Raw versus linear acceleration in the recognition of wrist motions related to eating during everyday life

---

A Thesis
Presented to
the Graduate School of
Clemson University

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

---

by
Shaurya Gupta
August 2021

Accepted by:
Dr. Adam Hoover, Committee Chair
Dr. Yongkai Wu
Dr. Linke Guo

Abstract

This thesis investigates the difference between raw and linear acceleration in wrist motion for detecting eating episodes. In previous work, our group developed a classifier that analyzed linear motion and achieved good accuracy. However, the classifier can be volatile in the sense that when retrained and tested on the same data, accuracy varies, especially when trained on small amounts of data such as for a single individual. We hypothesize that in part this may be due to the noise in linear acceleration which is significantly larger relative to normal human wrist motions as compared to the noise in raw acceleration. We therefore perform a set of experiments to determine if classifier accuracy and/or stability can be improved by analyzing raw acceleration instead of linear acceleration.

The dataset used for this work is the Clemson All-Day Eating (CAD) dataset. This was collected over a period of one year, in 2014. In the process of data collection, 351 participants were recruited and 354 days of wrist data was recorded. The recorded data contained 1,133 meals spread over 250 hours of eating. The total length of the recorded data was nearly 4,680 hours. In this work, the CAD dataset was reduced to 342 days and 1034 meals because for some recordings, raw acceleration data was not saved.

Previous work developing a classifier based on linear acceleration achieved a time-based weighted accuracy of 80%, a true positive rate of 89% on eating episodes, and a false positive per true positive rate of 1.7. However, these results were based upon a single run of train and test. Recently we discovered that the model accuracy varies somewhat between runs. We therefore perform a replication experiment on the linear classifier to confirm these results by rerunning the entire experiment 10 times. We report the average and standard deviation of all metrics across these runs. This helps establish a better baseline for comparison of our new classifier that analyzes raw acceleration.

We next analyze the same set of data, using the same neural network model and general approach as for the linear acceleration-based classifier, to compare its accuracy and stability. Evaluating all results, we found that the linear acceleration classifier achieved (average $\pm$ standard deviation across 10 runs) a TPR of 86% $\pm$ 1.2% and a FP/TP of 1.7 $\pm$ 0.3. It also achieved a weighted accuracy of 79 % $\pm$ 0.5 %. Thus, we concluded that the results of original experiment were above the average results and could either be due to a freak training and testing run or due to contamination of the testing data. These results set up a new baseline with which we compare the raw acceleration model metrics. We found that the raw acceleration achieved a TPR of 84% $\pm$ 1.3 % and a FP/TP of 1.7 $\pm$ 0.3. In the case of time metrics, the raw acceleration model achieved a weighted accuracy of 78% $\pm$ 0.4%. Thus, on average, we found that the linear acceleration performed slightly better than raw acceleration in episode detection. The time metrics for both raw and linear acceleration were more or less similar but we did see a higher standard deviation for the raw models.

Our results indicate that linear acceleration does provide greater accuracy than raw acceleration. Even though raw acceleration has a higher signal-to-noise ratio than linear acceleration, in terms of normal human wrist motions, our classifier model has relatively equal volatility when analyzing either signal. We conclude that the main source of model volatility is still unknown. Thus, we found that linear acceleration is, overall, a better predictor of eating as compared to raw acceleration. It should be noted that the difference in the accuracies is very minor and the volatility in the training process could account for some of the differences.

Acknowledgments

First of all, I would like to thank Dr. Adam Hoover for the time that he spent in the process of helping me formulate the research methodology, guiding my work and navigating the field of technical research. Furthermore, even during the uncertain period of pandemic, he did his best to keep up with our work and help us through whatever means available. Without him, this work would not have been possible.

I would also like to thank Dr. Yongkai Wu and Dr. Linke Guo for serving on my defense committee and listening to my thesis defense. I would like to extend my gratitude to all the professors that taught and shaped my academic career at Clemson University over the past years without whom I would not have gained the professional skills to pursue higher education.

I would also like acknowledge my fellow students and colleagues that I worked with for their support. I would like to thank Dr. Surya Sharma for helping me in transitioning to and navigating the early part of my research and for providing a solid base that I could build off of.

Finally, I am forever indebted to my family for their continued support. I am grateful to my parents who worked hard so I could have this wonderful opportunity and to my brother who was always there when I needed him, both as a friend and as someone I could look up to.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

This work considers the problem of detecting eating episodes using embedded sensors on a wrist device that track acceleration and tilt of the wrist. Previously, our group created deep learning models to classify wrist motion on a dataset of 351 people. The models were trained on linear acceleration which was extracted from raw acceleration data collected by the sensors. Our work uses a Shimmer3 device that contains sensors and can be worn like a smartwatch. This section reviews the physics of these sensors and explains how they operate.

Linear and raw acceleration differ in the magnitude as well as the interpretation of the values. Since the magnitude of normal human wrist motions are small as compared to the acceleration due to earth's gravity, linear acceleration gives us an indication of linear motion, i.e. how fast the body is moving linearly along the 3 axes while raw acceleration gives us an estimation of the orientation of the body with respect to the earth. One of the main problems with using linear acceleration is that the noise to value ratio is really high. Since our wrists do not experience high magnitudes of acceleration, the various sources of noise can affect the readings significantly as the noise magnitude is comparable to the acceleration magnitude. This is in contrast to raw acceleration, where the magnitudes of the readings are much higher owing to inclusion of gravitational component. Thus, noise sources do not play a significant part in the overall measurement, providing far more stable values that are less affected by noise.

There are arguments to be made for both raw and linear acceleration, therefore we need to determine which one is more feasible to use for future research in this area. The data collected by our research group contains data regarding 351 days of wrist motion. In this work, we use the previously collected data to train two different sets of models. The first uses linear acceleration and gyroscope data, which is a replication of the work previously done by our research group and will work as both peer review and as the baseline. The other model uses raw acceleration and gyroscope data. Both models classify the activity taking place during a given window of data as eating or non-eating. Since we use raw/linear acceleration and gyroscope data to train the models, they must learn to differentiate between the patterns of motion that the wrist makes during eating and non-eating. By using metrics, described later in the thesis, we measure the accuracy of the models. The training process is described in more detail in the methodology section. The accuracy of the models and the metrics are presented in the results.

The rest of this section provides the necessary background required to understand the work. Section 1.2 explains the differences between raw and linear acceleration as well as how to convert one into the other. We explain the issue of noise in section 1.2.5, which is the core reason for this experiment. In section 1.3, we describe some of the previous work done in this field and finally in section 1.4, we discuss the novelty of this thesis.

1.2 Motivation

1.2.1 Obesity

Overweight and obesity affect almost a third of entire worlds population. It is estimated that by the end of 2030, almost 38% of the people will be overweight and 20% will be obese [11]. The United States is one of the most severely affected country in the world with nearly 38.5% of the US adults and 18.5% of US youth suffering from obesity [6]. The CDC estimates that overall, obesity rates in US reached 42.4% in 2017/18 [7]. A study by Wang. et. al tracked the progression of the disease and projected that by the end of 2030, 85% of US adults will be afflicted by obesity [27]. Furthermore, Obesity is can give rise to many problems that affect people later in life. Obesity has been shown to correlate with heart disease, stroke, type 2 diabetes and certain types of cancer [16]. Obesity not only directly impacts our health but also makes us susceptible to other virus based diseases. A study in US [22] found that having BMI of over 30, the definition of obesity, increased the risk of catching COVID-19 by almost 113%, and of dying by 48%.

Historically, there have been many reasons for obesity, the current epidemic is thought to be largely caused by modernization [29]. Most jobs in the 21st century are sedentary [19]. Moreover, as the world climbs the socio-economic ladder, changes in our food habits also contribute to the problem. While there are many factors that play into the spread of obesity, it all really comes down to energy balance [26]. Consistently consuming more calories than we burn will slowly cause weight gain and if the calorie intake is left unchecked, will eventually lead to obesity.

The only method of treating obesity is weight loss, and this can be achieved in different ways. There exist many surgical solutions for weight loss like liposuction, gastrectomy, abdominoplasty etc. These surgical procedures all carry complication risks. A study conducted on 551 consecutive patients [25] over 5 years found that 4.2% of the patients treated with liposuction developed some form of complication. This number was almost 50% among patients treated with abdominoplasty. While half of this number were just minor scar deformities, it does show that risk involved in cosmetic surgeries. Moreover, surgeries like liposuction directly remove fat from areas of concentration but do not treat the underlying issue of energy consumption or the physiological response to food [13]. Thus, often these surgeries do not have the intended effect in long term.

The simpler, non-surgical method of weight loss is to monitor energy consumption. This idea of eating less works in theory but is much harder to implement in practice. When a study [5] asked the participants to estimate the number of calories in a breakfast meal bought from a fast food chain (a very common routine in developed countries), they found that, on average, adults underestimated their purchases by 175 ± 636 calories. This leads us to the crux of the problem, i.e. tracking our health.

There exists a significant correlation between self-monitoring and weight loss [2]. There are many ways to monitor calorie intake, with the most common being the use of mobile phones. This idea of practicing medicine or monitoring public health through the assistance of mobile devices is known as mHealth.

1.2.2 mHealth

mHealth is a sweeping term used to define any telecommunication technology that is integrated with mobile phones and wearable devices to provide health support. While mHealth has its roots in developed countries, its rapid advancements are largely due to its wide use in developing nations, where providing in-person help and diagnosis isn't always possible [1]. Mobile technology has penetrated even the furthest and most remote corners of the world, and as more and more people embrace mobile devices, the reach of mHealth has grown beyond what was thought possible.

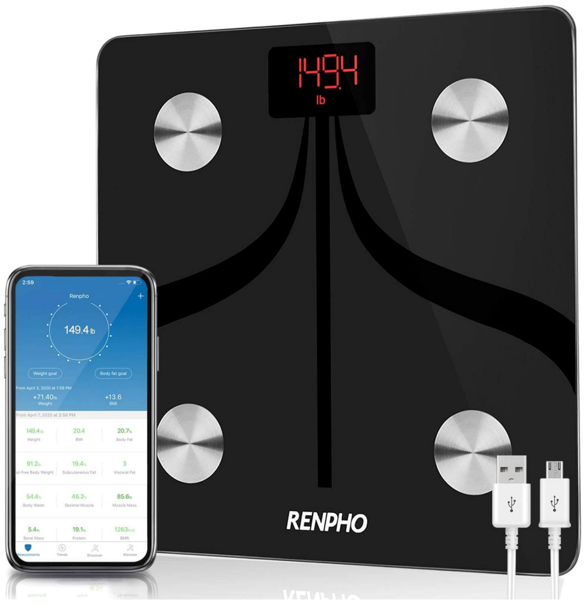


Figure 1.1: Renpho Body Scale : an example of monitoring weight using mHealth.

mHealth is a wide all encompassing term that has various sub-disciplines. In relation to this work, we focus on the self-monitoring aspect of the term. Advancements in data processing and mobile phone technology has revolutionized the field of mobile health monitoring. Use of smart-watches, smart-scales has become common place in our society. Modern scales, as pictured in figure 1.1 can not only measure your weight, but also, through the use of impedance technology, measure body fat percentage, BMI, muscle mass etc. It also stores these readings allowing a user to view the changes and trends in the quantities, dating back to more than a year.

1.3 Sensors

Sensors are devices that measure changes in variables of an environment and are often used in conjunction with a computer processor. Sensors are used everywhere, from detecting vision and imaging to measuring position and proximity of objects.

Our work uses sensors mounted on a smartwatch-like device to track wrist motion. This section reviews the physics of these sensors and explains how they operate.

1.3.1 Accelerometers

Accelerometers are devices used to measure the proper acceleration of an object. On earth, proper acceleration is the acceleration relative to free - fall, or the acceleration experienced by an object in its own rest frame. Thus, an accelerometer that is in free-fall will measure an acceleration of 0 g while one at rest will record an acceleration of 1 g in the direction directly opposite to the center of the earth. Thus, two primary components of an accelerometer reading are the gravitational component and the linear component.

$$a_{reading} = a_{linear} + g \tag{1.1}$$

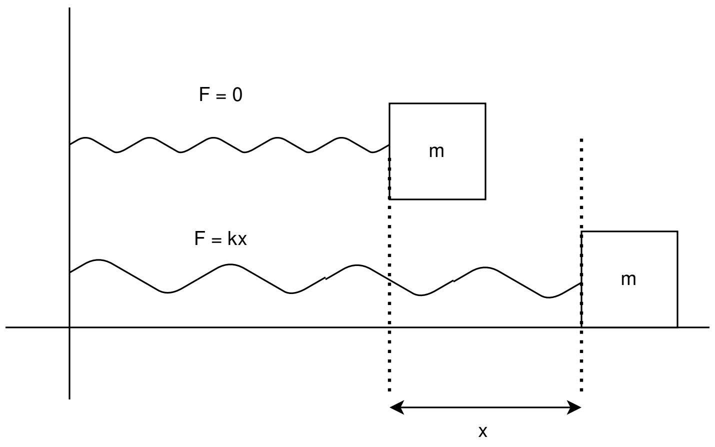


Figure 1.2: Hooke's Law: The force on the spring is linearly proportional to the extension or compression experienced by the spring.

Though there are many different types of accelerometers, all find their basis in Hooke's law. Imagine an object which has a spring attached to it and a small mass at the end of the spring. Whenever the object wants to accelerate, the mass will want to remain stationary due to its inertia. This will cause the spring to compress and elongate, creating the force that can be measured using Hooke's law and can be directly associated with the acceleration of the object.

Modern accelerometers are micro-electro mechanical systems (MEMS), as depicted in figure 1.3. When the mass is displaced from its neutral position, the capacitance between the fixed electrodes and the mass is measured. Thus, MEMS accelerometers report their readings in terms of millivolts which is then internally converted into g-force.

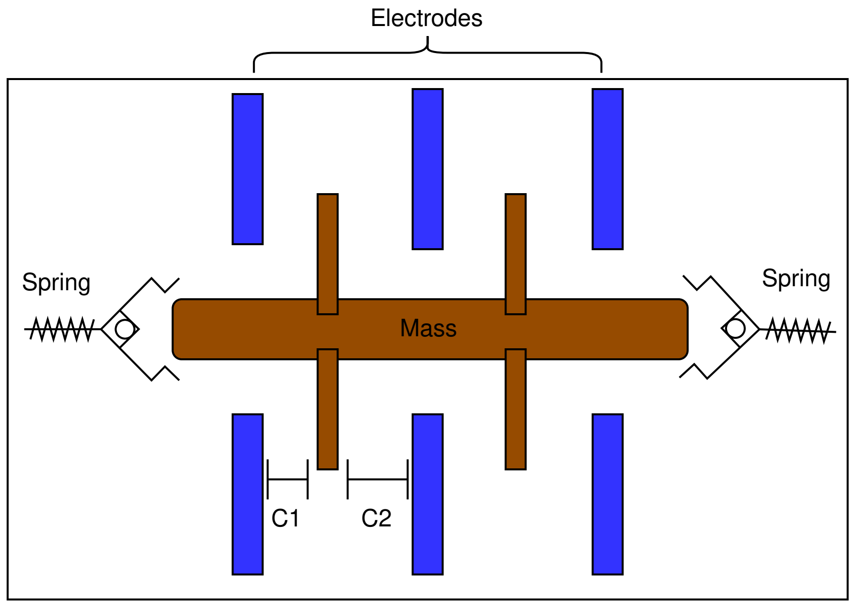


Figure 1.3: MEMS accelerometer

1.3.2 Gyroscope

Gyroscopes are devices used to measure the angular velocity of an object. The angular motion of the wrist varies according to the activity that is performed. For example, angular velocity will be low when we are doing a steady task, like writing or brushing teeth but will be high when doing tasks such as throwing a ball. Since we tend to eat in a consistent manner, a deep learning network could potentially learn the specific motions and differentiate between eating and non-eating activities.

Modern MEMS Coriolis vibratory gyroscopes (CVG), like the ones present on the Shimmer device, use the Coriolis effect to measure the angular motion. They contain a vibrating mass attached to a rotating support. When the support rotates, the vibrating mass will continue to vibrate in the same plane, applying a force on the support. By measuring this force, the angular velocity can be measured.

1.4 Raw vs Linear Acceleration

Linear acceleration, also called inertial acceleration, is acceleration caused by any force other than gravity. A body moving with constant velocity or at rest experiences zero net force and

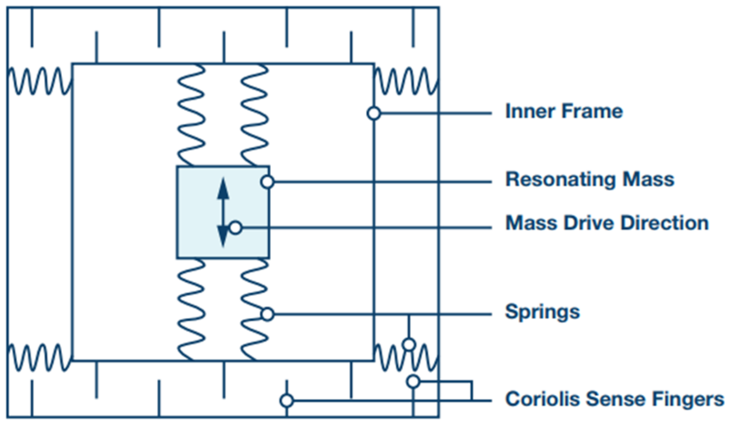


Figure 1.4: MEMS CVG mechanical structure [MEMS Gyroscope Provides Precision Inertial Sensing in Harsh, High Temperature Environments by Jeff Watson].

consequently zero linear acceleration. If a body is in any other state of motion, then it implies that there is some inertial force acting on the body which gives rise to the linear acceleration. This acceleration is usually measured in m/s^2.

Raw acceleration, on the other hand, is a measure of both the physical acceleration of an object as well as the normal forces which contributes in keeping the device from going into free fall. This measured quantity, which contains both gravitational component and a linear component of acceleration is called the raw acceleration. Thus, every raw acceleration measurement contains the linear acceleration measurement in itself.

While both raw and linear acceleration are measurements of an acceleration vector, their interpretations vary. In the case of tracking wrist motion, figure 1.5 demonstrates the practical difference between linear and raw acceleration. Linear acceleration is caused by back-and-forth hand motions generally is in the range of 0.00-0.04 g. On earth, these are dominated by the force of earth's gravity (1 g). The net effect is that linear acceleration provides an estimate of the lateral motion of the hand, while raw acceleration provides an estimate of the tilt of the hand relative to earth.

Accelerometers are only capable of reading and recording raw accelerations. While the modern accelerometers can convert the reading into linear acceleration, directly reading linear acceleration is impossible because of the equivalence principle.

The simplest way to explain what this means is to imagine a person in an elevator with a ball in his hand. When that person drops the ball, the ball accelerates towards the floor at a rate of

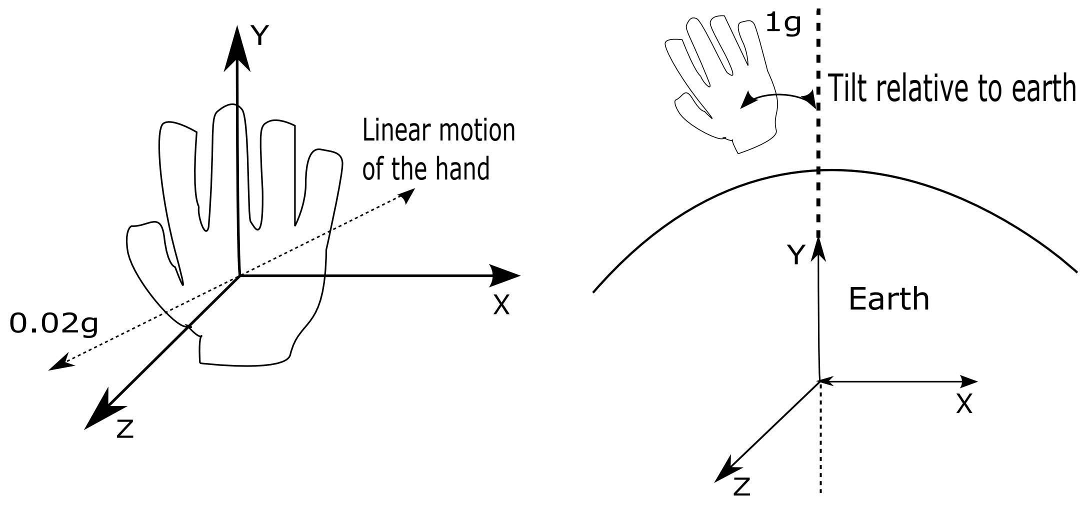


Figure 1.5: Difference in interpretation of raw and linear acceleration.

9.8 m/s^2. Does it imply that the elevator is situated somewhere in the gravitational field of earth? For instance, you can achieve the same result if the elevator was in deep space, away from any source of gravitational force but accelerated in the upwards direction at a rate of 9.8 m/s^2. It would be impossible for the person in the elevator to know which is the case, is the object falling towards the floor, and in extension to a source of gravitational force or is the floor accelerating upwards to the ball which is stationary?

Similarly, accelerometers, being inertial-frame sensors, cannot tell the difference between acceleration due to the effect of gravity and the acceleration due external force acting on the device, i.e., movement of the wrist. Thus, it is impossible for a sensor, present on the device for which it is taking measurements, to be able to capture linear acceleration directly.

1.4.1 Pose Estimation and Raw to Linear Conversion

A tri-axial accelerometer, like the one present on Shimmer device, measures acceleration in three mutually orthogonal axis. Each axis measures a certain proportion of the linear acceleration as well as the gravitational component. In the special scenario, when one of the sensors axis aligns perfectly with the direction of gravity, we can estimate the linear acceleration by simply subtracting gravitational acceleration from the readings, but in all other cases it becomes necessary to first estimate the orientation of the device.

Consider a case of bi-axial sensor measuring in X and Y axis as depicted by figure 1.7. In the

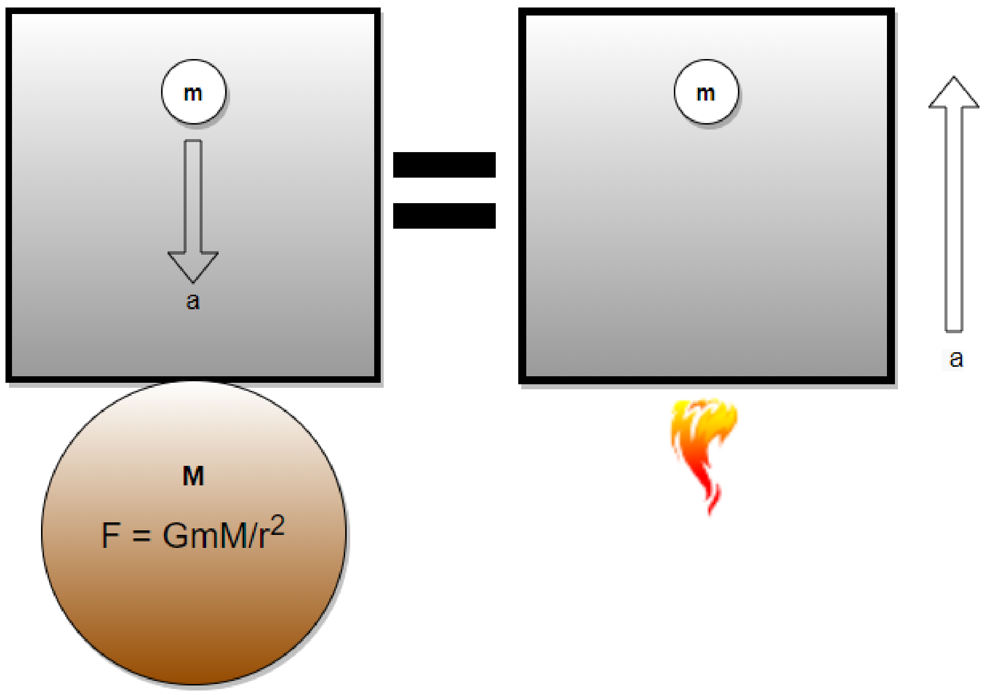


Figure 1.6: Equivalence principle - The ball can't tell the difference between acceleration due to gravity and artificial acceleration.

first case, there is no gravitational component along the X axis, thus extracting linear acceleration from the readings is straightforward. In the second case, when the gravity is no longer in alignment with the sensor axis, gravity is distributed proportionally along both the sensor axis and we must know the angle θ (angle that gravity makes with the sensor axis) to calculate the gravitational component along each axis.

The same concept can be extended to a tri-axial accelerometer and in that this case, the total acceleration can be expressed as:

$$a = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} a_L \cos(\theta_x) + g\cos(\phi_x) \\ a_L \cos(\theta_y) + g\cos(\phi_y) \\ a_L \cos(\theta_z) + g\cos(\phi_z) \end{bmatrix}$$

Here, a is the total acceleration which is the vector sum of gravity and linear acceleration, g is the acceleration produced by gravity, a_x, a_y, a_z are the acceleration measurements of the sensor in each of its axis, a_L is the total linear acceleration from which we can separate out the linear

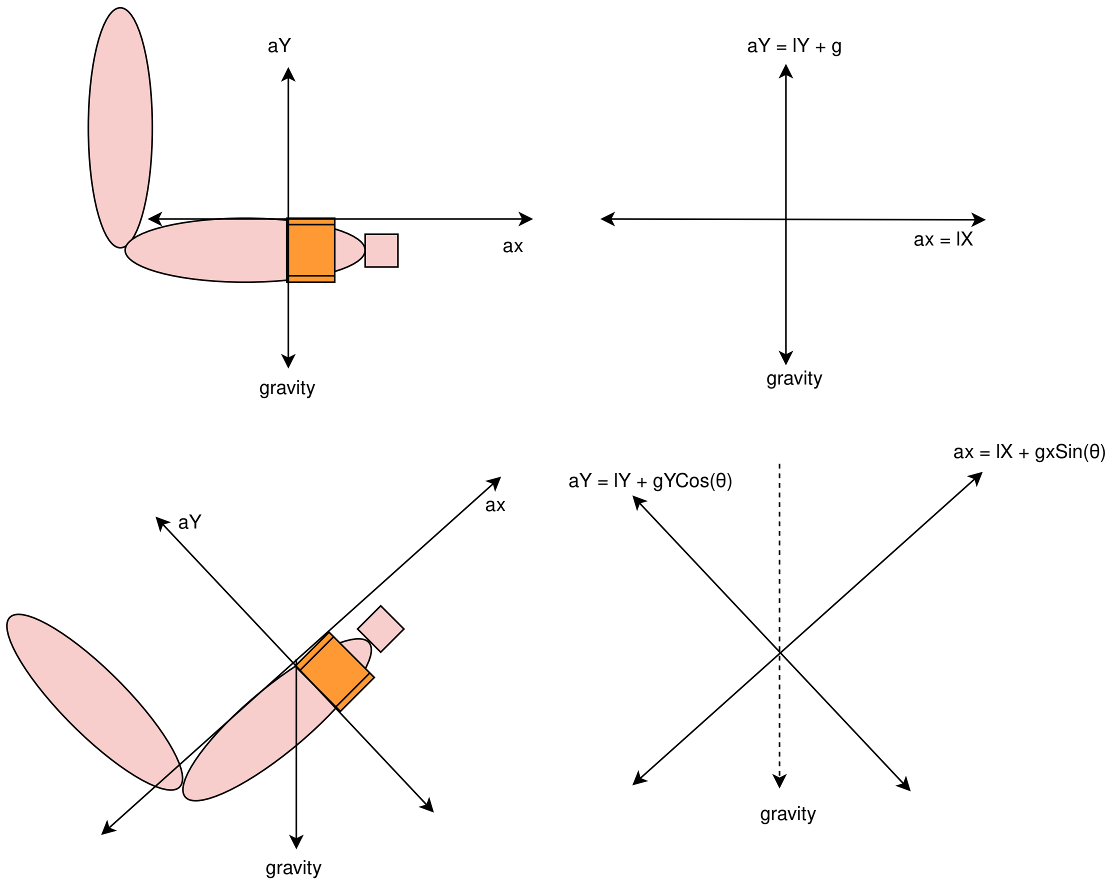


Figure 1.7: Bi-axial accelerometer (orange cube) attached to the wrist. In the first case, gravity aligns with one of the sensor axes. In the second case, the tilt in the wrist causes sensor axes to deviate from the gravity vector. θ is the angle that gravity makes with the X axis.

acceleration in each axis if we know θ, which is the angle that the linear acceleration makes with the sensor axis. $g\cos(\phi_x)$, $g\cos(\phi_y)$, $g\cos(\phi_z)$ are the gravitational components along each of the sensor axis and ϕ is the angle between gravity and each of the sensor axis.

In order to extract linear components of acceleration from raw acceleration, we first need to estimate the orientation of the device. For the Shimmer device, this information can be estimated from the MPU-9150 chips that are on-board the device. Due to computational efficiency and compactness, the MPU-9150 estimates the orientation in form of quaternions which can be converted into rotation matrix using the following equation:

$$R = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2-q_0q_3) & 2(q_0q_2-q_1q_3) \\ 2(q_1q_2-q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_0q_1-q_2q_3) & 1-2(q_1^2+q_2^2) \end{bmatrix} \tag{1.2}$$

Assuming that the rotation matrix is represented as R, and gravity in earth frame is represented as G$_e$, then the gravity, G, in the device frame can be calculated as :

$$G = [R \cdot G_e] \tag{1.3}$$

Once the gravity in device frame is extracted, the linear acceleration (L) can be estimated from the raw acceleration (R) by subtracting the gravity components (G) from the raw acceleration components as :

$$L = R–G \tag{1.4}$$

It should be noted that the direction of gravity vector, measured by the sensor, points away from the earth's surface. Thus gravity must be subtracted from the raw acceleration reading instead of being added to it. The Shimmer guide [17] explains this using the example of a hollow cube with a ball inside.

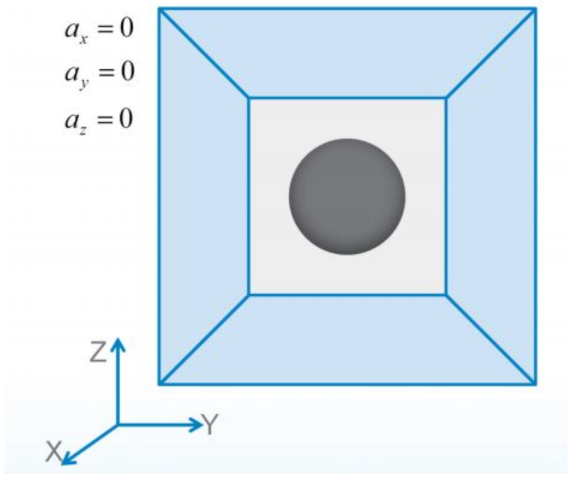


Figure 1.8: Accelerometer measurements if the ball was weightless.

Consider the scenario in figure 1.8. If the ball is weightless, then the measurements along each axis of the accelerometer will be 0 because there will be no force acting on the ball. This case is akin to the device being in free fall.

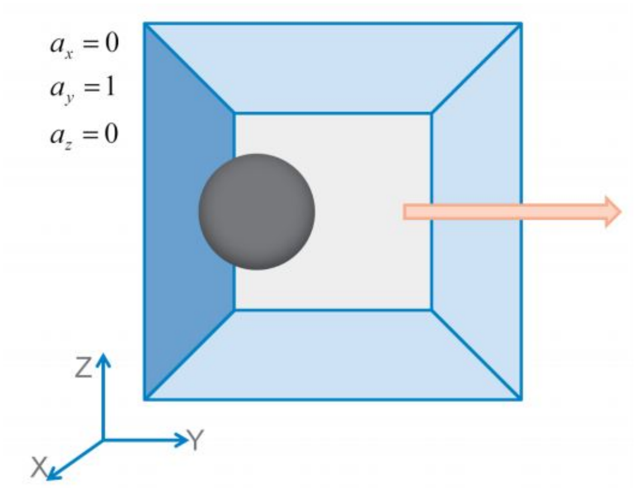


Figure 1.9: Accelerometer measurements if the ball was weightless.

Now consider the case in figure 1.9, when the box is accelerated to the right. In this case, the ball will experience acceleration in positive Y axis due to the force applied by the opposite face of the box.

Finally, lets drop the assumption of weightlessness and include the effect of gravity on the ball as in figure 1.10. In this case, the ball will rest motionlessly on the bottom face of the box, as long as the box is not accelerated. Just like the previous scenario, the ball will experience a force in the positive Z direction, produced by the surface of the cube. This force, called normal force, works against the weight of the ball. The ball (or the accelerometer), in this case, will read an acceleration in positive Z direction produced by the normal force.

1.4.2 Noise in Linear Acceleration

While it is possible to extract linear acceleration from raw acceleration readings, the accuracy of the orientation tracking is still a known issue and a major source of error. A study [21], investigating the orientation error of IMUs found that the overall errors were less dependent on the algorithm used to track the orientation, and more dependent on the amplitude and frequency of the movement. Furthermore, they found that one of the biggest sources of error emanated from the

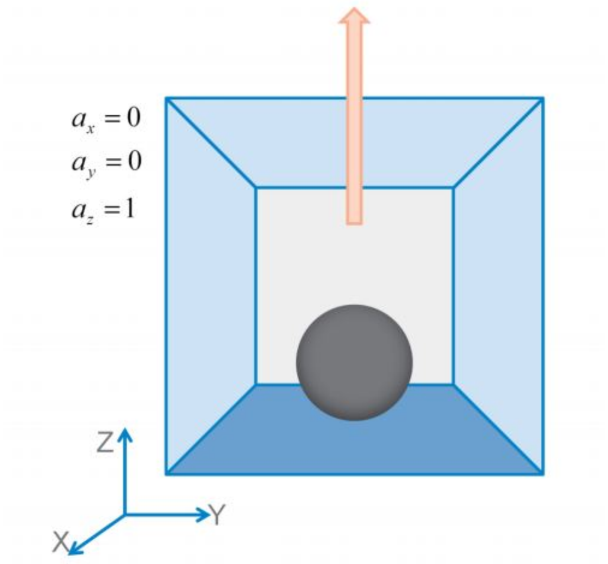


Figure 1.10: Accelerometer measurements if the ball wasn't weightless.

orientation of the rotation axis with respect to the gravity vector.

To estimate orientation, we use a combination of the accelerometer, gyroscope and the magnetometer but even then 1 degree errors in estimation are very common. This is often due to the sensitivity of the sensors to external influence. For example, magnetometers are highly sensitive to all magnetic fields and are even affected by other magnetic materials inside the device. While it is possible to compensate for constant magnetic fields, any moving source of magnetic field can destabilize the readings.

Thus, errors in orientation tracking, which can be as high as 1 degree, and high frequency movement of the wrist, can cause large errors in extracting linear acceleration and because of the difference in the orders of magnitude between linear hand acceleration and earth's gravity, the magnitude of these errors can be comparable to the magnitude of linear acceleration itself.

Furthermore, orientation tracking errors are not the only sources of errors in linear acceleration estimation. When we remove gravitational components from raw acceleration, we ignore the minor changes in gravity along the earth's surface which can vary from 9.76 m/s^2 to 9.83 m/s^2 [10]. Any minor manufacturing defects and biases in the sensors can also severely affect the linear acceleration estimates because of the difference in the order of magnitude, as explained earlier.

In our work, these noise artifacts can be seen in the linear acceleration values extracted from the raw acceleration. During certain short periods of time, noise can be seen as plateaus with non-zero heights. Figure 1.11 plots a sample of the net linear acceleration $(a_x + a_y + a_z)$ of a subject's wrist over the course of 3 hours. As seen in figure, there are plateaus of almost constant linear acceleration with different heights, indicating different accelerations. Such constant linear accelerations distributed randomly are most likely periods of rest and highlight the noise that is present in the linear acceleration reading.

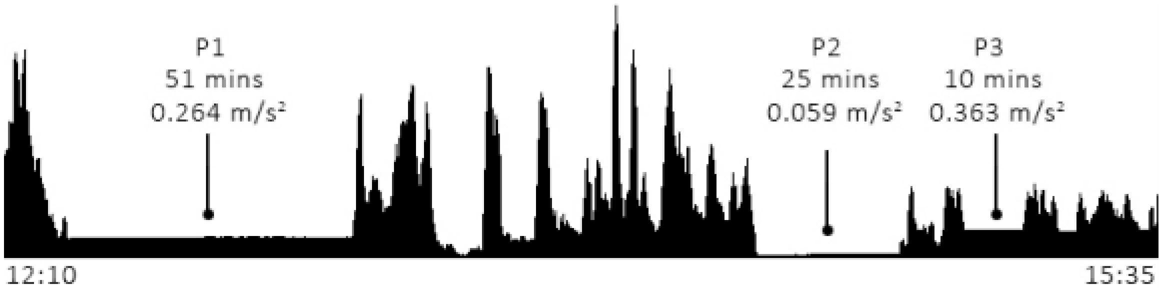


Figure 1.11: Magnitude of net linear acceleration (x+ y + z) for a 3 hour time period. Noise can be seen in the different level plateaus.

Since the error is non-constant, as is visible from the non-constant heights of the plateaus, there is no known way of separating this noise from the readings during motion. The noise can be easily mitigated during rest periods because the ideal value of linear acceleration during those periods should be 0. Thus, any deviation from that value can be attributed to noise. This creates a problem of identifying periods of rest when there is no ground truth available and any type of estimation that we make will likely have error in it.

In his previous work [23], Sharma used a variance based algorithm to detect periods of rest in the CAD dataset. For each datum, a window of fixed length t was centered on the datum and the standard deviations of the accelerometer and gyroscope outputs were calculated. Then thresholding was performed on the sums of the deviations to classify each datum as either being in a state of rest or in state of motion.

Figure 1.12 shows the distribution of the acceleration values for the data detected as being in rest and being in motion. Sharma noted that 90% of wrist acceleration was less than 0.04 g. For data at rest, the acceleration varied from 0.00 g to 0.06 g and that 70% of wrist motion overlapped with rest periods. Since there is a significant overlap between periods of rest and motion, and the noise in the rest periods is almost as much as the acceleration in periods of motion, we can conclude that conversion from raw to linear adds a significant amount of noise to the data. On the other

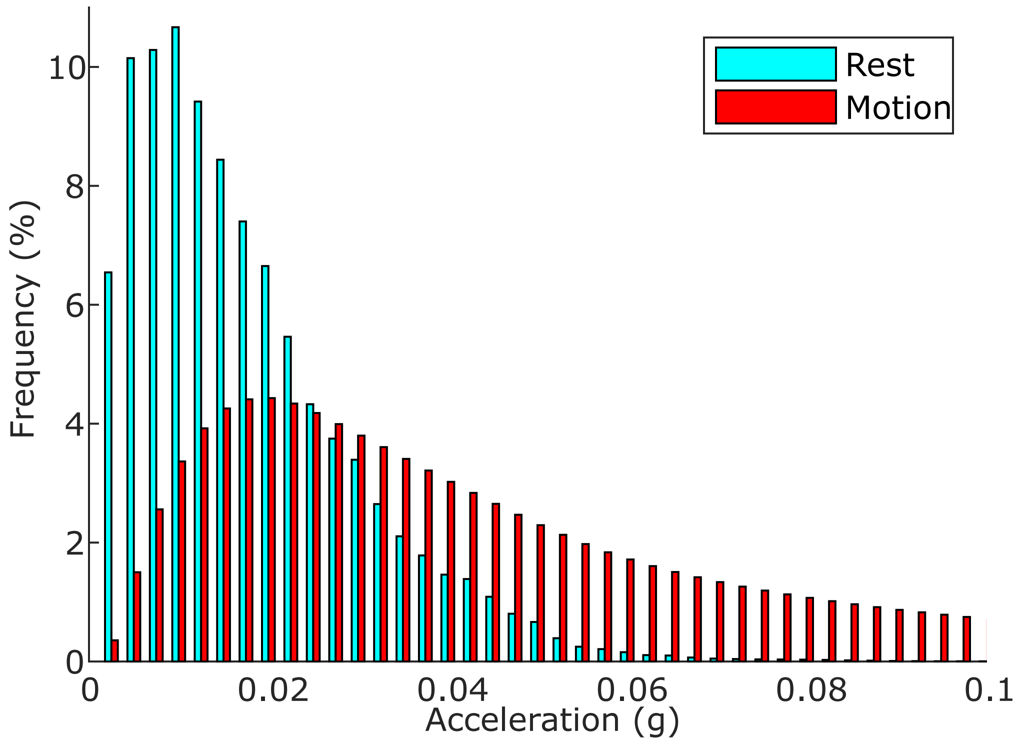


Figure 1.12: The spread of linear acceleration for data detected as rest (blue) and motion (red).

hand, since raw acceleration has a much higher magnitude than linear acceleration, the ratio of noise to the reading is likely to be much lower.

As indicated in Sharma's work [23], a zero mean filter can be used to reduce the artifacts during motion. This is accomplished by averaging the value of linear acceleration over a 10 second window along each of the axis. The window is centered on each datum on a rolling basis. The averaged value, along each axis, is then subtracted from the linear acceleration value for that datum. This essentially works as a high pass filter, retaining information regarding global movements while filtering out slow motions. Thus, using a zero mean filter may help with reducing the error artifacts but will cause you to lose out on information.

1.5 Related Work

The traditional way to track energy consumption is by using food diaries but this method is not optimal as it requires you to carry a diary with you at all times, and/or remember accurately how much you ate and when. Furthermore, most people do not want to spend time required to keep a consistent journal and often people tend to skip writing down the snacks, which account for the bulk of calories. Thus, researchers in this field are now looking for ways to automate this process.

In past, Dong et. al. [3] have used a iphone, mounted on the wrist, to track wrist motion of subjects in free living. A total of 43 subjects participated in the data collection. The subjects recorded 449 total hours of data consisting of 116 eating episodes. Using the data, the group developed a Bayesian classifier to identify meals. The classifier achieved a weighted accuracy of 81%, detecting 100/116 meals and producing 379 false positives.

In 2019, Kyritsis et. al [14], worked on a method to detect food intake cycles using wrist micro-movements. They defined wrist micro-movement as a short sequence of actions that the wrist undertakes when eating a meal, like operating a utensil. They used the publically available FIC dataset containing triaxial acceleration and orientation velocity from 12 subjects over 21 meal sessions. In the first part of the experiment, they define 6 micro-movements and use a CNN to classify windows of eating data as either one of the 6 possible movements. In the second part of the experiment, the output from the CNN is fed into a LSTM (Long Short Term Memory) network to classify window sequences as food intake cycles or not. Overall, their second experiment achieved a F1 score of 0.913 and Precision of 0.895 while their first experiment had an average accuracy of 79%.

In another work by the same group, Kyritsis et. al [15] propose another approach for detecting in-meal eating behaviour and classifying eating episodes. In addition to the FIC dataset, this time they also use FreeFIC and FreeFIC held out which are free-living datasets. In this work, the group takes a bottom-up approach to meal detection. They first attempt to localize "bite" episodes, using a combination of CNN and LSTM networks and then use the distribution of detected bites to segment eating episodes. Their bite detection algorithm achieved a F1 score and precision of 0.923 while the meal start/end point detection on in-the-wild dataset yielded a weighted accuracy of 0.788. It should be noted that their model has 160,000 trainable parameters which reduce the ability of the model to work in real time without a dedicated GPU.

It should be noted that the experiments that use laboratory setting to collect data report high accuracies but are not able to replicate those values on 'in-the-wild' datasets. For example, when the bottom-up classifier in [14], was used on a dataset collected by 11 people outside the lab setting under free-living conditions, the precision decreased to 46% and recall decreased to 63% [15]. This highlights the need for more research under free-living conditions with a larger number of subjects.

In an attempt to classify eating episodes in less restricted settings, Sharma [23] took a top down approach to episode detection. Instead of detecting bite events or micro-movements, he attempted to classify windows of data directly using a CNN. In his work, he collected the Clemson All Day dataset (CAD), which comprised of 351 subjects, and over 4,680 hours of wrist motion data. Using a CNN with approximately 7,500 parameters, he attempted to classify windows of varying sizes (0.5 - 15) as eating or non-eating. A larger window size could potentially allow the model to learn all the micro-movements related to an ingestion event and thus removing the need for a bottom up approach. His method achieved a window classification accuracy of nearly 82%. Furthermore, he found that higher window sizes ($>$ 6 min) did not increase accuracy but did increase inference, prediction and training times. He recommend a window size of 6 minutes which yielded a FP/TP of about 1.7.

In a recent work, Wei [28] trained 8 individual models corresponding to wrist motion data from 8 people and compares their accuracy with a group model trained on the entire CAD dataset. While the individual models averaged a weighted accuracy of 0.819, the group model only averaged 0.780. Thus, the individual models outperformed the group model, but the amount of improvement varied depending on the individual. While in one case, there was a 12% increase in the weighted accuracy, in another individual model, the increase was only 0.2%.

1.6 Novelty

The goal behind this thesis is to determine whether extraction of linear acceleration from raw acceleration is required for using accelerometer data in a CNN model. The thesis answers this question by training and testing two sets of models, one using linear acceleration, and other using raw acceleration. The process of training linear acceleration model doubles as a replication experiment to compare the previous work and set up a new baseline with which we compare the raw acceleration models. To account for the model volatility, we rerun this process multiple times to determine the stability of the model and therefore report accuracies in terms of average $\pm$ standard deviation instead of reporting accuracy from a single run only. We also aim to find the best set of hyper-parameters for both raw and linear models and see if there is a difference between the two.

To summarize, the thesis tries to find the answers to the following questions:

1. Can the results obtained previously using linear acceleration be replicated?
2. Can a classifier recognize eating episodes by analyzing raw acceleration? If so, does the model perform better or worse than linear acceleration?
3. What set of hyper-parameters yield the best results? Is there a difference in the hyper parameters for linear and raw models?

Chapter 2

Methods

This chapter provides a detailed description of the methodology used in the experiment. We begin be describing the dataset and its procurement in section 2.1. Then, in section 2.2, we discuss the steps taken to pre-process the data and prepare it for the model training. In section 2.3, we discuss the model architecture and hyper-parameter settings. Following that, in section 2.4, we describe the various metrics used to evaluate the models. Finally, in section 2.5, the entire training, testing and evaluation process is described. Figure 2.1 overviews all the methods described in this section.

2.1 Data Set

The data for this research, called Clemson All Day (CAD) dataset, was collected in 2014 over a course of one year using Shimmer3 units manufactured by Shimmer Sensing. The participants were recruited from the student, faculty and general population living near the University. The participants were instructed to wear the wrist motion tracker for a period of 1 day and the device recorded the acceleration and gyroscopic measurements for the wrist during that time period. To mark the start and end of a meal, the participants were required to press the button present on the device. The press times were later extracted and were cross checked with the user to ensure accuracy.

A total of 408 subjects were recruited which resulted in a total 354 days of usable recordings.

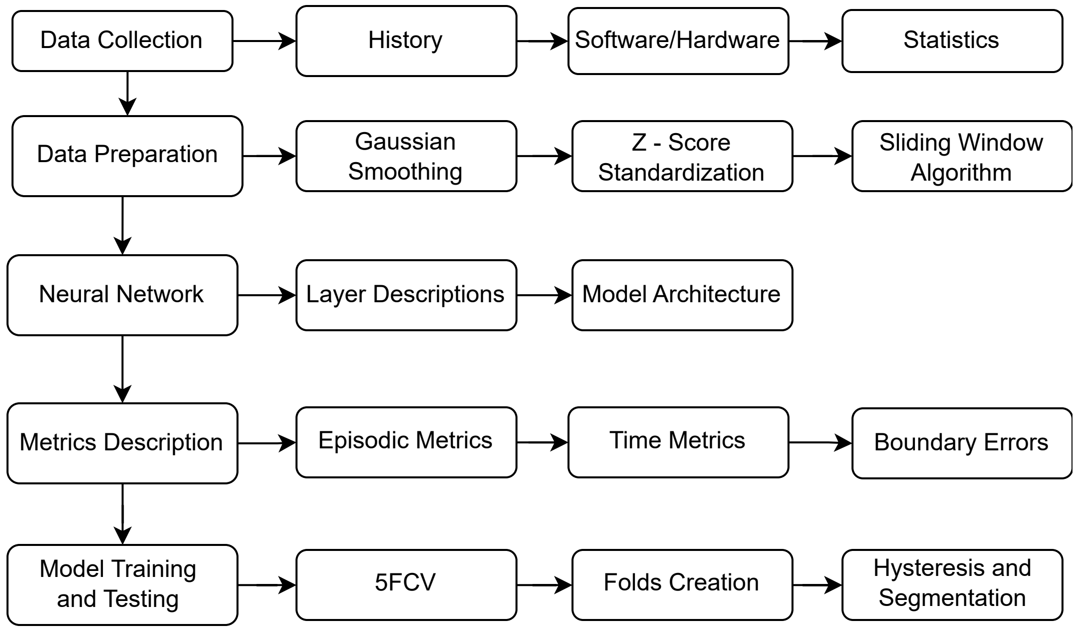


Figure 2.1: Outline of the methods section and the subsections.

Some recordings could not be used due to various reasons like device failure, incorrect usage, or failure to follow procedure by the participant.

2.1.1 Shimmer3 Device

Shimmer3 is a wearable wireless sensor developed by Shimmer Sensing for recording human body motion. While it can be work anywhere on the body, for the collection of this dataset, it was worn on the wrist. The Shimmer device used a MSP430F5437A micro-controller in combination with a 3 axis low noise accelerometer, 3 axis wide range accelerometer, a gyroscope and a magnetometer.

The accelerometer reported its measurements in terms of g-force or g, where 1 g (9.8 m/s^2) represents the standard gravitational acceleration of earth on sea level. So, a 3 axis stationary accelerometer would measure 0 g in x and y axis but 1 g in the +z axis due to the gravitational force exerted by the Earth.

From the two available accelerometers, the low noise accelerometer was preferred over the wide range because the low noise accelerometer offered more accurate reading with a lower sensitivity in measurements. Moreover, acceleration of the wrist was not expected to go beyond the range of the low noise accelerometer which was ± 2 g. For reference, a high speed roller coaster develops 4 g to 5 g of acceleration.

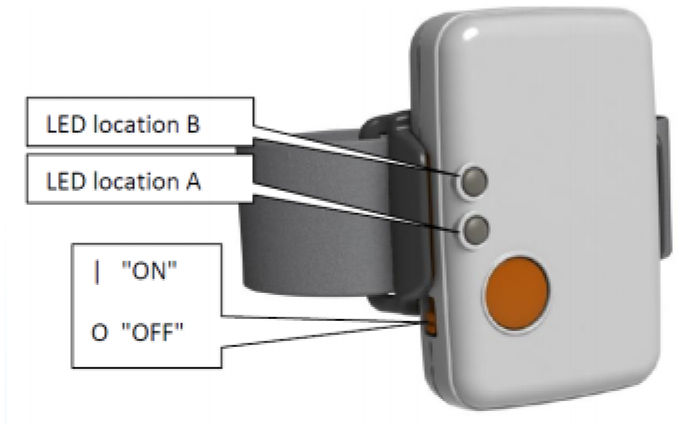


Figure 2.2: Front view of the Shimmer device with button labels

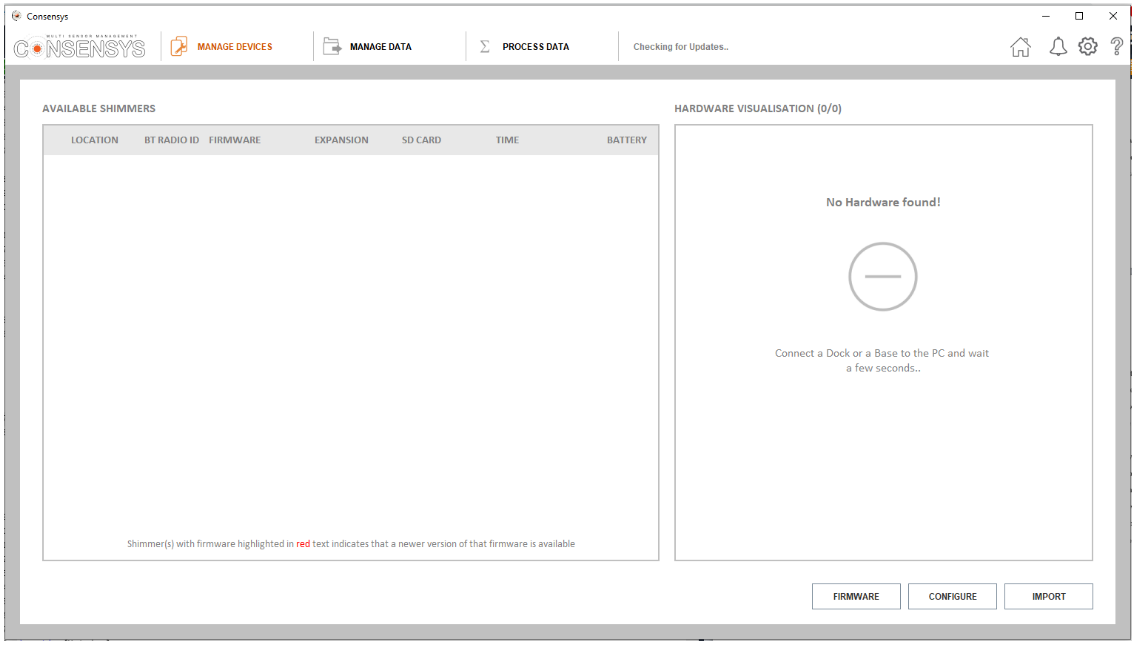
The MSP430 microcontroller is capable of writing the data to an external microSD card. To simplify user interaction, 2 LED lights were available which would signal whether the device was recording the wrist movements or not.

2.1.2 Data Collection History

The device recorded the raw acceleration and the angular velocity of the wrist in each of the x, y and z directions. At the time of the data collection, the accelerometer only recorded the raw acceleration of the wrist and the conversion to linear acceleration, if required, was done offline after recording.

As stated earlier, the data was collected using the wrist worn Shimmer device capable of storing the information on microSD card. From the microSD, the data was exported to a PC and written out to a CSV file using Consensys Software. Consensys could also be used to configure the device and set the sampling rate for the data collection. The default sampling rate was set to 15.06 Hz and because of device limitations could not be changed. Therefore, the data was under-sampled to 15 Hz, once the data collection was completed.

After the ConsenSys software exported the collected wrist data to CSV files, a tertiary program called MarkerParser was used to collate the ground truth data regarding the user-reported periods of eating during the day of data collection. This required the user to either have memorized the eating times or to have marked the eating start and end times using the button provided on the Shimmer Device itself. Aside from the start and end times of the different meals, information



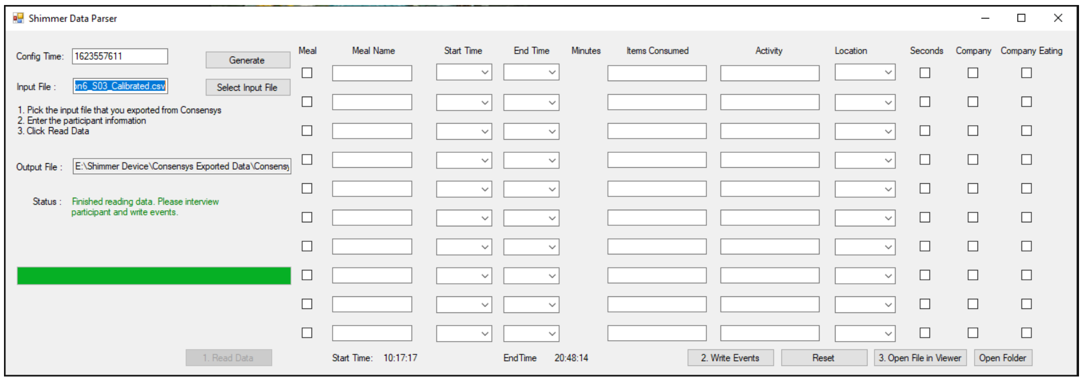
No Hardware found!

Connect a Dock or a Base to the PC and wait a few seconds..

Shimmer(s) with firmware highlighted in red text indicates that a newer version of that firmware is available

Figure 2.3: Interface of Consensys software showing options to import and configure individual Shimmer devices.

regarding food items consumed, the number of servings, whether the meal was consumed in company and where the meals were consumed was also recorded.



Config Time: 1623557611

Input File : n6_S03_Calibrated.csv

1. Pick the input file that you exported from Consensys
2. Enter the participant information
3. Click Read Data

Output File : E:\Shimmer Device\Consensys Exported Data\Consensy

Status : Finished reading data. Please interview participant and write events.

Start Time: 10:17:17

EndTime 20:48:14

Figure 2.4: Interface of MarkerParser software showing the various options related to the meals.

2.1.3 Creating Raw Acceleration Dataset

The raw CSV files recorded by the Shimmer devices were stored in 5 separate zip files. The raw acceleration and gyroscopic measurements were extracted from these CSV files. As stated earlier, due to the sampling frequency of 15.06 Hz, the data had to be under sampled to bring it to

15 Hz. This was done using the time indices that were stored in the CSV files alongside the data. Starting the clock from the third recorded time index, the clock was incremented at the frequency of 15 Hz, or 66.66 ms and the closest data point to the clock time was recorded. Sometimes, the device would stop recording data for anywhere from few seconds to few minutes. In these cases, we inserted the values for the missing time indices based on the length of time that the device would remain turned off. For a few seconds worth of missing data, we replaced the missing values with the closest known values on the either borders of the missing data. For larger lengths of time, i.e., longer than 5 minutes, we replaced the missing data with 0 in the acceleration and gyroscope measurements.

The CSV files also had a column which recorded the internal clock times of the Shimmer Device. This clock had the same tick rate as the sampling rate. In some cases, where the date time column was either corrupted or missing entirely, the internal clock column was used to under sample and extract the raw data. Unfortunately, some files had both the internal clock and date-time column missing so those files could not be used to generate the data. Statistics regarding the final dataset are expanded upon in the next section.

2.1.4 Data Statistics

During the process of original data collection 408 participants were recruited and data was collected over all 7 days. Of the 408 participants, 351 managed to successfully complete the data collection, yielding a total of 354 days of usable data. One participant collected 3 days of data while another collected 2 days and the rest 1 day. A total of 4,680 hours of data was collected, containing 265 hours of eating activity across 1,133 separate meals in the original data collection. The average amount of data recorded per participant was 13.2 hours. The average start time for a recording was 8:50 am, while the average end time for recordings was 22:06 pm.

Of these 354 days/files of data, 12 files did not have a consistent and surviving date-time or clock column so for the purposes of this thesis, our data only consisted of 342 days of use able data. Each day of data was stored in a unique file, yielded 342 files. Table 2.1 shows some key details regarding the demographics.

1,101 meals were extracted from the 342 files of which 67 meals were further ignored. This was because either the meals were too long, too short or did not fulfill the criteria to be considered a meal. For example, sipping coffee for an hour while driving. Therefore, in total, 1,034 meals from 342 days form the dataset for this thesis.

n	
342	
Age	
mean (±SD)	27.53 ± 11.5
Gender	
Female	207
Male	125
Not Documented	9
Ethnicity	
Black	66
White	202
Other	64
Not Documented	9
BMI	
mean (±SD)	25.7 ± 5.65

Table 2.1: Demographic details for participants forming the raw acceleration dataset

2.2 Data Preparation

This section details the techniques used in prepossessing the data to create training and testing sets as well as the model that was ultimately used to train both liner and raw acceleration models.

2.2.1 Gaussian Smoothing

Signal smoothing is the process of creating an approximate function that attempts to capture the important patterns in the data while leaving out noise and outliers. The Shimmer3 stored data at a frequency of 15.06 Hz meaning that the device sampled roughly 15 values from accelerometer and gyroscope every second. When sampling at such a high frequency, there is bound to be significant noise in the readings and when it comes to time series, individual measurements are not as important as the patterns they form. Thus, the first part of pre-processing was to filter the readings from the sensors independently using a 1D Gaussian filter.

Gaussian filter is implemented using a Gaussian kernel, which essentially acts as a weighting function. The kernel is slid across each of the axis and convolution is performed across each data point, yielding a new value which is weighted average of its nearby points.

$$G(x) = e^{\frac{-x^2}{2\sigma^2}} \tag{2.1}$$

In this work, the kernel generated is of size 15, with a variance of 10. Equation 2.4 is used to generate the 1D kernel. Figure 2.5 shows the effect of smoothing on the original signal.

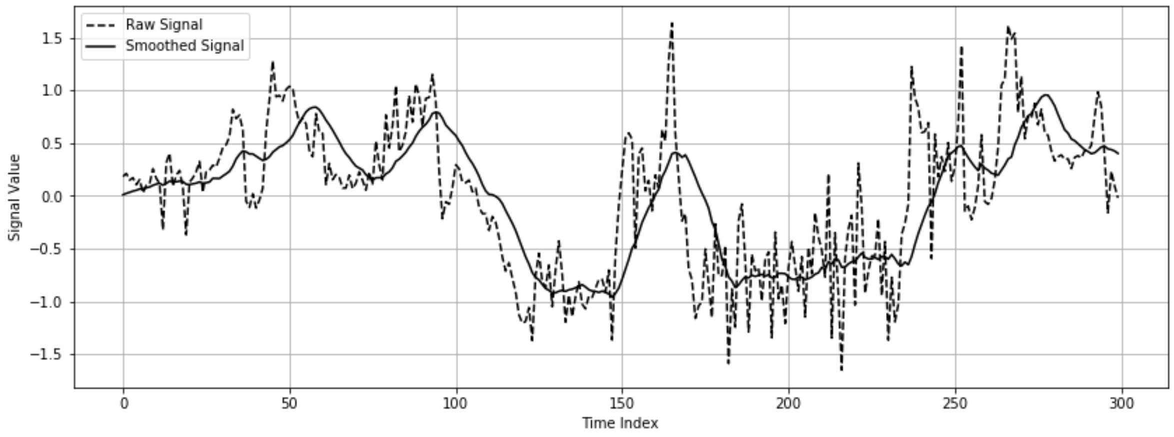


Figure 2.5: Gaussian smoothing on noisy signal.

2.2.2 Z - Score Standardization

Accelerometers and gyroscopes measure two very different quantities so the range of possible values that the sensors output also varies. This variation is also present in each of the axis that the sensors measure. Since, any machine learning algorithm only sees numbers and not the units associated with those numbers, inputs that have higher variance are treated as more important and can have more effect on the model. This is a well known phenomenon in machine learning and in order to avoid forming this internal bias in the model, it is important to feature scale the inputs so that the model treats them with equal importance. Furthermore, feature scaling has shown to also increase the robustness of the model and allow the gradient descent to converge quicker.

In this work, Z score normalization was used to feature scale each of the axis. In practice, this is accomplished by subtracting the mean of the variable from each value and then dividing by the standard deviation of the variable. This makes the value of each feature have zero-mean with a unit variance.

$$x' = \frac{x - \bar{x}}{\sigma} \tag{2.2}$$

Previous work [4] has shown that Z Standardization works better than the alternative like min-max scaling.

2.2.3 Sliding Window

Since the dataset was under sampled to a constant 15 Hz, we can treat the collected data as a time series. When working with a time series data, the most common way of forming training and testing datasets is to frame the data as continuous windows of fixed length. Providing individual data points to the model as inputs will not yield good results as these points only tell instantaneous acceleration and tilt.

In this work, windows of varying lengths were cut, from 0.25 seconds to 15 seconds, and a set of models were created for each window length, in accordance with the cross validation technique discussed later. For forming the dataset, we used a slide of 15 seconds which corresponded to 15x15 = 225 data points. Slide refers to the number of data points that the window was moved before we cut a new slice. While any value of slide could be used, earlier work [23] has shown that a slide of 225 seems to work best for training the models. Slide of 1 was avoided because it would create more data than we needed to train the classifier and the windows would be too similar to each other, and that could lead issues like over fitting. We also cannot set the slide to a value greater than the window length because this would not only decrease the number of windows that can be cut but also, the uniqueness of each window would mean that the model is looking at completely different frames rather than a continuous motion.

Figure 2.6 explains the sliding window algorithm as used on the CAD dataset. Each file in the CAD dataset was stored as a Tx6 array, where T was the total number of data points sampled by the sensors during the course of the day. The 6 dimensions of the data corresponded to the 6 axis (ax, ay, az, gx, gy, gz) that the sensors measured values in. The size of the new dataset that is created by sliding window can be calculated by the following equation 2.3:

$$N = \frac{T - W}{S} + 1 \tag{2.3}$$

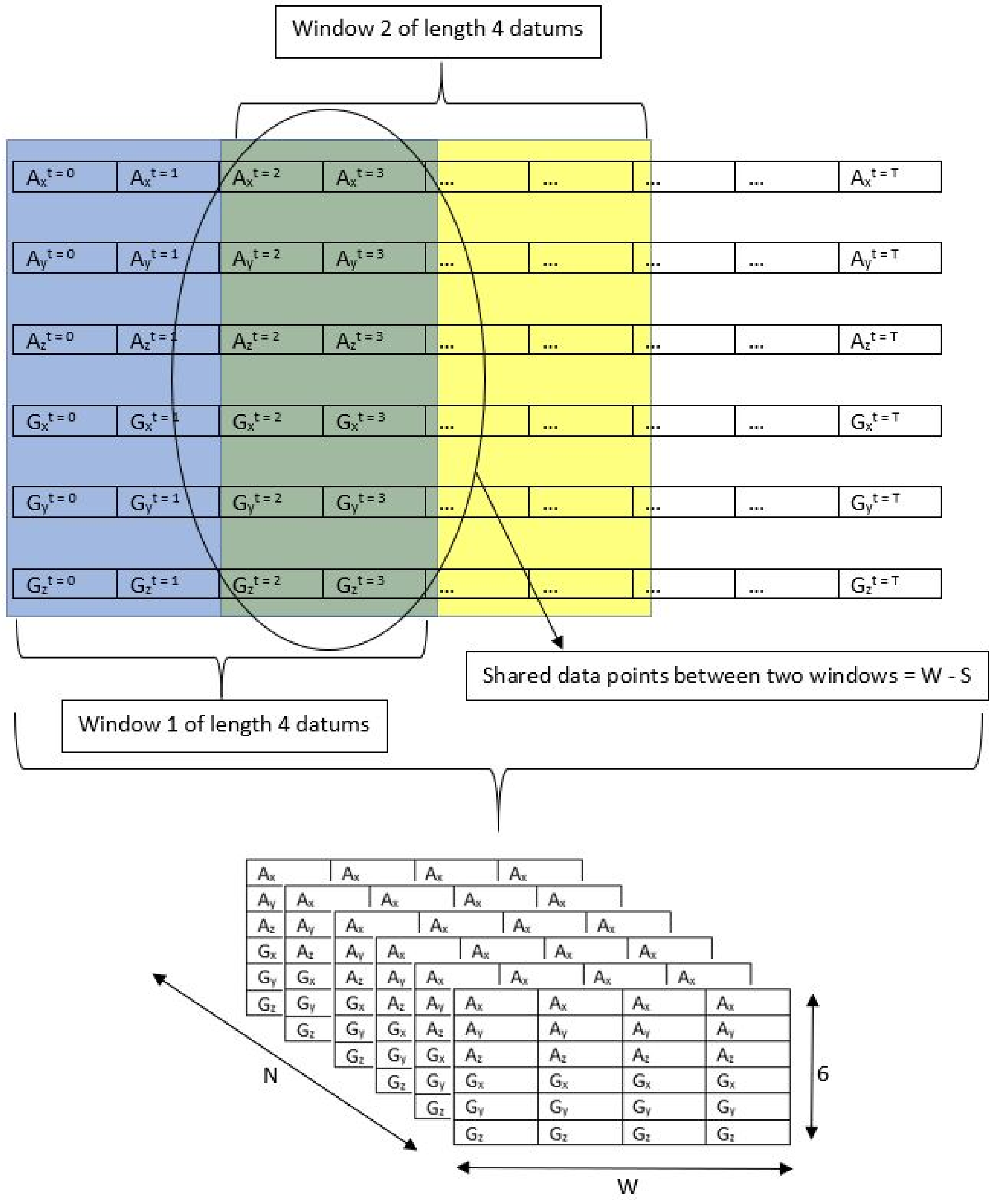

Window 2 of length 4 datums
Window 1 of length 4 datums
Shared data points between two windows = W - S
N
6
W

Figure 2.6: Using sliding window to cut windows of data. Here slide used is 2 and window length is 4. The final windows, stacked together form the dataset of shape NxWx6.

Here, N is the total number of windows that can be cut from a data that is of length T, W refers to the size of the window and S is the slide. Thus, for a given day of recorded data, stored in Tx6 form, we could create a dataset of size NxWx6.

The labels for the dataset were also created using the similar approach. For each window cut from the dataset, its corresponding self reported action was observed. If at least 50% of the data points in the window overlapped with a self-reported meal then the window was labeled as '1', otherwise a '0'.

2.3 Neural Network

This section describes the neural network architecture used to train the models as well as the different layers and their purposes.

According to a 2019 survey [9], the most commonly used approaches to detecting eating by tracking upper limb movements were support vector machines (SVMs), random forests, decision trees and hidden Markov models. Further, the survey found that only 5 works used deep learning approach to detecting eating. All of those works used a bottom – up approach to finding the eating periods i.e. they first identified the gesture and then classified it as eating or non - eating. Previous work [23] by our research group has shown that a top down approach using CNN is also viable.

2.3.1 Convolutional Layer

Convolutional layer contains a set of filters, with teachable weights that are tuned using back propagation. The depth of the layer is determined by the number of filters that the layer has. Convolution of filters across the input generates a feature map. Each filter generates a unique feature map (also called activation map) and the output of the layer are the feature maps stacked together.

The output shape of each convolutional layer is determined by the depth of the layer, the stride of the filters and the padding. Stride refers to the number of data points that the filter moves between successive convolutions and padding refers to the number of pixels added to the borders of the activation map to control the shape of the output. Padding is often done to keep the shape of the feature map same as the input to the layer.

As seen in figure 2.7, when working with time series data we often prefer to use 1D convolutional layers. This is because we expect the kernel (another name for a filter) to move in 1D only. While the use of 2D layers is possible, it is discouraged because all the variables together paint a more holistic picture of the movement as opposed to two or three.

For a 1D convolutional layer, we can determine the output shape as indicated by the equation

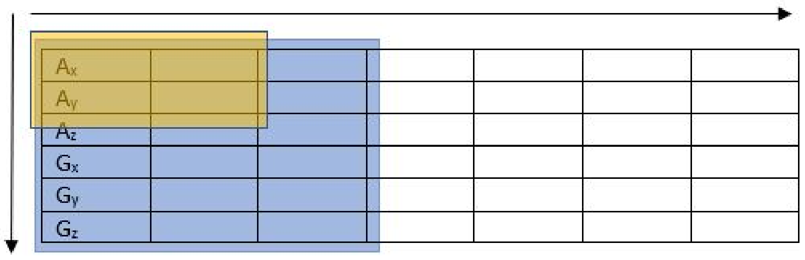


Figure 2.7: 1D Conv vs 2D Conv : The blue filter can only move in 1D, while the yellow can perform convolution in 2D. We prefer the blue as it can see the acceleration and tilt in all 6 axis.

2.4. Here, O refers to the output size, I is the input size, f is the kernel size and S is the stride.

$$O = \frac{I - f}{S} + 1 \tag{2.4}$$

While convolutional layers are often the first layers in CNN, they can also be stacked together and have their inputs and outputs connected such that the input to the next layer is the output of the previous layer. This is a very powerful idea and is in fact what allows the model to gain a deeper insight into the data and learn more powerful patterns that may not be visible to human eye.

For this work, we used a total of three convolutional layers. All of the three layers have 10 filters each and perform convolution with a stride of 2 but have different size of filters. The first layer has a filter size of 44. This corresponds to about 3 seconds of data and was set to this value because according to earlier work [24, 20] average bite length is about 3 seconds. The second layer has a filter size of 20 while the last layer has a filter size of 4. We also place a activation layer after each convolutional layer, described in more detail in the next section.

2.3.2 Activation Function

Taking a closer look at a neuron, we see that the output is a linear combination of the weight, input, and the bias, as represented in equation 2.5. Thus, there are no bounds on the output value of a neuron, the value could range anywhere from -inf to +inf. As stated earlier, if the inputs are not bounded, the model tends to perform worse. Moreover, if the output becomes too small or too large, it could cause computational issues.

$$output = (weights * input) + bias \tag{2.5}$$

Perhaps, the most important problem with the convolutional layer is the linearity. While a model like this will work well for linearly separable data, in the real world, data like that almost never exists. Thus, in order for a network to learn more complex patterns, we need to introduce non-linearity into the model. This is the purpose of the activation function.

In this work, we used the ReLU activation function (Rectified Linear Unit). ReLU is a well-researched and the most widely used activation function because of its simplicity. A study in 2011 showed that the ReLU function improved the learning ability of a neural network . The function is mathematically represented as :

$$f(x) = max(0, x) \tag{2.6}$$

In the model we use, there is a ReLU activation function present after each convolutional layer to allow the model to learn complex, hidden arm movement patterns.

2.3.3 Batch normalization Layer

As explained earlier, before we input data to the first convolutional layer, we normalize the input such that it has zero mean and unit variance. While this remains true for the input to first layer, due to non-linearity of the model, the output from that layer no longer maintains this mathematical trait. The value of the mean and the variance of the input changes as it travels through the model. This is called internal covariance shift, and in order to fix this issue, Loffe et.al [12] proposed a new layer called Batch Normalization Layer.

This layer works similar to the standardization that is performed during pre-processing. Before the inputs of the previous layer are fed into the next one, the mean and the variance of the input batch is calculated, as shown by equations 2.7 and 2.8. Here, B represents the batch, and m represents the number of training samples in the batch.

$$\mu_B = \frac{1}{m}\sum_{1}^{m} x_i \tag{2.7}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{1}^{m}(x_i - \mu_B) \tag{2.8}$$

Then, each dimension of the input is normalized separately (2.9) and transformed (2.10).

The parameters, γ and β learned during the training period.

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2}} \tag{2.9}$$

$$y_i = \gamma \hat{x_i} + \beta \tag{2.10}$$

While there is some evidence that this does not actually solve the internal covariance shift rather only smooths the objective function, it still reduces the training time and positively affects the learning rate. In this work, we added a batch normalization layer between each convolutional layer.

2.3.4 Global Average Pooling

For complex training data, we need to create a sufficiently complex model which requires stacking more convolutional layers on top of each other. While this works in theory, one of the major drawbacks, in practice, is the ever increasing size of the outputs. Large outputs not only slow down the learning rate, but also can cause issues of over-fitting.

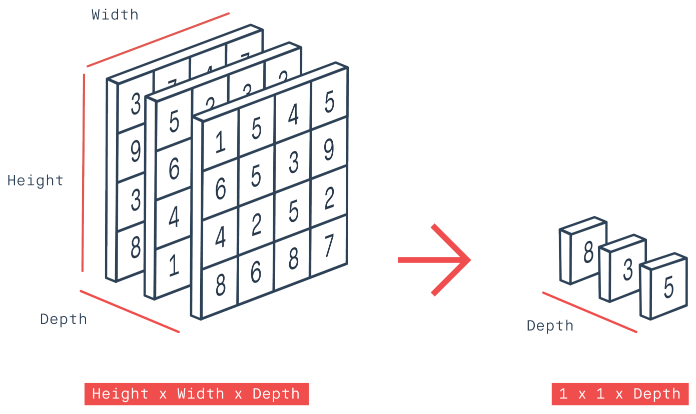


Figure 2.8: Example of global pooling. Each feature map is replaced by its average value.

The most common approach to this problem is to use Pooling layers, which reduce the size of the feature maps by performing a mathematical operation on them. In this work, we use 1D Global Average Pooling operation, which replace the feature map by their average values, as depicted in figure 2.8. The averaged value still contains all the necessary information for the model to learn the underlying structure without associating the classification to a particular sample.

2.3.5 Fully Connected Layer

A fully connected (or dense) layer is composed of neurons which are interconnected with the neurons of the previous layer i.e each neuron in a dense layer receives input from all the neurons of the previous layer. The input is a linear function of the weight matrix, the previous inputs, biases and the activation function, as represented by equation 2.8.

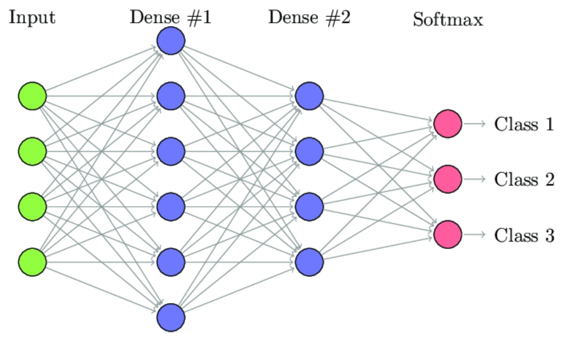


Figure 2.9: Example of interconnections between two dense layers in a multi-class model. In both dense layers, all the neurons are connected to the output of the previous layers. In the actual model, these connections are scalar values called weights.

In this work, we have two fully connected layers stack at the end. The first layer has 200 neurons, with 2200 trainable parameters, which takes input from the preceding global average pooling layer. This layer is connected to the output layer which has a single neuron, with 200 trainable parameters. The purpose of the final dense layer is to convert the feature space into a single scalar value. This value is then fed into a sigmoid activation function (equation 2.11), which compresses the value between 0 and 1, allowing us to interpret the output of the model as the probability of eating. If the output is higher than 0.5, then we label the input window as eating, while a value of less than 0.5 implies non-eating.

$$\sigma(z) = \frac{1}{1 + e^{-x}} \tag{2.11}$$

2.3.6 Loss Function

Use of back-propagation to train neural networks requires us to select a loss function for which the gradients are calculated. The most common function that is used in tandem with sigmoid output is the cross entropy loss function. This is because the output of a sigmoid activation is bounded between 0 and 1, thus loss functions which feature exponents in their derivative will face the problem of saturating neurons, as the calculated gradient will be very small.

In information theory, cross entropy is used to differentiate between two probability distributions. If the vector of true labels, y_i is treated as the primary distribution and the vector of predicted class labels $p(y_i)$ is treated as the secondary distribution, then the cross entropy loss can be calculated as :

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i) \tag{2.12}$$

2.3.7 Network Hyper Parameters

While the layers determine the model architecture, the hyper-parameters determine the way the model is trained. Picking the right set of hyper parameters is a challenging task, and one that is mostly done though trial and error. There are many hyper-parameters that need to be set, the major ones are detailed here :

1. **Batch Size** : Batch size refers to the number of training samples that are fed through the training loop before updating the parameters of a model. While it is possible to use all the training samples before each update, this not only costs memory but also takes a lot of time. Thus, using a smaller batch size is often the best practice. When using a batch size other than the number of samples in training set, we can calculate the total number of parameter updates that will be performed as :

$$Updates = \frac{N}{b} \tag{2.13}$$

   Where, N is the total number of samples in the set and b is the size of the mini-batches that we use to train. In this work, we used a batch size of 256 to train all the models.

2. **Learning Rate** : The learning rate determines the amount of change made in the weights of the model during the update cycle. A high learning rate can cause the model to either not

converge at all where as a very low learning rate could cause the model to take a long time to converge or get stuck in a local optimum. In this work, we used a learning rate of 0.01.

3. **Optimizer** : Optimizers are algorithms that monitor and control certain parameters of the model to minimize the loss function. In this work we use the Adam (Adaptive Momentum Estimation) Optimizer. Adam works on the idea of momentum i.e. by adding the exponentially decaying average of the previous gradients, we can either increase or decrease the weight update value to optimize the learning rate. Equations 2.14 and 2.15 show the first and second order momentum calculated by the Adam optimizer and equation 2.16 shows the modified weight update equation.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.14}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.15}$$

$$\theta_{t+1} = \theta_t - \frac{\mu}{\sqrt{\hat{v}_t} + \eta} \hat{m}_t \tag{2.16}$$

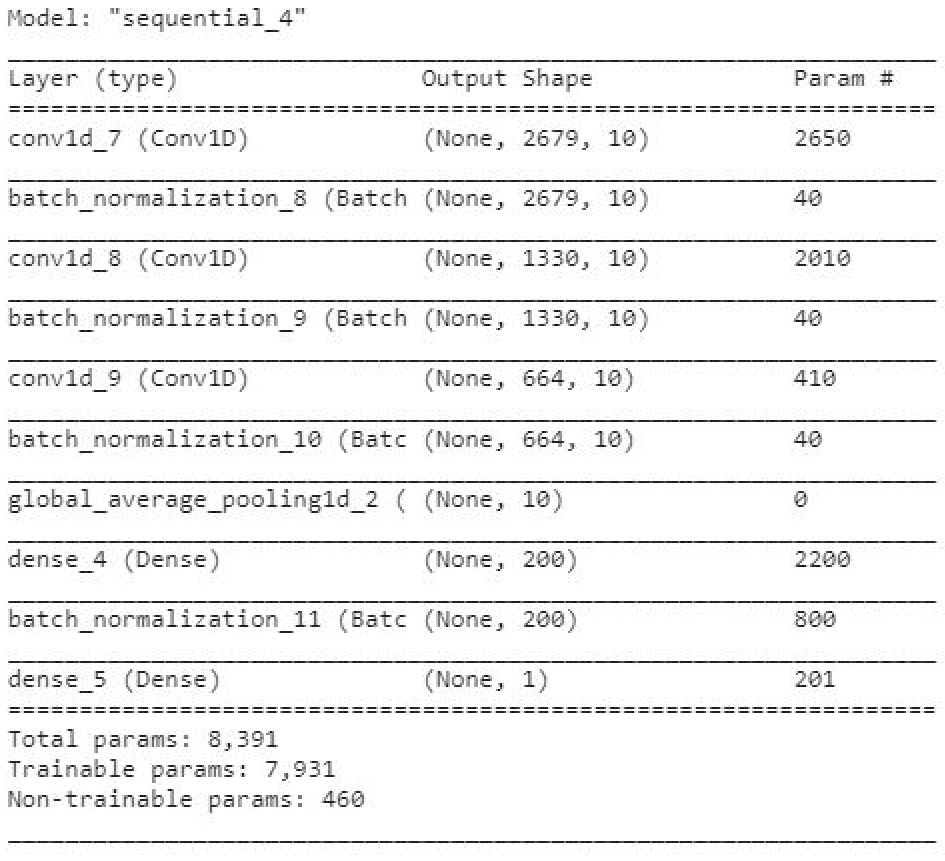
4. **Epochs** : The number of epochs determines the total number of times that the model goes through the entire dataset. In this work, we set the epoch value to 150, to maintain the training environment as setup by the group previously. By combining this with early stopping, we can avoid the problem of over fitting.

5. **Early Stopping** : As the name suggests, early stopping is the mechanism of stopping the training process early, before it reaches the specified number of epochs. This occurs if a certain metric used to monitor the process stops improving. In this work, we used the validation loss as our early stopping criteria with patience of 15 epochs and minimum delta of 0.01.

6. **Model Checkpoint** : In the process of being trained, a deep learning model goes through many iterations of back-prop. Each iteration is bound to change the model accuracy, and not always for the best. Model Checkpoint is a system of saving the model where the model is saved only if it beats a certain metric that is monitored. In this work, we save the model with the lowest validation loss.

2.3.8 Model Architecture

Figure 2.10 shows the final model architecture that was used to train the models. In summary, the network has 3 convolutional layers, each with a stride of 2 and L1 norm for regularization. After stack of convolutions layers, there exists a global pooling layer, a dense layer and finally, the output layer. All 3 convolutional layers and the dense layer are followed by a batch-normalization layer. In total, the model contains approximately 7,500 trainable parameters.



```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_7 (Conv1D)            (None, 2679, 10)          2650
_________________________________________________________________
batch_normalization_8 (Batch (None, 2679, 10)          40
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 1330, 10)          2010
_________________________________________________________________
batch_normalization_9 (Batch (None, 1330, 10)          40
_________________________________________________________________
conv1d_9 (Conv1D)            (None, 664, 10)           410
_________________________________________________________________
batch_normalization_10 (Batc (None, 664, 10)           40
_________________________________________________________________
global_average_pooling1d_2 ( (None, 10)                0
_________________________________________________________________
dense_4 (Dense)              (None, 200)               2200
_________________________________________________________________
batch_normalization_11 (Batc (None, 200)               800
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 201
=================================================================
Total params: 8,391
Trainable params: 7,931
Non-trainable params: 460
_________________________________________________________________
```

Figure 2.10: Architecture of the CNN model.

2.4 Hysteresis and Meal Segmentation

Since the model outputs the prediction as a scalar value between 0 (non-eating) and 1 (eating), we can use the sliding window algorithm to generate a probability curve, *p(t)*. For each file in the held out fold and for each window size W, we generate windows of data using the sliding window algorithm, with a slide of 1. This generates a window for each datum in the file. Using that as the input, we generate a continuous curve that peaks when the person is predicted to be eating a meal.

A Hysteresis algorithm is used to segment the predicted meals from the probability distribution. The algorithm uses two thresholds, T_s and T_e. When the probability of eating, $p(t)$ becomes higher than T_s, we mark that datum as the start of a eating episode and when the probability becomes lower than T_e, we mark that as the end of the eating episode. Thus, the two thresholds act as a bounding system for the probability, requiring a strong probability to mark the start of an eating episode but a more relaxed value to mark the end of the eating episode. The value of T_s is higher than that of T_e because we generally eat faster and thus accrue more eating micro-movements when we are hungry. The number of micro-movements related to eating are less and more spread out over time as we become satiated. The models are trained to recognize the movement of the hand and eating more creates more ingestion events, allowing the model to recognize it with a higher probability while eating less creates lesser ingestion events, causing a lower probability towards the end of the meal. Figure 2.11 shows the predicted meals segmented from the probability curve generated by the model. Note, that while probability axis has ticks up to 1.4, the probability can never go above 1. Ticks from 1 - 1.4 are only used to show the ground truth and the predicted meals.

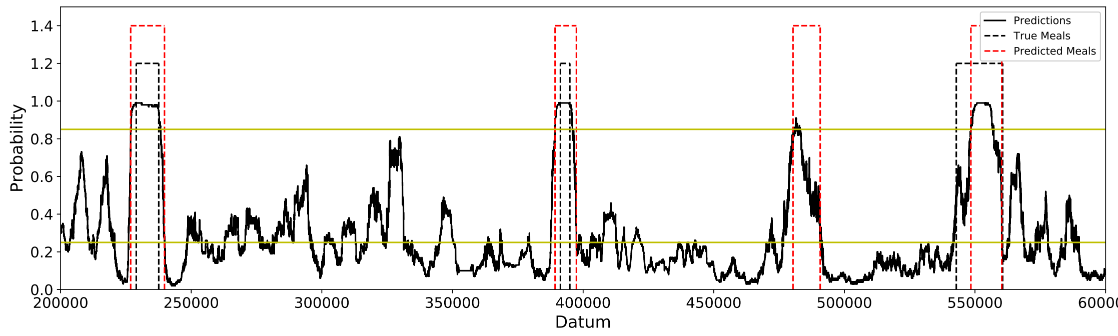


Figure 2.11: Shows hysteresis segmentation with Ts = 0.8 and Te = 0.4. The solid black line is the probability of eating. The dotted black line mark the start and end of the true meals eaten while the red dotted line marks the predicted meals. The horizontal lines mark the Ts and Te.

2.5 Metrics

This section details the various metrics used to evaluate the classifiers and compare the models trained with raw and linear acceleration. In this work, we used two different types of metrics, one using balanced but randomly distributed dataset and second using ordered but imbalanced dataset. The metrics are used at different stages and serve different functions. Each metric is expanded on in the following subsections.

2.5.1 Metrics for Balanced and Randomly Sampled dataset

These metrics were used to analyze the model after the training stage. The goal of this metric was to configure the model hyper-parameters and the training set to reduce possible over-fitting and catch any problems before we moved on to more complex, and expansive quantitative assessments.

Since at this stage the testing data was balanced, we used the raw accuracy score of the classifier on the testing data as the main metric. In a binary context, accuracy refers to how well does the classifier differentiate between the two possible classes. If there existed a large gap between the training accuracy and the testing accuracy of the models, we would stop and reconfigure some of the parameters like epochs, learning rate etc to reduce the problem. This served as the preliminary measure of how well is the model doing on unseen testing data.

2.5.2 Metrics for Continuous and Ordered Data

2.5.2.1 Episode Metrics

Episodic metrics quantify the ability of the model to recognize and classify meals, irrespective of the total time spent eating. For our purposes, any self identified meal counts as an eating episode and is weighted equally against other meals, no matter the length of the meal. Thus, an eating episode that takes 13 minutes is statistically as important as the one that takes 5 minutes. Furthermore, the accuracy of the detected meal, in terms of the amount of overlap in predicted and true meal is also ignored. This gives rise to 4 possible scenarios, as explained by figure 2.12.

In this work, we label each detected meal segment as either a true positive (TP), false positive (FP) or a miss. A TP detected meal is a meal that overlaps with a true meal. A detected meal will be labeled as a TP as long as any amount of overlap exists between the two meals. A FP

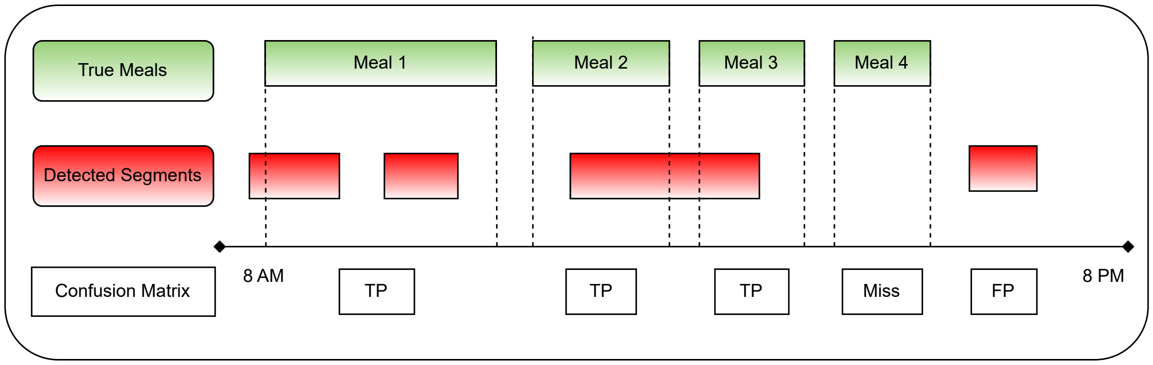


Figure 2.12: Graphic explains the metric definitions. We consider a case of a person eating 4 meals across 8 AM to 8 PM. The classifier detected meals are marked in red while the true meals are marked in green. The confusion matrix row tells us what each segment is classified as.

meal is a predicted meal which has no overlap with any of the true meals. A true meal that has no overlap (and thus remained undetected by the model) is labelled as a miss. There is no definition of a false negative (FN) in the context of this work and thus is not used as a metric. We do not report true negatives for episodic metrics because we are only concerned with measuring the accuracy of detecting eating episodes.

Consider the example in figure 2.12. Meal 1 will be considered detected because at least one detected segment overlaps with the true meal. Meals 2 and 3 will also be considered detected as one segment overlaps with both, classifying both as detected. Meal 4 remains undetected and will be classified as a miss while the last segment will be counted as an false positive because there is no overlap with a true meal.

The above definitions allow us to calculate the true positive rate (TPR) as :

$$TPR = \frac{TP}{TP + miss} \tag{2.17}$$

and FP-TP ratio as :

$$FP/TP = \frac{FP}{TP} \tag{2.18}$$

The TPR tells us the proportion of meals that were correctly detected by a model and should be as high as possible, ideally being 1. The FP-TP ratio tells us the number of falsely detected meals for every correctly detected meal. This number should be as low as possible, ideally 0.

2.5.2.2 Time Metrics

Time metrics quantify the ability of the models to classify each moment of time (each datum) as eating or non-eating. We classify each datum as either a true positive, false positive, true negative or a false negative. The definition of the labels remains consistent with episodic metrics, in addition of true negative and false negative. A true negative is a datum that is correctly classified as non-eating, while a false negative is a datum that is incorrectly classified as non-eating i.e. its true label is eating.

Using these labels, we further calculate the sensitivity, precision, F1 score and weighted accuracy (ACC_W) of the model. Considering that there will always be more non-eating than eating datums, there exists a class imbalance in the dataset. F1 score and precision are known to be adversely affected by class imbalance, i.e. they can give a sense of false confidence. Thus, weighted accuracy is considered the more useful and correct measure of accuracy. Since non-eating occurs roughly 20 more times than eating, we can calculate ACC_W as done previously [18] :

$$ACC_W = \frac{TP \times 20 + TN}{(TP + FN) \times 20 + (TN + FP)} \tag{2.19}$$

2.5.2.3 Boundary Error

Boundary error, calculated only for meals that are true positives, measures the accuracy of identifying a meal with respect to its start and end times. In this metric, we calculate the average difference between the start times and the end times of the actual meal and the predicted meal. In case that a meal is overlapped by two predicted meals, we use the start time of the first meal and the end time of the second meal. Figure 2.13 explains the various scenarios that can occur.

2.6 Model Training and Evaluation Process

Having explained the metrics used in evaluation, we now look at the entire training and evaluation process, from the beginning to the end.

2.6.1 Balancing the Dataset

Imbalance in the dataset occurs when the distribution of classes in the ground truth is uneven i.e. there are more samples of one class as compared to another class. Such an imbalance can pose

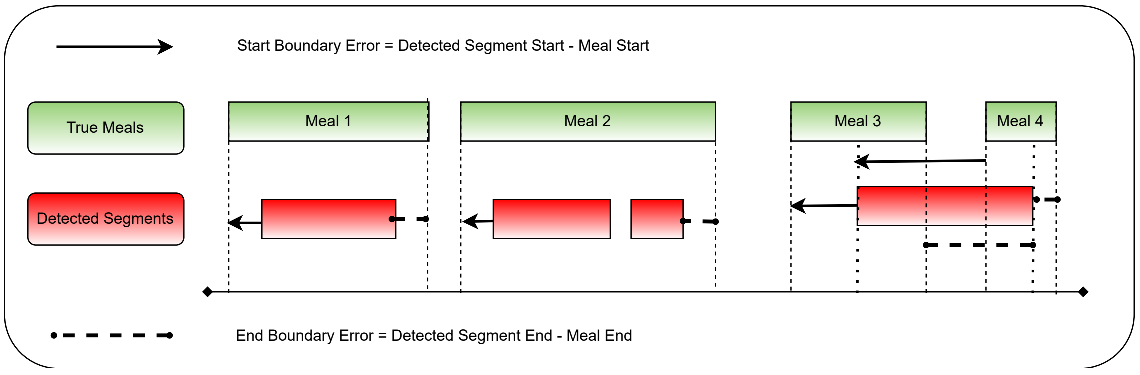

Start Boundary Error = Detected Segment Start - Meal Start
True Meals
Detected Segments
Meal 1
Meal 2
Meal 3
Meal 4
End Boundary Error = Detected Segment End - Meal End

Figure 2.13: Evaluating the boundary error of the TP meals. Meal 1 is identified by 1 segment. Meal 2 is identified by 2 segments, we use the boundaries of the combined segments. Meal 3 and 4 are identified by a single segment, in which case we calculate the boundaries for both separately.

significant problems in predictive modeling because machine learning algorithms expect there to be equal examples of all classes. Using imbalanced data to train can cause problems like over-fitting while using imbalanced data to test the model can result in false statistical measurements.

In our case, there exists a natural and expected imbalance in the data. According to a study conducted by USDA [8], an average American adult only spends about 67.8 minutes of their day in primary eating and drinking activities. This means, over a 24 hour period, most people only spend about 1 hour eating food. This is also reflected in our recorded data where there is a roughly 20:1 ratio of eating to non-eating hours.

There are many ways of solving the problem of class imbalance. Depending upon the data complexity, creating synthetic data is a possibility and there exist many algorithms to do so. Given the intricate nature and dimensionality of our data, this was not an option. In this work, we balance our dataset through the method of under sampling the over-represented class. In both training and preliminary testing stage, after cutting windows of data through sliding window algorithm, we under sample without replacement the non-eating class so that the number of samples in that class are equal to the eating class.

2.6.2 K - Fold Cross Validation

K fold cross validation (KFCV) is a model validation technique that can be used to estimate how well will a model generalize to unseen data. In KFCV, we split the entire dataset into K non-overlapping subsets, also called folds. Of the K folds, we retain one fold to serve as the ”unseen” data and train the model using the remaining K - 1 folds. This process is repeated K times, and each fold serves as the testing data exactly once.

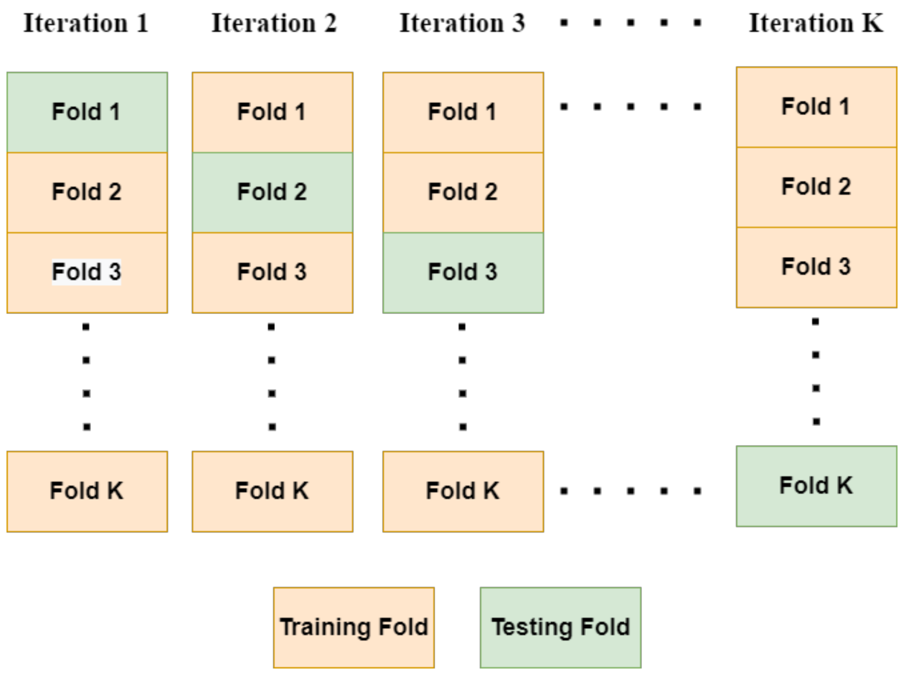


Figure 2.14: K fold CV in action. Each fold is rotated as the test set. This also means that K models are trained in the process.

Model performance is indicated by the averaged value of any performance metric over the K folds. KFCV has many advantages over training with entire dataset. The variance of any estimate is lower when measured using KFCV. Moreover, it is extremely easy to implement and works well with large datasets, where training with the entire set may not be feasible due to limitations on memory. It also has certain disadvantages like the process has to run K times which means more compute time and this method does not work well with small datasets where dividing in K folds may not leave you with enough data to train.

In this work, we use 5 folds to train and evaluate the model. We divide the 342 available days of data into 5 unique folds. We then train 5 models, with each model being trained on set of 4 folds and tested on the 5th. Training data for each model is generated using sliding window and non-eating class is under-sampled to create a balanced training set. Once model training is completed, we generate a test set from the held out fold. The held out fold is also under sampled to create a balanced testing set. By comparing the training and testing accuracy, we can determine the degree of over-fitting.

Once the models have been trained to satisfaction, we then use each of the models to make prediction over their respective held out fold. The model generates a probability curve of eating over the course of the day. We then use hysteresis algorithm to extract the predicted meals from the curve. Statistics regarding time and episode metrics are then calculated by comparing the timeline of predicted and actual meals.

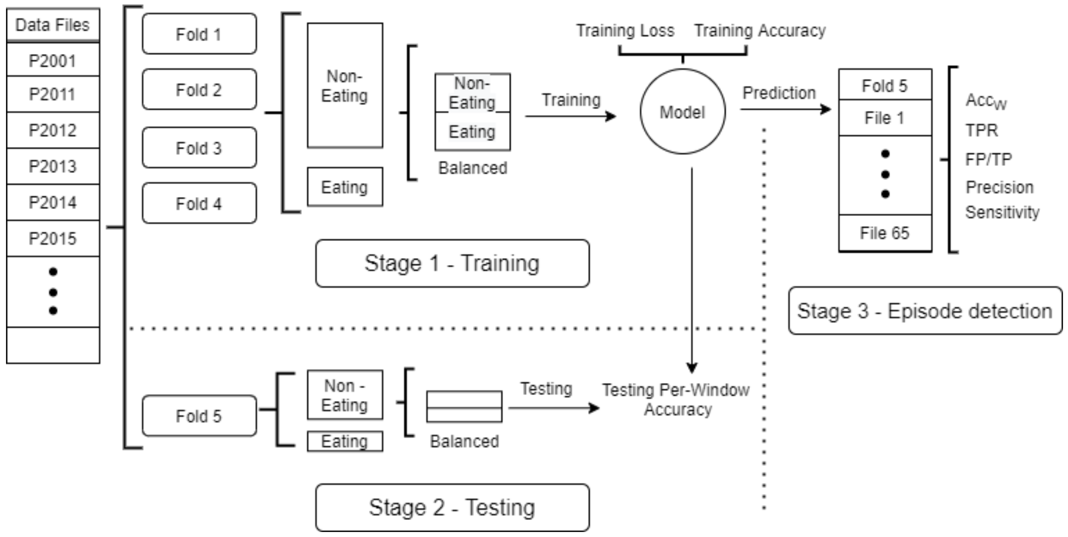


Figure 2.15: 5 fold cross validation and data preparation process followed by model training, testing and episode detection. This process is repeated 5 times, each held out fold acts as a testing and prediction fold once.

2.6.3 Striped Folds Creation

The general method of creating folds is to shuffle the dataset randomly and then split it into K separate groups. This system works well for data that didn't have any underlying bias in the collection process. In our case, that is not true. Data collection for this dataset happened in stages, over the course of a year. Data for students of the Clemson University was collected first, followed by faculty and staff and then finally local townsfolk. Ideally, there would be no difference in the eating habits of people of different ages and occupation but we cannot make that assumption without proof. Therefore, if we follow a random fold creation method, we risk grouping certain demographics together. If the eating habits do in fact differ across the sections then a model trained on a particular demographic will not be able to generalize to the rest of the population.

To avoid this problem, we follow the method of striped fold creation. Instead of randomly creating five folds, we organize the data in the order of collection (indicated by file name) and then split it into 5 groups. Each subgroup is then further split into 5 smaller groups and their order is then randomized within the subgroup. Finally, we pick one group from each of the five subgroups and combine them to form a fold. This improves the distribution of demographics across all the 5 folds and will assist in reducing over-fitting.

2.6.4 Differences in Replication and Original Work

There exist certain differences between Sharmas work in [23] and this work. Sharma, in his work, trains the models for 150 epochs and saves the model with the highest training accuracy. In this work, we instead use validation loss as the metric for saving the model and also implement early stopping. If the validation loss does not improve over a period of 10 epochs, then the model training is automatically stopped and last model with the lowest validation loss is saved. The validation data is split from the training data with a ratio of 20:80.

The reason behind this change is that when we save models based on training accuracy, we do not account for possible over-fitting. Training for 150 epochs can easily cause the model to over-fit on the training data, which would yield a high training accuracy. This may lead us to a over-confident model which would perform poorly on unseen testing data.

2.7 Measuring Model Volatility

One of the goals of this thesis is to quantify the volatility in the model training process. We define this volatility as the change in accuracy and the measured metrics (boundary, time, episode) every time a model is trained.

Testing Fold	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Sharma Testing Accuracy (%)	**81**	**83**	**85**	**76**	**81**

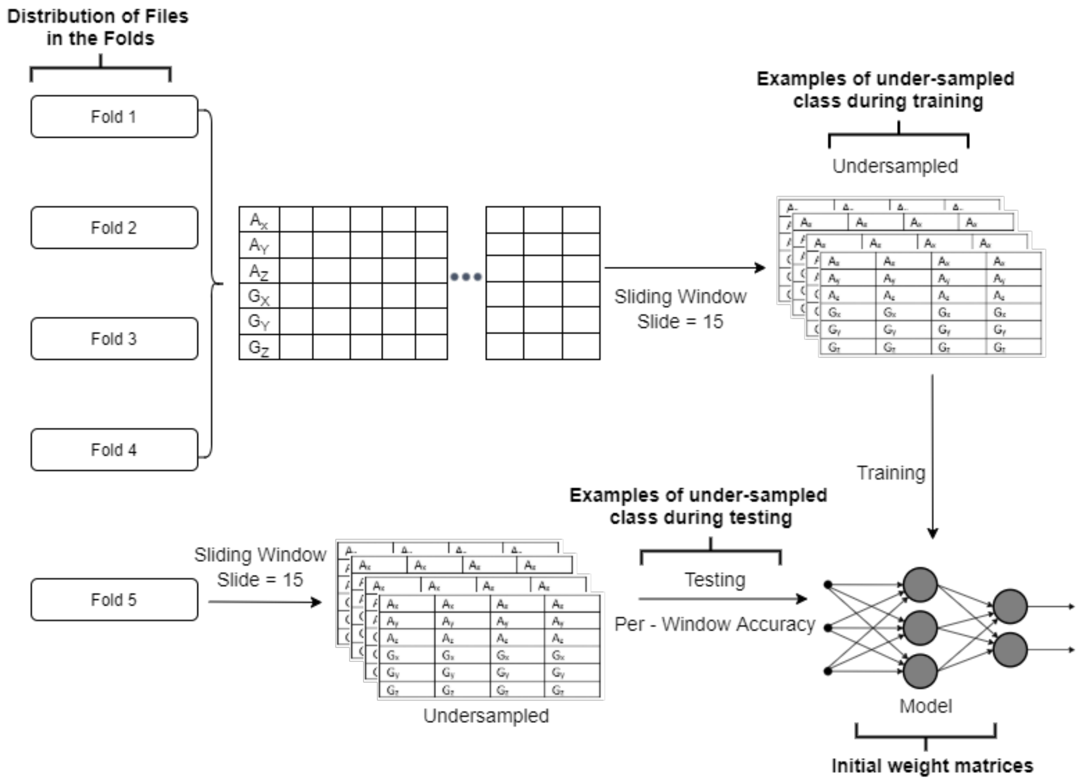
Table 2.2: Per-window testing accuracies for W = 6 min across the 5 folds obtained by Sharma in previous work.

Table 2.2 shows the per-window testing accuracy of the models across the five folds for window size of 6 minutes as achieved by Sharma's experiment. In his work, Sharma saw a difference of up to 9% in the accuracy across the folds. Given the large variability in the magnitudes, we suspect that volatility in the training process might play a key role in the overall model fitting. This in part motivated us to more deeply test the model volatility.

There are a lot of important variables that are determined during run-time which can potentially have a impact on the training process. For example, since we train on a balanced dataset, the choice of windows that are under-sampled from the over-represented class can positively or negatively affect the model. In our case, in the non-eating class, we want there to be a equal distribution of various non-eating actions so the model can better learn to distinguish eating actions from the various non-eating actions. Consider the case where a random under-sampling leads to a training set where the non-eating dataset is composed of similar actions. While the classifier will be able to tell this particular movement apart from eating, it won't be able to generalize well to other movements. The choice of under-sampled windows is not the only source of volatility. The initial weight matrices of the model can also determine how the model trains and converges. Each time we run the model, these matrices can change which could lead the model to converge at a different minima. Distribution of files in the folds is also a contributing factor to volatility. The data collected has inherent bias in the sense of demographics. As mentioned earlier, in section 2.6.3, data for students of the Clemson University was collected first, followed by faculty and staff and then finally local townsfolk. Since we cannot assume that the eating habits are same for each demographics, randomizing the folds can cause a lot of volatility if certain demographics get grouped together. Figure 2.16 marks some of the known sources of volatility during the training and testing process.

In his work, Sharma took a top down approach to detecting windows of eating. He trained neural network models of varying window lengths and analyzed the testing accuracy to determine the optimum window size for meal prediction but he only reported the values of a single run, thus his results do not account for the potential volatility in the training process.

In this work, in an attempt to account for this volatility, we choose to report our results in terms of average $\pm$ standard deviation of the measurement obtained from 10 runs. We define a single run as a complete five fold cross validation training and testing process. Thus, we repeat the entire training and testing process 10 times. During each run, we measure the metrics described earlier in section 2.5, and finally report the averaged value across the runs, along with the standard deviations. By repeating the training process 10 times and allowing for different set of under-sampled indices and



Distribution of Files in the Folds
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
A_X
A_Y
A_Z
G_X
G_Y
G_Z
Sliding Window Slide = 15
Examples of under-sampled class during training
Undersampled
Training
Sliding Window Slide = 15
Examples of under-sampled class during testing
Testing
Per - Window Accuracy
Undersampled
Model
Initial weight matrices

Figure 2.16: Sources of volatility : Various sources of volatility are marked in bold during the stage 1 and stage 2 of the training and testing process.

weight initializations, the averaged value is a more complete measure of the metrics when compared to a single run. It should be noted that all 10 runs are made on the same distribution of files, i.e. the folds remain constant over the runs. This was done to reduce the affect of volatility and make the results more reproducible.

Chapter 3

Results

In this section we present the three main results. Firstly, in section 3.1, we show the results of replicating the original experiment [23] on linear acceleration to confirm its accuracy and establish a new baseline for the comparison with raw acceleration. We also select the hyper-parameters Ts and Te for final metrics calculation. Then, in section 3.2, we show the accuracy of the new method, which uses raw acceleration and select the optimal hyper-parameters for this work. Following this, in section 3.3, we compare the episodic metrics and time metrics of the raw and linear models based on the selected hyper-parameters from the previous sections. Finally, we present some secondary results regarding volatility measure and per-window TPR/TNR of the models. As explained earlier, we also report model volatility in the experiment by reporting each statistic as average $\pm$ std.

3.1 Replication Experiment on Meal Detection

We first show the results of the replication experiment on the CAD dataset. We start by showing the effect of window size on the accuracy of the network on balanced dataset. We then show the effect of Ts and Te on the boundary error, true positive rate and FP/TP ratio and select the optimal value for the parameters.

3.1.1 Effect of Window Size

The figure 3.1 plots the effect of window size on the average testing accuracies obtained from the five fold cross validation. We notice a curve, similar to what was obtained by Sharma in

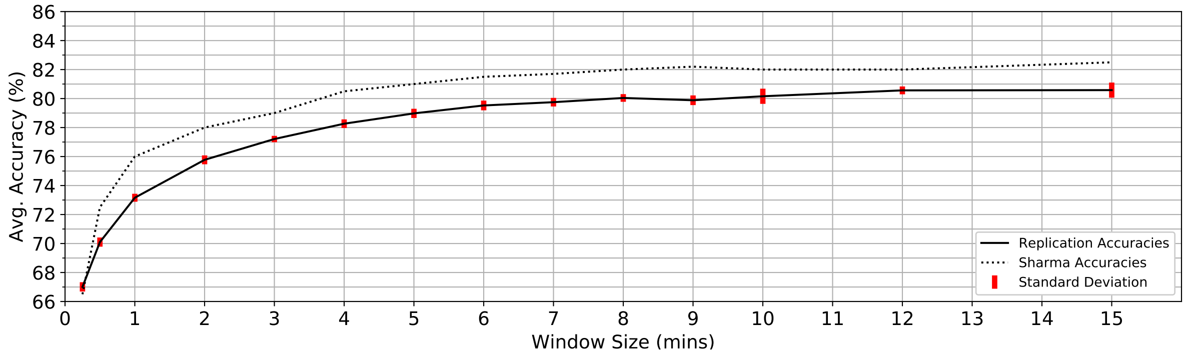


Figure 3.1: Average accuracy (5-fold) of CNN on windows from test split vs window size W, over 10 runs

his work. As the window size grows larger, the testing accuracy also increases. This is likely due to the fact that while smaller windows capture individual ingestion events, larger windows are able to capture continuous patterns of eating which results in a higher accuracy.

Comparing the values with Sharma's work, we see that Sharma's values are higher by 1 - 2%. We suspect that this could be due to a leak between training and testing data. Leakage occurs when certain parts of testing data become available to the model during training, which otherwise would only be limited to testing phase. Contamination of training data could lead to inflated values, giving a false sense of confidence in the model.

Contamination is not the only explainable cause of the higher values for example, in the replication experiment, we average the testing accuracies obtained over the period of 10 runs, thus accounting for volatility in the training phase. Sharma's results, on the other hand, were obtained from a single run. Thus, his numbers could simply be a result of freak training and testing run. However, this idea is invalidated by the standard deviations obtained by repeating the replication experiment 10 times. The standard deviations lie comfortably outside the range of the accuracies obtained by Sharma's experiment, thus, it is unlikely that a single run would achieve the kind of accuracy that his experiment did.

There also exists a difference in the fold creating method used by Sharma and in this experiment. Sharma created the five folds by sorting the files and then dividing them into 5 separate but continuous sets. In this work, we opt for a stratified fold creation method, where we try to distribute the demographics across each fold equally, while still maintaining certain amount of randomness in the distribution of files.

3.1.2 Effect of Ts and Te on FP/TP

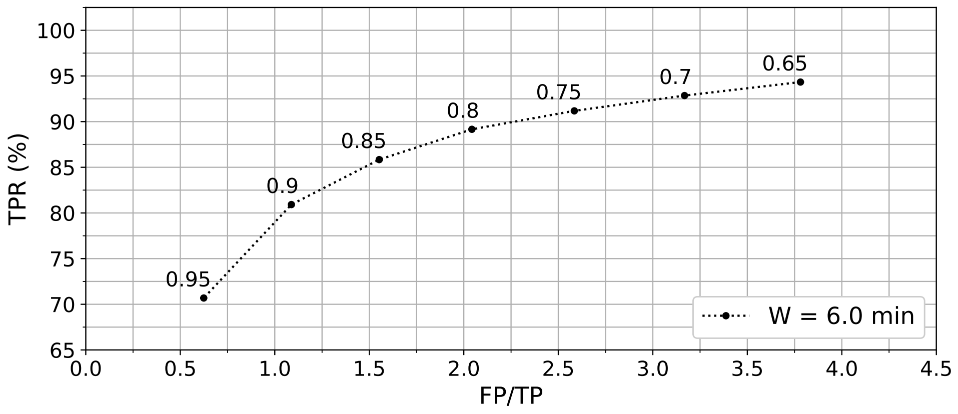


Figure 3.2: Effect of Ts on TPR and FP/TP. The value of Ts is varied while Te is fixed at 0.25.

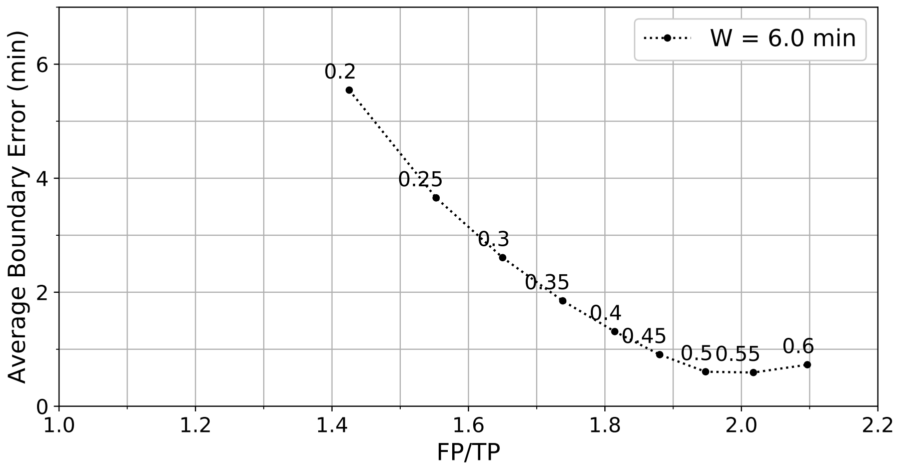


Figure 3.3: Effect of Te on boundary error and FP/TP. The value of Te is varied while Ts is fixed at 0.85.

Figure 3.2 shows the effect of Ts on TPR and FP/TP ratio, with Te held constant at 0.25. As the Ts value is increased, the FP/TP ratio decreases because the model needs to be more confident in order to indicate the start of an eating episode. This makes the model much more selective in terms of eating detection. A low value of Ts tends to have a higher TPR because it allows more meals to be detected but conversely this also increases the FPR of the model. We pick the value $Ts = 0.85$ as the optimal value.

Figure 3.3 shows the effect of varying Te on the average boundary error (ABE) and FP/TP ratio, while keeping Ts fixed at 0.85. We see the trend of decreasing ABE with increasing Te. This trend is consistent with Sharmas work but we do get a lower FP/TP ratio and a higher boundary error. We suggest Te value of 0.3 which provides a balance between average boundary error and FP/TP.

The start boundary error with $Ts = 0.85$ and $Te = 0.3$ is -1.8 ± 0.5 minutes and the end boundary error is +3.3 ± 0.4 minutes. This means that on average, the model predicts that an eating episode starts 1.8 minutes before it actually does and marks the end of an eating episode 3.3 minutes after it actually ends. The average boundary error of the model is 2.6 ± 0.4 minutes, average of the start and end errors.

Work	EA	Subjects	TPR (%)	FP/TP	Dataset
Sharma CNN	1063	351	89	1.7	CAD
Sharma Replication	**1034**	**342**	**86 ± 1.2**	**1.7 ± 0.3**	**CAD - Linear**

Table 3.1: Episodic metrics (avg ± std) on CAD dataset for raw and linear models.

Method	Precision (%)	Recall (%)	TNR (%)	F_1 Score (%)	ACC_W (%)
Sharma CNN	36	69	93	48	80
Sharma Replication	**37 ± 1.7**	**72 ± 1.2**	**90 ± 1**	**49 ± 1.3**	**79 ± 0.5**

Table 3.2: Time metrics (avg ± std) on CAD dataset for raw and linear models.

Tables 3.1 and 3.2 show the performance of the model on episodic and time metrics (average ± standard deviation) respectively. Statistics of other works have been shown for completeness. From the 1,034 meals, we detected 886 ± 13, missed 153 ± 13 and had 1,329 ± 275 false positives. This yields a TPR of 86% ± 1.2% and a FP/TP ratio of 1.7 ± 0.3. When compared to the original work, we see an average 3% decrease in the TPR. The FP/TP remains consistent between the original and the replication experiment, with 1.7 FPs per every TP.

In case of time metrics, the replication experiment outperformed the precision and the recall of the original experiment as well, improving the precision by 1% and the recall score by 3%. We did see a decrease in the TNR by 3%. The F1 score and weighted accuracy were found to be comparable.

Thus, we conclude that Sharmas results were slightly inflated due to a mix of contamination of testing data and only reporting the best run. In this section, we replicated the experiment and setup a new baseline with which we can compare any future work.

3.2 Raw Acceleration

In this section, we detail the results of model training and testing using raw acceleration and gyroscope data. We first report the window accuracy of the models and then do a hyper-parameter search to find the optimal parameter value for Ts and Te.

3.2.1 Effect of Window Size

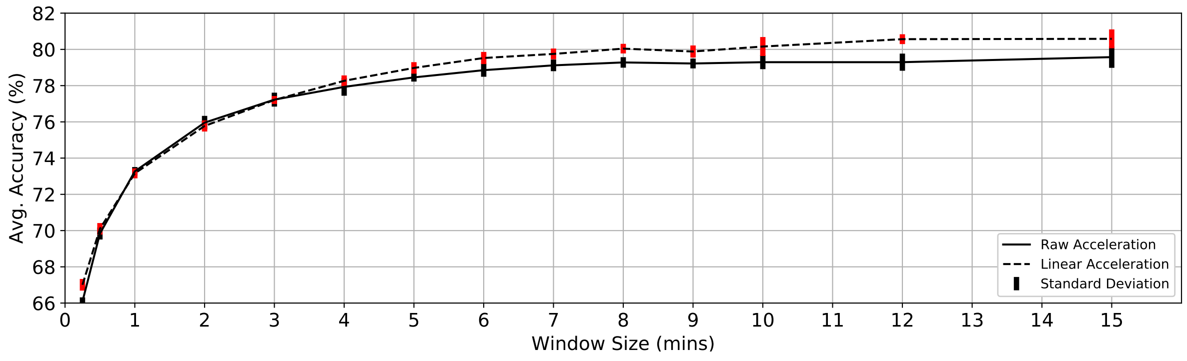


Figure 3.4: Testing accuracy of raw acc. model (solid line) vs linear acc. model (dotted line).

The first thing we notice is that the models trained on raw acceleration are able to distinguish between eating and non-eating windows. Though, when tested on a balanced dataset, the linear models outperform raw models on window lengths larger than 4 minutes. Similar to linear acceleration, windows larger than 6 minutes do not increase accuracy significantly. Thus we use a W = 6 min for measuring the various other statistics.

3.2.2 Effect of Ts and Te on FP/TP

Figure 3.5 shows that while smaller values of Ts yield a larger TPR rate, the FP/TP ratio also increases accordingly. The value of 0.85 maximizes the TPR rate while also maintaining a small FP/TP ratio. Thus, we recommend a Ts value of 0.85 for the raw acceleration.

Figure 3.6 shows the effect of varying Te while keeping Ts fixed at 0.85. We see a common trend of decreasing boundary error and increasing FP/TP ratio as the value of Te increases. While there is no clear knee in the graph, the value of $Te = 0.3$ maintains a low boundary error and a small FP/TP ratio. Thus, we recommend Te value of 0.3 for raw acceleration.

The start boundary error (average ± standard deviation) with $Ts = 0.85$ and $Te = 0.3$ is -1.6 ± 0.2 minutes and the end boundary error is +2.05 ± 0.4 minutes. This means that on average, the model predicts that an eating episode starts 1.6 minutes before it actually does and marks the end of an eating episode 2.05 minutes after it actually ends. The average boundary error of the model is 1.7 ± 0.3 minutes, average of the start and end errors.

Using the recommended Ts value of 0.85 and Te value of 0.3, the models are able to identify (average ± standard deviation across 10 runs) 869 ± 13 meals and miss 165 ± 13. They also trigger a total of 1470 ± 218 FP. The result is a TPR of 84% ± 1.3% and a FP/TP ratio of 1.7 ± 0.2.

3.3 Comparison of Raw and Linear models

Method	EA	Subjects	TPR (%)	FP/TP	Dataset
Linear	**1034**	**342**	**86 ± 1.2**	**1.7 ± 0.3**	**CAD - Linear**
Raw	**1034**	**342**	**84 ± 1.3**	**1.7 ± 0.2**	**CAD - Raw**

Table 3.3: Episodic metrics on CAD dataset for raw and linear models.

Method	Precision (%)	Recall (%)	TNR (%)	F_1 Score (%)	ACC_W (%)
Linear	**37 ± 1.7**	**72 ± 1.2**	**90 ± 1**	**49 ± 1.3**	**79 ± 0.5**
Raw	**38 ± 2.1**	**69 ± 1**	**92 ± 1.4**	**49 ± 1.6**	**78 ± 0.4**

Table 3.4: Time metrics on CAD dataset for raw and linear models.

Table 3.3 shows the episodic metrics statistics for the replication experiment and the experiment with raw acceleration. In case of episodic metrics, linear acceleration performs better than raw acceleration. Linear acceleration has a 2% higher TPR rate (on average). Since the replication model has a lower per-window accuracy, the episodic metrics are also lower than the original work. The FP/TP is similar across both raw and linear acceleration.

In case of time metrics (table 3.4), the linear acceleration slightly outperformed raw acceleration. We see a mix of results with linear acceleration having a roughly 3% higher recall and a 1%

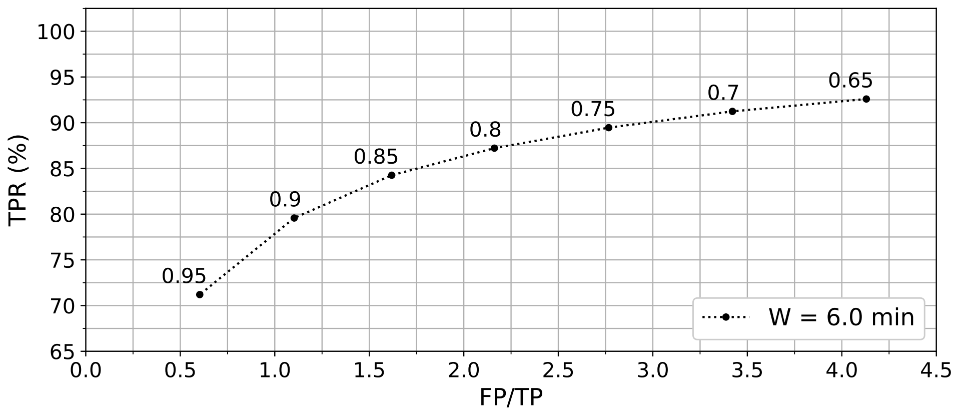


Figure 3.5: Effect of Ts on TPR and FP/TP. The value of Ts is varied while Te is fixed at 0.25.

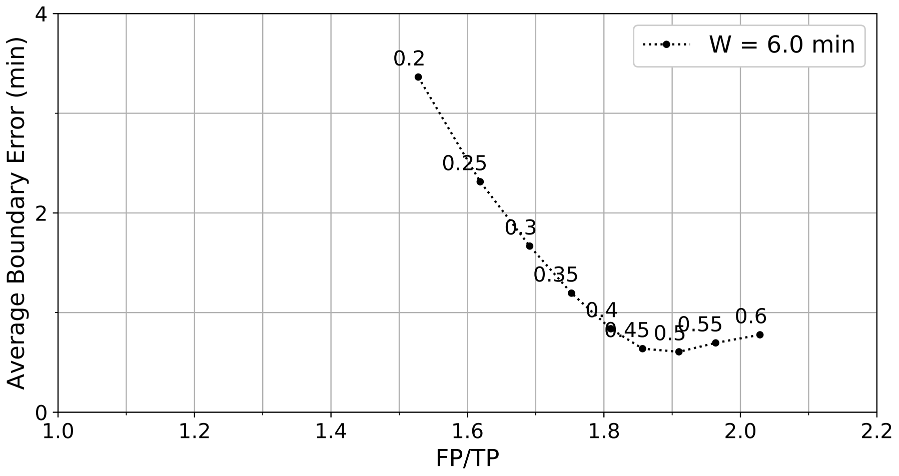


Figure 3.6: Effect of Te on boundary error and FP/TP. The value of Te is varied while Ts is fixed at 0.85.

higher weighted accuracy score than the raw acceleration. The F1 scores of both raw and linear were similar but raw acceleration had a 2% higher TNR. Thus, based on the episodic and time metrics, while both raw and linear perform very similar to each other, overall, linear beats raw by a very thin margin.

Finally, in case of boundary errors, we see a lower average boundary error (ABE) with raw acceleration as compared to linear acceleration. Using the optimum values of Ts and Te, the ABE in linear acceleration was 2.6 ± 0.4 minutes while in raw acceleration, the ABE was 1.7 ± 0.3 minutes. Thus, raw acceleration resulted in lower boundary errors than linear acceleration.

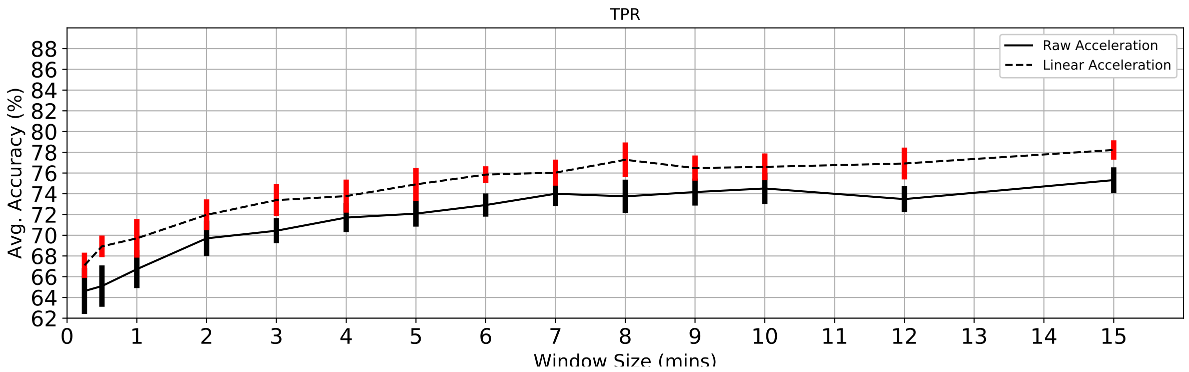


Figure 3.7: Per Window TPR of the raw and linear models. The linear models have a higher window TPR than the raw models.

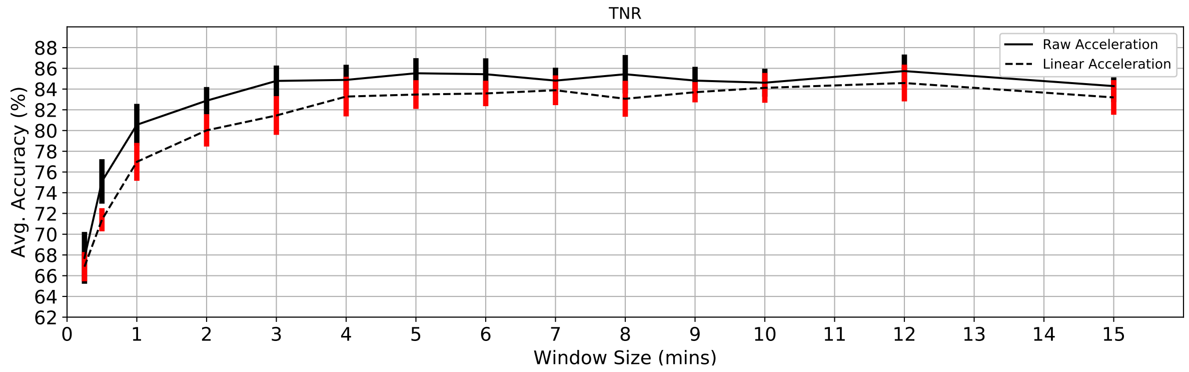


Figure 3.8: Per Window TNR of the raw and linear models. The raw models have a higher window TNR than the linear models.

The per-window accuracy, depicted in figure 3.4 is a mix of the TNR and TPR of the raw and linear models. When TPR of the model increases, the TNR decreases. During the training process, the model attemps to find the best balance of TNR and TPR that maximizes the average accuracy on the validation data.

Comparing the per-window TPR (figure 3.7) and TNR (figure 3.8) of the raw and linear models, we find that the raw models have a higher TNR than linear models. Raw models are more likely to correctly predict a window as non-eating than linear models. Linear models, on the other hand, have a higher TPR rate than the raw models. It should be noted that the difference in the TPR of linear and raw models is larger than the difference in the TNR of the models.

Furthermore, TNR of both raw and linear models reaches a higher value than the TPR. This means that it is easier to recognize non-eating actions than it is to recognize eating actions. This could simply be because non-eating occurs 20 more times than eating so models learn to identify that behaviour better than eating. This could also be a result of the higher variability in non-eating actions as compared to eating actions.

Figures 3.9 and 3.10 show the distribution of the length of the meals as well as the number of meals detected by raw and linear models respectively. The values are averaged over the 10 runs and rounded to the nearest integer. We notice that both raw and linear had similar distributions of detected meals. Linear outperforms raw models in identifying meals between 4-10 minutes. Both models have almost the same number of identified meals for eating episodes that last longer than 10 minutes.

Finally, we look at the volatility across per window accuracy in the models trained by raw and linear acceleration across the five folds. Figure 3.11 and 3.12 plot the accuracies of the five folds vs the window size. The spread of the accuracies over the folds is very similar in both cases. Fold 1 has the lowest accuracy while fold 5 generally outperforms all the other folds.

Furthermore, we noticed that there was a higher volatility across the folds as compared to the volatility across the runs. Moreover, since the testing accuracies vary so wildly over the folds, this implies that in order to train a network on wrist movements, we require all the five folds. Training a model using only a single fold can result in both over or under confident model, depending on the files in the fold. When we compare the distribution of the accuracies in the linear model with Sharma's work, we see that the variation of the accuracies is smaller in the replication work. This is likely because of striped fold creation where we distribute the demographics equally. Thus, another factor in accuracy distribution is the fold creation method.

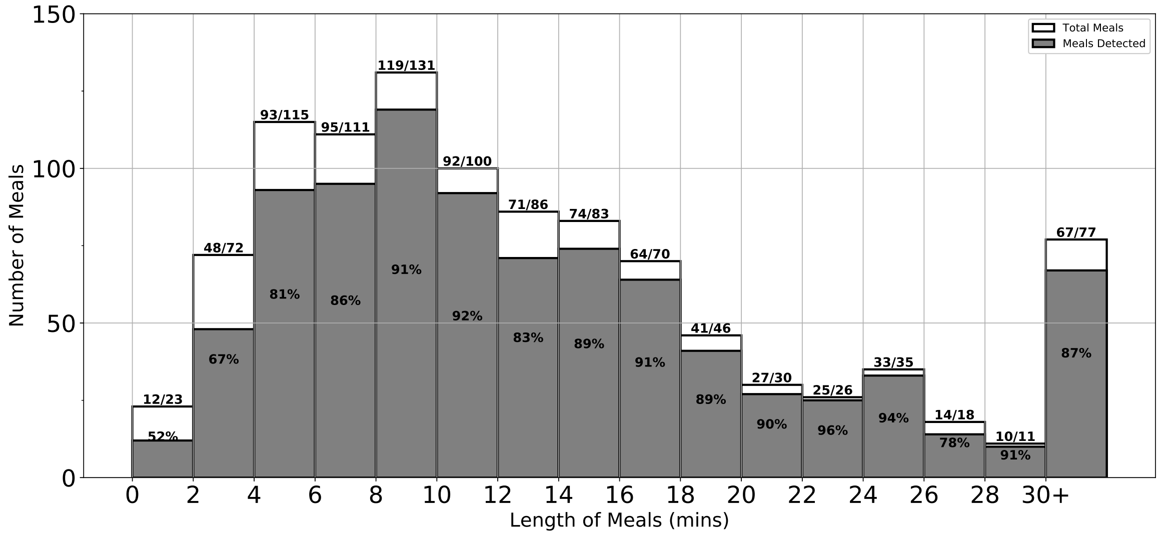


Figure 3.9: Distribution of meals detected by linear model.

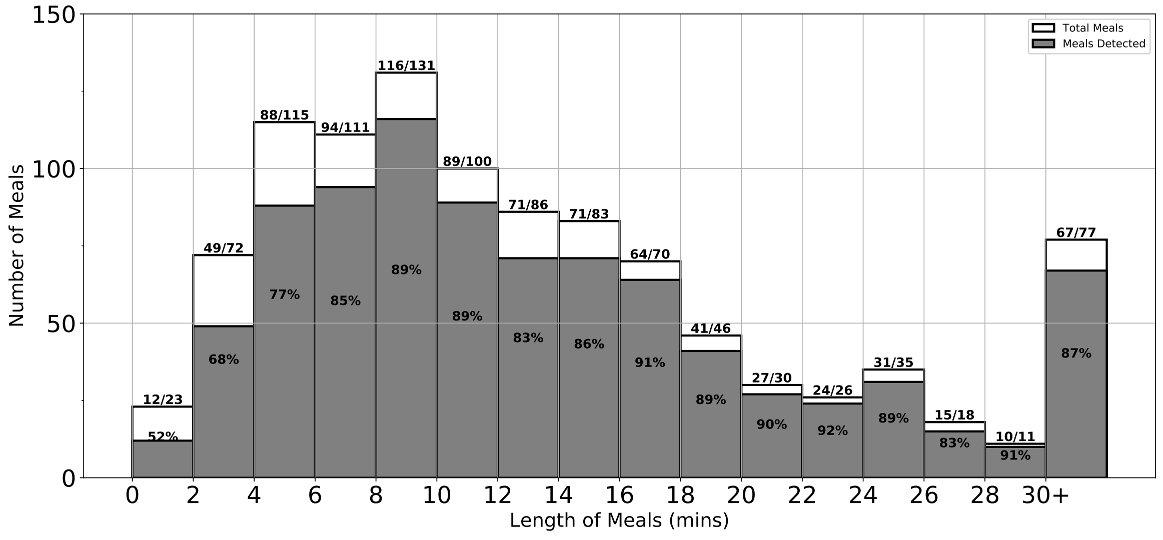


Figure 3.10: Distribution of meals detected by raw model.

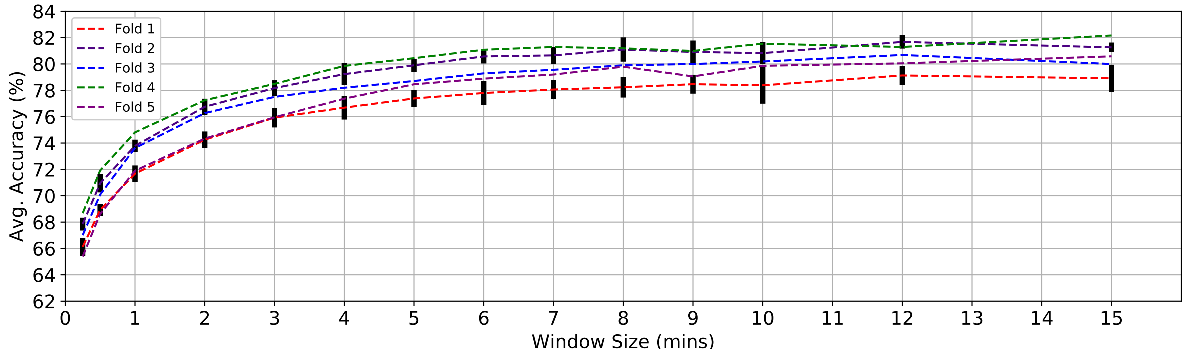


Figure 3.11: Accuracy and standard deviations per fold for linear acceleration models.

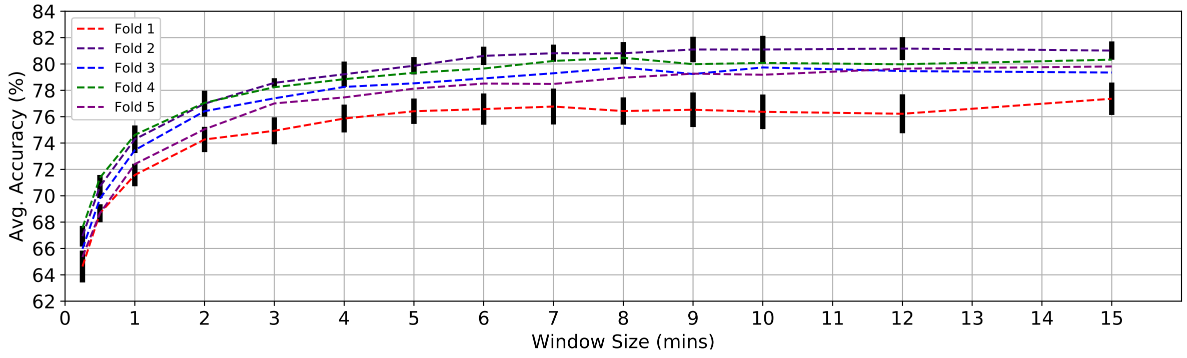


Figure 3.12: Accuracy and standard deviations per fold for raw acceleration models.

Chapter 4

Conclusions and Future Work

4.1 Conclusions

In this work we considered the problem of detecting eating episodes by tracking wrist motion. The motion of the wrist can be characterized by using accelerometer and a gyroscope, giving us the information of movement and tilt respectively. There are two kinds of acceleration related to an accelerometer, linear and raw. In this thesis, we compare two sets of models, one trained on linear acceleration and gyroscopic data and the other trained on raw acceleration and gyroscope data to determine if raw acceleration can outperform or replace linear acceleration as a variable.

The very first question that we posed was whether or not the work done previously by our group, on linear acceleration, can be replicated? The first part of the results confirms that the results of the previous experiment on the CAD dataset can be replicated. With our replication work, we established a new baseline for comparison and found that the combination of linear acceleration and gyroscope data is able to detect (average $\pm$ standard deviation across 10 runs) 86% $\pm$ 1.2% of the total meals with a FP/TP ratio of 1.7 $\pm$ 0.3 which equals to roughly 5 FP a day. We found that the per-window testing accuracy of our model is 1% - 2% lower than the previous work. The difference could be attributed to a leak in training and testing data, which lead to inflated values. In case of time and episodic metrics, Sharma only reported the best results of multiple runs, and since our values are within 3 standard deviations of his values, we have reason to believe that his results are the upper end of the metrics distribution

Secondly, we posed the question, can the results of the linear acceleration be matched by using raw acceleration? We found that raw acceleration tends to perform slightly worse than linear acceleration. The raw model achieved a TPR of 84% ± 1.3 % with a FP/TP ratio of 1.7 ± 0.2. The raw model does achieve a higher TNR rate on the time metrics, beating linear by 2%. The linear outperforms raw with a higher TPR on time metrics, beating raw by 3%. This was consistent with the per-window accuracies where we noted that the raw model achieved a higher per-window TNR and a lower per-window TPR than the linear model. Furthermore, we also found that in free - living datasets, we need large amounts of data to account for the volatility in the training process. Training on any single fold, which contain up to 71 files - a relatively large number in itself, will not be enough and could lead to inflated or deflated metric measurements. Finally, we suspected that the low signal-to-noise ratio in linear acceleration may be a significant contributor to model volatility. However, training the model with raw acceleration, which has a much higher signal-to-noise ratio, did not reduce model volatility. Instead, both models showed approximately equal volatility across 10 train-and-test runs. We therefore conclude that sensor noise is not the cause.

Finally, we conducted a space search for hyper-parameters, Ts and Te, for both raw and linear models. We found that, for both, raw and linear models, $Ts = 0.85$ and $Te = 0.3$ are the most optimal values. Tuning these hyper parameters was also important because the various statistics that we calculated, detailed in the results section, were dependent upon the choice of the hyper-parameters.

Thus, in conclusion, we demonstrated that linear acceleration seems to out-perform raw acceleration by a slight margin.

4.2 Limitations and Future Work

A limitation to this work is that we used the same model to train both linear and raw models. Since linear and raw acceleration are two different variables, we cannot reasonably expect the same model to perfectly fit both data types. Moreover, considering the minor gap between the linear and raw model metrics, raw acceleration could potentially match or outperform linear acceleration if a better model, that is designed for raw acceleration can be developed. One approach to this would be to look for deeper networks because the information that we want the model to extract and learn from raw acceleration is hidden behind the gravitational component.

Secondly, more testing needs to be done across different datasets with data from devices other than Shimmer device. In this work, we only looked at the difference in raw and linear acceleration across the CAD dataset which is collected from the Shimmer device. Since linear acceleration requires pose estimation, the quality of linear acceleration magnitude will depend on how good the device that estimates the pose is. On the other hand, raw acceleration readings are directly taken from the accelerometers and the magnitude of noise in the readings across devices is less likely to affect the reading because of the higher magnitude of raw acceleration. Thus, raw acceleration could potentially work more consistently across devices with different chips as compared to linear acceleration.

Thirdly, the difference between the measured metrics of linear and raw isn't very large. While linear outperforms raw acceleration, in this dataset, more testing needs to be done in real time, online system to determine whether the conversion from raw to linear is worth the costs.

Fourthly, we believe that a major source of volatility is the file distribution. Since, in this work we keep the file distribution across folds consistent for reproduce-ability, the volatility we see is here is unique to the given folds. More work needs to be done to measure the volatility across file distribution by varying the files across the folds and increasing the number of runs.

Finally, there are known errors in ground truth collected from the participants. For example, we do not consider the affect of pre-meal snacking and post-meal snacking on the model. Participants may have lightly snacked outside what they perceived as the start and end times of their meals. Thus, any snacks during pre and post meals are registered in the ground truth as non-eating. These obviously confuse the model and are likely to reduce the accuracy. Thus, more research needs to go into how to identify and ignore/include the snacks during the data collection.

Bibliography

[1] S. Adibi. *mHealth Multidisciplinary Verticals*. CRC Press, Melbourne, 2014.

[2] L. Burke, J. Wang, and M. Sevick. Self-monitoring in weight loss: A systematic review of the literature. *Journal of the American Dietetic Association*, 111(1):92–102, 2011.

[3] Y. Dong, J. Scisco, M. Wilson, E. Muth, , and A. Hoover. Detecting periods of eating during free-living by tracking wrist motion. *IEEE Journal of Biomedical and Health Informatics*, 18(4):1253 – 1260, 2014.

[4] M. Farooq and E. Sazonov. Detection of chewing from piezoelectric film sensor signals using ensemble classifiers. *38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 33(7):4929 – 4932, 2016.

[5] R. Franckle, J. Block, and C. Roberto. Calorie underestimation when buying high-calorie beverages in fast-food contexts. *American Journal of Public Health*, 106(7):1254–1255, 2016.

[6] C. Hales, M. Carroll, C. Fryar, and C. Ogden. Prevalence of obesity among adults and youth: United states, 2015–2016. *NCHS data brief*, (288), 2017.

[7] C. Hales, M. Carroll, C. Fryar, and C. Ogden. Prevalence of obesity and severe obesity among adults: United states, 2017–2018. *NCHS data brief*, (360), 2020.

[8] K. Hamrick, M. Andrews, J. Guthrie, D. Hopkins, and K. McClelland. How much time do americans spend on food? (1476-2019-2786):64, 2011.

[9] H. Heydarian, M. Adam, T. Burrows, C. Collins, and M. Rollo. Assessing eating behaviour using upper limb mounted motion sensors: A systematic review. *Nutrients*, 11(5):1168, 2019.

[10] C. Hirt, S. Claessens, T. Fecher, M. Kuhn, R. Pail, and M. Rexer. New ultrahigh-resolution picture of earth's gravity. *Geophysical Research Letters*, 40(16):4279 – 4283, 2013.

[11] A. Hruby and F. Hu. The epidemiology of obesity: A big picture. *PharmacoEconomics*, 33(7):673–89, 2015.

[12] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[13] S. Klein, L. Fontana, V. L. Young, A. Coggan, C. Kilo, B. Patterson, , and B. Mohammed. Absence of an elect of liposuction on insulin action and risk factors for coronary heart disease. *New England Journal of Medicine*, 350(25):2549–2557, 2004.

[14] K. Kyritsis, C. Diou, and A. Delopoulos. Modeling wrist micromovements to measure in-meal eating behavior from inertial sensor data. *IEEE journal of biomedical and health informatics*, 2019.

[15] K. Kyritsis, C. Diou, and A. Delopoulos. A data driven end-to-end approach for in-the-wild monitoring of eating behavior using smartwatches. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):85, 2020.

[16] C. Lavie, R. Milani, and H. Ventura. Obesity and cardiovascular disease: risk factor, paradox, and impact of weight loss. *Journal of the American college of cardiology*, 53(21):1925–1932, 2009.

[17] Realtime Technologies Ltd. Shimmer user manual. 2017.

[18] M. Mirtchouk, D. Lustig, A. Smith, I. Ching, M. Zheng, and S. Kleinberg. Recognizing eating from body-worn sensors: Combining free-living and laboratory data. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):85, 2017.

[19] S. Blair R. Anderson. Encouraging patients to become more physically active: the physician's role. *Annals of Internal Medicine*, 127(5):395–400, 1997.

[20] R. Ramos-Garcia, E. Muth, J. Gowdy, and A. Hoover. Improving the recognition of eating gestures using intergesture sequential dependencies. *IEEE Journal of Biomedical and Health Informatics*, 19(3):825–831, 2015.

[21] L. Rici, F. Taffoni, and D. Formica. On the orientation error of imu: Investigating static and dynamic accuracy targeting human motion. *PLOS ONE*, 11(9):1–15, 9 2016.

[22] M. Senthilingam. Covid-19 has made the obesity epidemic worse, but failed to ignite enough action. *BMJ*, 372(411), 2021.

[23] S. Sharma. *Detecting periods of eating in everyday life by tracking wrist motion - What is a meal?* PhD thesis, Clemson University, 2020.

[24] 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*, 21(3):599–606, 2017.

[25] E. Swanson. Prospective clinical study of 551 cases of liposuction and abdominoplasty performed individually and in combination. *Plast Reconstr Surg Glob Open.*, 1(5), 2013.

[26] United Kingdom National Health Service. Obesity - causes, 2019.

[27] Y. Wang, Beydoun, L. Liang, B. Caballero, and S. Kumanyika. Will all americans become overweight or obese? estimating the progression and cost of the us obesity epidemic. *Obesity (Silver Spring, Md.)*, 16(10):2323–30, 2008.

[28] W. Wei. Individualized wrist motion models for detecting eating episodes using deep learning. Master's thesis, Clemson University, 2021.

[29] Y. Yadav. Obesity : A modern day plague. *Medical journal, Armed Forces India*, 58(1):60–65, 2002.