This is designed to only train nine models at a time to allow for parallelization through `launch_5fold_train.sh`.

16. **kfold_test.sh:** This shell script tests all files in `PedometerData/` with the corresponding model in a given window size, across all 5 folds. This can be parallelized across multiple window sizes with `launch_5fold_test.sh`.

17. **Stepcounter PAIR:** This C++ Windows program is a modification of Stepcounter VIEW that allows us to visualize SDA. Predicted step indices generated by `predict_steps.sh` can

be imported and automatically paired with corresponding ground truth files. Markers above the sensor are ground truth steps, while markers below are predicted steps. Paired steps are in green, while false positives and false negatives are in red.

Appendix B Complete Testing Results

This appendix lists the full details of the five-fold validation results from Chapter 3. Training results are omitted.

B.1 Full Five-Fold Test Results

Full training and testing data is shown across all five folds for a window size test of five seconds are shown in Table B.1.

Full training and testing data is shown across all five folds for a window size test of two seconds are shown in Table B.2.

B.2 Full Window Size Test Results

Full window size testing results are shown in Tables B.3, B.4, and B.5. Results are averaged across five folds. Table B.6 shows average accuracies and standard deviation across participants, with a window size of two seconds.

Window size of 5 seconds			Testing Accuracies	
Fold	Gait	Sensor	RCA	SDA
1	Regular	Wrist	0.98	0.97
1	Regular	Hip	1.00	0.96
1	Regular	Ankle	0.98	0.98
1	SemiRegular	Wrist	1.03	0.88
1	SemiRegular	Hip	1.02	0.91
1	SemiRegular	Ankle	1.00	0.92
1	Irregular	Wrist	0.98	0.64
1	Irregular	Hip	1.01	0.81
1	Irregular	Ankle	1.03	0.85
2	Regular	Wrist	0.88	0.85
2	Regular	Hip	0.80	0.85
2	Regular	Ankle	0.99	0.99
2	SemiRegular	Wrist	0.93	0.85
2	SemiRegular	Hip	1.00	0.91
2	SemiRegular	Ankle	1.00	0.91
2	Irregular	Wrist	0.91	0.66
2	Irregular	Hip	0.99	0.82
2	Irregular	Ankle	0.94	0.81
3	Regular	Wrist	0.73	0.92
3	Regular	Hip	1.00	0.97
3	Regular	Ankle	1.00	0.99
3	SemiRegular	Wrist	0.92	0.87
3	SemiRegular	Hip	1.02	0.93
3	SemiRegular	Ankle	1.00	0.93
3	Irregular	Wrist	0.90	0.53
3	Irregular	Hip	1.05	0.81
3	Irregular	Ankle	0.91	0.82
4	Regular	Wrist	1.00	0.98
4	Regular	Hip	0.93	0.94
4	Regular	Ankle	0.99	0.99
4	SemiRegular	Wrist	1.02	0.87
4	SemiRegular	Hip	0.89	0.85
4	SemiRegular	Ankle	0.97	0.90
4	Irregular	Wrist	1.05	0.66
4	Irregular	Hip	0.85	0.77
4	Irregular	Ankle	1.00	0.83
5	Regular	Wrist	1.01	0.97
5	Regular	Hip	0.89	0.92
5	Regular	Ankle	0.95	0.95
5	SemiRegular	Wrist	0.96	0.89
5	SemiRegular	Hip	1.00	0.91
5	SemiRegular	Ankle	0.98	0.91
5	Irregular	Wrist	1.09	0.67
5	Irregular	Hip	0.98	0.74
5	Irregular	Ankle	1.05	0.78

Table B.1: Full five-fold training and testing results for a five second window.

Window size of 2 seconds			Testing Accuracies	
Fold	Gait	Sensor	RCA	SDA
1	Regular	Wrist	1.00	0.97
1	Regular	Hip	1.00	0.97
1	Regular	Ankle	1.01	0.99
1	SemiRegular	Wrist	0.94	0.87
1	SemiRegular	Hip	1.04	0.93
1	SemiRegular	Ankle	1.01	0.94
1	Irregular	Wrist	0.88	0.62
1	Irregular	Hip	0.98	0.81
1	Irregular	Ankle	1.04	0.88
2	Regular	Wrist	0.90	0.93
2	Regular	Hip	0.93	0.95
2	Regular	Ankle	1.01	0.99
2	SemiRegular	Wrist	0.95	0.87
2	SemiRegular	Hip	0.95	0.90
2	SemiRegular	Ankle	0.97	0.93
2	Irregular	Wrist	1.02	0.63
2	Irregular	Hip	0.97	0.83
2	Irregular	Ankle	0.97	0.88
3	Regular	Wrist	1.02	0.98
3	Regular	Hip	1.04	0.98
3	Regular	Ankle	1.02	0.99
3	SemiRegular	Wrist	0.94	0.89
3	SemiRegular	Hip	1.03	0.94
3	SemiRegular	Ankle	0.97	0.94
3	Irregular	Wrist	1.15	0.57
3	Irregular	Hip	1.19	0.82
3	Irregular	Ankle	0.90	0.88
4	Regular	Wrist	1.02	0.97
4	Regular	Hip	0.97	0.97
4	Regular	Ankle	0.96	0.98
4	SemiRegular	Wrist	1.03	0.87
4	SemiRegular	Hip	0.88	0.84
4	SemiRegular	Ankle	0.99	0.92
4	Irregular	Wrist	0.95	0.62
4	Irregular	Hip	0.81	0.77
4	Irregular	Ankle	0.91	0.88
5	Regular	Wrist	0.97	0.96
5	Regular	Hip	0.93	0.95
5	Regular	Ankle	0.99	0.98
5	SemiRegular	Wrist	1.00	0.89
5	SemiRegular	Hip	1.00	0.92
5	SemiRegular	Ankle	0.98	0.91
5	Irregular	Wrist	1.07	0.63
5	Irregular	Hip	1.00	0.80
5	Irregular	Ankle	1.08	0.81

Table B.2: Full five-fold training and testing results for a two second window.

Gait	Regular					
Sensor	Wrist		Hip		Ankle	
Window Size (seconds)	RCA	SDA	RCA	SDA	RCA	SDA
1	0.92	0.93	1.01	0.97	1.00	0.98
2	0.98	0.96	0.98	0.96	1.00	0.98
3	0.89	0.92	0.97	0.96	1.00	0.99
4	0.91	0.95	0.96	0.96	0.99	0.99
5	0.92	0.94	0.92	0.93	0.98	0.98
6	0.90	0.93	0.93	0.93	0.99	0.98
7	0.91	0.92	0.95	0.94	0.99	0.99
8	0.94	0.93	0.94	0.93	1.00	0.98
9	0.90	0.90	0.95	0.93	1.00	0.98
10	0.89	0.93	0.92	0.91	1.00	0.98

Table B.3: Full window size results for the regular gait. Accuracies are averaged over five folds.

Gait	SemiRegular					
Sensor	Wrist		Hip		Ankle	
Window Size (seconds)	RCA	SDA	RCA	SDA	RCA	SDA
1	0.92	0.84	1.00	0.90	0.96	0.92
2	0.97	0.88	0.98	0.91	0.98	0.93
3	0.94	0.87	0.98	0.90	0.98	0.93
4	0.96	0.88	0.96	0.89	1.00	0.92
5	0.97	0.87	0.98	0.90	0.99	0.91
6	0.96	0.86	0.99	0.88	1.01	0.91
7	0.94	0.86	0.97	0.88	0.99	0.90
8	0.95	0.85	0.95	0.85	1.01	0.90
9	0.96	0.86	1.00	0.89	1.00	0.90
10	0.94	0.84	0.99	0.88	1.00	0.89

Table B.4: Full window size results for the semi-regular gait. Accuracies are averaged over five folds.

Gait	Irregular					
Sensor	Wrist		Hip		Ankle	
Window Size (seconds)	RCA	SDA	RCA	SDA	RCA	SDA
1	1.05	0.53	1.01	0.76	1.05	0.84
2	1.01	0.61	0.99	0.81	0.98	0.87
3	1.03	0.62	0.98	0.80	1.04	0.85
4	1.03	0.63	0.96	0.78	0.93	0.70
5	0.98	0.63	0.98	0.79	0.98	0.82
6	1.08	0.63	1.01	0.78	1.00	0.80
7	1.00	0.60	1.01	0.77	1.03	0.78
8	1.04	0.62	0.98	0.75	1.00	0.76
9	1.00	0.60	0.98	0.72	0.98	0.74
10	0.97	0.59	0.96	0.70	0.99	0.73

Table B.5: Full window size results for the irregular gait. Accuracies are averaged over five folds.

Participant	Average RCA	Average SDA	σ RCA	σ SDA
1	1.00	0.90	0.05	0.07
2	0.96	0.89	0.06	0.13
3	0.97	0.90	0.04	0.09
4	0.99	0.89	0.05	0.10
5	1.02	0.91	0.06	0.08
6	1.02	0.87	0.05	0.15
7	0.95	0.87	0.09	0.13
8	0.98	0.89	0.05	0.11
9	0.92	0.89	0.06	0.11
10	0.99	0.91	0.06	0.08
11	0.97	0.90	0.06	0.10
12	1.08	0.91	0.09	0.07
13	1.06	0.91	0.10	0.11
14	0.99	0.91	0.07	0.09
15	0.98	0.93	0.04	0.06
16	0.94	0.87	0.09	0.14
17	1.01	0.88	0.09	0.14
18	1.01	0.91	0.06	0.10
19	0.99	0.90	0.06	0.09
20	0.97	0.88	0.08	0.11
21	1.05	0.91	0.06	0.07
22	0.98	0.90	0.07	0.09
23	0.96	0.86	0.15	0.17
24	0.93	0.91	0.08	0.10
25	1.00	0.87	0.07	0.11
26	1.10	0.88	0.14	0.10
27	0.93	0.89	0.07	0.10
28	0.97	0.86	0.05	0.15
29	1.01	0.90	0.07	0.09
30	0.99	0.89	0.06	0.12

Table B.6: Table of RCA and SDA averages across all 30 participants, found with a window size of two seconds. Averages are calculated across all gait-sensor pairs and all five folds. σ denotes the standard deviation.

Bibliography

[1] J. A. Noah, D. K. Spierer, J. Gu, and S. Bronner, "Comparison of steps and energy expenditure assessment in adults of fitbit tracker and ultra to the actical and indirect calorimetry," *Journal of Medical Engineering & Technology*, vol. 37, no. 7, pp. 456–462, 2013, pMID: 24007317. [Online]. Available: https://doi.org/10.3109/03091902.2013.831135

[2] J. Y. Leong and J. E. Wong, "Accuracy of three android-based pedometer applications in laboratory and free-living settings," *Journal of Sports Sciences*, vol. 35, no. 1, pp. 14–21, 2017, pMID: 26950687. [Online]. Available: https://doi.org/10.1080/02640414.2016.1154592

[3] R. Mattfeld, E. Jesch, and A. Hoover, "A new dataset for evaluating pedometer performance," in *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2017, pp. 865–869.

[4] M. Narici, G. D. Vito, M. Franchi, A. Paoli, T. Moro, G. Marcolin, B. Grassi, G. Baldassarre, L. Zuccarelli, G. Biolo, F. G. di Girolamo, N. Fiotti, F. Dela, P. Greenhaff, and C. Maganaris, "Impact of sedentarism due to the covid-19 home confinement on neuromuscular, cardiovascular and metabolic health: Physiological and pathophysiological implications and recommendations for physical and nutritional countermeasures," *European Journal of Sport Science*, vol. 0, no. 0, pp. 1–22, 2020, pMID: 32394816. [Online]. Available: https://doi.org/10.1080/17461391.2020.1761076

[5] H. J. Lim, H. Xue, and Y. Wang, *Global Trends in Obesity.* Cham: Springer International Publishing, 2020, pp. 1217–1235. [Online]. Available: https://doi.org/10.1007/978-3-030-14504-0_157

[6] S. Compernolle, A. DeSmet, L. Poppe, G. Crombez, I. Bourdeaudhuij, G. Cardon, H. van der Ploeg, and D. Dyck, "Effectiveness of interventions using self-monitoring to reduce sedentary behavior in adults: a systematic review and meta-analysis," *International Journal of Behavioral Nutrition and Physical Activity*, vol. 16, 12 2019.

[7] E. P. Abril, "Tracking myself: Assessing the contribution of mobile technologies for self-trackers of weight, diet, or exercise," *Journal of Health Communication*, vol. 21, no. 6, pp. 638–646, 2016, pMID: 27168426. [Online]. Available: https://doi.org/10.1080/10810730.2016.1153756

[8] "North american wearables market q2 2020," Sep 2020. [Online]. Available: https://www.canalys.com/newsroom/canalys-north-american-wearables-market-Q2-2020

[9] G. M. Turner-McGrievy, A. Boutté, A. Crimarco, S. Wilcox, B. E. Hutto, A. Hoover, and E. R. Muth, "Byte by bite: Use of a mobile bite counter and weekly behavioral challenges to promote weight loss," *Smart Health*, vol. 3-4, pp. 20 – 26, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352648316300071

[10] Y. Shen, J. Salley, E. Muth, and A. Hoover, "Assessing the accuracy of a wrist motion tracking method for counting bites across demographic and food variables," *IEEE journal of biomedical and health informatics*, vol. PP, 09 2016.

[11] B. Lin and A. Hoover, "A comparison of finger and wrist motion tracking to detect bites during food consumption," in *2019 IEEE 16th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, 2019, pp. 1–4.

[12] F. Shaffer and J. P. Ginsberg, "An overview of heart rate variability metrics and norms," *Frontiers in Public Health*, vol. 5, p. 258, 2017. [Online]. Available: https://www.frontiersin.org/article/10.3389/fpubh.2017.00258

[13] S. W. Lichtman, K. Pisarska, E. R. Berman, M. Pestone, H. Dowling, E. Offenbacher, H. Weisel, S. Heshka, D. E. Matthews, and S. B. Heymsfield, "Discrepancy between self-reported and actual caloric intake and exercise in obese subjects," *New England Journal of Medicine*, vol. 327, no. 27, pp. 1893–1898, 1992, pMID: 1454084. [Online]. Available: https://doi.org/10.1056/NEJM199212313272701

[14] P. Wark, L. Hardie, G. Frost, N. Alwan, M. Carter, P. Elliott, H. Ford, N. Hancock, M. Morris, U. Mulla, E. Noorwali, K. Petropoulou, D. Murphy, G. Potter, E. Riboli, D. Greenwood, and J. Cade, "Validity of an online 24-h recall tool (myfood24) for dietary assessment in population studies: Comparison with biomarkers and standard interviews," *BMC Medicine*, vol. 16, 12 2018.

[15] R. MacManus, *Health Trackers: How Technology is Helping Us Monitor and Improve our Health.* Rowman Littlefield, 2017.

[16] P. F. Saint-Maurice, R. P. Troiano, D. R. B. Jr, B. I. Graubard, S. A. Carlson, E. J. Shiroma, J. E. Fulton, and C. E. Matthews, "Association of Daily Step Count and Step Intensity With Mortality Among US Adults," *JAMA*, vol. 323, no. 12, pp. 1151–1160, 03 2020. [Online]. Available: https://doi.org/10.1001/jama.2020.1382

[17] R. S. Mattfeld, "Evaluation of pedometer performance across multiple gait types using video for ground truth," Ph.D. dissertation, Clemson University, 5 2018.

[18] Y. Dong, "Tracking wrist motion to detect and measure the eating intake of free-living humans," Ph.D. dissertation, Clemson University, 5 2012.

[19] J. Yin, B. Chen, and Y. Li, "Highly accurate image reconstruction for multimodal noise suppression using semisupervised learning on big data," *IEEE Transactions on Multimedia*, vol. 20, no. 11, pp. 3045–3056, 2018.

[20] R. Becerra-Vicario, D. Alaminos, E. Llamas, and M. Fernández-Gámez, "Deep recurrent convolutional neural network for bankruptcy prediction: A case of the restaurant industry," *Sustainability*, vol. 12, p. 5180, 06 2020.

[21] A. O. J. Kwok and S. G. M. Koh, "Neural network insights of blockchain technology in manufacturing improvement," in *2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA)*, 2020, pp. 932–936.

[22] M. Afana, J. Ahmed, B. Harb, B. Abu-Nasser, and S. Abu-Naser, "Artificial neural network for forecasting car mileage per gallon in the city," *Journal of Advanced Science*, vol. 124, pp. 51–59, 12 2018.

[23] S. Rezaei, S. Shokouhyar, and M. Zandieh, "A neural network approach for retailer risk assessment in the aftermarket industry," *Benchmarking An International Journal*, vol. 26, 07 2019.

[24] Y. Y. Luktuke, "Segmentation and recognition of eating gestures from wrist motion using deep learning," Master's thesis, Clemson University, 5 2020.

[25] S. Sharma, P. Jasper, E. Muth, and A. Hoover, "The impact of walking and resting on wrist motion for automated detection of meals," *ACM Trans. Comput. Healthcare*, vol. 1, no. 4, Sep. 2020. [Online]. Available: https://doi.org/10.1145/3407623

[26] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *ArXiv*, vol. abs/1603.07285, 2016.

[27] B. H. Shumpert, "A method to automatically learn appearance variability in machine parts during appliance manufacturing," Master's thesis, Clemson University, 5 2019.

[28] S. Kiranyaz, T. Ince, and M. Gabbouj, "Personalized monitoring and advance warning system for cardiac arrhythmias," *Scientific Reports*, vol. 7, 08 2017.

[29] Y. Yu, C. Wang, X. Gu, and J. Li, "A novel deep learning-based method for damage identification of smart building structures," *Structural Health Monitoring*, vol. 18, p. 147592171880413, 10 2018.

[30] F. Gu, K. Khoshelham, J. Shang, F. Yu, and Z. Wei, "Robust and accurate smartphone-based step counting for indoor localization," *IEEE Sensors Journal*, vol. PP, 03 2017.

[31] N. Zhao, "Full-featured pedometer design realized with 3-axis digital accelerometer," *Analog Dialogue*, vol. 44, no. 6, pp. 1–5, 2010.

[32] A. Rai, K. Chintalapudi, V. Padmanabhan, and R. Sen, "Zee: zero-effort crowdsourcing for indoor localization," pp. 293–304, 08 2012.

[33] G.-L. Chen, F. Li, and Y.-Z. Zhang, "Pedometer method based on adaptive peak detection algorithm," *Zhongguo Guanxing Jishu Xuebao/Journal of Chinese Inertial Technology*, vol. 23, pp. 315–321, 06 2015.

[34] G.-L. Chen, Y.-Z. Zhang, and Z. Yang, "Realization of pedometer with auto-correlation analysis based on mobile phone sensor," vol. 22, pp. 794–798, 12 2014.

[35] H.-H. Lee, S. Choi, and M.-j. Lee, "Step detection robust against the dynamics of smartphones," *Sensors (Basel, Switzerland)*, vol. 15, pp. 27 230–27 250, 10 2015.

[36] M.-S. Pan and H.-W. Lin, "A step counting algorithm for smartphone users: Design and implementation," *IEEE Sensors Journal*, vol. 15, pp. 2296–2305, 04 2015.

[37] M. Susi, V. Renaudin, and G. Lachapelle, "Motion mode recognition and step detection algorithms for mobile phone users," *Sensors*, vol. 13, pp. 1539–1562, 02 2013.

[38] S. Münzner, P. Schmidt, A. Reiss, M. Hanselmann, R. Stiefelhagen, and R. Dürichen, "Cnn-based sensor fusion techniques for multimodal human activity recognition," in *Proceedings of the 2017 ACM International Symposium on Wearable Computers*, ser. ISWC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 158–165. [Online]. Available: https://doi.org/10.1145/3123021.3123046

[39] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019.

[40] W. Shao, H. Luo, F. Zhao, C. Wang, A. Crivello, and M. Z. Tunio, “Depedo: Anti periodic negative-step movement pedometer with deep convolutional neural networks,” in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[41] S. Bagui, X. Fang, S. Bagui, J. Wyatt, P. Houghton, J. Nguyen, J. Schneider, and T. Guthrie, “An improved step counting algorithm using classification and double autocorrelation,” *International Journal of Computers and Applications*, pp. 1–10, 02 2020.

[42] Gofusowa, “File:k-fold cross validation en.svg,” 2019, gufosowa, CC BY-SA 4.0 ¡https://creativecommons.org/licenses/by-sa/4.0¿, via Wikimedia Commons. [Online]. Available: https://commons.wikimedia.org/wiki/File:K-fold_cross_validation_EN.svg

[43] M. Panwar, S. Ram Dyuthi, K. Chandra Prakash, D. Biswas, A. Acharyya, K. Maharatna, A. Gautam, and G. R. Naik, “Cnn based approach for activity recognition using a wrist-worn accelerometer,” in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2017, pp. 2438–2441.

[44] Y. Chen and Y. Xue, “A deep learning approach to human activity recognition based on single accelerometer,” in *2015 IEEE International Conference on Systems, Man, and Cybernetics*, 2015, pp. 1488–1492.

[45] J. Huang, S. Lin, N. Wang, G. Dai, Y. Xie, and J. Zhou, “Tse-cnn: A two-stage end-to-end cnn for human activity recognition,” *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 1, pp. 292–299, 2020.

[46] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: JMLR Workshop and Conference Proceedings, 11–13 Apr 2011, pp. 315–323. [Online]. Available: http://proceedings.mlr.press/v15/glorot11a.html