Identifying Noisy Labels in the Ground Truth of Eating Episodes Self-Reported by Button Press on a Wrist-worn Device

A Thesis
Presented to
the Graduate School of
Clemson University

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

by
Tianyi Zhang
May 2022

Accepted by:
Dr. Adam Hoover, Committee Chair
Dr. Richard Groff
Dr. Harlan B. Russell

Abstract

This thesis considers the problem of identifying noisy labels in the ground truth of eating episodes (meals, snacks) as self-reported by participants collecting data in the wild. Participants wore a smartwatch-like device that tracked their wrist motion all day. They were instructed to press a button on the device at the start and end of each eating episode. The device and instructions were designed to be as simple to use as possible, but post-review of the ground truth provided by participants revealed a strong likelihood that a significant portion of the button presses may contain errors. For example, an error could be caused by a participant forgetting to press the button until halfway through a meal, thus misidentifying the start boundary. This thesis seeks to determine if these types of errors can be identified with confidence, how often they occurred, and if they can be fixed.

The correctness of the ground truth is important because it is used to generate labels for the wrist motion data to train a classifier to detect eating. If the button presses have errors, then data will be mislabeled, which will diminish classifier performance. The problem of noisy labels is well-known in other domains, such as image segmentation, where it is expected that a certain amount of pixels or images have been mislabeled. In the domain of wearable devices used to monitor human behavior or health, the concept of noisy labels is relatively new and less work has been done. In particular, this is the first work to consider the challenge of identifying errors in the identification of start and end times for eating episodes.

The data used for this work is the Clemson all-day (CAD) data set. It contains 354 days of wrist motion data from 351 different participants. The total length of the data set is 4,680 hours with 1,133 meals indicated by 2,266 button presses (start and end for each meal). This data was used previously to develop a classifier that outputs a continuous probability of eating P(E) all day for each recording. In this work, we visually compare the P(E) plot against the ground truth button presses reported by participants. This comparison highlights intervals where the classifier disagrees with the ground truth. We developed a schema for quantifying these disagreements, and had three raters independently use it to assess and modify the ground truth for 71 days of data. Two raters achieved an agreement of 79% on the adjustments, while three raters achieved an agreement of 64%.

To further test the viability of identifying and updating the ground truth, all 354 days of data were reviewed and adjusted by a single rater. This updated ground truth was then used to retrain the wrist motion classifier. Its performance was evaluated on the original unadjusted ground truth in order to prevent bias. When trained on the original ground truth, the classifier achieved a per-datum weighted accuracy of 79.1%, an episode true positive rate (TPR) of 87.5%, and a false positive to true positive ratio (FP/TP) of 1.9. When trained on the adjusted ground truth, the classifier achieved a per-datum weighted accuracy of 80.1%, an episode TPR of 85.9%, and a FP/TP of 1.7. These results indicate that adjusting the ground truth yielded a 1% improvement in weighted accuracy, and a decrease in the detection of false positive episodes, but also a decrease in the detection of true positive episodes.

Collectively, these results indicate that it is possible to identify button press errors in the ground truth of data used to train a wearable device for detecting eating. However, our methods also need improvement in order to obtain a higher inter-rater reliability, which would potentially yield additional improvement in classifier performance.

Acknowledgments

First of all, I want to thank Dr. Adam Hoover for his continuous guidance and patience throughout this research topic. From formulating methodology to professional writing, Dr.Hoover has taught me a lot, without which I would not be able to complete this work.

I would also like to thank Dr. Richard Groff and Dr. Harlan Russell for serving on my committee and listening to my thesis defense. I want to further extend my gratitude to the Clemson faculty, professors and friends who have helped me to shape my academic life and skills in Clemson University. Without the help, I would not be able to acquire the skill and knowledge that lead me through the graduate life.

Finally, I express absolute gratefulness to my family who has worked very hard to support me through my college life. I would also like to express gratitude to other friends who have provided guidance and help outside the university. Without you, I will not be able to progress to this point.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis considers the problem of identifying noisy labels from self-reported data collected from the wild. Previously, our group has collected data from 351 subjects, which is called Clemson all-day data set (CAD), and has trained a deep learning classifier based on the CAD data set. Since the data set is collected from the wild and relies on self-reported meal periods, the classifier suffers from the noisy labels. The motivation of this work is to provide a solution to safely identify the noisy labels in the CAD data set, based on the knowledge of self-reported meal periods, classifier detected segments and output eating probability distribution P(E). Noisy labels could happen more than expected during the data collection stage, especially for a data set collected from the wild. A meal interval could be marked incorrectly or inaccurately due to subjective errors. For example, subjects may misunderstand the instructions and mark a meal event before the meal intake starts, causing a period of erroneous labels. Also, subjects may forget to mark an entire eating period, or mistakenly mark a eating period due to device malfunction. Even more, a subject may have a meal period that is dominated by secondary activities such as cell phone usage, and socializing, in which case, a meal may exhibit features closer to normal days activities rather than decent meal. Unlike natural noise which is random and does not cause large bias if the data set is large enough, the noisy labels in the CAD data set introduces large bias and could make the classifier training derail from the true optimal, and lead to poor performance of the final model.

By identifying and adjusting noisy labels, the data set could contain less noise, and more likely to lead the model converging to the correct optimal, providing better performance in classification. This work will use the previous data collected from 351 subjects. Each subject has approximately 1 day of wrist motion data collected. This work will also use the previous classifier to perform the classification, and utilize the P(E) of the classifier to safely identify noisy labels. In addition, we explore the improvements we could have by fixing the noisy labels. The challenge of the CAD data set is that no hard ground truth is provided other than subjects self-reported time marks. This work attempts to adjust noisy labels under the context of classifier P(E) and detected segments. We start the experiments by adjusting individual datum label during the classifier training process, and eventually propose a schema and an algorithm to perform label adjustment on a contiguous sequence of data. We will first measure the reliability of our schema, then analyze the prevalence of label errors in the CAD data set and compare the performance of the same classifier trained on the original data set and the adjusted data set. The rest of this chapter will provide necessary background information related to this work. Section 1.1 provides background information about the motivation, technology used for data collection, neural network related knowledge and denoising. Followed by section 1.2, we introduce the noisy labels related works, ranging from wearable sensor data to other domain such as image classification.

1.1 Background

1.1.1 Obesity

Obesity is a health problem caused by excessive body fat and can raise the risk of many severe illness, such as type II diabetes, cancers, and heart diseases. Obesity is defined by body mass index (BMI). According to the U.S. According to Centers for Disease Control and Prevention (CDC), for obesity, the BMI is defined to be over 30. In U.S., the prevalence of obesity remains at a increasing trend. Based on report from CDC, from 1999 to 2018, the obesity prevalence has increased from 30.5% to 42.4%. Specifically, the age group from 40 to 59 has the highest obesity prevalence of 46.4%. Childhood obesity is also notable. In 2017-2018, childhood prevalence obesity has achieved 19.3%, affecting 14.4 million children and adolescents nationwide [14]. Moreover, obesity could persist over chronological periods. According to a recent research, obese children and obese adolescence are likely to become obese adults [32, 33]. Most children who suffer from the obesity are likely to bring it to their adulthood as well.

Obesity is in general caused by an imbalance in physical activity (energy expenditure) and consumption (energy intake). On a large scale, multiple studies have demonstrated that losing weight and a balanced nutrients intake could effectively reduce obesity, and bring medical and economical gain [6, 11, 12, 26, 27]. Regulations from government and family level could also help to reduce the cases of childhood obesity [7, 9]. Study points out that, humans have weak psychological against weight gain when food is abundant. Thus, in today's world where food supply is superb, people need self-motivated mechanism to prevent themselves from obesity [17]. Moreover, for people who want to loss weight and reduce obesity, not only a dietary is needed but also a form of supervision is needed so that the people could enforce their plan and guarantee the success in battling obesity [24].

Although methods to prevent or lose obesity are available, few people achieve the goal. One major cause for most people who cannot get away from obesity is lacking a method to keep track of their daily intakes. Keeping track of calorie intakes could be a time-consuming and daunting task. It could be so overwhelming that people could easily give up during the process. For example, a person may be willing to write down calories for each major meal, however, to ask a person to record everything he or she eats on a day will become a burden. Dietary intervention requires a long period of recording, few people has the time and energy to do that. To solve this problem, mobile health or mHealth is being adapted. The mHealth enables people to automatically track their calorie intake by specific digital devices and we will discuss it in the section 1.1.2.

1.1.2 mHealth

By definition, mobile health or mHealth is a method that uses wireless or mobile technologies to achieve health objectives [19]. Since mobile and wireless technology have a wide coverage worldwide, by adpating the concept of the mHealth, countries and nations could provide more accessibility to basic health status monitoring to wider population [25]. According to World Health Organization (WHO) reports from 2011, on the global scope, more than 80% of the countries has adapted at least one mHealth initiative [19]. More and more countries are adapting similar strategies to improve their health conditions. Moreover, mHealth does not only make the scope larger but also collect subject health information more accurately. Physicians also argue that mHealth could improve diagnose efficiency and accuracy by collecting immediate data when subject suffers a health risk [28]. All the benefits mentioned above has made mHealth a worthy field to be investigated.

In recent years, some mHealth device and applications, such as smartwatch and various applications on mobile phone, have been developed. For example, iphone has the first FDA approved electrocardiogram (ECG) sensor embedded in the device, which is able to monitor heart rate [16]. In

Figure 1.1: Example of Apple Watch health tracker. The GUI on the watch screen displays heart beat rate and health advice on physical activities.

figure 1.1, Apple Watch uses the embedded sensors in iphone to collect cardinal health status of the human heart from user's daily activities. The result is then analyzed and displayed by the health app installed on iphone. Not only Apple Watch has the ability to monitor health, but also other wrist-worn devices are capable of capturing useful positional information from daily activities. Such information, combining with machine learning techniques, can produce many useful health analysis, such as calorie intakes, and provide health suggestions according to personal needs.

1.1.3 Shimmer3 Device

The data set used in this thesis is collected by the Shimmer3 device [29]. Shimmer3 device is a wrist-worn wireless device that performs motion capture from the wrist motions. As a inertia measurement instrument (IMU), Shimmer3 device possesses kinematic components to measure positional changes, such as the accelerometer, gyroscope, and magnetometer [4, 13]. In CAD data set, our group has recorded data from accelerometer and gyroscope (magnetometer is not used) [29]. In other words, we will use the angular accelerations and linear accelerations on 3 axes-x, y, z recorded by the Shimmer3 device, for further data analysis.

Figure 1.2 demonstrates the Shimmer3 device used to collect the data. Our group has programmed the device to operate on 15 Hz or 15 data points per second. After turning on the device by the side switch, the device will start to record motion data happened on the accelerometer and the gyroscope. The big orange button on the front of the device is then used to mark the event start and end. In the data collection, subjects are asked to turn on the device in the morning.

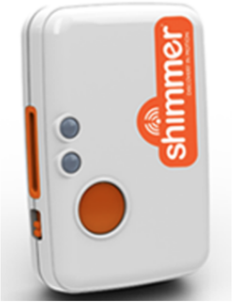


Figure 1.2: A Shimmer3 device for data collection. The device has an orange switch on the side and an orange button for starting the record or ending the record.

Whenever they eat a meal or snack, subjects are instructed to press the circle orange button at the meal start and press the button again when they finish eating. Shimmer3 device also provides an interface-ConsenSys to export the recorded data in the CSV format. The output file from Shimmer3 device contains a total of 6 axes and approximately 15 data point per second. The 6 axes include 3 axes-x, y, z on gyroscope and accelerometer respectively. Further details of gyroscope and accelerometer will be introduced in the 1.1.4 and 1.1.5 sections.

1.1.4 Accelerometer

Accelerometer is a motion sensor that is widely used in multiple disciplines. Specifically, in physical activities assessment, accelerometer is able to capture enough of the motions of the human daily activities with the minimal discomfort to the subject [35]. Structurally, accelerometer is built by damped spring and use Hook's Law to calculate the acceleration. Note that, instead measuring a absolute acceleration, accelerometer measures the relative acceleration or the proper acceleration of the object.

In figure 1.3, a traditional structure of accelerometer is shown in the graph. The traditional accelerometer utilizes the Hook's Law. When the whole setting is moved, the change of the inertia in the "Mass" object is converted to the shape change of the spring. In other words, when the whole setting moves, the "Mass" object will squeeze the spring on one side. According to Hook's Law:

$$F = -kx \tag{1.1}$$

The force F is related to the spring constant (material property) k and the spring displacement x.

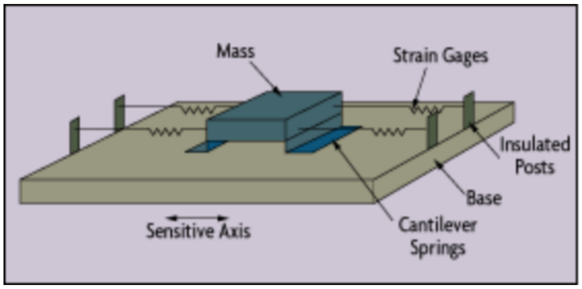


Figure 1.3: A demonstration of Hook's Law on a mechanical accelerometer. The spring connecting the mass and the insulated posts is used to transform the acceleration on the mass into measurable shape-change.

The Hook's law gives an idea of how we could measure the change in inertia or acceleration by spring displacement. For modern accelerometer, the Micro Electro Mechanical System (MEMS) is used to minimize the mechanical and electrical components onto a single chip. In modern accelerometer, the mass will squeeze the piezoelectric material which produces an electrical charge that is proportional to the force exerted upon it [1]. The final ouput of the accelerometer is in meters per second square (m/s^2), or in standard gravity g.

1.1.5 Gyroscope

The gyroscope is typically used to measure angular acceleration or maintain angular velocity. Traditional gyroscope is built by rotor, spin axes, and gimbals. And modern gyroscope is built on MEMS which utilizes optic fiber and sensitive laser but could perform the same functionality as the traditional one. Both of the modern and traditional gyroscope is built based on Coriolis effect, which is shown below:

$$F_c = -2m(\Omega * v) \tag{1.2}$$

In Coriolis effect equation, Ω is the angular velocity and v is the tangential velocity. m is the mass of the object. The equation gives an idea of how we could quantify velocity into force. The Coriolis effect demonstrates that the a force is being exerted while the object has a rotational movement. By measuring the force rate of the rotating object, we could determine the rate of the rotations. An IEEE standard Coriolis vibrator gyroscope (CVG) is built on such foundation [5]. Moving on from the traditional gyroscope, figure 1.4 shows the world's first single-

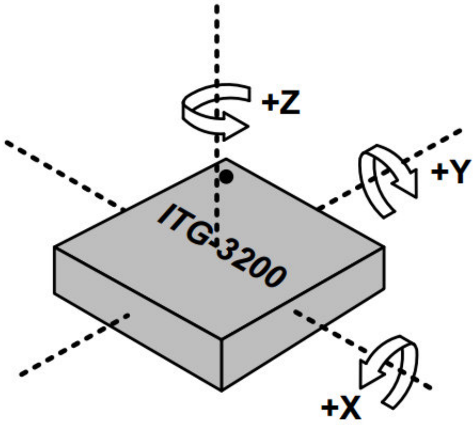


Figure 1.4: An example of modern gyroscope with three axes and is encapsulated into a single chip.

chip, digital-output, and 3-axis MEMS gyro optimized motion sensor [2]. The measured output from the component is in unit degrees per second ($^\circ/sec$) or revolutions per second (RPS).

1.1.6 Machine Learning

Machine learning is the collection of the computer algorithms that can be improved automatically with the feeding data. There are already a variety of the machine learning algorithm in the field of computer vision, natural language processing, and recommender system. Overall, machine learning, as a important data analytic method, is ubiquitous in multiple disciplines.

In practice, machine learning algorithm is built based on a mathematical model, the parameter of which is mutable. Unlike traditional algorithm which has an established set of parameters, machine learning model is subject to change from the feeding data. In machine learning, unlike traditional data analytic methods learn data patterns from given data set, machine learning models have a considerable amount of "training data" and "validation data". The training data is used to train the model to update its parameter towards the optimal. Intuitively, a machine learning model will know the patterns in a data set better and better throughout the training phase. The validation data is then used to evaluate the performance of the trained model generalizing to unseen data.

The key of machine learning model can be successful in complex problems is that, machine learning model has the ability to update its parameters to an optimal, which is usually too complicated to reach by traditional data analytic tools. Machine learning algorithms train a set of parameters in complex combinations in order to undercover deeper data patterns. As one may imagine, the accuracy of the training data is one of the significant components of getting a good model at the end of the training. To see this effect, we discuss training data that affects machine learning models under two categories: supervised learning, and unsupervised learning.

In supervised learning, the training data and the validation data is in the form of (x, y), where x represents the data and y represents the label of the data. For example, in CAD data set, the accelerometer and the gyroscope output $(x, y, z, x_{angular}, y_{angular}, z_{angular})$ which will be the x data and, depending on the mark in the recording, the label y will be "eating" if the subject indicates a meal period, or "non-eating" otherwise. In the training phase, the machine learning model will calculate its own prediction y_{train} on the data x, then compare the y_{train} with y, and finally, update its model according to the error $Error(y_{train}, y)$. In general, supervised learning is very similar as traditional data analysis but with a more sophisticated set of parameters. Error in y will certainly degrade the final model performance.

In unsupervised learning, the validation data remains the format of (x, y). However, the training data only has x. Instead of updating parameters according to $Error(y_{train}, y)$ mentioned in the supervised learning, the unsupervised machine learning model reaches the optimal state when it has the ability to cluster the data x into a target number of groups. For example, the classical algorithm K-nearest neighbor (KNN) will cluster the data into a certain number of groups (preset by user). Figure 1.5 demonstrates a KNN cluster preset to 3 groups. In the training phase, the KNN will update its parameters until it reaches certain optimal that could dissect the data set into 3 groups. Unsupervised machine learning model will group the data to such state, to uncover the unseen pattern from the unlabeled set. With erroneous labels, the unsupervised learning algorithm could group entirely different sets of labels, and thus making wrong predictions on unseen data.

1.1.7 Neural Network

With recent advancement in the semiconductor industry, higher and higher computation power is available to the research field. Neural network merges as an effective machine learning method but with high computing cost. Neural network originated from 1943 [20], mimics the component and the structure of human brain: neuron and how neuron is related to each other. In this work, we use a neural network architecture from previous work and train it on the CAD data set.

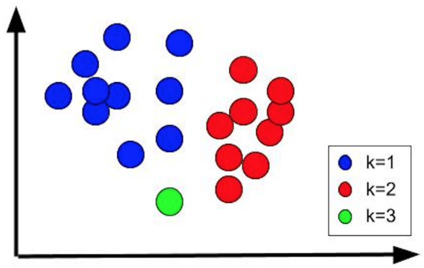


Figure 1.5: KNN preset with k=3 or 3 groups. The dots are clustered into 3 groups by the classifier with each group represented by a different color.

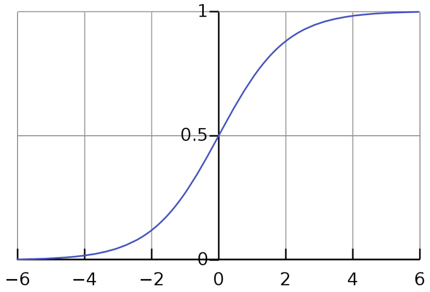


Figure 1.6: A demonstration sigmoid function graph for $x \in [-6, 6]$. The range of the sigmoid function is $(0, 1)$, making it useful to normalize input data while being unsaturated.

Neural network has resistance to noisy labels due to its nonlinear components and large number of neurons (large size of structure). However, large quantity of noisy labels could still lead the neural network to finalize on a non-optimal or diverged model.

In neural network, each neuron is consisted by a linear function and an nonlinear activation function. The linear function provides a weighted average of the neuron's input, and the activation function will decide if the neuron should be activated for the current computation iteration based on the output from the linear function. Here, we give an example of the activation function which we have used in our work, the sigmoid function:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{1.3}$$

The sigmoid function exhibits a nature that can be demonstrated by its graph in figure 1.6. The sigmoid function serves as a role to decide whether the input is relevant (≈ 1) or not (≈ 0). A nonrevalent input will thus be discarded and do not affect further computations.

When combining neurons together, we could get a neural network. In figure 1.7, we see a neuron network of 5 layers (from left to right) with one input layer and one output layer. The first layer of neurons is fully connected to the neurons in the next layer, meaning each neuron on the secondary layers will receive the output from all outputs from the previous layer. The structure of the neural network, in a simple version, could be categorized into input layer, output layer, and the hidden layer. The input layer takes the data vector as the input of the training data x. The hidden layers are a complex structure that involve potentially more parameters. These layers are necessary to describe complex data patterns. Finally, the output layer is the final layer in a neural network architecture. It usually reduce the data dimensions and output a class value indicating the network's prediction on the given data. To make the neural network work, we still have more mechanism to introduce. At training phase, an error is calculated by a cost function, i.e. $Error(y_{train}, y)$. Such error value is then used to update the neurons in the network. A typical equation for error calculation is the mean square error (MSE) function for N output classes shown below:

$$MSE = \frac{1}{N}\Sigma_{i=1}^{N}(y_{train_i} - y_i)^2 \tag{1.4}$$

To update the network's neurons, neural network implements a mechanism called back propagation at the end of each iteration. In back propagation, the error is propagated backwards to each neuron of each layer. The neuron's error is then calculated by the product of the activation function's first derivative and the error, and we call the neuron's error value $error_{neuron}$. The $error_{neuron}$ is then used to update the weights of the neuron's linear function and we call this weight w [15]. The equation below demonstrates this update to the w:

$$w_{update} = w - a * x * error_{neuron} \tag{1.5}$$

As discussed before, x is the input data. The parameter a stands for stepsize. In equation 1.5, we see the higher a stepsize is, the more different w_{update} would be. So, a large stepsize will accelerate the convergence to the optimal w; on the other hand, a small stepsize will make the convergence slower. Although large stepsize could reduce the time to reach the optimal parameter, large stepsize is not necessarily the best option in every case. In figure 1.8, we demonstrate the difference between large and small stepsize, and the potential problems with using a large stepsize. The convex curve represents a relationship between $error_{neuron}$ and w. Although small stepsize (red path) tends to

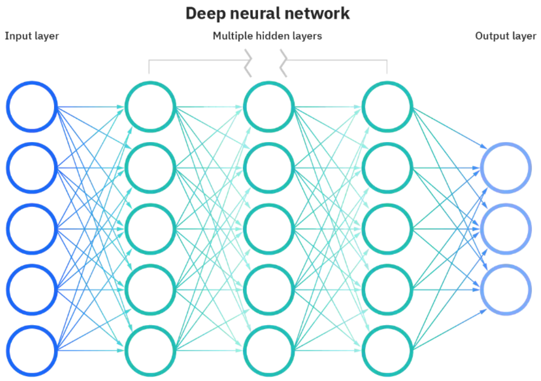


Figure 1.7: A simple demonstration of a multi-layer deep neural network from input layer (taking input data of arbitrary dimension) to the output layer (giving results in an one-hot form across target classes) [3].

reach the optimal value slower, the large stepsize entirely misses the optimal and diverges to infinity $error_{neuron}$. In machine learning, the $error_{neuron}$ is usually used by its first derivative instead of applying the results directly, since the first derivative of a cost function describes the trend of error better than error itself. So, what we have shown in figure 1.8 is also called gradient descent in machine learning. Each neuron is to update its w by gradient descent based on the output y_{train}. The neural network is said to be converged when each neuron weights converge to their optimal value. Noisy labels could cause the model to converge on wrong optimal, and in a worse case, the neural network could diverge because too many labels are noisy and contradicting with each other.

1.1.8 Loss Function

In neural network, loss function is used to measure the distance between the target value and the classifier output value. In convolutional neural network (CNN), we need a loss function at the end of the network architecture to calculate the loss of each training iteration and use the back-propagation method to push the network parameters towards the optimal value. In this work (CAD data set), the objective for the classifier is to predict data sample from two classes: ”eating” class represented by the number 1, and ”non-eating” represented by the number 0. For classifier with two targeted class, this work implements a binary cross entropy loss function. The binary cross entropy measures the difference between a target distribution and a given distribution. Given a target class y and an predictive output $p(x)$ for a random sample x, the cross entropy loss L is

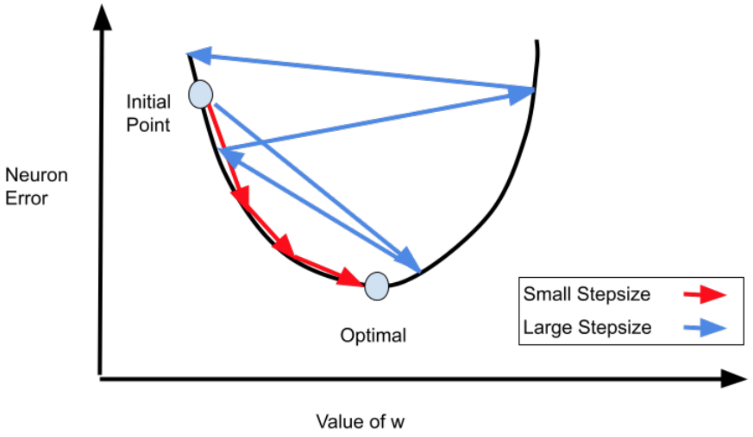


Figure 1.8: A demonstration of the weights update. The red path takes a small stepsize and eventually reaches the optimal. The blue path takes a large stepsize and misses the optimal, causing the further neuron error to bounce to infinity.

calculated by:

$$L = \frac{1}{N}\Sigma_{k=1}^{N}[y_k \cdot log(p(x_k)) + (1 - y_k) \cdot log(1 - p(x_k))] \tag{1.6}$$

In equation 1.6, N is the total length of the data sample. Classifier output P(E) for each sample is scaled by logarithmic function, and the takes an weighted average to produce the cross entropy loss. The model loss is then calculated by the average sum of all data samples.

1.1.9 Global Average Pooling Layer

Pooling is an essential mechanism for CNN to avoid bias and overfitting, as well as compressing input data dimensions further. The method of average pooling for 1 dimensional data is fairly straight forward, which is the calculation of the average for the given length. Given x as input data and a average pooling kernel of length N, this method can be formulated as:

$$y = \frac{\Sigma_{i=1}^{N} x_i}{N} \tag{1.7}$$

We also have a stride value (or stepsize) for the average pooling layer, to move the kernel across all data points. However, this work uses the tensorflow keras library to perform the global average pooling and we follow the default settings of the given function. This layer will not require any parameters in real implementation.

1.1.10 Fully Connected Layers

Fully connected layers is the cardinal components in a CNN architecture. These fully connected layers are where the classification happens. Each of the layers consist of arbitrary number of neurons. Specifically, each of this neuron is represented by a linear function with a bias and an activation function. The linear function takes the weighted average of the output from the previous layer and the form of:

$$y = \mathbf{WX} + b \tag{1.8}$$

Each neuron will calculate its own results and if the results is not accurate, it will be reflected by the loss function at the end of the CNN architecture. The weight $\mathbf{W}$ and bias b will be fixed towards the optimal value in back propagation. Besides linear estimation, each neuron has another essential component: the activation function. The activation function decides if a neuron should be deactivated (output equals to zero) or activated (provide its estimation to the next layer). There are a number of activation function and most of which are nonlinear function. In this work, we use rectified linear unit (ReLU) function in the middle layers and sigmoid activation function in the output layer (the last layer of the multi-layer fully connected layers). We have talked about the sigmoid function in the previous chapter. The ReLU function can be formulated as below:

$$y = max(x, 0) \tag{1.9}$$

It is not hard to tell that the ReLU function deactivate any value below to zero but has a unlimited upper bound before saturation. The unlimited upper bound allows w to keep updating if necessary instead of gradually diminish the error fix in the back propagation process. The deactivation feature gives the advantage for a neuron to be deactivated in certain classification and thus escape the possibly unnecessary w adjustments during back propagation. In addition, deactivation allows the network to discard biased values, which is quite useful.

1.1.11 Gaussian Smoothing

Gaussian smoothing is a common technique used to high frequency signal preprocessing. The goal of the smoothing technique is to eliminate redundant details that could cause the classifier confusion. In Gaussian smoothing, each data point x_j is to modify its current value by consulting its neighbors in a kernel range $2N$ (from x_{j+N} to x_{j-N}), which could be demonstrated by the equation below:

$$x_j = \Sigma_{k=-N}^{N}(x_{j+k} * G_k) \tag{1.10}$$

In equation 1.10, G is the Gaussian kernel that is used to convovle with the data point x_j. The Gaussian kernel has length of $2N$, and can be formulated as below:

$$G(k) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{k^2}{2\sigma^2}} \tag{1.11}$$

In equation 1.11, k stands for the distance from the current origin, which is the same k in 1.10. For example, if we have a kernel of length $2N$, then $x \in [-N, N]$ and x is integer. The σ^2 is the variance that is decided by the actual problem. The data processed by the Gaussian smoothing will keep its major features while erases any minor traits.

1.1.12 Sliding Window Data Sample Generation

In this work, to better train the classifier, data sample that contains a window of data points is being created for training and evaluation. The intuition behind this method is that individual data points simply need more context to be reasonably considered as an individual sample that describes a certain movement. To solve this problem, we use a window of time length T to define a single data sample.

In figure 1.9, we show how the sliding window is done. Each data sample is generated with a window length of 3 data points, and a sliding window stepsize of 1 data point. There are 6 axes with 3 axes-x, y, z from the accelerometer and 3 axes-roll, pitch, yaw from the gyroscope. As shown in the figure, each sample has overlap with each other and gradually slide over till the end of the data file. To cover the whole data set and produce a contiguous prediction distribution, the window will slide along the time axis with step size S. Assuming we have a total number of L data, and

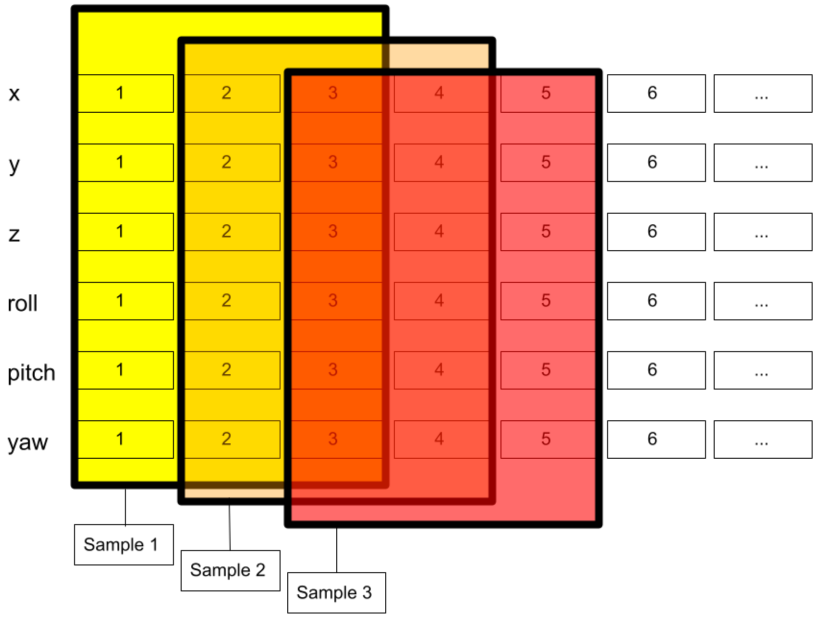


Figure 1.9: A demonstration of the sliding window method on Shimmer3 device data.

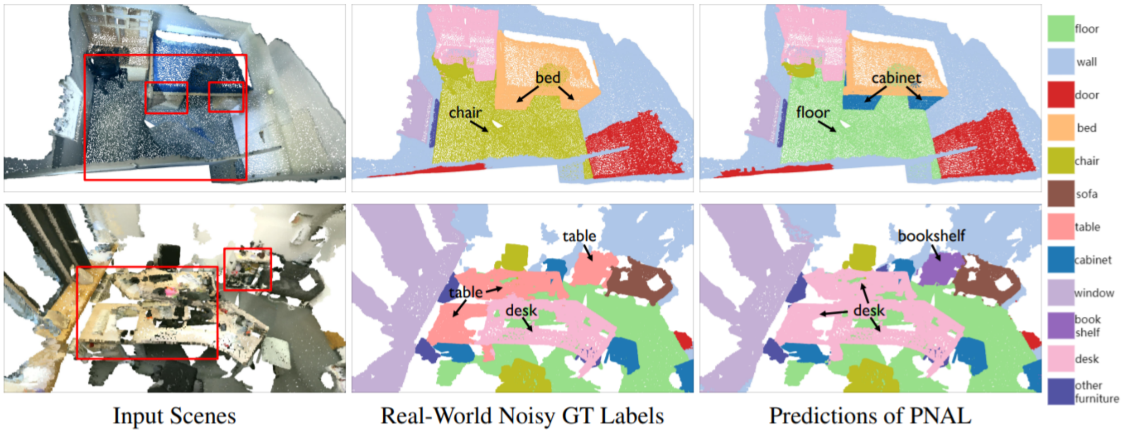
then the total samples we could extract from the data can be calculated by:

$$Total\ Number\ of\ Samples = \frac{L - T}{S} + 1 \tag{1.12}$$

Since a single data point from the Shimmer3 device does not say much about a gesture, we use this process to generate training and testing sets for the classifier, so that the data sample could encapsulate a window of action.

1.1.13 Denoising

Noisy labels are a well-known issue in image classification. A large data set contains natural noise which could be neutralized by each other. However, if a data set is too small to neutralize the noise or all noise is biased to the same direction, then the data set is likely to cause a problem. The noisy labels will most likely inflict a defect on the final classifier performance. In deep learning with neural network, noisy label is also a common problem that the model has to deal with. Noisy label could result the neural network converging to wrong distribution or diverging in the training



Input Scenes

Real-World Noisy GT Labels

Predictions of PNAL

Figure 1.10: Illustration of the GT noisy labels contained in real-world data set ScanNetV2. From left to right are the input (noisy instances highlighted red boxes), the manual annotation given by the data set, and the prediction of the classifier with denoising features. The dataset suffers from label noise, such as mislabeling the floor as a chair, even that it is already a re-labeled version of ScanNet. [37]

process. To mitigate the effects of noisy labels, various attempts has been made. In figure 1.10, Ye et. al attempt to solve the noisy labels in the ScanNetV2 data set. The image demonstrates the error ground truth (GT) label in the popular image data set. The image has also shown Ye's method could overcome the noisy GT labels, and make the correct prediction. The paper utilizes a method to identify noisy labels by classifier past predictions throughout the training process. The method is based on the fact that most classifier could make correct predictions even if the labels are erroneous.

Some studies also attempt to utilize the neural network regularizers to mitigate noisy effects. For example, Krizhevsky et. al's ImageNet [21] uses the data augmentation to prevent the data from overfitting into the data set, which could help to prevent some noisy labels (outliers) being fit by the model. As mentioned in the paper, data augmentation has enabled the researchers to utilize a larger neural network structure; otherwise, a smaller network is needed to avoid overfitting. Data augmentation could be done in various ways; the key idea is to make a copy of the data by making slight changes according to reasonable methods. In other words, some noisy labels have been manually added into the original data, which though seems to be contradictory with our intention, helps the network overcome more bias on a bigger picture. Because of its simplicity and effectiveness, data augmentation has been a common method in image classification by deep neural network.

Although regularizers are easy to implement, minor improvements could be done by just regularizer when the bias is large. More complex approach is needed. MentorNet [18] builds a new network architecture, which consists a mentor net and a student net. In this network structure, unlike usual networks, student net gets its weight w from the mentor net, and produces the data features as its output. On the other side, the mentor net learns the features output from the student net and provides predictions on the data set, as well as migrating the weight it has learned to the student net. One of the key ideas from MentorNet is that some of the crucial features (either from original data or from other network outputs) have resistance towards noise. For example, a highly biased data set is possible to have a less biased average value, and by learning the average value, the model could avoid some noisy labels. Overall, with the knowledge of more reasonable features, network could become more sensitive towards noisy data samples.

In CAD data set, subjects are asked to report their eating period by the orange buttons on the Shimmer3 device. Since the data is collected from the wild, error button presses could happen more often then expected. Since the collector only confirms the time marks with the subjects at the end of the data collection, we cannot guarantee a subject will remember the accurate time mark and cannot assume all subjects will follow the instructions as intended. For example, subjects may forget to record a snack period and choose not to tell that 1 minute snack to the data collector. Even more, subjects may fail to record a time mark either at the meal start or meal end, and fix them by pressing the button at a later time. In figure 1.11, we show an example file of eating probability distribution P(E) and a possible error button press. The classifier has given a clear peak indicating eating activities, however, the ground truth is slightly off to match the peak, making the the button press a possible erroneous ground truth. Such potential noisy labels could jeopardize the training results. Thus, the current model could be improved by using a noisy label detection method to adjust the original ground truth.

1.2 Related Work

A number of studies has done to fight noisy labels in data set collected from the wild. For time-series data set such as motion sensor, Soma et. al [8] proposed a method to automatically generate labels for a data set with only a small group of known clean labels. The data set used in the study is time-series data collected by accelerometer, motion sensors, and ECG. The method

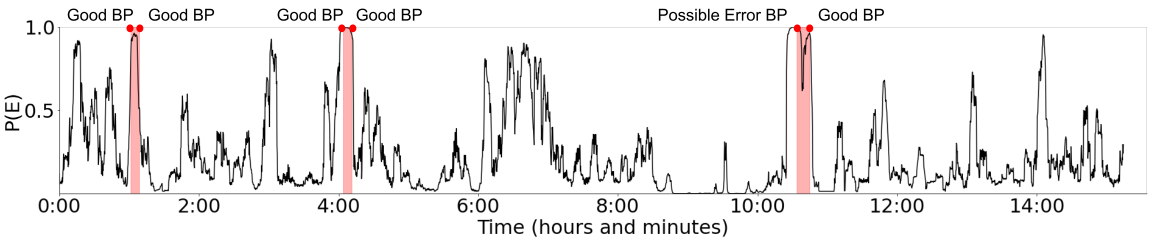


Figure 1.11: An example of classifier output eating distribution P(E) (black curve), participant labeled ground truth eating period (red area) and button press (BP) (red dots). The first two pairs of BP overall match with the P(E) peaks. In the third pair, the first BP is slightly delayed than the start of the P(E) peak, making it a possible error.

clusters all unlabeled sample into different groups, and looking for the clean labeled sample that is closest to the specific cluster. Each unlabeled group will be generated with a label that is same as the closest clean labeled sample. In this way, the group cleverly reduces the number of noisy labels and establishes consistency inside the whole data set. Eventually, the method is able to produce an improved accuracy by 1% to 15% on the majority of the classifiers they have tested on. Terry et. al [34] explores data augmentation method in wearable sensor data for Parkinson's disease monitoring. Since the sensor data is both limited in number and noisy, by augmenting the original data set, the study prevents the neural network training from overfitting and improve its generalization to the clean labels, as well as generating enough data to train a deep learning classifier. The study proposes to augment the original data with different positions of wearable sensor, different temporal locations of data samples, and scaling. The final result is able to achieve 86.88% accuracy on a convolution neural network. Another study relates to time-series data and wearable device, Das et. al [10] proposes a method to classify the noisy ECG sensors data from heart beat by training a neural network with noise tolerance methods. The study has preprocessed the ECG data with high-pass, low-pass filtering and apply Fourier transform before sending the data into neural network for training, which proves to be efficient on reducing noise. The final result of the proposed method is able to achieve 22% over the current state of the art method of that field.

For image classification field, where annotation could be much higher cost, noisy labels happen more frequently. Various studies have investigated possible solutions. Wu et. al [36] proposes a method to generate clean labels from the data set on the fly. The method utilizes the topological relationship between clean labels and noisy labels, where clean labels tend to form regular clusters while the noisy labels tend to be outliers. The study uses a early stop method on training to generate an initial set of clean labels and noisy labels. At each iteration, the classifier will only be trained on the clean labels and keep peeling off more edge labels that have loose connection to the cluster (more susceptible to be noisy labels). In this way, the final output of the method would be a clean set of labels and a fully optimized model trained by clean labels. The method has outperformed most of the best noisy label detection methods by at least 1% and achieved better accuracy on noisy image data set.

Studies also look into the nature of noisy labels. Malach et. al [23] proposes a general intuition into detection of noisy labels. The study brings up two classifiers that are trained simultaneously, and select the training samples based on the disagreement of two classifier output. The study finds that, for a sample label that two classifier yields low accuracy or the classifiers do not agree on is usually a noisy label. The study has achieved an accuracy that is over 90% on image classification problems, which are comparable to most of the state of the art methods. Furthermore, Junnan et. al [22] proposes a method to divide data samples into clean labeled and noisy labeled data based on classifier output loss value for each sample. The method must be applied with a semi-supervised learning model, with supervised training on clean labeled data set, while for the noisy label data set, the classifier is trained with unsupervised learning fashion. The core of dividing data samples is by using a two-components and one-dimensional Gaussian mixture model that calculates the probability of a data sample being clean labeled. The method outperforms most of the noisy label detection methods by at least 2% in image classification field.

Finally, Yiru et. al [31] proposes a method to detect noisy labels and classify data samples generated from wearable motion device. The study explores the bite detection for different foods and beverages by using a thresh-hold based algorithm with a hidden Markov model, on 271 participants with a total of 24,088 bites recorded. The data set is collected in a laboratory setting, however, relies on human annotation to label the eating gestures. The study proposes a method to instruct human raters label eating gestures by video frame. Only the portion of raters agree on is used for classifier training. The study has achieved a 75% sensitivity (true detect rate), proving the algorithm to be an effective way to detect bite-level eating data.

1.3 Novelty

This thesis considers the problem of noisy labels in the CAD data set. The novelty of this work is to develop a schema and an algorithm to reliably identify and adjust noisy labels based on the classifier output eating probability distribution P(E) and classifier detected eating segments. The adjustments use a combination of data analysis and human raters. Besides, this work compares the results between the model trained by original labels and the model trained by the adjusted labels. By comparing the model performance, we want to know if label adjustment based on classifier detections is a viable way to identify the noisy labels and improve the model training. In summary, this work responds to the following questions:

1. Can we reliably identify errors by using the existing ground truth and the classifier output probability of eating P(E)?
2. By using the method, how many errors do we think exist in the CAD data set?
3. By fixing these errors, how much improvements can we get from the classifier training?

Chapter 2

Methods

The following paragraph provides an overview of this chapter. In section 2.1, we briefly introduce the CAD data set used for this work. The following section 2.2 outlines the preliminary experiment we have done and provides a brief result. The preliminary experiment leads us towards a more profound approach: adjusting a contiguous sequence of data by following a schema. In section 2.3, the schema approach is discussed in detail. In section 2.4, we discuss the methods to perform inter-rater reliability test that examines the quality of the schema. From section 2.6 to section 2.8, we introduce the model used to evaluate the adjusted labels. We will discuss the methodologies from data preprocessing to neural network architecture, as well as post training process for meal segments. In the last section 2.9, we outline the model evaluation metrics for the adjusted labels and the results will be presented in the next chapter.

2.1 Data Set

This work uses the Clemson all-day (CAD) data set collected by Sharma et. al [30]. The participants are random personals recruited from the wild. Each participant has approximately 10 hours of recording at the end of the data collection. In the data collection process, 408 participants are recruited and 351 participants have completed the one day recording, resulting 354 days of data available. In the data set, there are 1,133 separate meals/snacks with each activity cross checked with the participant at the end of collection to ensure accuracy. Each recording consists of 3-axis accelerometer data, 3-axis gyroscope data, and ground truth time marks made by participants on the Shimmer3 device. The Shimmer3 device is programmed to operate at 15 Hz, which means there are 15 data points from 6 axis for each second. A total of 4,680 hours of activities is available with: 265 hours of eating activities are recorded and non-eating activities are 20 times more than the eating activities.

When collecting CAD data set, instructors relies on the subjects to follow the instructions and record meal start and end time accordingly. Each participant will receive a short 5 minutes training before providing with a Shimmer device. According to Sharma et. al, each participant was instructed to wear the device on waking up, and long press the button to start recording data. The subjects were requested to tap the button before the start of any period where the primary activity was eating, such as a meal or a snack, or any other contiguous period of eating. Subjects were instructed not to timestamp any grazing consumption activity such as drinking coffee while working or reading a book over a long period of time, where the primary activity may not be eating. At the end of the day, participants were instructed to stop recording data, and take off the device [30]. The researchers will then extract data from the Shimmer3 device and ignore any erroneous meal labels.

In this process, although the steps are seemingly simple, some details may impair the accuracy of the meal time marks. On Shimmer3 device, only limited user feedback features have been designed to signal recording or labeling time mark. When the device is turned on (start recording regular activities), it does not provide any obvious feedback, and the green light will only start to blink when the orange button is pressed (recording eating period). Participants may mistaken blinking light as device turning on, and not realizing it is a signal for recording eating period. In this situation, participants will likely to overlook the blinking light and not fix the error until the start of first meal, where they will press the button again and stop recording, causing a erroneous ground truth. Furthermore, minimal training is given to the participants, which may not make every participant to understand the instructions. In the data collection process, the button presses are defined as the moments of first intake and last intake, but this definition is probably confusing to some participants. If participants ”snack” or ”taste” while preparing their ”meal” they may hit the buttons early/late. Also, some participants may consider a ”meal” as a 45-minute period during which they only actually are making intake gestures for 15 minutes scattered throughout the period, causing eating gestures to be overwhelmed by secondary activities. These misunderstanding or misconduct on recorded eating period will cause erroneous button presses, and thus creating noisy labels.

At the end of the data collection, instructors will check each recorded meal period with the participants in a short interview. If a participant has a regular meal schedule, then he or she will probably remember the meals they ate and gave a rough time check with the instructor. However, if a participant does not have regular meal schedule, it is very likely that the participant does not remember when they ate a day before. Although the short interview is enough to eliminate any obvious errors (participants realize the erroneous button presses), it may not be enough to get the exact desirable eating period time marks without any noisy labels. Furthermore, activities like a 5-minutes snacks have short period of time, making it even harder to memorize. These conditions make checking noisy labels a hard process at the data collection stage. We address to unknown erroneous button presses from the data collection stage and analyze them to see how we could fix such errors.

2.2 Preliminary Experiment for Adjusting Noisy Labels

For the first attempt to adjust noisy labels and to get an idea of how it may work, we take the approach of label refurbishment with neural network model. Essentially, a classifier that built with neural network architecture is used, and labels are being refurbished based on the classifier loss value after each training epochs. In this preliminary experiment, we treat each datum as an individual point, and apply adjustment to each of them separately.

Here we define a variable y_{clean} to be the collections of the clean labels, a variable $y_{corrupted}$ to be the collections of the corrupted labels and $y_{refurbish}$ to be the refurbished labels. The classifier will generate its labels for the next learning epoch by:

$$y_{refurbish} = (1 - \alpha) \cdot y_{clean} + \alpha \cdot y_{corrupted} \tag{2.1}$$

In equation 2.1, α is defined as noisy ratio, or the percent of expected noisy labels in the original data set. α can be formulated as:

$$\alpha = \frac{\# \ of \ y_{corrupted}}{\# \ of \ y} \tag{2.2}$$

In CAD data set, the information of the noise ratio is unknown and we do not have a secondary source to confirm the accurate original time marks. However, one thing we are sure about is that, the data set has more clean labeled datum than the corrupted ones. Since it would be impossible for a general classifier converging to a minority data set, we safely assume more clean data is presented since the model from Sharma [30] has demonstrated a decent performance on identifying eating periods. Another important observation we have is, by the nature of the neural network classifier, the model tends to converge to clean labels faster than the corrupted labels. Given the condition that $\alpha < 0.5$, we suspect the classifier has the innate ability to adjust noisy labels. In CAD data set, we are trying to classify the eating gestures against the non-eating gestures. Clean labeled data of eating and non-eating class will be learned by the classifier quickly, while noisy labels will converge later. Thus, a tentative way to distinguish between noisy labels and clean labels is by their convergence rate.

Additionally, we observe that although noisy labels can be misleading, classifier tends to hold strong indicator on those data sample it is certain of, regardless of the labels. In other words, we could observe a data sample holds a ground truth label ”non-eating” but keeps getting ”eating” label from the classifier, which is a strong indicator of noisy labels. Since classifier has given strong certainty on some noisy labels, we suspect this as a useful information for doing label refurbishment. In the preliminary experiment, we put together these ideas and let the classifier to adjust individual datum at the end of each training epoch and create its own training set for the next epoch. And to quantify model certainty, we use a data array H to store class prediction history that classifier has made on a particular datum.

Furthermore, when predicted label from the classifier does not agree with the ground truth label, the classifier will generate a high loss value. We use loss value to quantify the degree of the disagreement between ground truth and the model predictions. A data sample with a large loss value is likely to be a targeted noise label. By adjusting such label could improve the consistency of the training set. In this preliminary experiment, we assume the CAD data set will contain at most 20% of the noisy labels or $\alpha < 0.2$. And, we use the loss value to filter out the bottom 80% of the datum, and only look at the top 20% of the datum for refurbishment (the datum that holds the top 20% of the highest loss values).

Finally, an tentative algorithm is developed to adjust labels based on loss value. In algorithm 2.2, we put all the points together and demonstrate the execution. In the algorithm, we define class ”eating” as 1, and class ”non-eating” as 0. We wait for 7 training iterations before we start to adjust labels. By the 7th iteration, the classifier has fit into the y_{clean} and has not overfit the $y_{corrupted}$.

At this “golden” stage, we use the class prediction history to measure the certainty of the classifier. For a specific datum, if classifier has generate more labels on one class than the other, we will adjust the labels for the sample according to the majority in H. Although this method does not define a hard line on lots of the criteria and is quite loose, it allows the classifier to adjust some highly noisy labels back and forth instead of locking them down to wrong labels. We want this algorithm to provide an intuition of adjusting noisy labels by individual datum.

Algorithm 1 Adjusting Ground Truth Based on Loss

```
Require: Training set $\mathbf{X}$, class prediction $Y$, class prediction history $H$
procedure For Epoch N
    $\mathbf{X} \rightarrow$ Model
    Read the model loss $L_i$ for each sample
    Read the model class prediction (0 or 1) $P_i$ for each sample

    for $x_i$ in $\mathbf{X}$ do                    ▷ We update the class prediction history
        if $Y_i = 1$ then
            $H_i[1] \rightarrow H_i[1] + 1$
        else
            $H_i[0] \rightarrow H_i[0] + 1$

    if $N < 7$ then                    ▷ We do not do any adjustment before the 7th epoch
        continue

    for $x_i$ in $\mathbf{X}$ do
        if $L_i$ is in the highest 20% then        ▷ We assume 20% of the training samples are noisy
            if $H_i[1] > H_i[0]$ then
                $x_{i_{label}} = 1$
            else
                $x_{i_{label}} = 0$

```

2.2.1 Preliminary Experiment Results

The resulted model is evaluated on the 354 files from the CAD data set. We have also trained a baseline model from the original labels for comparison. The baseline model adapts all hyperparameter and methods from Sharma [30]. The results are recorded in table 2.2.1. Both models are evaluated on the original ground truth. From the results, we see the model trained with the adjusted labels has 2% higher weighted accuracy than the baseline mode.

In the figure 2.1, we show an example file with ground truth meal and the adjusted labels by the refurbishment model. The adjusted model is flipping some reasonable labels under the ground

	Weighted Accuracy
Baseline	79%
Adjusted	81%

Table 2.1: Preliminary experiment results for the baseline model and the label refurbishment model.

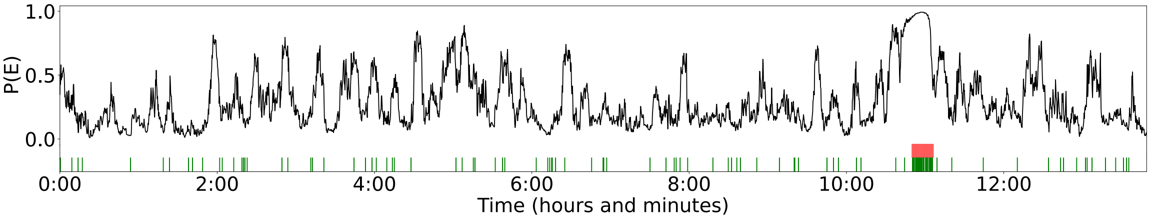


Figure 2.1: Preliminary experiment results to compare the baseline model and the label refurbishment model. The black curve represents the classifier output eating distribution P(E) by the baseline model. The ground truth meal (red) is a contiguous rectangle underlying a strong P(E) peak. The adjusted labels (green) are scattered dash lines demonstrate no continuity that could form a complete meal period.

truth meal. However, the adjusted model has also flipped a number of “non-eating” data samples to “eating” samples with no obvious reasons. The adjustments made by the adjusted model seems to be random and do not show any useful pattern. While we are looking for a contiguous meal period like the ground truth meal (red), the adjusted model is aggressively adjusting individual datum that does not have connections to each other. From this preliminary experiment, we conclude that the current method is not adjusting the targeted noisy labels. Each datum is treated individually instead of a contiguous sequence of data that represents a regular meal period. Based on this conclusion, we move forward with adjusting noisy labels by human expert following a certain schema that considers a contiguous sequence of data.

2.3 Schema to Identify Button Press Errors

In this section, we introduce a scheme approach to identify button press errors. In the CAD data set, a pair of button press could define an interval of meal period, containing a contiguous sequence of data. In the schema approach, instead of treating each datum individually like what we did in the preliminary experiment, we adjust the original ground truth (GT) by considering a sequence of data together. In this way, we believe the GT will be more reasonably adjusted instead of following single separated datum flipping, which does not show clear rational behind it. For the rest of this section, we will begin by introducing our motivation behind this method in section 2.3.1, and introducing the interface we use to analyze the button error (BP) in section 2.3.2. In section 2.3.3 and 2.3.4 respectively, we will present the schema and the algorithm to identify error BP with GT meals and classifier detected segments. Last but not least, in section 2.3.6 and 2.3.5, we will introduce the method to adjust each erroneous BP and the way to identify a missing GT meal.

2.3.1 Motivation

In the preliminary experiment, we have shown a possible approach to fix erroneous GT by adjusting individual datum. Random scattered data is adjusted without following our perception of what a normal human eating activity would be like. We have decided that we could better fix the noisy labels by changing this method. In this section, we take the approach to adjust a contiguous sequence of datum by adjusting the button press(BP). By adjusting an erroneous ground truth BP, we flip the label of a contiguous sequence of datum. In figure 2.2, we show the process of adjusting an erroneous ground truth BP and create a new complete adjusted GT meal. By changing the first BP of the original GT meal, we convert a contiguous non-eating period to contiguous eating period. In this way, we are creating adjusted labels more reasonably and the resulted adjusted GT will be a complete meal period instead of scattered datum across the whole P(E) distribution.

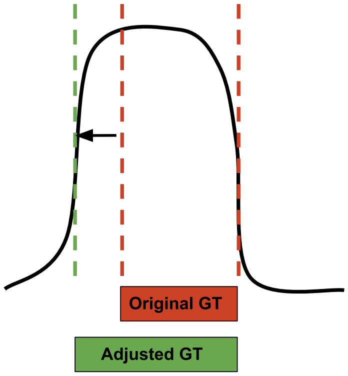


Figure 2.2: The original GT (red) is adjusted by moving the first BP (red dash line) to the new index (green dash line) according to the P(E) distribution (back curve), and a new adjusted GT with complete eating period is created (green rectangle).

2.3.2 Rater Adjustments Interface

To fix the noisy labels by adjusting erroneous BP, we have human rater to examine the GT one by one for all 354 files. To do such work, rater will start to learn about identifying BP errors by looking at files. In figure 2.3, we show such a file that is used for training a rater. In the file, there are 3 essential components components: P(E) distribution, ground truth meal (GT), and the classifier detections. Specifically, each components stand for:

- Classifier Ouput Eating Probability Distribution P(E): represented by the black curve reflects classifier confidence.
- GT Meal Period: represented by the red areas and describes the ground truth eating time period. The two vertical edges of the red area are BP reported by the participants
- Classifier Detections: represented by the blue rectangles and describes the eating periods detected by the classifier.

In figure 2.3, there are two GT meals, and four classifier detected segments. Rater will be looking at mainly these two things, and judging if a BP is erroneous by comparing the boundaries of the classifier detections with the GT meals. When the classifier detects a meal segment with high probability while the GT meal does not entirely agree, it usually suggests the GT has one or two error BPs. When identifying an error BP, the P(E) distribution will be used as a secondary reference when boundary conditions are not enough to make the judgement. Although P(E) distribution demonstrates a more complex and complete nature of the classifier response, we believe it requires more experience to make good judgements. The classifier detections could be a more obvious version of classifier opinion, and could help inexperienced rater to make better judgements. Overall, since P(E) distribution has more variants and is hard for inexperienced rater to comprehend, we aim to pick the erroneous BP by mainly looking at GT intervals and classifier detected segments. In this way, we could prevent the more complicated P(E) to confuse rater, and provides certain constraint on judging error BP. After understanding the basic intuition behind identifying error BP by looking at classifier detections and GT, we develop a schema that puts each GT into different categories, so that rater could apply adjustments accordingly. We will further discuss this schema and methods for adjustments in the following sections.

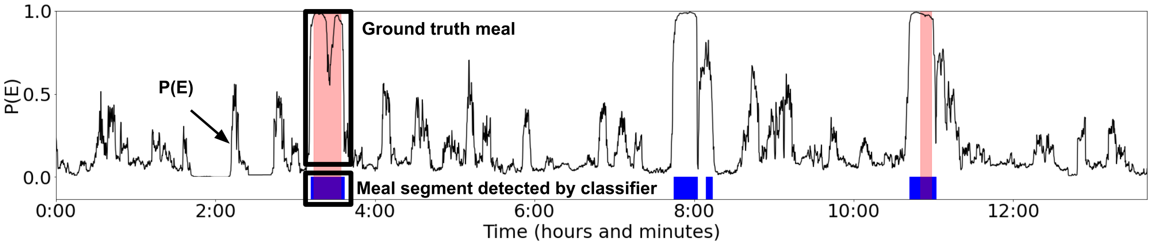


Figure 2.3: An example of data shown to a rater to identify BP errors. P(E) is shown in black, GT intervals are shown in red, and classifier detected segments are shown in black rectangle. Raters are instructed to prioritize comparing GT intervals (red rectangles) to classifier intervals (blue rectangles) to look for BP errors.

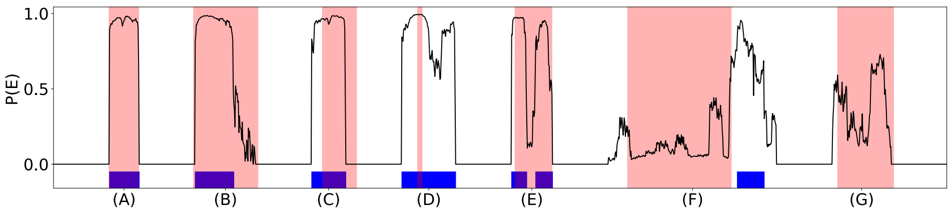


Figure 2.4: Examples for each category in the schema with GT meal interval represented by the red area, the classifier detections represented by black rectangles, and the P(E) distribution represented by black curve (see more details in text).

2.3.3 Categorizing GT by Schema

In this section we will demonstrate different categories of erroneous GT meals by looking at the GT intervals, classifier detections, and P(E) distribution. In figure 2.4, we demonstrate seven different examples that are representative to each category.

A. **Nicely Aligned**: The GT meal interval overlap with the classifier detection. The boundaries of the two intervals match nicely. The P(E) distribution shows a strong P(E) peak that is close to a square wave. This case has every thing for a correct GT meal.

B. **Slightly Off**: Although the GT meal interval overlap with the classifier detection, we could see that the latter edge of the GT meal does not match with the classifier detected segment. The GT interval covers a small portion of low P(E) distribution, which is a non-eating period decided by the classifier. In this case, the classifier disagrees with the GT interval on a small portion, so the GT interval is slightly off by having an error BP. In the example, the delayed BP suggests the participant probably forgets to press the button after finishing eating.

C. **Slightly Off**: This is another case of “slightly off”. The GT meal and the classifier detection have overlapped area, however, both edges are missed. Comparing to the GT, the classifier has detected a shifted meal period. In this case, the GT has two error BP, but still covers some portion of the classifier detection. The participant may have misunderstood the instruction and thus included some of the non-eating period into the eating period.

D. **Interval Squeezed**: The GT interval overlaps with the classifier detection, however, two edges do not match and the classifier is significantly larger than the GT interval. Since a large portion of GT non-eating period is being classified as eating, the classifier could be overly aggressive. However, it is also possible that the original GT meal has two error BPs and does not cover the entire eating period. Due to its ambiguous nature, we put this case as an individual category.

E. **Not Continuously Eating**: The GT interval overlaps with the classifier detection and the boundaries match, however, the detected meal segment is discontinued. In other words, the GT interval is covering multiple classifier detected segments instead of a complete one. Note that, the classifier detected segments are usually discontinued by relatively small time breaks instead of long breaks. The participants may have rested during the eating period, and causing the GT eating period contains small portions of non-eating time intervals.

F. **Misidentified Meal**: The GT interval does not overlap with the classifier detection at all. However, a detected segment may be nearby the GT interval and is very close to overlap. In this case, GT interval may cover a long period of non-eating P(E) distribution and have a strong P(E) peak or detected segment close to it. We suspect that subject has misunderstood the instruction, and recorded time period irrelevant to the eating gestures.

G. **Unusual Wrist Motion**: No eating periods is detected under this category. Although the classifier finds nothing of a period under the GT interval, the P(E) distribution is notably higher than the surrounding area, which suggests that a eating alike period has happened but is not strong enough to be identified formally. GT meals of this category looks very much similar to the one shown in 2.4(G), the overall P(E) distribution could cross 0.5 or even close to 0.8. Such case suggests a mixture of eating gestures with secondary activities. Although this category may demonstrate useful GT intervals, we decide it has overwhelming secondary activities and thus becomes less worthy for eating activities detection.

H. **Cannot Identify**: Finally, we put any meals that do not belong to the former categories in here. Usually, GT meals in this category demonstrate erroneous traits, however, the classifier detection and the P(E) distribution are also corrupted or too ambiguous to tell what kind of error does the current GT has. This often happens when the file P(E) distribution is noisy, creating a number of detected segments and not showing clear patterns. In figure 2.5, we show an example of noisy files, in which the third GT interval is hard to judge.

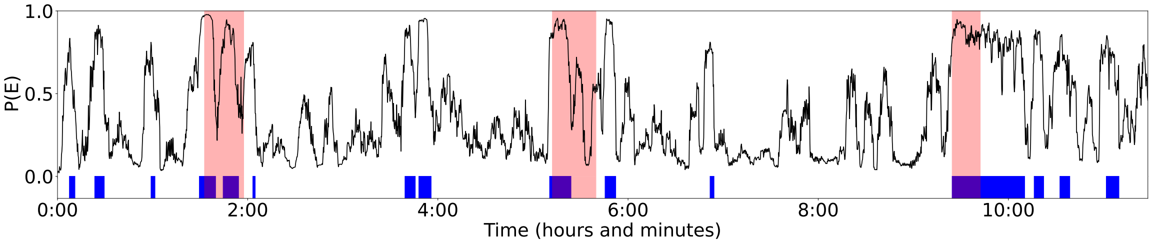


Figure 2.5: An example of a noisy file. There are a number of classifier detections, suggests file could have multiple unlabeled eating period or the classifier is confused on the file data. The third GT interval covers a portion of the classifier detection but then, the detected segments extend and scatter around about 2 hours of period, which are too long for normal eating period. We cannot make a safe judgement on the third GT interval despite its erroneous traits.

The schema outlines a range of different GT interval relating to the classifier detections. Although each category is clearly defined, rater may still have dilemma when look at a complex file. To help the rater make better judgements, we develop a step to step algorithm to put each GT under the seven categories.

2.3.4 Flowchart for Categorizing GT

To better put GT into each category, we form an algorithm for raters to follow when they are looking at GT intervals. In figure 2.6, we show a step by step flowchart that illustrates how a GT interval is put into the respective category. Essentially, we compare the boundaries of the classifier detections with the GT intervals, and make decision accordingly. Starting from the first box in top left box "for each GT interval", we progress by following the arrow. The very first thing we check is if a single classifier detection overlaps with the GT interval. If the two intervals share some agreement (overlap), we proceed to check if their boundaries match. Boundary condition is pertinent to judge if the GT interval is “slightly off” with BP. On the other hand, if the boundaries match correctly, the GT interval is considered to be correct and a “nicely aligned” interval with the classifier detection. Furthermore, if a GT interval is “slightly off”, however, the classifier detection is significantly longer than the GT interval, then we have found an “interval squeezed” GT. To separate the “interval squeezed” GT with the “slightly off” category, we check if the two boundaries of the GT are close to the detected segments’ boundaries. By doing such, we have found three categories where one individual classifier detection has overlapped with the GT interval.

Going back to the box which checks if a single classifier detection overlap the GT. If no overlap has been found, we have two possible cases: the GT interval overlaps multiple classifier detections, or the GT interval does not overlap with classifier detection at all. We first check if multiple classifier detections exist under the GT interval. If the classfier detection is discontinued and contains small time breaks under a contiguous GT meal period, then we have found a “not continuously eating” case. Following the arrow, if the GT interval does not share an agreement with the classifier detections at all, we check if there is a detected segment nearby the GT interval. If a detected meal segment exists nearby, that means the subject has “misidentified meal”. If there is no classifier detections at all, that means the GT is standing alone by itself. In this case, we would check the GT interval if the P(E) under it is notably larger than the surrounding area. If the P(E) does indicate a weak peak, we have found a “unusual wrist motion” which shows a high P(E) distribution under the GT interval but is not strong enough to be considered as an eating period. If the GT interval does not belong to any of these cases, then it suggests the current file is noisy or the classifier output is too ambiguous, which fall under “cannot identify”.

2.3.5 GT Adjustments for Each Category

After assigning each GT into its own category, we provide further actions to adjust the BP to correct the GT intervals accordingly:

A. **Nicely Aligned**: do nothing.

B. **Slightly Off**: adjust start and the end BP to match the classifier detection.

C. **Interval Squeezed**: do nothing.

D. **Not Continuously Eating**: split the GT according to the classifier detections.

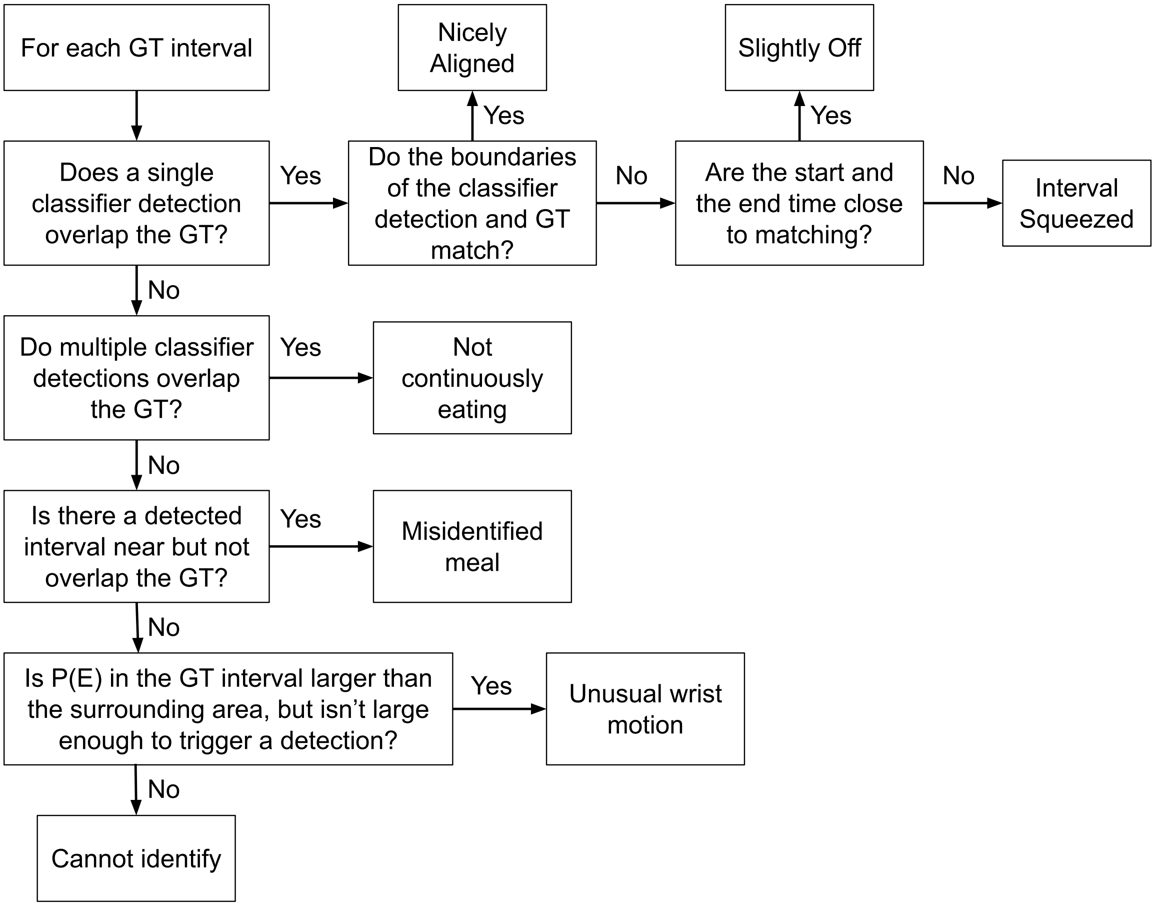

For each GT interval
Does a single classifier detection overlap the GT?
Yes
Do the boundaries of the classifier detection and GT match?
Yes
Nicely Aligned
No
Are the start and the end time close to matching?
Yes
Slightly Off
No
Interval Squeezed
No
Do multiple classifier detections overlap the GT?
Yes
Not continuously eating
No
Is there a detected interval near but not overlap the GT?
Yes
Misidentified meal
No
Is P(E) in the GT interval larger than the surrounding area, but isn't large enough to trigger a detection?
Yes
Unusual wrist motion
No
Cannot identify

Figure 2.6: Categorizing GT intervals according to the schema, starting from the top left box (see more details in text).

E. **Misidentified Meal**: shift, add, or delete BP to match the nearest classifier detection.

F. **Unusual Wrist Motion**: remove the entire GT meal.

G. **Cannot Identify**: remove the entire GT meal.

The intuition behind the adjustments follow the same as how we categorize each GT intervals. For fixable errors like “slightly off”, “not continuously eating”, and “misidentified meal”, we could adjust the BP and have a new adjusted GT interval. For GT intervals that we think are introducing noise or not worthy to pursue, such as “unusual wrist motion” and “cannot identify”, we remove them completely. If the GT “interval squeezed”, we do not delete it because it introduces a solid eating period that we trust, and we do not adjust it neither because we are not sure if adjusting the BP will introduce new noise due to the classifier detection ambiguity. The “nicely aligned” GT will be kept as it is.

2.3.6 Missing Meals

A difficult case in our schema is missing meals. Judging if a portion of P(E) distribution is an unlabeled meal requires extra cautions and experience since it has the risk of introducing a large period of noise to the original labels. To accomplish this process of finding missing meals, we have defined a robust list of things we would check before consider an unlabeled period as a missing meal period. For this category, we do not only look for strong match on GT intervals and the classifier detection but also a clean P(E) distribution throughout the file. The file must has the following qualities for us to identify any missing meals:

1. Each GT intervals overlap a classifier detection. And only one classifier detection exists outside the GT intervals (this segment will be the missing meal).
2. Subject indicates clear eating schedule with no snacks or short meals, and only has regular meals (breakfast, lunch, dinner).
3. Subject misses one of the three major meals.
4. More than 4 hours of time exist between the identified meal segment and the nearest ground truth meal (the missing meal needs to be 4 hours apart from its nearest GT intervals).

5. The P(E) that underlies the classifier detection (missing meal) demonstrates a solid shape as figure 2.4(A).

We do not expect a large amount of missing meals coming out from the definition above. As mentioned before, the cost of introducing a completely incorrect meal can be high, and we only want to add the meal that we are sure about. We will discuss the missing meals in the CAD data set in the result chapter.

2.4 Inter-Rater Reliability

To validate the consistency of the BP adjusting algorithm and schema, we perform inter-rater reliability (IRR) test on 3 raters across 71 subject files. Two of the raters are outsiders who do not have any knowledge of the schema prior to this test, and one rater posses a deeper knowledge to the schema. The 71 files contain a total of 227 GT meals if 454 BP. The rest of this section will introduce how we have done the inter-rater reliability test on 3 raters. In section 2.4.1, we introduce how raters adjust GT BP with EatMon GUI. In section 2.4.3 and 2.4.2, we introduce the evaluation methods by erroneous BP count and time adjustments in IRR.

2.4.1 Data Collection for IRR

In figure 2.8, we show an outline of the inter-rater reliability test. In the process, we give all three raters the schema, an explanation to the schema and the algorithm to perform GT classification. All raters will only classify the GT on the same 71 files and do not require to provide explanation on their adjustments. All three raters will perform adjustment using EatMon GUI. In figure 2.7, we show a complete P(E) distribution and the user features that are used to make adjustments. In the user interface, the green vertical lines are the GT time marks, and the red vertical lines are labels for adjustment. P(E) distribution is represented by the solid black curves. The GUI allows for meal addition and deletion. After raters making any changes to the red vertical lines, the GUI will document the adjusted BP in a separate file for future evaluation. Note that, the EatMon has a resolution of 10 sec/pixel or each pixel on the timeline represents 10 seconds of the P(E) distribution. So, the minimum adjustments that each rater could make is 10 seconds.

For training, two raters without previous knowledge about the schema have been given a 10-minute session on how to use the EatMon GUI to make label adjustments, and a 20-minute

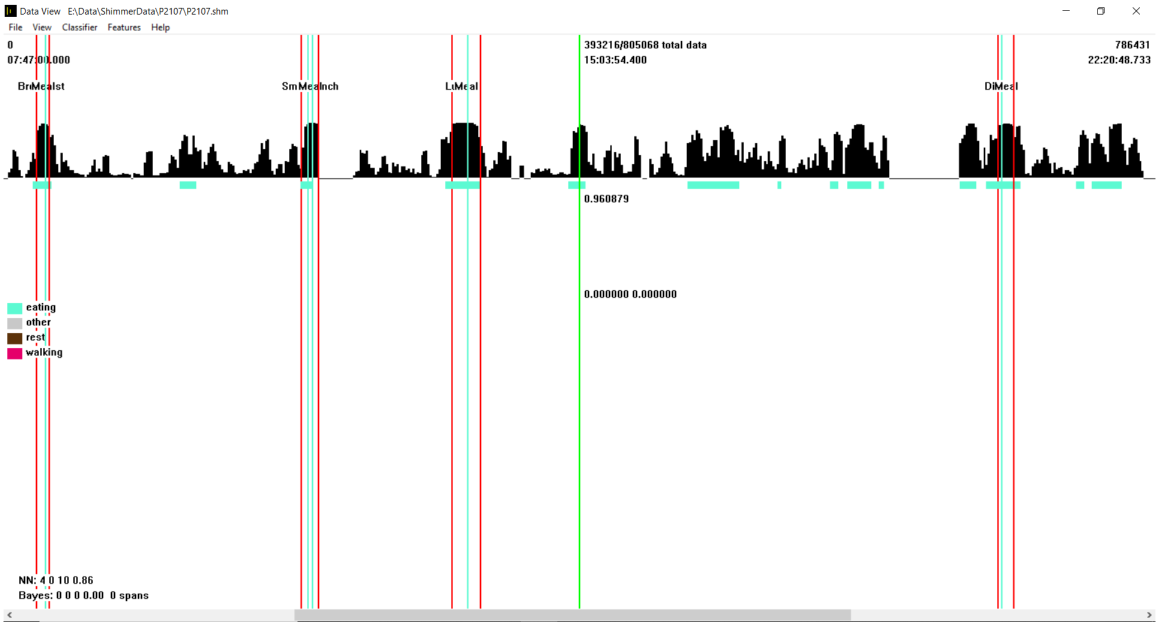


Figure 2.7: A demonstration of EatMon GUI. The GUI displays the GT by pairs of green vertical lines. The red vertical lines stand for the adjusted GT and can be manipulated by mouse click (see more in text for details).

session on how to use the schema and follow the algorithm to identify any suspicious GT and fix it accordingly. Two files other than the selected 71 files are picked for demonstration, to show the two raters a whole process in action. Two raters are then given a week to finish adjustments on the 71 files, though making adjustments on those files will take less than 1 hour. Raters are asked to work on the adjustments independently and only rely on the schema and the algorithm provided. Totally, there are 227 GT meals and 454 GT BP (2 BP for each meal). After the 3 raters finishing their adjustments on all the GT meals, we collect the adjustments files and do further evaluations. To measure the disagreement between raters, we evaluate the number of BP adjustments and the adjusted time resulted from the BP adjustments, which will be elaborated in section 2.4.2 and section 2.4.3 respectively.

2.4.2 Inter-Rater Reliability Evaluation by BP

For evaluation of BP adjustments, we compare each rater's BP with the original GT BP. Raters either agree that the original GT BP needs to be altered or should be remained as it is. We evaluate the IRR by counting the numbers of altered or remained GT that raters agree on. Although

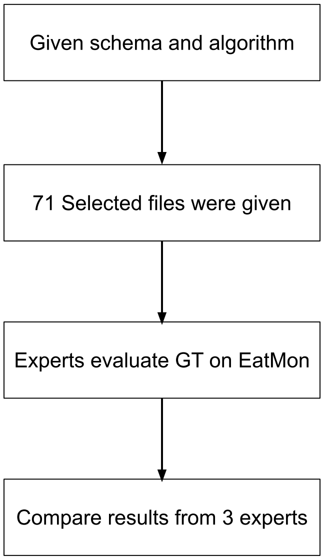

Given schema and algorithm
71 Selected files were given
Experts evaluate GT on EatMon
Compare results from 3 experts

Figure 2.8: Process walk through for inter-rater reliability

an intuitive way to compare BP adjustments would be to compare BP time stamps and see if they are equal, we decide it is not possible to have a fair comparison in this way to reflect the true IRR. In the process of raters' adjustments using EatMon, it is very likely that two raters do not make exactly same adjustments (one rater could click in one place, and another rater might click on a time stamp nearby). Also, some times raters have minor adjustments that have minimum significance. For example, move a BP for 2 minutes on a 30 minutes meal period to match a P(E) peak perfectly, or raters could changes a BP by 20 seconds just to match a P(E) peak. Minor adjustments do not change original GT interval or is making minimum impact. When measuring IRR, we want to skip any minor adjustments and reflect the true judgement of a rater on BP adjustments. To do this, we put each BP under the context of a meal segment, and compare the centroids of the meal segments, so that we could give tolerance to filter out any minor adjustments. The calculation of the centroid could be formulated by equation 2.3:

$$Centroid = \frac{Start - End}{2} \tag{2.3}$$

Since we are intended to compare single BP instead of an interval , each time we fix one rater's BP with its GT BP counterpart, and form a new meal segment for comparison. In figure 2.9, we show the comparison between one rater adjustment and the GT interval. When comparing the first BP (BP1), we fix the rater's BP1 with the ground truth BP2 and forms a new meal segment to compare with the GT interval. Similarly, when comparing the second BP (BP2), we fix the rater's BP2 with the ground truth BP1 and compare the resulted meal segments.

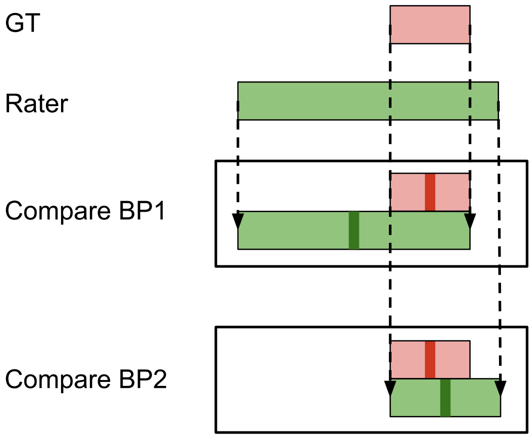


Figure 2.9: A demonstration of meal segments centroid comparison. The green rectangle represents the meal segment made by one rater, and the red rectangle represents the original GT segment. Rater has made a major adjustment on BP1, and rater has made a minor adjustment on BP2 (see text for more details).

After making the new meal segments, we compare the rater's adjustments against the original GT by centroid. The key idea is that, if the new meal segments and the original GT have included each other's centroid, we consider the rater's BP as a minor adjustment and thus rater does not alter the BP. If any of the two centroids (the original GT segment and the rater's adjusted BP segment) do not lie within the range of the other meal segments, we consider the rater has made a major adjustment and the original GT BP needs to be altered. In figure 2.9, when comparing BP1, the rater's adjustment has clearly shifted the centroid out of the GT interval, which is considered as a major adjustment. When comparing BP2, the rater's adjustment does not shift the centroid outside the GT interval, and the GT's centroid is also within the range of the rater's interval, so the rater's BP2 adjustment is considered as minor. We will evaluate the number of BP, out of 454, raters agree to alter, and the number the raters agree to remain.

2.4.3 Inter-Rater Reliability Evaluation by Adjusted Time

Besides counting the number of the adjusted BP, we also do adjusted time comparison to evaluate how long in time does the rater agree to alter. In this comparison, we count the total time length that each rater converts from GT eating to non-eating and vice versa. In the given 71 files, the total GT non-eating time is 895 hours and the total GT eating time is 53.9 hours. We measure the exact time length that each conversion happens for each rater. We also count the time length that raters intersect with each other without considering the original GT, which is aimed to evaluate the disagreement between raters. In figure 2.10, we show a legend for how this evaluation is defined. In the figure, we do not care about the start and the end time of the original GT or the rater's meal segments, we only care about where they overlap or disagree. For the area that the original GT does not label as eating and the rater suggests an eating period, we say the rater has made a conversion from non-eating to eating. The same judgements are made for conversion from eating to non-eating. The overlapped area would be the remaining correct GT.

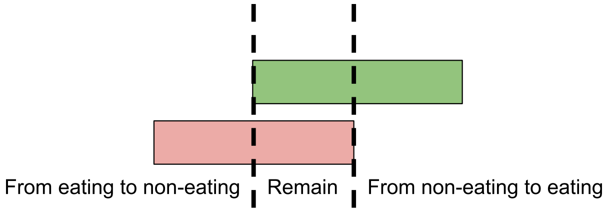


Figure 2.10: A demonstration for IRR evaluation in time metrics. The red rectangle is the GT meal period, and the green rectangle is the rater's adjusted meal segment. The time period where two meal segments overlap will be remained, and the other area is converted to what rater suggests.

2.5 Evaluation of the Adjusted Labels on Deep Learning Model

In order to evaluate the improvements gain by adjusting the original GT, we adapt the model and evaluation metrics from Sharma[30] which is elaborated from section 2.6 to 2.9. We use the model to test how adjusted labels could improve model performance.

To evaluate the effectiveness of the adjusted labels, we use all 354 files from the CAD data set and their adjusted labels. A total of 1,063 GT meals or 2,126 BP are included in our evaluation. Before going into details of the model, the figure 2.11 gives an overview of the baseline model evaluation (benchmark). We start by dividing the 354 files into 5 different folds with each fold containing 70 or 71 files. An original model is trained with the 4 folds containing the original GT labels and evaluated on the left-out testing fold (fold 5) with the original labels. The training set is a balanced data set with equal number of eating class and non-eating class. The testing set incorporates all data containing in fold 5.

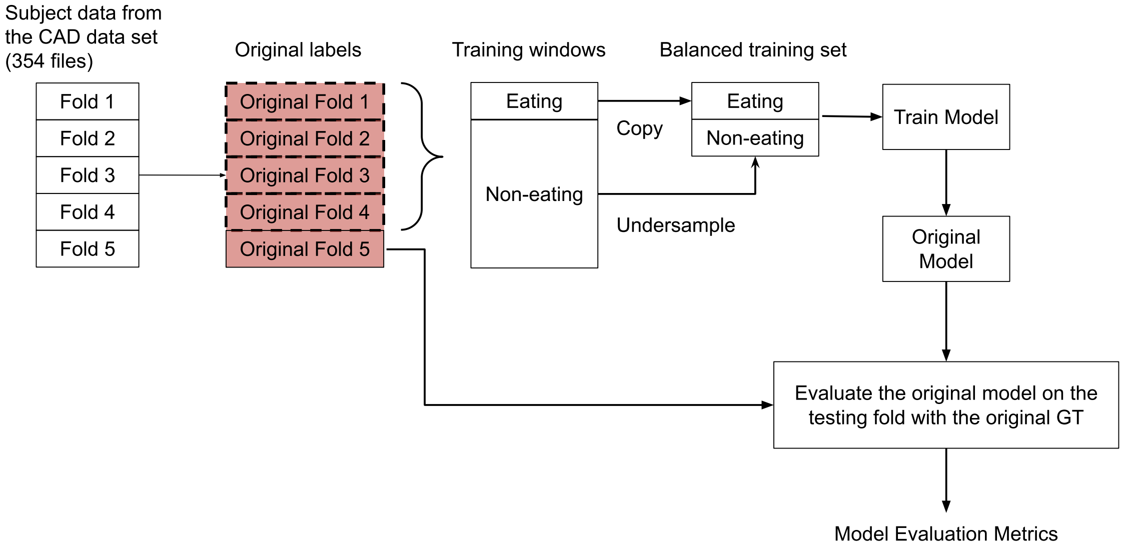


Figure 2.11: The process for training and evaluating the original labels on model.

On the other hand, the figure 2.12 gives an overview of our evaluation process for the adjusted model. Similarly, we start by dividing the 354 files into 5 different folds with each fold containing 70 or 71 files. And then we use the algorithm in figure 2.6 to adjust labels for all GT meals. We create a balanced training set from 4 folds of the adjusted labels. Finally, an adjusted model is trained and evaluated on the left-out testing fold of containing the original GT labels. We then use various evaluation metrics to exam the model performance. Similarly, an original model is trained with the four folds containing the original GT labels and evaluated on the left-out testing fold with the original labels. The original model will be evaluated with the same metrics as the adjusted model.

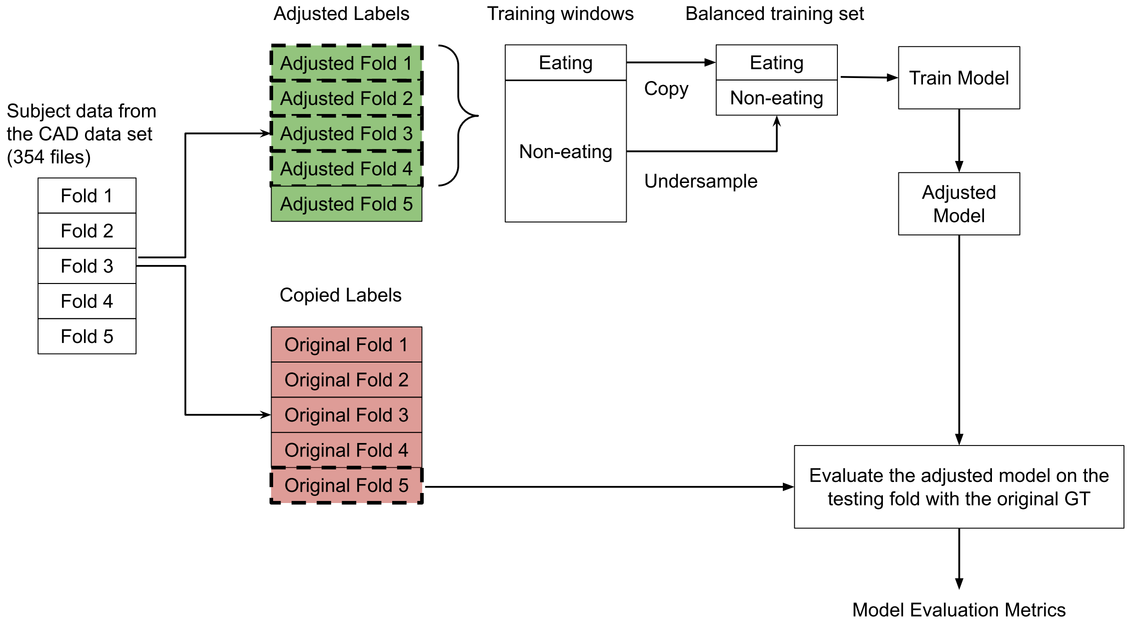


Figure 2.12: The process for training and evaluating the adjusted labels on model.

2.6 Data Preprocessing

In this section, we will introduce data preprocessing techniques used for this work.

2.6.1 Gaussian Smoothing

Before sending data into the classifier, we perform data smoothing with the Gaussian smoothing method. In figure 2.13, we compare the shape of raw data and data after Gaussian smooth. The raw data used is a accelerometer x-axis segment from the CAD data set file. The raw data is visually noisy, and the Gaussian smooth carries out the key features of the raw data but has filtered out the confusing noise. For example, the two peaks around index 300 is carried out perfectly, as well the valley before the first peak. For this thesis, the parameters used in Gaussian smooth is adapted from the previous work. Variance is set to 10 and the kernel length is 15. We implement the Gaussian smoothing on each of the six-axis.

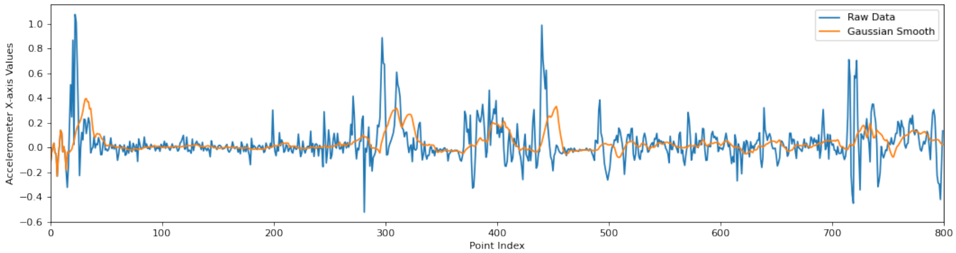


Figure 2.13: A comparison between raw data and raw data after Gaussian smooth. The Gaussian smoothed has kept major features of raw data and eliminated minor noise.

2.6.2 Normalization

Besides Gaussian smoothing, we also apply normalization to the the 6 axes data in the CAD data set. In figure 2.13, we could see how unsettling raw data could be from a motion sensor. Not only the raw data fluctuating in high frequency but also has a wide value range due to bias. The goal of normalization is to minimize the bias contained in the data set and scale the raw data into a certain range. In this work, we use the z-score normalization to rescale data according to the axis-specific standard deviation and the average value. Each normalized data point is given by:

$$x_{normalized} = \frac{x - \bar{x}}{\sigma} \tag{2.4}$$

In equation 2.4, $\bar{x}$ and σ are the mean value and the variance of the current axis respectively. Previous work has proved the effectiveness of z-score normalization over other normalization methods. This work we will adapt the same approach. Note that, we apply the normalization after Gaussian smoothing.

2.6.3 Sliding Window

The CAD data set is collected with 15 Hz sample rate. In other words, each second has 15 data points, which can hardly describe any human movements. In this work, we adapt the S (stepsize) and T (window length) combinations from the previous work that has the best experimental results on the classifier. We will use 15 seconds for sliding window stepsize and 6 minutes for the window length. In this case, 6 minutes or 360 seconds, will cover 360 × 15 for each of the 6 axes, which leads to a total of 32,400 data points. We apply the sliding window to the CAD data set and generates around 95,000 samples for each model training phase. Each sample label is decided by the majority labels contained under the current window. For example, if a window contains 5400 datum, it will be defined as ”eating” if more than 2,700 datum has label ”eating” and vice versa.

2.7 Neural Network

This section will introduce the convolutional neural network (CNN) used for this work, as well as the parameter used for the CNN.

2.7.1 Convolution Neural Network Architecture

The previous work has constructed a decent CNN architecture for use. In figure 2.14, we show a flow chart of the whole process from preprocessing raw data, going through CNN, and finally producing output P(E). After raw data going through the data preprocess, i.e. Gaussian smoothing, normalization, we uses the sliding window approach to extract the raw in the dimension of 6×5400, recall this means 6 axes with each axes covers a time window of 6 minutes. The data sample is then sent into the input layer or the convolution layer with parameter set as (filter length=44, depth=10, stride=2). The data sample will go through the architecture and eventually come out from the fully connected layer as an individual output P(E). The architecture will judge the final class of the current label by the output P(E), and can be formulated as:

$$Class\ Prediction = \begin{cases} \text{eating} & P(E) > 0.5 \\ \text{non-eating} & \text{Otherwise} \end{cases} \tag{2.5}$$

In figure 2.15, we compile this model with tensorflow keras API, and the summary gives the detailed information about shapes as well as the trainable parameter. The architecture takes up 7,471 parameters for optimizing, and about 120 GB of RAM to run the complete CAD data set.

2.7.2 CNN Hyperparameters

1. **Training Epochs**: Training epoch is defined as one pass through all individual samples generated from the CAD data set. Theoretically speaking, a lengthy training epochs, for example, 1,000 epochs will not affect the resulted model. The bottom line is that we need to see the CNN convergence before we could stop. At convergence, the validation rate will

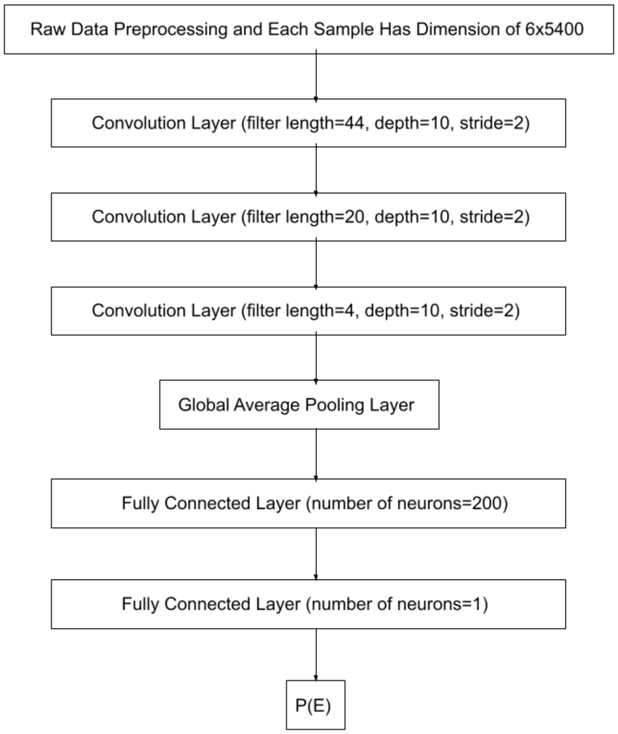
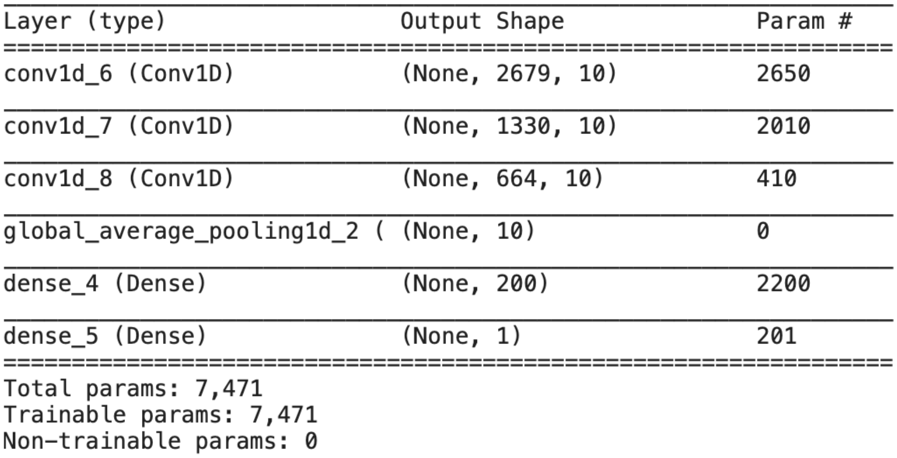

Raw Data Preprocessing and Each Sample Has Dimension of 6x5400
Convolution Layer (filter length=44, depth=10, stride=2)
Convolution Layer (filter length=20, depth=10, stride=2)
Convolution Layer (filter length=4, depth=10, stride=2)
Global Average Pooling Layer
Fully Connected Layer (number of neurons=200)
Fully Connected Layer (number of neurons=1)
P(E)

Figure 2.14: A summary of CNN architecture in a flow chart adapted from Sharma[30].



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_6 (Conv1D)            (None, 2679, 10)          2650
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 1330, 10)          2010
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 664, 10)           410
_________________________________________________________________
global_average_pooling1d_2 ( (None, 10)                0
_________________________________________________________________
dense_4 (Dense)              (None, 200)               2200
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 201
=================================================================
Total params: 7,471
Trainable params: 7,471
Non-trainable params: 0
```

Figure 2.15: A summary of CNN architecture at the code compile time. Total number of parameters for training is 7,471 and containing 6 layers.

become steady at a certain number. In this work, we set the training epochs to be 150, which is long enough for our model to converge.

2. **Batch Size**: Batch is a unit of training samples getting by dividing the whole training set into smaller units. In the process of assembling a batch, we do not change any contents or structure of the original data samples. However, the batch incurs loss as a whole, which means, each batch will be sent into the model and the model will not update its parameters until the batch is depleted. Subsequently, a bigger batch could help with quick convergence but also fewer iteration of back-propagation. A large draw back of big batch size is that, the classifier may converge to a local optimal instead of a global optimal. Thus, not carrying the best performance. A small batch size, on the other hand, will force the classifier to "see" the samples one by one and update weights accordingly. The running time of small batch size is quite slow but could sometimes get better results than a larger batch. In this work, considering the trade off, we use batch size of 256.

3. **Learning Rate**: The learning rate is a parameter used in the optimizer. It decides the portions of update that will be applied on the parameters. A large learning rate will accelerate the converging speed, however, have the risk of skipping the targeted optimal and make the classifier diverge. A small learning rate would be a better option for a good convergence, however, could make the training time too long to be practical. In this work, we use learning rate of 0.01 which proves to be effectively leading optimizer towards optimal while maintaining a reasonable convergence speed.

4. **Optimizer**: The optimizer is the formula we use to update the neuron parameters according to loss. In this work, we use the adaptive moment estimation (Adam) for our optimizer. This method is built on two gradient descent methods: momentum and root mean square propagation (RMSP) and combining their characteristic together. In summary, Adam optimizer is able to approach the optimal parameter values with the minimum step size so it will not miss it, while keep a big enough step size to walk over any other shallow minimum without the parameters being trapped in it. In equation 2.6 and 2.7 are the two methods to calculate the

aggregate of the gradients at a given time t:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)[\frac{\partial L}{\partial w_t}] \tag{2.6}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)[\frac{\partial L}{\partial w_t}] \tag{2.7}$$

The variable $\beta_1 = 0.9$ is $\beta_2 = 0.999$ are decay weights with respect to iterations. The parameter w_t is the target parameter that we update. At each iteration t, the Adam optimizer calculate m_t and v_t, and use equation 2.8 and 2.9 to avoid any bias (the two β values starts from a value close to 1 and decays gradually together):

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{2.8}$$

$$\hat{v_t} = \frac{v_t}{1 - \beta_1^t} \tag{2.9}$$

In the end, we use equation 2.10 to update the targeted parameter w. The variable ε is a very small positive number to prevent the divisor become 0 and has no actual meaning in the equation.

$$w_{t+1} = w_t - \hat{m}_t(\frac{\alpha}{\sqrt{\hat{v_t}} - \varepsilon}) \tag{2.10}$$

5. **Early Stopping**: Sometimes, the training epoch becomes too long that the classifier overfits the given data set. Thus, causing the performance to decrease. We use the checkpoint model at the training iteration when the classifier reaches its highest validation accuracy, which equivalent of stopping the model training. The checkpoint feature is provided by the tensorflow API. In this work, we give the classifier a patience of 20 epochs, and use the best model trained from the past epochs.

6. **Hardware**: The architecture is ran on Clemson University Palmetto cluster with 2 CPU, each with 120 RAM. Besides, this work uses two Nvidia Tesla P100 graphical processing units. The total running time for training the models are 30 minutes.

2.8 Hysteresis and Meal Segmentation

The class prediction from the CNN architecture is not accurate due to the naive approach of categorizing them by threshold of 0.5. We need a more complex method to look for meal events from the model output. In the previous works, P(E) is used to find meal segment through the complete files, which is defined as the hysteresis algorithm. In the hysteresis algorithm, we process a complete P(E) distribution and find the meal segment with a meal starting threshold T_s and a meal ending threshold T_e.

Recall in section 2.6.3, similarly, we use the sliding window approach to generate consecutive data samples for a given file. Instead of using a step size of 15 seconds, we move the sliding window one data point ($\frac{1}{15} seconds$) at a time, and send the data samples to the trained model for P(E) distribution. By connecting the P(E) together, we get a complete distribution for the file, which is the same as the file shown in section 2.3.2. Based on observations, when a meal starts, the starting point usually demonstrates a high probability, while the end point usually demonstrates a low probability. Based on this observation, we find the best T_s that signals the start of a meal and a T_e for meal ending. Previous work has examined an abundant combinations of T_s and T_e. For this work, we adapt the combination that gives the best results $T_s = 0.8$ and $T_e = 0.4$. In figure 2.16, a demonstration of the hysteresis shows how we get the meal segment from a P(E) distribution. The blue rectangle at the bottom represents the classifier detected segment. The black dash lines represent the begin and the end of the segment.

Based on the meal segments from the P(E) distribution, we further process these meal segments, since various problems may emerge. A common problem of this method is over segmentation. The classifier detections will get extremely small meal segments or a number of discrete meal segments which could be group together as a whole. Redundant segments create visual confusion when used in practice and a lot of them do not compromise model's performance by getting aggregated. So, we adapt the same approach from previous work. This work will delete any meal segments under 1 minute, as well as combing segments that are under 1 minute apart. We use this hysteresis algorithm for all post-training analysis and evaluation, which will be further introduced in section 2.9.

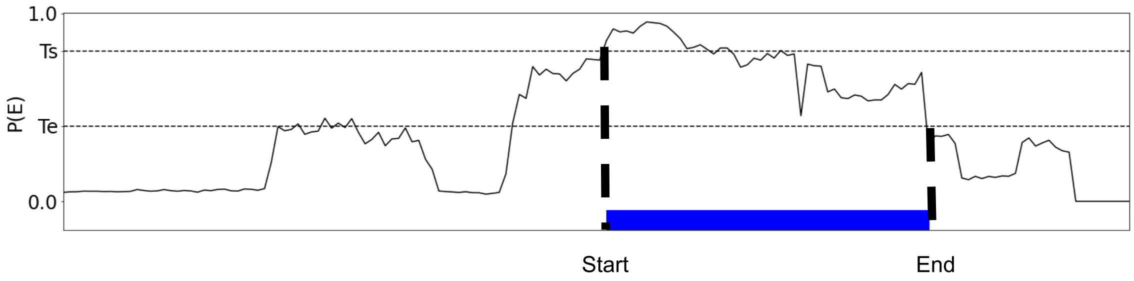


Figure 2.16: A demonstration of classifier detected meal segment. The black curve represents the classifier output eating probability P(E). The meal start P(E) is labeled by T_s (0.8) and the meal end P(E) is labeled by T_e (0.4).

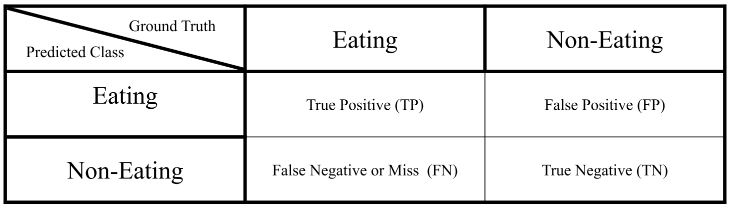
2.9 Evaluation Metrics

This section will introduce the evaluation methods used to examine the model performance. we perform evaluation on the model to measure how well it could detect meal events and generalize its result to unseen data. From here, we use unbalanced and timely ordered original data samples from the CAD data set. The CAD data set adjusts labels by using the algorithm in 2.6 will follow the exact same evaluation process for comparison. The main points of this section covers: time metrics in section 2.9.1, episode metrics in section 2.9.2, and 5-folds cross validation method in section 2.9.3.

2.9.1 Time Metrics

Time metrics measures the accuracy of individual data points. The evaluation is based on the meal segmentation method. The data points that are within the range of a classifier detected segment will be counted towards ”eating” class. Any data points that are outside the meal segmentation will become a ”non-eating” class. Based on the given ground truth from the CAD data set, we categorize the data points into true positive, false positive, true negative, and false negative. These could be presented by a confusion matrix shown in figure 2.17.

In figure 2.18, we show a ground truth and a classifier detection to demonstrate how we define the metrics. Note, since this work only considers the problem of detecting a meal period, we do not care about any true negative cases. As shown in the figure, the overlapped portion between the GT and the classifier detection is considered as TP. The portion of the detection that does not cover by ground truth will be FP. The ground truth that does not get overlap with any portion of



Ground Truth Predicted Class	Eating	Non-Eating
Eating	True Positive (TP)	False Positive (FP)
Non-Eating	False Negative or Miss (FN)	True Negative (TN)

Figure 2.17: The confusion matrix used to illustrate the TP, FP, FN, and TN for classifier time and episode metrics evaluation.

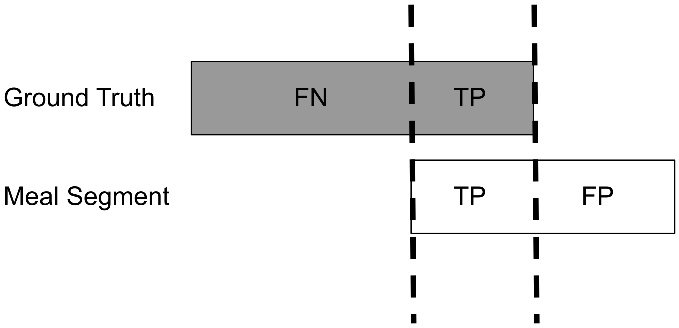


Figure 2.18: A graphical demonstration of time metrics evaluation including FN, TP, and FP datum.

detection is a FN or missed. With the knowledge of the basic definition, we use a weighted accuracy (W_{acc}) to evaluate the performance of the model on time metrics:

$$W_{acc} = \frac{TP \times 20 + TN}{(TP + FN) \times 20 + (FP + TN)} \tag{2.11}$$

In equation 2.11, we weight all ground truth eating samples with a positive 20. This is because in the CAD data set, we have 20 times more non-eating samples than eating samples. For a data set with unbalanced class samples, we should use the weighted accuracy to better describe the performance of the model. For example, without keep the weight as part of the equation, model that identifies 1 non-eating sample is equivalent to 1 eating sample. Since we have 20 times more non-eating sample, the classifier could get up to 95% accuracy rate without identifying any eating sample, which is clearly not reasonable. Based on this concern, we use the weighted accuracy to evaluate the performance of the model for time metrics.

2.9.2 Episode Metrics

Besides time metrics, we also perform evaluation based on episode metrics. Episode metrics measure the accuracy of the classifier detection as a complete period. In figure 2.19, we demonstrate how the classifier detections are categorized by ground truth. If there exists any overlap between the ground truth meal period and the detection, we will count the classifier detected segment as a TP. On the other hand, if a detection does not have any overlap with a ground truth meal, then the segments belong to a FP. Any ground truth that is missed by all classifier detections, is counted as a FN.

For episode metrics, it is meaningless to calculate the weighted accuracy like time metrics. Instead, we evaluate the model based on their truth positive ratio (TPR) and false positive to true positive ratio (FP/TP). The TPR is calculated by the following equation:

$$TPR = \frac{TP}{TP + FN} \tag{2.12}$$

In short, equation 2.12 calculates the percentage of identified meal out of total meals. $TP + FN$ is the total number of the GT meals and TP is the number of correct classifier detections. Besides TPR, we also measure the FP/TP ratio, and it can be formulated by the equation below:

$$FP/TP = \frac{FP}{TP} \tag{2.13}$$

The ratio is fairly straightforward to understand. We use this ratio measure the number of incorrect detection for every correct detection. A model could have high TPR demonstrates a strong ability to identify meal intervals, however, the model still has to have a reasonable FP/TP to be practical. This ratio mainly prevents the model from cheating the evaluation by giving excessive amount of detections while acutally being bad. A good model will have a high TPR while maintaining a low FP/TP ratio.

2.9.3 5-Folds Cross Validation

In our work, we use a 5 folds cross validation method (5FCV) for evaluation of the model's ability to generalize to unseen data. The CAD data set is split into 5 different folds before training begins. In figure 2.20, we evenly split the CAD data set into 5 folders . Since there are 354 files

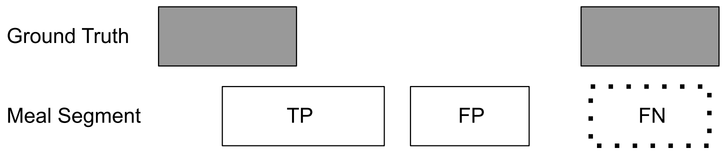


Figure 2.19: A graphical demonstration of episode metrics evaluation including TP, FP, and FN detections from the classifier output P(E).

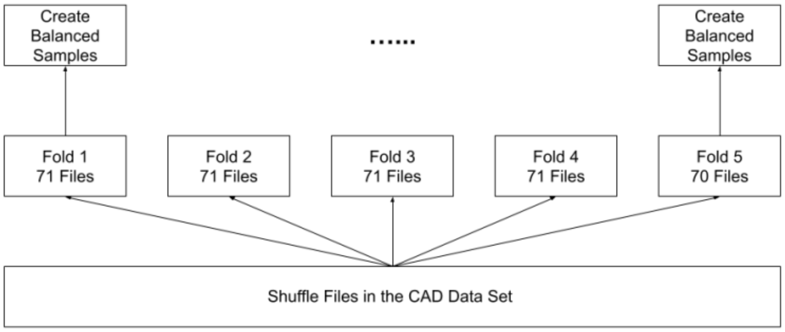
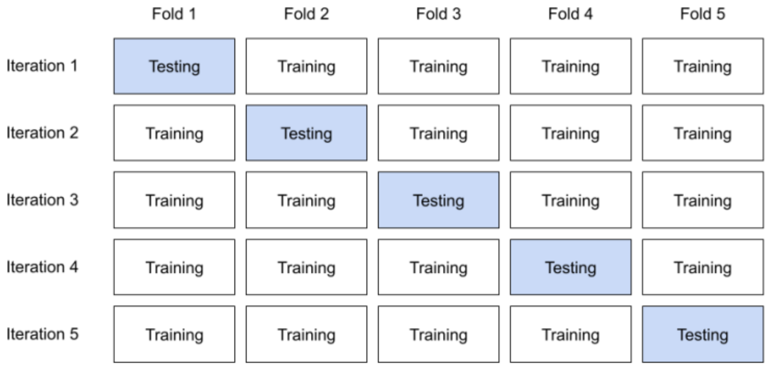


Figure 2.20: A graphical demonstration of creating 5 folders after data preprocessing.

in total, one of the 5 folders contain only 70 files instead of 71. After creating 5 folders, we create balanced samples from each folder. So, at the end of this first step, we will have 5 folders of balanced samples and each folder contains different number of samples.

Moving into the training phase, each time, we use four folders of data sample for our training set and the last folder of balanced data will be used as the validation set. During evaluation phase, we will collect unbalanced and ordered data samples from the files of validation folder. In other words, we need 5 models to be able to evaluate over all files for the CAD data set. We show this process in figure 2.21, each combination of training set and validation set is iterated by once. At the end of the process, we will take the TP, FP, FN from each folder and combine them together to get the final model performance evaluation.



	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Iteration 1	Testing	Training	Training	Training	Training
Iteration 2	Training	Testing	Training	Training	Training
Iteration 3	Training	Training	Testing	Training	Training
Iteration 4	Training	Training	Training	Testing	Training
Iteration 5	Training	Training	Training	Training	Testing

Figure 2.21: A demonstration of 5 folds cross validation used to evaluate the classifier's ability to generalize to unseen data.

Chapter 3

Results

This chapter presents the results of this work. In section 3.1, we present the result of IRR tests, demonstrating the reliability of the schema and the algorithm. In section 3.2, we analyze the adjusted labels versus the original labels, providing data to illustrate the change made to the original labels. In section 3.3, we compare the model performance trained by the adjusted labels and the model trained by the original labels, which reflects the improvements gain from the adjusted labels.

3.1 Inter-Rater Reliability

The inter-rater reliability test (IRR) is done among three raters, with two raters A and B do not possess any prior knowledge, and C has more knowledge. The following results are collected on 71 given files, containing a total of 227 GT meals and 454 BP. First, we present a summary on the three raters. In table 3.1, rater B and C have more aggressive approach on adjusting GT, which rater A has more conservative adjustments.

	Alter (%)	Remain (%)
A	42	58
B	59	41
C	51	49

Table 3.1: BP adjustments for each rater on 454 GT BP, with rater B being the most aggressive on adjustments.

We count the number of changes each rater has made and plot them according to the adjustments length. In figure 3.1, three raters have shown a similar distribution. However, rater C has less adjustments made in 2-4 minutes, and the distribution is more spread out comparing to the other two raters.

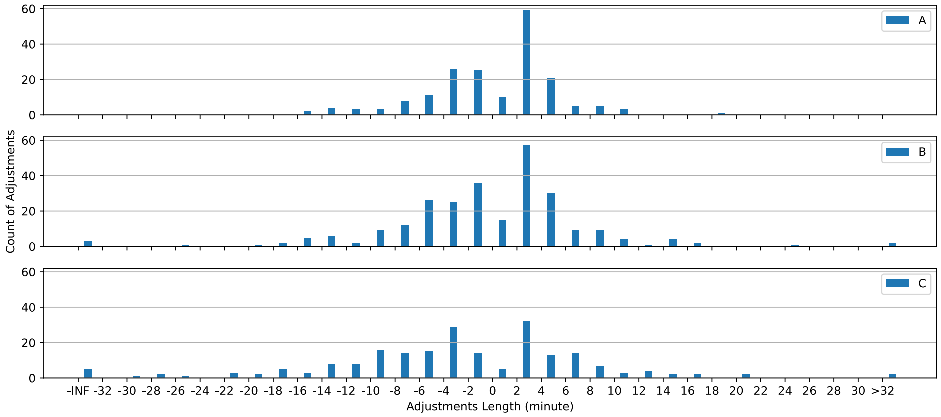


Figure 3.1: Counts of adjustments with respect to adjustment length (in minutes) for three raters. Three raters shown similar distributions with more adjustments made between -6 minutes to 6 minutes. Rater C has less adjustments made for 2-4 minutes.

	Alter Concur (%)	Remain Concur (%)	Total Concur (%)
A and B	40	40	80
C and A	31	40	71
C and B	42	34	76
Three Raters' Intersection	31	33	64

Table 3.2: Three raters have concurred on 31% of bad BP that needs to be adjusted, and concurred on 33% of correct GT BP. Each of the two raters have shown more than 70% agreements.

We also count the number of BP adjustments that raters agree on. In table 3.2, we show the percentage of agreement for each combinations of two raters, and three raters all together. Three raters share an agreement over 60%. The agreement is lower than our expectation because we suspect each rater has his or her own style on adjustments. Three raters could concur on some meals being obvious error or good GT, however, for some meals that are ambiguous, three raters tend to make different judgements. We also provide a comparison of adjustments over time. In table 3.3 and 3.4, we show the hours that each rater adjusts from non-eating (NE) to eating (E) and vice versa. Three raters agree on 8.2 hours of GT NE should be adjusted to E, and 2.7 hours of GT E should be adjusted to NE. In the table 3.4, rater C has more aggressive approach and adjusted double the amount of rater B, quintuple the amount of rater A. Eventually, about 30 hours of E time is adjusted to NE time by rater C. In table 3.3, all three raters share a more conservative approach. Rater B and rater C adjust around 2% of the NE time to E time. Although rater A adjusted even less, around 1.5% or 13.3 hours of NE time is still adjusted. Note that, since we have a strict definition on adding missing meals to the original GT, 3 raters have only added 1 missing meals out of 227 meals. Although all 3 raters agree on adding the meal, it only has limited impact on our analysis.

	GT NE (hour)	NE to E (hour)	NE to E (%)
A	895	13.3	1.5
B	895	18.7	2.1
C	895	17.3	1.9
Intersection	895	8.2	1

Table 3.3: A summary of three raters' agreement on GT non-eating (NE) adjustments based on time. Three raters agree on 8.2 hours of GT NE labels should be adjusted to eating (E), which is 1% of the total GT NE hours.

	GT E (hour)	E to NE (hour)	E to NE (%)
A	53.9	3.6	6.7
B	53.9	8.6	16.0
C	53.9	16	29.7
Intersection	53.9	2.7	5.0

Table 3.4: A summary of three raters' agreement on GT eating (E) adjustments based on time. Three raters agree on 2.7 hours of GT E labels should be adjusted to non-eating (NE), which is 5% of the total GT E hours.

In figure 3.2, we provide examples when raters disagree or agree with each other. For GT in 3.2(a) and 3.2(b), rater C makes adjustments that differ from the other two raters. The original GT has short periods, rater A and rater B make no adjustments or minor adjustments, while rater C makes major adjustments on BP2 to line up with the P(E) peak. For GT in 3.2(c), rater A and rater C decide to shift the GT to the nearest P(E) peak, but rater B does not think the P(E) peak is a valid classifier detection for the original GT and the original GT should be left as it is. In GT in 3.2(d), rater C has made adjustments differ from the other two raters. In this case, we have a longer P(E) peak than the original GT. Since the valley demonstrates a lower P(E) which might signal that the current period is not eating, rater A and B stops to match the peak. On the other hand, rater C classifies the long P(E) peak as a complete meal, and adjusts the starting BP to line up with the P(E) peak. Lastly, GT in 3.2(e) has a short period and all raters agree that GT should be extended to match the peak.

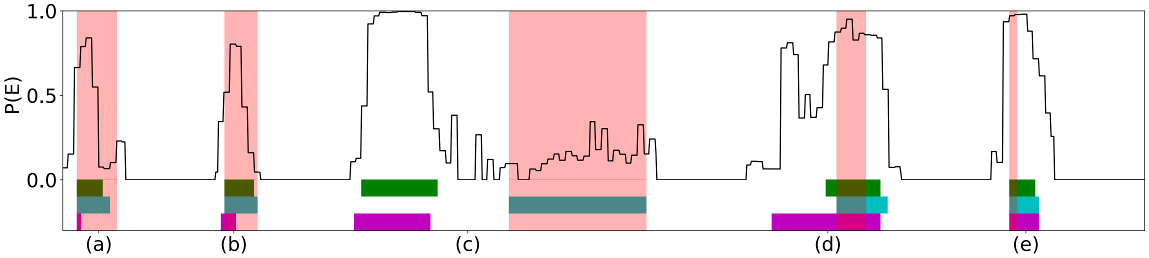


Figure 3.2: Adjustments made by rater A (green), rater B (blue), and rater C (purple) are compared against the GT meal (red) in 5 cases. For GT (a), (b), (c), and (d), three raters do not share a consensus for the adjustments. On GT (e), all three raters reach an agreement (see more details in text).

3.2 Analysis of GT Errors Prevalence in the CAD Data Set

In this section, we we compare the adjusted labels with the original labels, and analyze how much errors we have found in the original GT. This section includes the GT BP errors we have found across 354 files or 1,063 meals. We make adjustments to all erroneous GT according to the schema, and in table 3.5, we provide the number of GT meals adjusted in each category. The total number adds up to 1,063. We have 41.5% of correct meals ("nicely aligned"), and for the other 58.5% of the meal we are either unsure about if there is an error ("cannot identify", "unusual wrist motion", or "GT interval squeezed") or we are confident the GT is wrong based on the classifier detections ("BP slightly off", "not continuously eating", and "misidentified meal"). In table 3.6, we summarize error GT meals under the unit of BP. Out of 2,126 BP, 62.1% of them is correct and does not need any adjustments. The other 37.9% BP appears to be possible errors. Note that, error ratio of BP is much lower than that of the meal unit. This is because, categories with obvious errors such as "BP slightly off", could have 1 error BP instead of 2. A meal is not necessarily adjusted on both BP. In table 3.7, we analyze the adjustments under the time unit, and their effects on "eating" and "non-eating" classes. About 83.1 hours of total GT eating is adjusted to non-eating, and about 55.4 hours of non-eating is converted to eating. Note that, although about 33.2% or 83.1 hours of GT eating is adjusted, we also have 55.4 hours of non-eating adjusted to eating, which has maintained an abundant data samples for the "eating" class.

Category	Count	Percentage (%)	Summary (%)
GT Nicely Aligned	441	41.5	41.5
BP Slightly Off	306	28.9	43.8
Not Continuously Eating	140	13.2	
Misidentified Meal	18	1.7	
GT Interval Squeezed	26	2.5	14.8
Unusual Wrist Motion	54	5.1	
Cannot Identify	77	7.2	
Total	**1063**	**100**	**100**

Table 3.5: Label adjustments summary based on categories. About 41.5% of GT meals is correct and does not need any adjustments. About 43.8% of the GT meals contains fixable errors. About 14.8% of the GT meals cannot be analyzed by the current method.

Category	Count	Percentage (%)
Correct BP	1320	62.1
Error BP	806	37.9
Total	**2126**	**100**

Table 3.6: Total error BP and correct BP. More than 60% of BP is correct.

	Total (hour)	Adjusted (hour)	Percentage (%)
Eat to Non-eat	250.3	83.1	33.2
Non-eat to Eat	4453.8	55.4	1.2

Table 3.7: A summary of GT meal adjustments based on time. About one third of the eating time is converted to non-eat and around 1 percent of the non-eat is converted to eat.

Also, we notice that the order of the subjects participating in the data collection is not a factor that creates the noisy labels. We have divided 354 data files into 5 folds. In figure 3.3, we compare the adjustments in each folder by categories. All folds has a relatively good amount of correct GT that dominates the other categories. The second largest category is "slightly off" which requires minor modification by adjusting BP. Thirdly, the "not continuously eating" category has a notable cluster. A lot of this category can be fixed by simply divide the meal period according to detected meal segments. The rest of the categories are the minorities in our adjustments. From the figure, we could tell that each of the 5 folds is following similar distributions across the categories, proving the consistency of our schema across all folds. We suspect that, at early stage of the data collection, the collector might be inexperienced and thus causing more noise. Based on the similar distribution in figure 3.3, we are confident to say that the order of the subjects participate in the data collection, is not a factor that creates the noisy label.

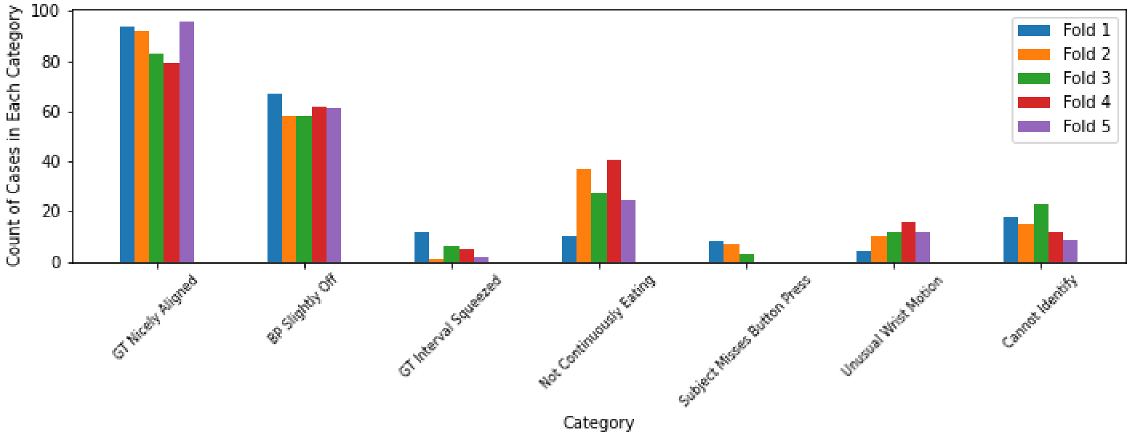


Figure 3.3: A comparison between 5 folds for each of the 7 categories from the schema. 5 randomly assembled folds demonstrate a similar distribution on 7 categories (See text for more details).

After making adjustments to the original labels, the total meal number shrinks from 1063 to 1051. In figure 3.4, we compare the distribution of GT meals and adjusted meals by plotting the number of meals with respective to their length. We could observe a clear shift of the distribution from longer meals to shorter meals, with 0-4 minutes meals being reduced. This is probably due to a lot of original GT contains delayed button press. The number of meals over 30 minutes is also greatly reduced. We suspect that the large number of adjustments happening in long meal is because subjects do not press button when they are in rest, and some of the subjects simply forget about the button press. Most of the meals is clustered around 4-12 minutes, for which is likely to be a normal human eating length. Throughout the process of adjusting error GT, we do not encounter much of the missing meals. Since we have put a strict limitation on adding a new meal, the total new meal that we have added to the original GT is 4, which has limited impact on the analysis.

3.3 Model Performance

In this section, we compare model trained on the adjusted labels and the original labels. In the table 3.3, by comparing the baseline model with the adjusted model, the adjusted model demonstrates an improvement in weighted accuracy, about 1.0%, and FP/TP ratio, from 1.9 to 1.7. However, the adjusted model has made a trade off between FP/TP and TPR. With a better FP/TP

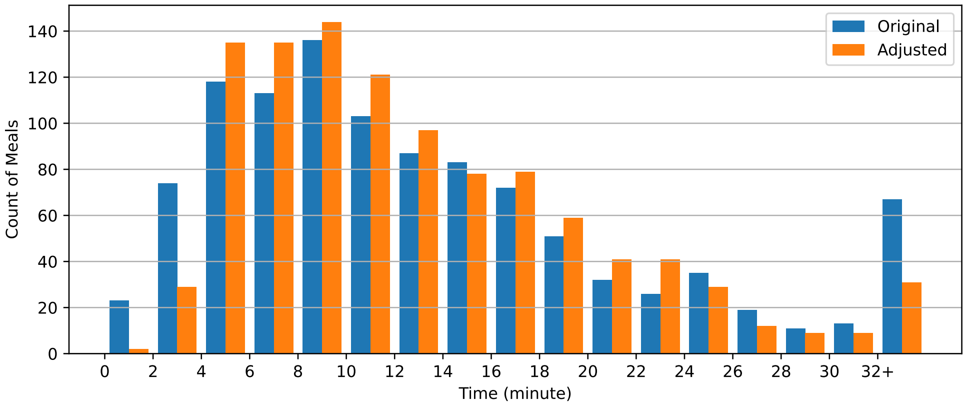


Figure 3.4: Meals of lengths 0-4 minutes are greatly reduced. Meals of length >30 minute are reduced. Meals of length 6-12 minutes are increased. (See text for more details)

ratio, the TPR performance is degraded by 1.6%, or identifies 17 meals less than the original model. We also show a distribution of the classifier detections for both the baseline model and the adjusted model in figure 3.5 and 3.6 respectively. The adjusted model has become worse at identifying meals that are under 14 minutes. Instead, there are minor improvements for identifying the longer meals.

Training Set	Weighted Accuracy	FP/TP	TPR	FP	FN	TP
Original Label	79.1%	1.9	87.5%	1786	133	930
Adjusted Label	80.1%	1.7	85.9%	1571	150	913

Table 3.8: A comparison of performance between training with the adjusted label and original label

Furthermore, we show two examples where the adjusted model has improved performance and worse performance. Figure 3.7 shows the classifier results on subject A for original model (top) and adjusted model (bottom) respectively. The blue rectangle in the lower axis shows the identified interval (II) by the classifier. We see the adjusted model has done perfectly: not only all original GT is identified but also the border has lined up almost perfectly. The adjusted model yields no false positive meal across the time span. In figure 3.7 top side, we demonstrate the classifier results for subject from the original model. The first meal is not identified by the original model and 2 false positive meal segments are produced. Overall, the P(E) distribution from the original model is much noisier than that of the adjusted model. Next, we demonstrate an example where the adjusted model has actually become worse. In figure 3.8, we show the classifier results on subject B for

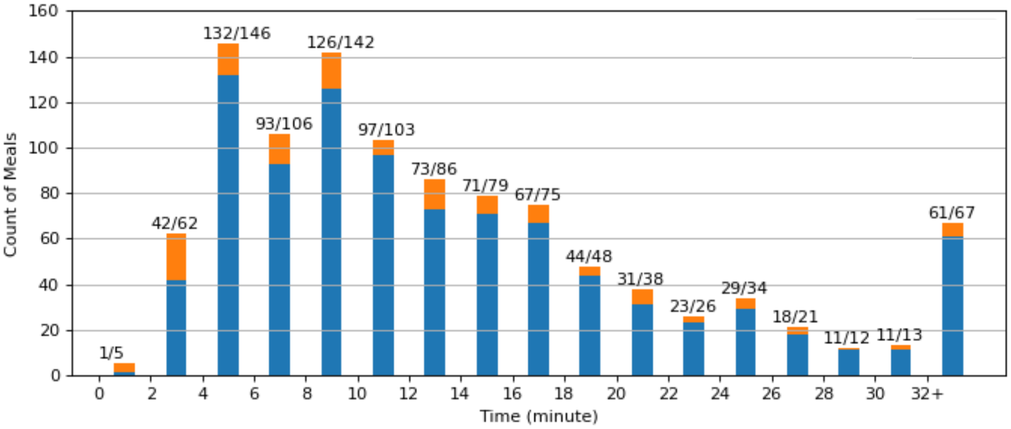


Figure 3.5: Original model detected segments (blue) versus total original GT (orange) distribution with respect to the meal length. Less percentage of meal in 2 to 4 min time interval is being identified.

original model (top) and adjusted model (bottom) respectively. The original model, though quite noisy on the second half of the recording, has successfully identified all GT meals, and yields 5 false positives. On the other hand, the adjusted model missed the second meal entirely and produced 6 false positives. Overall, we could tell the P(E) distribution from the adjusted model is cleaner than the original model, however, the performance on subject B has been degraded.

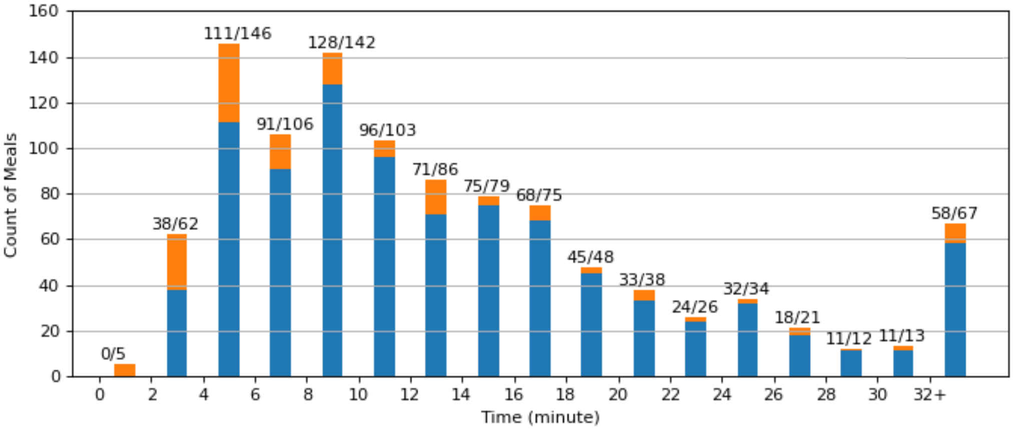


Figure 3.6: Adjusted model detected segments (blue) versus total original GT (orange) distribution with respect to the meal length. More percentage of long meal ($\leq 14min$) has been identified while less percentage of short meals ($< 14min$) are identified by the adjusted model.

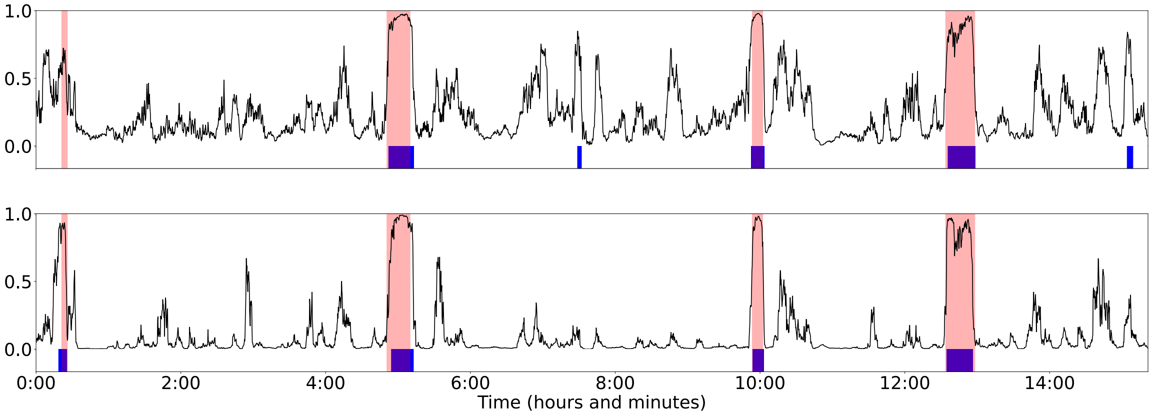


Figure 3.7: Top Graph: P(E) distribution for subject A from original model. Two detected meal segments (blue) are false positives. The classifier has also missed the first meal. Bottom Graph: P(E) distribution for subject A from adjusted model. All GT meals is identified by meal segments and no false positive is produced. The adjusted model has performed better.

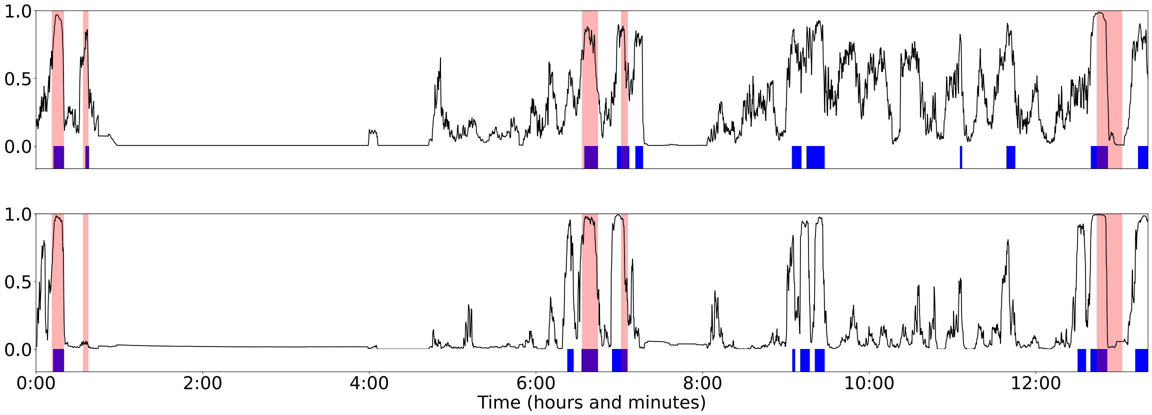


Figure 3.8: Top Graph: P(E) distribution for subject B from original model. Five detected meal segments (blue) are false positives. The classifier has identified all GT meals. Bottom Graph: P(E) distribution for subject B from adjusted model. The second GT meal is missed. Six detections are false positives. The adjusted model has performed worse.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

This thesis considers the problem of noisy labels in the eating data collected by wrist-worn device. We develop methods to reliably identify and minimize the damage of noisy labels in the CAD data set. In this section, we will discuss the results from IRR tests, the analysis to the prevalence of errors in the CAD data set, and evaluation of label adjustment method by model performance. We answer the questions from section 1.3 based on the results:

1. Can we reliably identify errors by using the existing ground truth and the classifier output probability of eating P(E)?

2. By using the method, how many errors do we think exist in the CAD data set?

3. By fixing these errors, how much improvements can we get from the classifier training?

First of all, we see from the IRR tests, each pair of the raters reach $> 70\%$ agreements on the adjustments, and three raters has a 64% agreement. Although the agreement is not perfect, raters agree on more than half of the adjustments by following the schema. More importantly, raters have similar distributions for the adjusted time length. We could say raters are making similar adjustments for the majority of the erroneous GT. The reliability of the schema could be improved but for now, we say we could reliably identify the erroneous GT by comparing the original GT with the classifier detections and P(E) distribution.

Secondly, by using the schema we have identified 37.9% or 806 total error BP. We have adjusted 83.1 hours of GT eating to non-eating, and 55.4 hours of GT non-eating to eating. In other words, more than 30% of the BP and the GT eating time is erroneous according to our method. We think the real error in GT is lower than this percentage, which requires us to update the schema to a better shape for more reasonable adjustments. For now, the answer to question 2 is about one third of the ground truth BP and the GT eating time is erroneous.

Lastly, we have evaluated the adjusted models by comparing the adjusted model and the original model on the original labels. By looking at the results, the adjusted model out performs the original model on the weighted accuracy by 1% and FP/TP ratio by 0.2. Thus, we conclude that the adjustments to the GT has indeed brought performance improvements to the classifier.

By analyzing all the results, we have realized that the schema should be updated to promote more caution in adjusting button presses, with a goal of perhaps 10-20%. The current schema caused raters to change 40% or more BP, which is likely too aggressive since that means about half of the meal periods is not recorded correctly. For example, figure 3.8 shows a snack is now missed after adjustments. It is likely that snack-length data is noisy to begin with, and that adjustments that remove it will hurt the model rather than help it. The problem with the schema may be that it overemphasized looking at classifier detected boundaries, instead of focusing more on the P(E) plot. In other words, the schema has converted a complex judgements into a binary scenario where raters emphasize too much on only detected meal segments and the GT meal. The original schema tried to simplify training raters by having them look at binary boundaries, to avoid having to interpret the P(E) curve. However, it may be that more training and a deeper understanding of the P(E) curve is necessary to better judge button press errors.

In summary, the schema can reliably identify the error GT from the original GT. The error GT is about one third of the CAD data set. Finally, by adjusting the error GT, we could achieve model improvement and accomplish better accuracy.

4.2 Limitations and Future Work

In this work, several limitations are encountered in the process. First of all, we have limited raters and limited time to do IRR. In the best case scenario, we could have more raters to perform adjustments and longer time to train the raters, so that we could see the places where raters have agreement and the places that schema is ambiguous. We could refine the schema if more time is given. Due to limitations, we only able to have 3 raters and perform adjustments on a portion of the data set. The algorithm could be biased due to the large percentage of adjustments. Secondly, we could also have more time on training raters to look at P(E) and make their judgement with more degree of freedom. The current schema and algorithm is biased towards the classifier detections alone. Thirdly, to make the process of adjusting GT faster, we have picked a low resolution on EatMon GUI to display the detections and the P(E) distribution, which limits the ability of the raters to make more accurate adjustments according to the schema.

Besides the limitations, we also expect some exciting future works on this topic. Throughout the experiment, we are excited to see how noisy labels may bring improvements to model performance, and may be detected by classifier outputs. Especially for the CAD data set, which is collected from the wild, is expected to have a decent amount of noise. To solve this challenge, this work has explored two ways, by fixing noisy labels with flipping individual datum labels, and by human raters adjust BP on for a contiguous sequence of data . Although the results have achieved slight improvements, there are still much to explore. For example, some research points out the effectiveness of using data analytic feature to help with deep learning. The classifier will learn not only the raw data samples but also the features of the data segments (mean, standard deviation, mode, median, and etc.), which helps the classifier to identify the noisy labels in the data set. Also, some groups take a small handful of clean data set to begin with, and make the classifier to learn from other noisy data. The intuition is that, the model will overfit to the clean data set, and the noisy data will be used as a way to correct the model. Although there are a lot of interesting works, the majority of them either requires a clean data set to begin with, or their training data is collected from a laboratory setting. For future work, more efforts need to be put in dealing with wild data. Instead of looking for short contextual window like classifier detections, we need more context like a complete P(E) distribution. Furthermore, we could refine and extend the current schema, which proves improvements on the model, to a more robust and automated process that enables us to adjust labels on each training iteration.

Bibliography

[1] How to measure acceleration. https://www.omega.com/en-us/resources/accelerometers. Accessed: 2021-10-20.

[2] Itg-3200 integrated triple-axis digital output gyroscope. https://invensense.tdk.com/products/motion-tracking/3-axis/itg-3200/. Accessed: 2021-10-20.

[3] Neural networks. https://www.ibm.com/cloud/learn/neural-networks. Accessed: 2021-10-23.

[4] Shimmer user manual. https://bmslab.utwente.nl/wp-content/uploads/2019/12/Shimmer-User-manual.pdf. Accessed: 2021-10-20.

[5] Vibrating structure gyroscope. https://en.wikipedia.org/wiki/Vibrating_structure_gyroscope. Accessed: 2021-10-20.

[6] HK Aduful. Obesity–a preventable disease by f. ofei. *Ghana Med J*, 39(3):98–101, 2005.

[7] Steven Allender, Erin Gleeson, Brad Crammond, Boyd Swinburn, Mark Lawrence, Anna Peeters, Bebe Loff, and Gary Sacks. Moving beyond'rates, roads and rubbish': How do local governments make choices about healthy public policy to prevent obesity? *Australia and New Zealand health policy*, 6(1), 2009.

[8] Soma Bandyopadhyay, Anish Datta, and Arpan Pal. Automated label generation for time series classification with representation learning: Reduction of label cost for training. *arXiv preprint arXiv:2107.05458*, 2021.

[9] HE Brown, AJ Atkin, Jenna Panter, Geoff Wong, Mai JM Chinapaw, and EMF Van Sluijs. Family-based interventions to increase physical activity in children: a systematic review, meta-analysis and realist synthesis. *Obesity reviews*, 17(4):345–360, 2016.

[10] Anup Das, Francky Catthoor, and Siebren Schaafsma. Heartbeat classification in wearables using multi-layer perceptron and time-frequency joint distribution of ecg. In *Proceedings of the 2018 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies*, pages 69–74, 2018.

[11] Antonio De Lorenzo, Lorenzo Romano, Laura Di Renzo, Nicola Di Lorenzo, Giuseppe Cenname, and Paola Gualtieri. Obesity: a preventable, treatable, but relapsing disease. *Nutrition*, 71:110615, 2020.

[12] Centers for Disease Control et al. Physical activity and good nutrition: essential elements to prevent chronic diseases and obesity 2003. *Nutrition in clinical care: an official publication of Tufts University*, 6(3):135–138, 2003.

[13] Grzegorz Glonek and Adam Wojciechowski. Kinect and imu sensors imprecisions compensation method for human limbs tracking. In *International Conference on Computer Vision and Graphics*, pages 316–328. Springer, 2016.

[14] C.M. Hales, M.D. Carroll, C.D. Fryar, and C.L. Ogden. Prevalence of obesity and severe obesity among adults: United states, 2017–2018. In *NCHS Data Brief, no 360*. Hyattsville, MD: National Center for Health Statistics, 2020.

[15] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[16] Kathleen T Hickey, Nicole R Hauser, Laura E Valente, Teresa C Riga, Ashton P Frulla, Ruth Masterson Creber, William Whang, Hasan Garan, Haomiao Jia, Robert R Sciacca, et al. A single-center randomized, controlled trial investigating the efficacy of a mhealth ecg technology intervention to improve the detection of atrial fibrillation: the iheart study protocol. *BMC cardiovascular disorders*, 16(1):1–8, 2016.

[17] James O Hill and John C Peters. Environmental contributions to the obesity epidemic. *Science*, 280(5368):1371–1374, 1998.

[18] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *International Conference on Machine Learning*, pages 2304–2313. PMLR, 2018.

[19] Misha Kay, Jonathan Santos, and Marina Takane. mhealth: New horizons for health through mobile technologies. *World Health Organization*, 64(7):66–71, 2011.

[20] Stephen Cole Kleene. *Representation of events in nerve nets and finite automata*. Princeton University Press, 2016.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[22] Junnan Li, Richard Socher, and Steven CH Hoi. Dividemix: Learning with noisy labels as semi-supervised learning. *arXiv preprint arXiv:2002.07394*, 2020.

[23] Eran Malach and Shai Shalev-Shwartz. Decoupling" when to update" from" how to update". *arXiv preprint arXiv:1706.02613*, 2017.

[24] Maureen T Mcguire, RR Wing, ML Klem, HM Seagle, and JO Hill. Long-term maintenance of weight loss: do people who lose weight through various weight loss methods use different behaviors to maintain their weight? *International journal of obesity*, 22(6):572–577, 1998.

[25] Patricia N Mechael. The case for mhealth in developing countries. *Innovations: Technology, Governance, Globalization*, 4(1):103–118, 2009.

[26] F Ofei. Obesity-a preventable disease. *Ghana medical journal*, 39(3):98, 2005.

[27] A Powell, Andrew J Teichtahl, Anita E Wluka, and FM Cicuttini. Obesity: a preventable risk factor for large joint osteoarthritis which may act through biomechanical factors. *British journal of sports medicine*, 39(1):4–5, 2005.

[28] Simon P Rowland, J Edward Fitzgerald, Thomas Holme, John Powell, and Alison McGregor. What is the clinical value of mhealth for patients? *NPJ digital medicine*, 3(1):1–6, 2020.

[29] Surya Sharma, Phillip Jasper, Eric Muth, and Adam Hoover. The impact of walking and resting on wrist motion for automated detection of meals. *ACM Transactions on Computing for Healthcare*, 1(4):1–19, 2020.

[30] Surya Prakash Sharma. *Detecting periods of eating in everyday life by tracking wrist motion—what is a meal?* PhD thesis, Clemson University, 2020.

[31] Yiru Shen, James Salley, Eric Muth, and Adam 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, 2016.

[32] Mark Simmonds, Alexis Llewellyn, Christopher G Owen, and N Woolacott. Predicting adult obesity from childhood obesity: a systematic review and meta-analysis. *Obesity reviews*, 17(2):95–107, 2016.

[33] Mark Simmonds, Alexis Llewellyn, Christopher G Owen, and N Woolacott. Predicting adult obesity from childhood obesity: a systematic review and meta-analysis. *Obesity reviews*, 17(2):95–107, 2016.

[34] Terry T Um, Franz MJ Pfister, Daniel Pichler, Satoshi Endo, Muriel Lang, Sandra Hirche, Urban Fietzek, and Dana Kulić. Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, pages 216–220, 2017.

[35] Klaas R Westerterp. Physical activity assessment with accelerometers. *International Journal of Obesity*, 23(3):S45–S49, 1999.

[36] Pengxiang Wu, Songzhu Zheng, Mayank Goswami, Dimitris Metaxas, and Chao Chen. A topological filter for learning with label noise. *arXiv preprint arXiv:2012.04835*, 2020.

[37] Shuquan Ye, Dongdong Chen, Songfang Han, and Jing Liao. Learning with noisy labels for robust point cloud segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6443–6452, 2021.