Individualized wrist motion models for detecting eating episodes using deep learning

---

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
Wenkang Wei
May 2021

Accepted by:
Dr. Adam Hoover, Committee Chair
Dr. Richard Groff
Dr. Yongqiang Wang

Abstract

This thesis considers the problem of detecting eating episodes such as meals and snacks, by tracking wrist motion using smartwatch device. Previous work by our group has trained a wrist motion classifier using a large data set collected from 351 people to learn general eating behaviors. We call this a group model. This thesis investigates training the classifier with the same model architecture on new data collected by 8 people, and training the individualized classifier separately for each person. We call these individual models. The main goal in this work is to determine if individual models provide higher accuracy in detecting eating episodes, with fewer false positives, compared to the group model. By comparing their performance, we can also know if the improvement from individual models varies for each individual.

In data collection, two data sets were used. One is the individual data set, which was collected from 8 participants and each participant has at least 10 days of wrist motion 6-axis time-series data. There are 115 days, 1,064.5 hours and 246 meals collected in total in this data set. The second one is the group data set, called Clemson All-day Data set (CAD), collected in previous work [27]. This group data set collected from 351 participants contains 354 days, 1,133 meals, 250 eating hours and 4,680 hours in total. Two data sets were first processed using smoothing and normalization techniques and then cut along time by a sliding window to generate training and testing samples for training models.

In model training and evaluation, all models used the same convolution neural network architecture. Only one group model was trained on CAD group data set and this group model was used to compare with all other individual models. We used 5-fold cross validation to train and evaluate 5 individual models per individual. In model evaluation, we selected weighted accuracy (WAcc) as time metric to measure the models' ability of classifying each window sample as eating or non-eating. We also selected true positive rate (TPR) and ratio of false positive over true positive

(FP/TP) as episode metrics to measure model's ability of detecting each meal episode. TPR measures how many true eating episode are detected correctly and FP/TP measures the ratio of wrong detection amount over true detection amount. Hence when TPR is larger and FP/TP is smaller, model performs better. WAcc, TPR and FP/TP were measured by cross validation.

When measuring the time metric, we found that over 8 participants, the average WAcc on all individual models is 0.819 and the average WAcc on the group model is 0.780. On average, the individual models outperform the group model. Moreover, the improvement of individual models over the group model can vary per individual. For example, in one individual data set, individual models with WAcc of 0.897 have obvious improvement compared to the group model with WAcc of 0.774. In another individual data set, WAcc of 0.958 from individual models is very close to WAcc of 0.956 from the group model.

In the measurement of episode metrics, we found that before tuning hyper-parameters T_s and T_e, compared to the group model, individual models have the average improvement of 8.6% on TPR, but -14.4% on FP/TP. After tuning T_s and T_e, individual models have the average improvement of 10.1% on TPR and 33.2 % on FP/TP. Tuning T_s and T_e can improve the individual models' episode metrics.

Acknowledgments

At first, I want to thank Dr. Adam Hoover for his patience and his time in guiding me to do this research and giving me precious suggestions on both technical research and communication skills. He has taught me a lot in doing a research step by step and presenting the ideas clearly and professionally. Without him, I can not complete this research and thesis.

I would like to thank Dr. Richard Groff and Dr. Yongqiang Wang who serve as my defense committee and take their time to listen to my defense. I also want to acknowledge the Clemson faculty and professors who have taught me the professional and technical skills and provided me an excellent learning environment to achieve my goals in the graduate study years.

Finally, I want to thank my friends and all people who gave me lots of help in either collecting data for this thesis or solving the problems I met during my graduate study. Their helps are inseparable and significant parts in the success of my graduate study.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

This thesis considers the problem of detecting eating episodes (meals, snacks) by tracking wrist motion using sensors embedded in a smartwatch. Previous work by our group has trained a wrist motion classifier using data collected by 351 people. We call this a group model. This thesis investigates training the same classifier on new data collected by 8 people, and training the classifier separately for each person. We call these individual models. The main goal is to determine if individual models provide higher accuracy in detecting eating episodes, and fewer false positives, compared to the group model.

The motivating idea for this work is that different people may move their wrists differently during eating episodes and other daily activities. For example, one person may habitually eat with utensils while another tends to use their hands. One person may tend to eat faster while another tends to eat slower. Typical food selections by different people may require different types of wrist motions. A group model must learn all these motion patterns which increases the complexity of differentiating eating activities from non-eating activities. An individual model can learn fewer patterns which may improve classification accuracy.

As people may have different wrist motions during eating, they may also have different wrist motion patterns caused by their other typical daily activities. For example, one person may wear glasses and frequently adjust them. These wrist motion patterns can resemble eating gestures and cause confusion in our classifier. However, another person may not wear glasses, and in this case

the classifier may benefit from the lack of this confound. Another difference between individuals is proclivity for facial and hair grooming. Daily work and hobby tasks may also cause a wide variety of wrist motion differences between individuals. Our classifier outputs two classes, eating and non-eating, which means that its challenge is to differentiate wrist motions caused by eating episodes from all other wrist motions observed throughout the day.

The data previously collected by our group consists of appx 1 day of wrist motion for each of 351 people. One participant recorded for three days, another participant recorded for two days and the others recorded for 1 day. For the experiments in this thesis, we collected new data consisting of appx 10 days for each of 8 people. We used the previous data set to train the group model, and the new data to train individual models. We also compared the accuracy of detecting eating episodes on the new data using the group model and individual models.

The rest of this chapter provides background for our methods and experiments. Section 1.2.1 discusses the obesity epidemic which is the motivating health problem for this research. Section 1.2.2 discusses the field of mobile health (mHealth) tools, which empower individuals to track data related to their individual health. Section 1.3 discusses previous work using mHealth tools for detecting eating episodes. Section 1.4 describes the wrist motion tracking device we use to collect data, including the physics of how accelerometers and gyroscope sensors work. Section 1.5 provides background in deep learning and neural networks, which is the type of classifier we use to analyze wrist motion. Finally, section 1.6 outlines the novelty of this work and lists the main questions examined in our experiments.

1.2 Background

1.2.1 Obesity

Obesity is a challenging healthcare problem. It is a medical problem that can increase the risk of diseases and health problems, such as heart disease, diabetes, high blood pressure and certain cancers. For example, diseases like liver cancer, kidney cancer and gastric cardia cancer are usually developed in people who are overweight or obese much easier than normal-weight people. As reported by National Cancer Institution, people who are overweight or obese are up to twice as likely as normal-weight people to develop liver cancer. The association between overweight/obesity and liver cancer is stronger in men than women [5, 17, 32]. According to a report from CDC in 2017,

obesity was prevalent in 39.8 % of the population and affected about 93.3 million of US adults in 2015-2016 [6]. China has the largest number of affected people worldwide, with about 46%of adults and 15 % of children being obese or overweight [36]. As the obesity problem is prevalent making people exposed to high risk of disease, it is necessary to figure out the causes of obesity and ways to lose weight.

According to previous researches, the factors contributing to obesity are genetics, lack of physical activity and poor diet. Although genetic factor could be a cause, the current epidemic of obesity is caused largely by an environment that promotes excessive food intake and discourages physical activity and the imbalance in Energy intake (EI) and energy expenditure (EE) [25]. Humans have evolved excellent physiological mechanisms to defend against body weight loss, but they have only weak physiological mechanisms to defend against body weight gain when food is abundant [18].

There are many methods to lose weight, such as trying intermittent fasting, eating mindfully, cutting back on sugar and refined carbohydrates, tracking diet and exercise [15, 19]. For example, in intermittent fasting (IF), it involves regular short-term fasts and consuming meals within a shorter time period during the day, like fast on 2 days out of 7 days in a week. There are some researches indicating that short-term intermittent fasting up to 24 weeks in duration, can lead to weight loss in overweight individuals [8]. The second easy method to lose weight is to eat mindfully. For example, people can eat slowly to take time to chew and savor the food. This technique helps with weight loss, as it gives a person's brain enough time to recognize the signals that they are full, which can help prevent over-eating and pay attention to select food and avoid picking food that contains high calories [15]. In addition to controlling the in-take of food, tracking the diet and exercise and encouraging people to take more exercise after eating is usually a useful way to lose weight. There is a study revealing that a positive correlation between weight loss and the frequency of monitoring food intake and exercise. Even a device as simple as a pedometer can be a useful weight-loss tool for monitoring the diet and exercise in daily life [4].

While the approaches mentioned above are easy and natural ways to lose weight, they require people to keep track of their diet and daily life mindfully. However, it is usually hard to let people record how they eat or do exercise every day for a long time. Especially when people are busy due to their job or study, they are likely to give up recording gradually. In this case, some devices like smart phone, smart watch can provide convenient and powerful ways for them to do that and this leads to a field of mobile health (mHealth), which is introduced in the next section.

1.2.2 mHealth

Healthcare is one of the significant demands in human life to improve life quality. In the last decade, healthcare spending in the OECD has increased from an average of 7.8 percent to 9.7 percent of the region's GDP [1]. Unfortunately, in emerging markets, inadequate health infrastructure limits the rural and the urban poor to the most basic care. Healthcare in hospital can be expensive and need patients to wait for a long time to get diagnoses and treatments. Observations from patients prior to a hospital visit are often anecdotal, which need doctors to guess details. It is usually inconvenient for patient to record the observations manually for a long period. Accurate tools for self-monitoring are necessary to help people monitor their daily life and diet and these tools should be easy to use and impose a minimal burden on the user [10]. Consequently, healthcare providers move their insights to cheaper and faster ways for healthcare diagnoses and pay more attention to develop mobile health service.

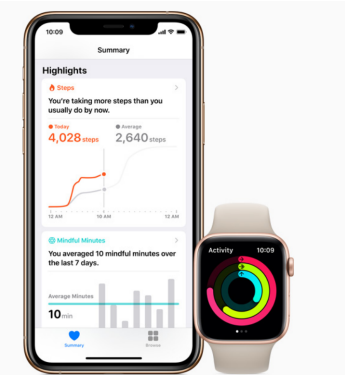

Figure 1.1: Example of iPhone smart watch and its application in mobile health

Mobile health, also called mHealth, is a term used for the practice of medicine and public health supported by mobile devices, which is an important part of healthcare. Mobile heath largely emerges as a mean of providing greater access to larger segments of a population in developing countries, as well as improving the capacity of health systems in such countries to provide quality healthcare [31]. With benefit from the rapid growth of computation techniques and big data period, mHealth techniques have growth rapidly. self-monitoring devices or software applications, like smartphone, smart-watch, become prevalent and powerful to monitor user's health and diet. For example, in figure 1.1, Apple combines iPhone and Apple smart watch to collect user activity data and applies machine learning in healthcare software to detect the exercise user takes and measure

how user's cholesterol or blood pressure has changed over the years. Another example is that using data collected from wrist-worn devices, we are capable of finding the number of bites taken from food by user, since the pattern of taking food usually consists of picking up food, rotating and moving hand to mouth and eat. By recognizing such kind of patterns, we can recognize if user is eating or not and how many meal events user has in daily life.

1.3 Related Work

In order to fight against obesity, measurements of energy intake are commonly used for the study and treatment of obesity. While the most widely used tools rely upon self-report and require a considerable manual effort, leading to under-reporting of consumption, non-compliance, self-monitoring devices provide powerful ways to detect eating period of users automatically. One of self-monitoring devices is watch-like device, tracking the wrist motion of user to detect eating activities.

In the past, Dong et. al proposed to use InertiaCube3 wearable device to track the wrist motion and segment the data based on task logs manually. The authors also developed a rule-based algorithm to classify the task of wrist motion [33]. The authors performed two experiments, in the first rule-based algorithm, it reported 91% accuracy in assigning these classes to a total of 125 tasks. In a second experiment, the authors applied a state machine method to categorize time into one of four categories: start of eating, end of eating, not-eating and inside or outside eating, which achieved a sensitivity of 82%.

Later, they also described an algorithm that segmented and classified such periods as eating or non-eating activities and also evaluated this method on a large data set (43subjects, 449 total hours of data, containing 116 periods of eating) collected during free-living that wrist based tracking could be used to count the number of bites of food consumed during a meal [34]. The authors reported that an accuracy of 81% for detecting eating at 1 second resolution in comparison to manually marked event logs of periods eating. The authors concluded that vigorous wrist motion is a useful indicator for identifying the boundaries of eating activities [34].

After Dong's work, Shen et. al collected data from 276 participants with a total of 44,873 gestures in data set and labelled the gesture using video synchronized with wrist motion tracking device. The authors examined if foreknowledge of the gender, age, and utensil used for eating could

improve recognition accuracy by using a machine learning method, called Hidden Markov Model (HMM). They reported that Age HMMs provided a 4.3% increase in accuracy and utensil HMMs provided a 6.2% increase in accuracy [35].

As mentioned in the paper [24], "Deep learning (DL) methods receive increasing attention within the field of human activity recognition (HAR) due to their success in other machine learning domains. Nonetheless, a direct transfer of these methods is often not possible due to domain specific challenges (e.g. handling of multi-modal sensor data, lack of large labeled data sets)", in recent years, more and more researchers start applying deep learning methods, like CNN, to human activity recognition field.

In recent years, our research group has utilized deep learning to build models to track and classify the wrist motion for gesture recognition and eating detection. Luktuke et. al applied a deep learning model called U-Net to segment 1D gestures in the same way of 2D image region segmentation, by treating each inertial measurement unit (IMU) datum like a pixel. A data set called Clemson Cafeteria Data set, collected from 276 participants with 1 - 4 courses in each meal, was used. The authors reported their neural network classifier recognized an average of 79.7% of 'bite' and 84.7% of 'drink' gestures correctly per meal. Overall 77.7% of all gestures were recognized correctly in average per meal [13].

In the research of eating activity detection, Sharma collected a data set called Clemson All-day Data set (CAD), a freeliving eating activity data set containing 354 days and 4,680 hours of wrist motion collected from 351 participants. One participant recorded for three days, another participant recorded for two days and the others recorded for 1 day. Then the author proposed to use a convolution neural network with a much longer window (0.5-15 min) that can contain other gestures related to eating, such as manipulating food, preparing foods for consumption, and resting. The final results reported in that work showed that the context of these other gestures can improve the detection of periods of eating, and that accuracy at detecting eating increased by 15% in longer windows compared to shorter windows as the overall results on CAD were 89% detection of meals with 1.7 false positives for every true positive (FP/TP), and a time weighted accuracy of 80% [27].

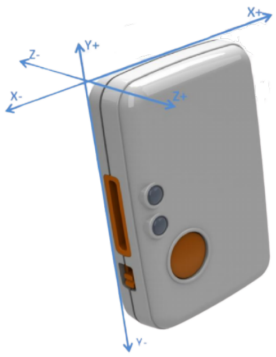

Figure 1.2: Shimmer3 device with x, y, z axes shown

1.4 Shimmer3 Device

In order to collect wrist motion, data, a wearable device for data collection is necessary. Usually wearable device contains inertial measurement unit (IMU) sensor, which combines accelerometer sensor and gyroscope sensor to measure and record the acceleration and angular velocity data from wrist motion. Shimmer3 is one of such type of sensors. Shimmer is a small wireless sensor platform well suited for wearable applications. The integrated kinematic sensors, large storage and low-power standards based communication capabilities enable emerging applications in motion capture, long-term data acquisition and real-time monitoring. Shimmer3 wearable device provides a convenient way to collect wrist motion time series data [21].

Shimmer wearable device contains 3-axis accelerometer (x, y, z axes shown in figure 1.2) and 3-axis gyroscopes (angular rate sensors), allowing us to keep track of wrist motion and record time series data for training our models. The collected data is loaded from the device to local laptop by using a software system called ConsenSys, which exports the time series data into a CSV file that contains acceleration data from internal 3-axis accelerometer and angular velocity data from internal gyroscope sensor. In this work, data was collected by using Shimmer3 devices. Section 1.4.1 and section 1.4.2 introduce how accelerometer and gyroscope work.

1.4.1 Accelerometer

Accelerometer is a microelectromechanical device, which measures the acceleration on the device due to external force. The principle of measuring the acceleration behind accelerometer is

based on Hooke's Law and Newton's Second Law.

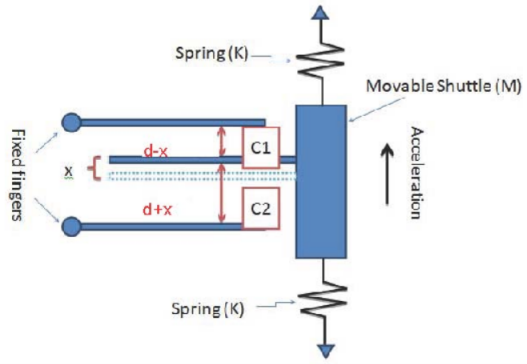

Figure 1.3: Example of the structure of accelerometer. The blue rectangel is the shuttle associated with two springs. When the shuttle moves, it changes the capacitance on the left and leads to a change of voltage.

According to Hooke's law, as shown in figure 1.3, when a movable shuttle with mass M attached to a fixed end with a flexible spring is displaced, the force from the spring is proportional to the spring constant K and the actual displacement x experienced by the body. This formula is

$$F = -K \times x \tag{1.1}$$

where x is the displacement of shuttle and K is the spring constant and F is the force exerted on the spring. According to Newton's Second Law, we have the formula

$$F = M \times a \tag{1.2}$$

where M is the mass of shuttle and a is the acceleration of that mass. By combining equations 1.1 and 1.2 Then the acceleration estimated in accelerometer is given by

$$a = -\frac{K \times x}{M} \tag{1.3}$$

The force that displaces the mass, causes capacitance change between the moving plate and the fixed plates. This change is measured to a voltage which can be converted to units like $meters/second^2$ or typically as a proportion of acceleration due to gravity $g = 9.81 m/s^2$.

1.4.2 Gyroscope

Gyroscope is a sensor that measures angular velocity, the rate at which a device is being rotated. In figure 1.4, there is a sensor called MPU-6050, which contains gyroscope sensor inside it. Gyroscope typically has 3 axes: roll, pitch and yaw. By utilizing the data from gyroscope, we can measure how fast the object rotates at different axes. These data can help us explore the patterns from wrist motion. For example, when eating, if we rotate our hand, the angular velocity along roll axis can be greater than the angular velocities in other axes. This helps us explore the patterns of our wrist motion.

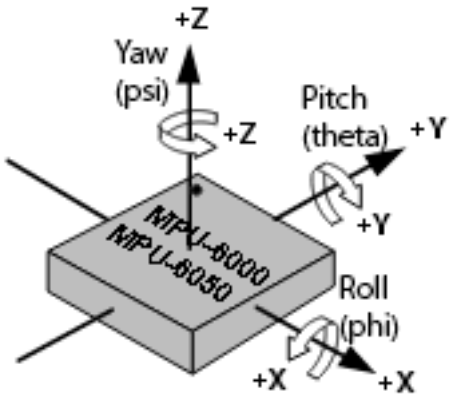

Figure 1.4: Example of 6 axes of gyroscope in MPU sensor: x, y, z for accelerometer and roll, pitch, yaw for gyroscope

The sensing principle behind gyroscope is based on the Coriolis effect for a vibrating body that experiences angular motion [30]. In figure 1.6, we can see when the platform is accelerated to the right, the mass moves to the outer edges due to the Coriolis effect. This compresses the outer springs, which causes capacitance changes in the sensing teeth. The displacement of the frame can be sensed through capacitive transduction in response to the force exerted by the mass. Typically this sensor output needs to be converted to unit of degree /sec.

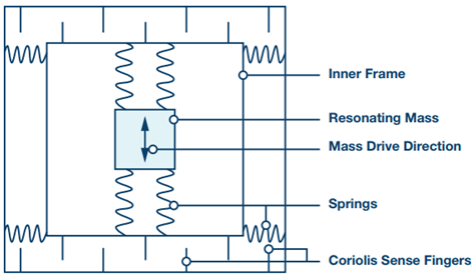

Figure 1.5: Schematic of the gyroscope's mechanical structure.

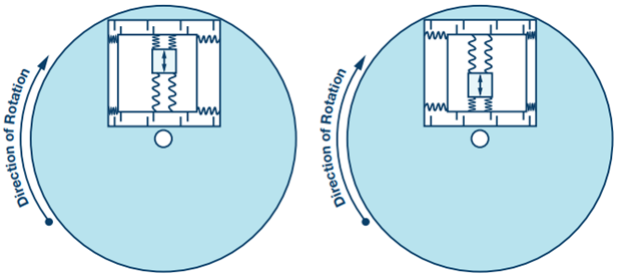

Figure 1.6: Example Coriolis effect in gyroscope. The blue circle is a rotating platform. Inside the circle, the square shows the mechanical structure of gyroscope

1.5 Machine Learning

Machine Learning is a sub-field of artificial intelligent, its goal is to learn specific patterns and mine interesting features from data so that people can utilize machine learning models to solve the real world problems. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve. Machine learning algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs [28]. According to the learning method, machine learning can be generally categorized into three types: supervised learning, unsupervised learning and semi-supervised learning.

In supervised Learning, machine learning model is given a data set of tuples (x_i, y_i), where x_i represents the i^{th} sample and y_i represents the corresponding label of the i^{th} sample. For example, if there is an image of cat, then this image is an sample x_i and the label "cat" is y_i. The task of detecting periods of eating belongs to a task of supervised learning as well, which is often formulated

as a classification problem, since it requires training samples x as well as the ground truth labels y, "eating" and "not eating".

Unsupervised learning utilizes data without labels, while supervised learning requires labels of training samples. The unsupervised learning algorithm is left to find commonalities among its input data. For example, in movie recommendation system application, it is usually necessary to cluster users into different groups based on their similarity on their interests of movie genres, their hobbies, countries and other features so that the system can better do recommendation based on the group's interests. In this case, there is no label used, but it needs to find similarity among people based on their information. Clustering task in Movie recommendation is an example of unsupervised learning.

Semi-supervised learning falls between supervised learning and unsupervised learning, it usually combines small amount of data with large amount of unlabeled data to train machine learning model [37]. In semi-supervised learning, it considers the problem of expensive label in supervised learning and permits harnessing the large amounts of unlabelled data available in many use cases in combination with typically smaller sets of labeled data. It is commonly used when there is a lack of labels for data set [29].

1.5.1 Deep Learning

Deep learning is a subset of machine learning, which is inspired by our human brain structure and utilizes a hierarchical level of artificial neural networks to carry out the process of machine learning. Artificial neural network simulates human brain and constructs a network using multiple layers of neurons in hierarchical style. There are many different deep learning architectures, such as deep neural networks, deep belief networks, recurrent neural networks and convolution neural networks, which have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs. Deep learning methods have produced results comparable to and in some cases surpassing human expert performance.

1.5.1.1 Neural Network

Neural network model is the core concept in deep learning. It was first inspired by the biological structure in human brain. A biological neuron contains dendrites, cell body, nucleus, axon and synapse. Similar to biological neuron, neuron in artificial neural network contains weights in the input branch, simulating the dendrites and synapse to have excitatory or inhibitory effect on input signal, activation function, simulating the cell body and nucleus. Neuron works as a non-linear function to sum up and filter the input and output the results.

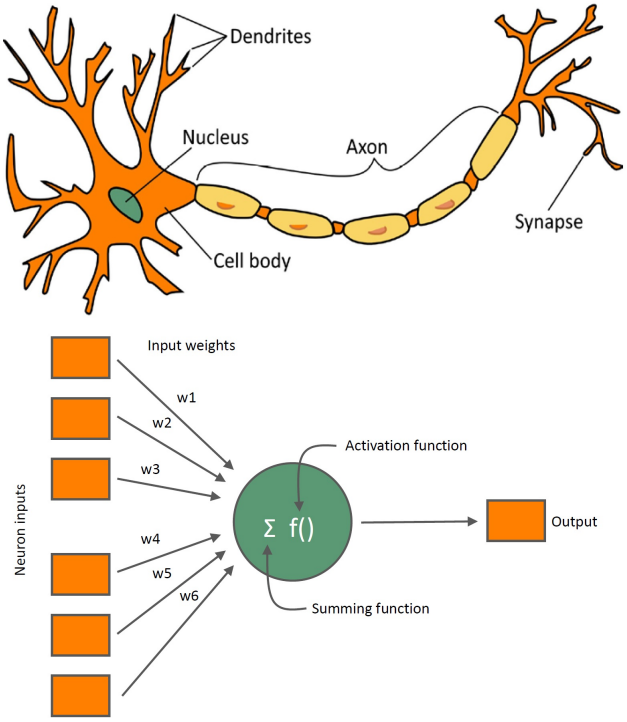

Figure 1.7: Comparison between biological neuron and neuron in neural network

Neural network works as a non-linear model, which takes the data feature vector as input and outputs the prediction in output layer. In classification problem, the output from neural network can be possibility that input feature belongs to a class. For example, when we build a network to classify if an animal in an image is a cat or not, then the network output is the possibility of being a cat, with value in the range from 0 to 1. In regression problem, for example, if we want to predict the housing price based on the input features like location of housing, longitude and so on, then the prediction from neural network becomes the estimated housing price in the range of $[0, \infty)$. Hence

the output from neural network can be various and it depends on the task. If we formulate the neuron, we can have the following expression:

$$a = \sigma(w^T x + b) \tag{1.4}$$

where w^T is a learnable row vector of weights, $w^T = [w_1, w_2 ... w_N]$ and x is an input column vector $[x_1, x_2 ... x_N]^T$. b is a scalar learnable bias term, $b \in R$. $\sigma(.)$ is usually a non-linear activation function, which is used to filter out the features and retain the most important features. There are many choices for the activation function and one of the most popular activation functions is rectified linear unit (ReLu) activation function, its formula is as follows:

$$\sigma(z) = \begin{cases} z & z \geq 0 \\ 0 & otherwise \end{cases} \tag{1.5}$$

This function outputs the scalar input value z directly when the input value is greater than 0, and outputs 0 when input is smaller than or equal to 0. This enables the network to filter out the values smaller than 0.

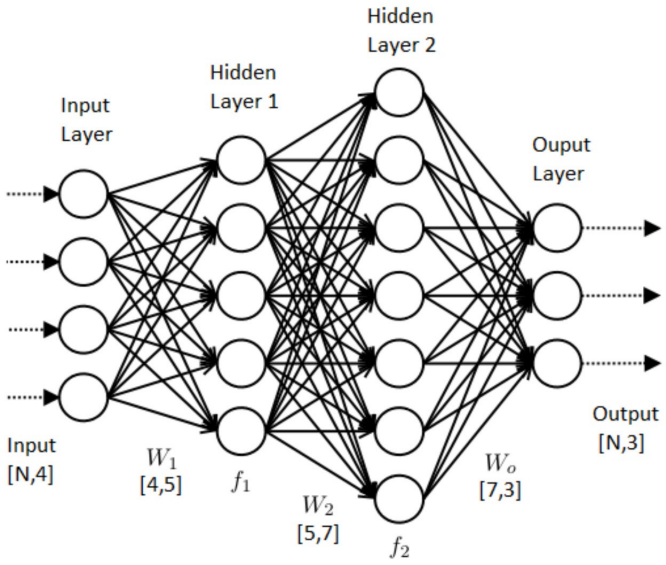

Figure 1.8: Example of neural network structure with fully-connected layers

Each layer of neural network is formed by multiple neurons and a neural network is constructed by multiple connected layers. When feeding the network with input data, the input feature

vector becomes the input layer in the neural network and this input feeds the next hidden layer inside the network. Then the output from one hidden layer becomes the input of the next hidden layer. This process is called feed-forward. Note that in figure 1.8, the layers, whose neurons have full connections to all neurons in the previous layer, as seen in regular neural networks, are called fully-connected layer. Their outputs can hence be computed with matrix multiplication.

When training the neural network, a technique called back-propagation, is used to update the parameters, like weights and bias terms in network [22]. In back-propagation, loss between prediction and target is first calculated and then used to compute the gradient of the parameters using chain rule method. After obtaining gradient values of parameters, gradient descent algorithm is applied to update the parameters in neural network so that neural network can learn from data.

1.5.1.2 Convolution Neural Network

Convolution neural network is a network that utilizes convolution layer in neural network. Convolution layer works as a function to detect and extract features from input by using adjustable masks, called kernel or filter, to perform convolution operation over the input feature matrix. The parameters in the kernels are the weights in the layer of neural network, which can be adjusted during training process to better fit data.

Convolution layer can be applied to either 1D feature, like time series data, or 2D features, like image. In 2D convolution, the input feature typically has width, height and channel. For example, in figure 1.9, the input is an image with width equal to 6 and height equal to 6 and channel equal to 1. It uses a 3-by-3 learnable kernel to do convolution on the upper left image patch by using element-wise multiplication and then summing up the result to get a new pixel in output matrix. After that, it slides the kernel by one pixel either rightward or downward and do the same thing to get the new output matrix. Hence the output matrix becomes 4-by-4 matrix.

1D convolution is the same process as a 2D convolution, except that the convolution kernel slides in a single dimension. Whereas we see the 2D convolution slide the kernel in both the horizontal and vertical orientation across the image, with our 1D convolution, we only see our kernel slide across the time domain [12]. In figure 1.10, we can see an example of 1D dimensional convolution, which uses a 1-by-3 kernel to do convolution on 1D data. The left figure shows the convolution with stride = 1 when moving kernel to the right while the right figure use a stride of 2. In addition, compared with fully-connected layer, convolution layer takes fewer learnable parameters by sharing

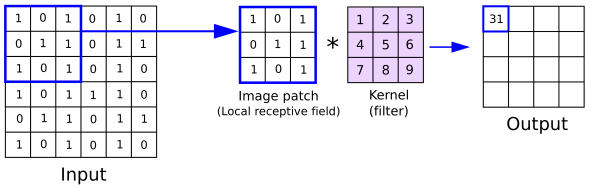

Figure 1.9: Example of 2D convolution on image feature in convolution layer with 3x3 kernel and 6x6 input image

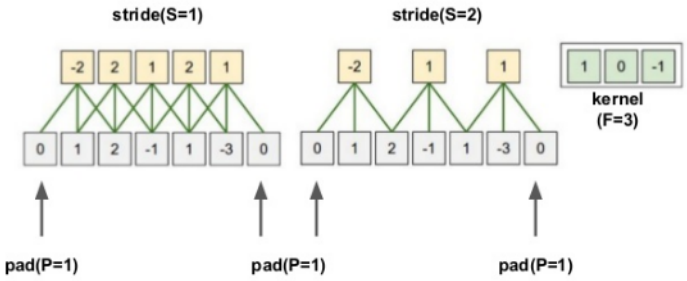

Figure 1.10: Example of 1D convolution on image feature in convolution layer with kernel size of 3

the parameters with different features and hence can be trained faster. As fully-connected layer considers the combination of all input features, convolution layer considers the features in a small local region only.

1.6 Novelty

The main novelty of this work is that it is the first to compare the performance of detecting eating activities of a CNN group model trained on a data set of a large group of people, Clemson All-day Data set (CAD) with the performance of individual CNN models that are trained on much smaller data set from different individual data sets, by testing the group model and individual models on the same individual data. The second novelty is that we attempted to find the thresholds T_s and T_e values for each individual data set using three different methods and improved the TPR and FP/TP performances of individual data sets. This work assumes that eating behaviors from a group of people share similar patterns with personal eating behaviors. The group model works as a more general model that requires a large amount of training data, while individual models work as

customized models for specific person, trained on fewer but specific data.

By comparing performance between general model and customized models, we can determine if the similarity of eating behavior between group of people and specific individual exists. This work can also make sure if customized individual models can work better than a more general model, group model. More specifically, this thesis aims to answer the following questions:

1) Do individual models trained on small individual data set perform better than the group model trained on large data from a group of people, when testing them on the same individual data?

2) How much improvement do individual models get, compared with group model? Does it vary depending on each individual?

3) Does changing individual hyper-parameters help improve performance of individual models?

Chapter 2

Methods

2.1 Data Collection

2.1.1 Data Collection Process

Two data sets used in this work were collected by Shimmer3 wrist motion tracking device, shown in figure 1.2. Participants were taught to use such device to collect the wrist motion of the hands they use to eat in their daily life.

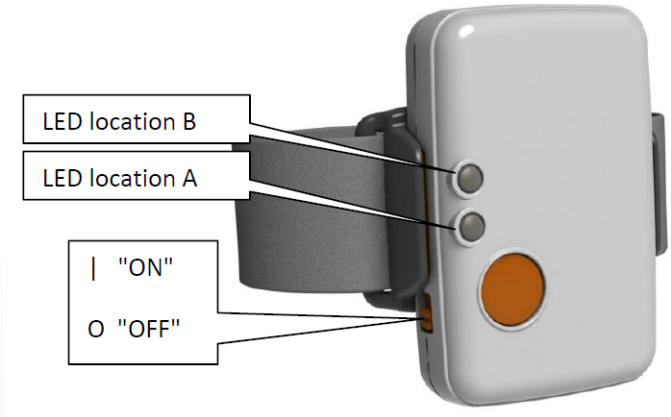

Figure 2.1: Explanation of using Shimmer device. The button on the left side is a power button to control power on/off state. The big orange button is a record button. Pressing this button about 5 second can start/stop recording. The recording state is indicated by LED lights

After collecting the data from one day, each participant used a software called ConsenSys,

to load the data from Shimmer3 embedding device into local laptop and export the data as a CSV file, which contains the date time of the signal data points, raw data from accelerometer sensor and gyroscope sensor. The wrist motion time series data sets were recorded in 15Hz frequency. In other words, there were 15 data points recorded in each second by Shimmer3. Figures 2.2 and 2.3 show the process of extracting raw data from device and convert data into CSV format.

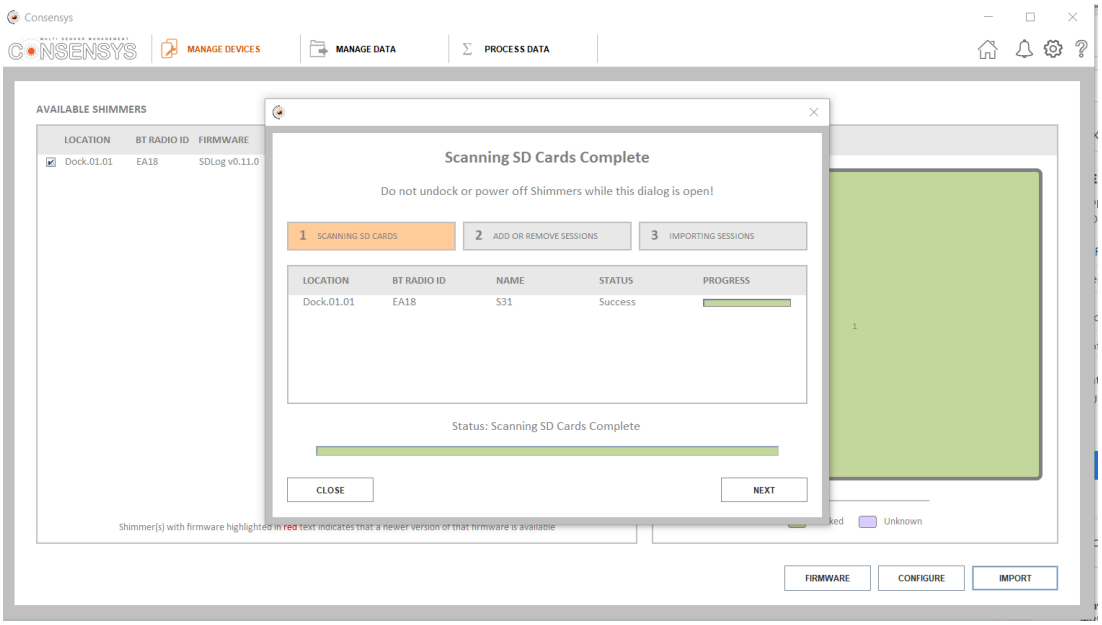

Figure 2.2: Example of loading data from Shimmer3 device to local laptop using ConsenSys Software

After exporting data into a CSV file, we used a software called MarkerParser to load the CSV file and label the eating episodes on time series data. When using the MarkerParse software, participants needed to memorize the eating episodes in that day in order to label the start time and end time of eating of each eating episode. The input information of a time series data includes meal names, the labels indicating if that period contains a meal or not, start time and end time of each eating episode, items consumed by user during eating period and location of eating. However, in this thesis, the information used includes labels of eating, start time and end time of eating period only, since the task in this thesis is to predict if participant is eating or not eating only. After the data collection and labelling steps, a C program software, called EatMon, was used to visualize the data set to check if data was recorded successfully without corrupted due to broken devices.

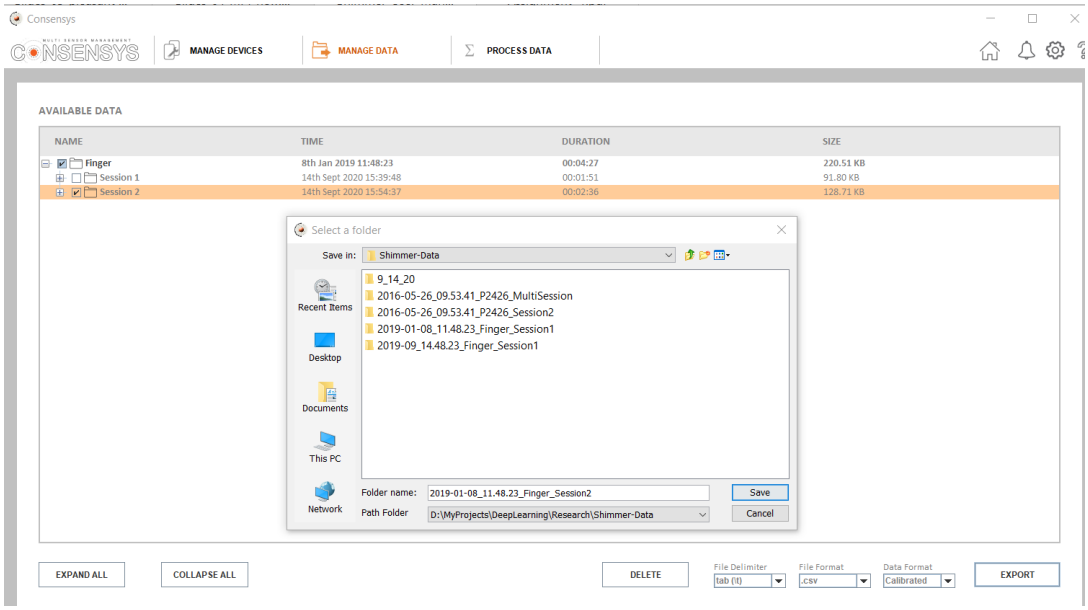

Figure 2.3: Example of exporting data from ConsenSys Software into a CSV file

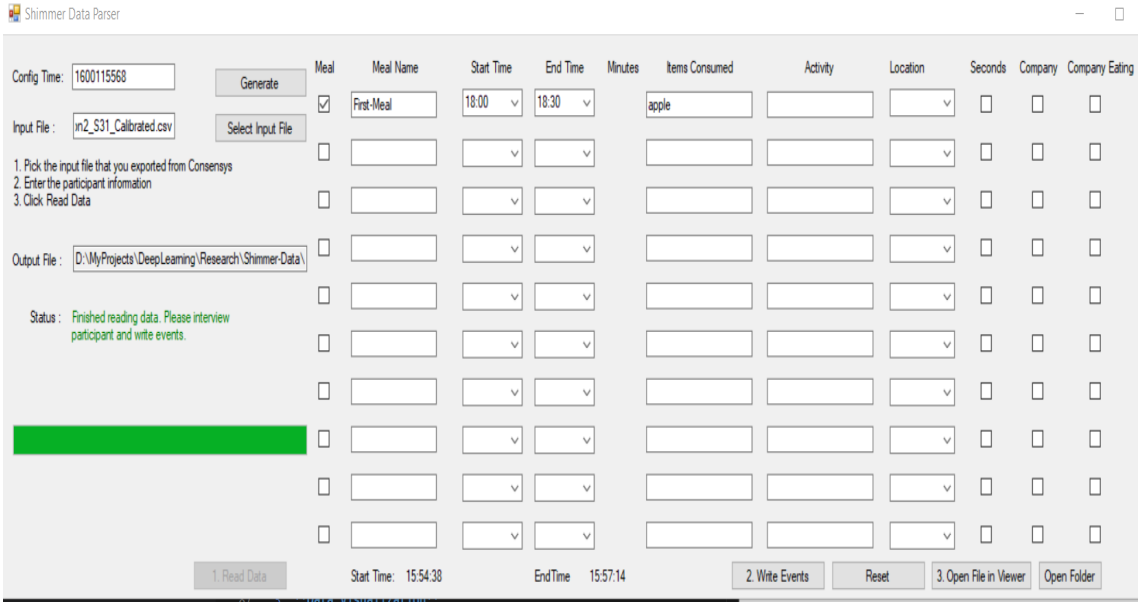

Figure 2.4: Example of software MarkerParser. User needs to fill the start time, end time, meal name and other information for each meal into the blank areas to label each meal

2.1.2 Data Description

In this work, there were two data sets utilized. One data set, called Clemson All-day Data set (CAD), was collected by our group previously [27]. As reported by the author, 408 participants were recruited in total to collect data and data was recorded on all seven days of the week. There were 351 participants completing the data collection providing a total of 354 days of usable data. One participant recorded for three days, another participant recorded for two days and the other participants recorded for 1 day. 4,680 hours of data was collected in total, containing 265 hours of eating activity across 1,133 separate eating activities. The average amount of data recorded per participant was 13.2 hours. The average start time for a recording was 8:50 am, while the average end time for recordings was 22:06 pm. The balance ratio of non-eating hours divided by eating hours is 20 in CAD data set [27].

The second data set from this work was collected from 8 participants. Each individual provides at least 10 days of data recorded successfully and hence there are 8 different individual data sets. There are 115 days, 1064.5 hours and 246 meal events recorded in total. A total of 57.5 of eating hours and 1007 non-eating hours were recorded in total. Balance ratio is the amount of non-eating hours over the amount of eating hours. For each individual, the balance ratio is computed by equation 2.1. A total balance ratio for the whole individual data set of 8 participants is 17.5.

$$balance\ ratio = \frac{no\ eating\ hours\ of\ an\ individual}{eating\ hours\ of\ an\ individual} \tag{2.1}$$

In table 2.1, each row, with data set id range from 1 to 8 in the first column, shows the information of a data set collected from one individual. For each individual, the features, including the number of days recorded, total hours recorded, meal count, average meal count per day, average hour per meal, eating hours, no eating hours and the balance ratio are computed. After that, the row, called "total", sums up the information over 8 individual data sets, to get total hours, total eating hours and total non-eating hours. The balance ratio in the row of "total" is equal to total non-eating hours divided by total eating hours. We also calculate the average records over the 8 individual data sets in "average" row. In this work, there is an average of 14.4 days over all individuals recorded. The average value of total hours of each individual data set is 133.1 and the average value of meal count over all participants is 30.8. The average balance ratio is 17.5, equal to the value of average non-eating hours divided by average eating hours.

dataset	Days	Total Hours	Meal Cnt	Avg Meal Cnt Per Day	Avg Hour Per Meal	Eating Hours	No Eating Hours	Ratio: noeat/eat
1	17.0	129.2	32.0	1.9	0.2	7.7	121.5	15.8
2	14.0	125.6	26.0	1.9	0.1	3.3	122.3	37.1
3	23.0	167.2	60.0	2.6	0.1	7.6	159.6	21.0
4	13.0	134.0	38.0	2.9	0.2	9.3	124.7	13.4
5	14.0	127.8	20.0	1.4	0.3	5.9	121.9	20.7
6	12.0	144.8	26.0	2.2	0.3	7.0	137.8	19.7
7	10.0	120.4	22.0	2.2	0.3	5.9	114.5	19.4
8	12.0	115.5	22.0	1.8	0.5	10.8	104.7	9.7
total	**115.0**	**1064.5**	**246.0**	**-**	**-**	**57.5**	**1007.0**	**17.5**
average	**14.4**	**133.1**	**30.8**	**2.1**	**0.2**	**7.2**	**125.9**	**17.5**

Table 2.1: There are 8 individual data sets collected from 8 participants respectively. The first column is the id of each individual data set. Total Hours: the amount of hours recorded in each individual data set. Meal Cnt: meal count of each individual. Avg Meal Cnt Per Day: average meal count per day of each individual. Ratio: the balance ratio computed by no eating hours over eating hours.

Data collected from Shimmer3 devices used contains 6 axes of features: linear acceleration x, y, z, from accelerometer and yaw, pitch, roll from gyroscope. Figure 1.4 and figure 1.2 show the axes of linear accelerometer (x, y, z) and the axes of gyroscope (roll, pitch, yaw) in Shimmer3 device. The acceleration data x, y, z output from accelerometer is in unit of m/s^2 while the angular velocity data from gyroscope, yaw, pitch and roll has unit of degree/second.

2.2 Data Pre-Processing

In this section, we discuss the common techniques used in processing the noisy time series data and the way used to extract the training and testing samples for training our deep learning models.

2.2.1 Smoothing

Usually raw time series data from sensors, like accelerometer and gyroscope, is noisy. Especially when collecting wrist motion data, user's hand may be shaking when they are eating, which generates unexpected noise to the wrist motion data. In order to diminish the noise effect and extract the main patterns that we are interested in, smoothing the time series data before usage is necessary.

One of the useful ways to smooth data is to use a Gaussian filter, which is to first generate a Gaussian kernel mask based on Gaussian distribution and then apply the kernel to do convolution over the time series data set. In this work, a Gaussian kernel with window size of 15 data points and variance $\sigma = 10$ is applied to each axis of data, which is as same as the previous work [27] and the convolution equation in smoothing is given by:

$$x_j^{smooth} = \sum_{k=-p}^{p} x_{j-k} w_k \tag{2.2}$$

where x_{j-k} is the $(j-k)^{th}$ data point in window and w_k is the k^{th} value in Gaussian filter [23]. The convolution operation from a software package, called Numpy, is applied. In figure 2.5, one example of comparing the smoothed data and data without smoothing is shown. This example shows a time series pattern with 900 data points from the x axis of accelerometer data. X axis in this figure represents the index of data point and Y axis represents the signal value. The blue curve is the data before smoothing, which is noisy especially around the x =200 and x = 800. After applying the Gaussian filter (the orange curve), noise is diminished obviously and the main pattern (a peak) still remains.

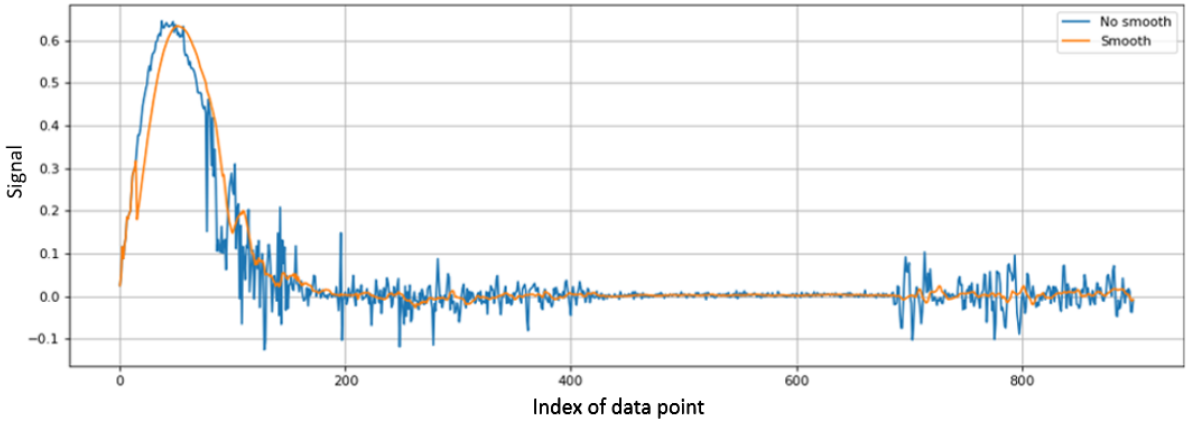

Figure 2.5: An example of comparison between smoothed data and non-smoothed data. Blue curve represents the signal before smoothing. Orange curve represents the signal after smoothing using moving average

2.2.2 Feature Scaling

In addition to smoothing, feature scaling is also a significant data pre-processing technique, which is to re-scale or regularize the range of data to reduce the bias effect on machine learning model due to different ranges of different features. Usually, model that depends on distance between features or uses gradient method to update hyper-parameters can be sensitive to the scale of feature. The convergence rate of such kind of models is usually affected by the range of feature. By applying feature scaling to data, it can speed up the model training and sometimes improve prediction performance [3].

In this work, z-normalization, also called standardization, one of the feature scaling methods, was applied to transform data. Rather than directly re-scale the range of features, z-normalization can convert data into data with zero-mean and unit-variance. The equation of z-normalization is given by:

$$x_{normalized} = \frac{x - \bar{x}}{\sigma} \tag{2.3}$$

where $\bar{x}$ and σ are the mean and the variance of time series data respectively. By subtracting the mean of the whole time series data and dividing by variance, it can shift the data to a zero center and then re-scale its variance to 1. According to the previous work , z-normalization method is better than other normalization methods, hence z-normalization was applied to transform data after smoothing the data using Gaussian filter in this work.

2.2.3 Sliding Window

As time series data is a kind of unstructural data, it can not be used to train model directly. This leads to a transformation method called sliding window, which extracts the training, testing samples from time series data by using a window with fixed size.

In sliding window method, a window with fixed size is moved along the axis of time in time series data. One sample is extracted from the region covered by the window in time series data. Then the window is moved forward and skip some data points to extract the next sample. The size of data pointed skipped by the window is called stride. For example, in figure 2.6, a window (blue box) with size of 3 is used to extract a 1-axis time series data. After that, the window moves forward by 1 data point (stride=1) to sample the next data frame. Note that when the stride is greater than 1, sometimes the window can fall outside the range of time series data and the extracted window

sample is incomplete and invalid, when the window moves to the end of time series data. In this case, we can simply drop the last invalid sample.

In this work, since the time series data contains 6 axes of data: x, y, z and roll, pitch, yaw, a sliding window extracts the 6-axis data to create a data frame with 6 axes. The window size used is 6 minute with sampling frequency of 15 Hz. That is, a window has the length of $6 \times 60 \times 15 = 5400$ data points. The stride used in CAD group data set is 15 seconds (15seconds $\times$ 15Hz $= 225$ data points). However, as for each individual data set, since the individual data contains fewer data than CAD data, in order to reduce under-fitting effect during model training due to small data set, we use a stride of 5 seconds to extract more training samples. The amount of extracted samples can be calculated by:

$$N = \frac{T - W}{stride} + 1 \tag{2.4}$$

where N is the amount of 6-axis data samples extracted from time series data. T is the length of time series data in time axis. W is the window size.

When generating labels for samples, label "1" represents eating and label "0" represents no eating. In data collection step, each meal period has been self-reported by user. Then each window is labelled as $y_i = 1$ (eating) if more than 50% of the window overlaps with a self-reported meal, otherwise $y_i = 0$ (non-eating). This is as same as the method in previous work [27].

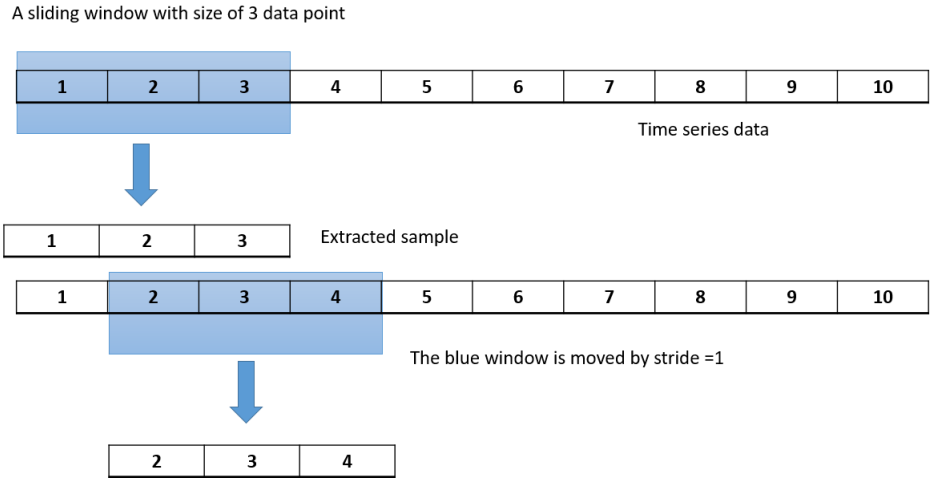

Figure 2.6: An example of sliding window with size of 3 data points to extract sample from 1-axis time series data of 10 data points. The stride in this example is equal to 1, so sliding window moves forward by 1 data point to extract the next sample

2.3 Neural Network Architecture

This section introduces the model architecture, convolution neural network, used in this thesis as well as the parameter settings used to train our models.

2.3.1 Objective Function

As the task in this work is to detect meal events in wrist motion data, which can be converted to a classification problem to classify if the window sample at a moment is 1 (eating) or 0 (non-eating). In classification problem, cross entropy loss is a common loss function used to train a classifier. Cross-entropy is a measure from the field of information theory, building upon entropy and calculating the difference between two probability distributions. When using cross entropy loss to train classifier, the output distribution from classifier is getting close to the distribution of ground truth in data set.

Binary cross entropy is a case of cross entropy. It considers that data set has two classes only. Two classes are mutual exclusive. Consider the definition of the binary cross entropy [9]. The feature space is denoted as $X \subset R^{d \times W}$. The label space is denoted as $\mathcal{Y} = \{0, 1\}$. There is a set of data tuples with size of N. The data set is denoted as $D = \{(x_i, y_i)\}_{i=1}^{N}$. x_i represents the i^{th} sample, $x_i \in X$. y_i is the label corresponding to x_i and $y_i \in \mathcal{Y}$. A binary classifier model outputs a scalar value with the meaning of possibility and it is denoted as $f : R^{d \times W} \rightarrow R$. θ is the set of parameters of the classifier model. Then the binary cross entropy has the following formula:

$$CELoss = -\frac{1}{N}\sum_{i=1}^{N}(y_i log(f(x_i;\theta)) + (1 - y_i) log(1 - f(x_i;\theta))) \tag{2.5}$$

Eating detection task in this work is a binary classification problem and there are only two classes. Eating is labelled as "1" and non-eating is labelled as "0". Sliding window with length of 5400 data points is used to extract sample x_i from 6-axis time series data. Hence dimension d=6 and $W = 5400$ and $x_i \in R^{6 \times 5400}$. The the neural network model with a set of parameters θ can output a scalar value, representing the possibility of eating. If this possibility is higher than a threshold, prediction becomes "1", otherwise, "0".

2.3.2 Model Architecture

The deep learning model architecture in this work is as same as previous work [27]. The convolution neural network used in this work consists of 3 different kinds of layers: convolution layer, 1D global average pooling layer and dense layer. Figure 2.7 shows the architecture of this convolution neural network as well as the output shape and the amount of trainable parameters used in each layer. In the column of "Output Shape", each 1D Convolution layers (Conv1D) has output shape tuple: (higher dimension, size, depth), and other layers have output shape tuple: (higher dimension, size). For all layers, higher dimension is None. For layers "G1" and "Dense", they output 1-dimensional vectors and hence their output shapes have size only, without the dimension of depth. Each sample input to this neural network has the size of 5400 data points (6minutes $\times$ 60 sec/minute $\times$ 15Hz) and depth of 6 (6 axes: x, y, z, roll, pitch, yaw). Hence each sample is a matrix with 6 columns and 5400 rows. The output from the neural network is a scalar value between 0 and 1, representing the possibility that the input sample belongs to the class of eating. In the later of this section, the function and principle of each layer are discussed.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 2679, 10)          2650
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1330, 10)          2010
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 664, 10)           410
_________________________________________________________________
global_average_pooling1d (Gl (None, 10)                0
_________________________________________________________________
dense (Dense)                (None, 200)               2200
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 201
=================================================================
Total params: 7,471
Trainable params: 7,471
Non-trainable params: 0
```

Figure 2.7: CNN architecture information printed by tensorflow

2.3.2.1 Convolution Layer

Convolution layer is a significant layer in neural network, which usually works as a feature extractor to extract the main important patterns from the input by using filters and convolution operation. Convolution layer consists of a set of filters with learnable weights and biases. Each filter is repeatedly convolved throughout the entire length of the data for generating a useful representation. Since the weights in filter are shared to do convolution along the length of data , convolution layer has the property of parameter sharing, which can control the number of parameters and boost the training speed by using fewer parameters.

The convolution operation can be 1-dimension or 2-dimension. 1D convolution layer is usually applied in time series data. Consider 1D convolution layer here. The input has the shape of $(size, depth)$. The number of data point along the time axis is called the size. The number of channels, or axes of sensors in input feature is called the depth. In 1D convolution layer, each filter, also called kernel, is a matrix and has the shape of $(k, depth)$, where k is called kernel size and it is the length of convolution filter. Depth of the filter is equal to the depth of input sample. During convolution, some additional data points can be added to the boundary of the input matrix to avoid filter moves outside the region of the sample. Such method of adding additional data points is called *padding*. In a convolution layer, it can contain multiple filters. The number of filters in the convolution layer is equal to the depth of the output feature map after convolution. During convolution, the stride of the kernel is denoted as s, which plays an important role in the feature generation part as it determines the shift of convolution filter, as it runs over the length of the sequence. It can be used to control the size of the output.

In 1D convolution layer, we can compute the size of its output based on the shape of input and shape of the filter. The size/length of the output feature can be calculated by the formula:

$$Output\ size = \frac{input\ size - k}{stride} + 1 \tag{2.6}$$

As mentioned before, k is the size or length of the kernel. The depth of the output is equal to the number of filters in this convolution layer, since each filter applied to the input sample can generate one channel of feature. Additionally, the amount of trainable parameters in one convolution layer is given by:

$$N = k \times d \times m + m \tag{2.7}$$

where N is the amount of trainable parameters. k is the kernel size. d is the depth of input. m is the number of filter in the convolution layer, or the depth of output. The total amount of weights in all filter is equal to $k \times d \times m$. Each filter contains one bias value and hence there are m bias values needed to be added.

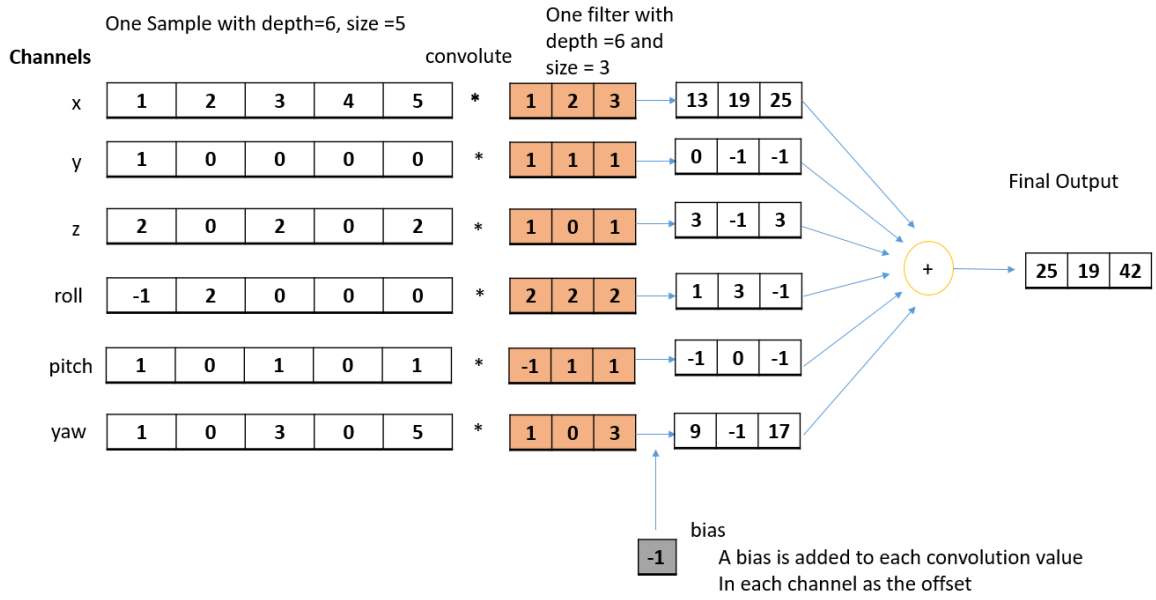

Figure 2.8: Example of convolution layer with 1 filter on 1D time series data with depth of 6 (6 rows) and size of 5 (5 columns), stride of 1. The orange matrix represents a filter in 1D convolution layer.The grey cell is a bias term in convolution layer

There is one example of 1D convolution in figure 2.8. In this example of convolution layer, the input sample is a time series data with size of 5 and depth of 6 (6 axes/channels: x,y,z,roll, pitch, yaw). There is only one filter (the orange array) in this convolution layer. The filter has kernel size of 3 and depth of 6 and stride of 1. According to equation 2.6, the output size is equal to $\frac{5-3}{1} + 1 = 3$. Each output value in each channel in convolution is given by:

$$y_{c,i} = \sum_{j=0}^{k-1} w_{c,j} x_{c,i+j} + b \tag{2.8}$$

where c is the index of channel/row and i is the index of position of column which starts from 0. $w_{c,j}$ is the weight at c^{th} row and j^{th} column of filter. b is the bias term. After convolution is

applied to each channel, all vectors are added up together to get the final output from this filter. For example, in the convolution of channel x, based on equation 2.8, the first output value is equal to $1 \times 1 + 2 \times 2 + 3 \times 3 - 1 = 13$. This process is as same as 2D convolution layer in [7].

In this work, three different convolution layers without using padding are used to transform the feature. The first convolution layer has the shape of $(k = 44, depth = 10, stride = 2)$, the second convolution layer has the shape of $(k = 20, depth = 10, stride = 2)$, while the third convolution layer has the shape of $(k = 4, depth = 10, stride = 2)$. Input sample has size of 5400 data points and depth of 6. Hence by using equation 2.6, output from the first convolution layer has size of (5400 - 44)/2 + 1= 2679 and depth of 10. According to equation 2.7, the first convolution layer has $44 \times 6 \times 10 + 10 = 2650$ parameters. Similarly, the output shape and parameter amount can be computed in the same way. All three convolution layers in the model use L1 norm regularization to reduce the effect of over-fitting [27, 20]. After each 1D convolution layer, a element-wise ReLu (rectified linear unit) activation function is used to filter the output values.

2.3.2.2 Global Average Pooling Layer

1D global average pooling layer is a layer that extracts the features and reduces the size of input by replacing each channel with the mean value of this channel, so that the dimension of the feature can be reduced. In figure 2.7, after applying the global average pooling1d layer, the dimension of input (None, 664, 10) is reduced to (None, 10) and only 10 scalar values are remained. The layer doesn't require any parameters, so the amount of parameters in this layer is 0.

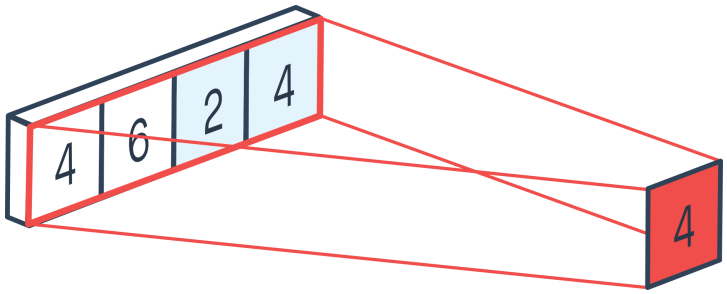

Figure 2.9: Example of 1D global average pooling. It averages the vector to get a scalar value as output and reduce the dimension of feature

2.3.2.3 Fully Connected Layer and Logistic Regression

After the global average pooling layer, a fully connected layer with 200 neurons. Fully connected layer, also called dense layer, can be viewed as a function that performs feature selection by tuning learnable weights and biases. In a fully-connected layer, each neuron takes the weighted sum of all output values from the previous layer as input and then an element-wise activation function is applied. Hence, it can be expressed by equation:

$$Y = \sigma(WX + b) \tag{2.9}$$

where X is an input vector, $X \in R^m$ and W is a weight matrix, $W \in R^{n \times m}$ and b is a bias vector, $b \in R^n$ and Y is the output vector from fully connected layer, $Y \in R^n$ and $\sigma(.)$ is an element-wise activation function. Hence the amount of parameters in fully connected layer is:

$$N = n \times m + n \tag{2.10}$$

where n is the length of output vector and m is the length of input vector. In the model architecture, there are $200 \times 10 + 200 = 2200$ trainable parameters in the first dense layer, whose dense layer use a ReLu activation function. In the second dense layer "$dense\_1$", it projects the feature vector into a scalar value and then apply a logistic function to project the range of value into [0, 1] as the possibility of eating. The logistic activation function, also called sigmoid function has the form:

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

In the dense layer with sigmoid activation function, there are 201 trainable parameters with 200 weights and 1 bias.

2.3.2.4 Summary of CNN architecture

Figure 2.10 summarizes the architecture of our CNN model, it is as same as the architecture in figure 2.7. The input to the network is a window of 6 minutes with 6 axes. Hence each input has size of 5400 (6minutes $\times$ 60 sec/minute $\times$ 15Hz) with depth of 6. In each convolution network, k is the kernel size or the length of kernel. The 1D global average layer converts the output from the last

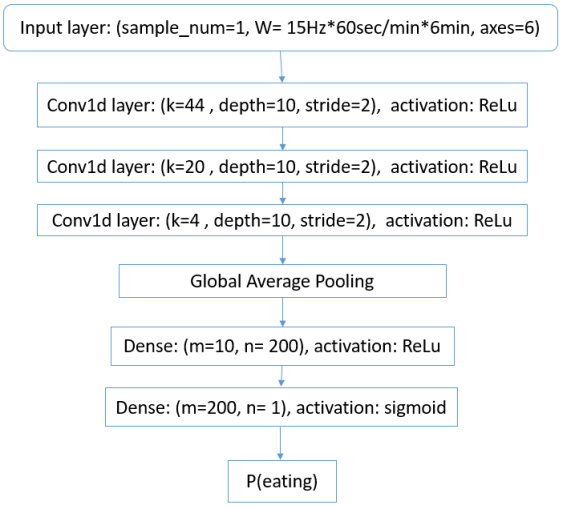

Figure 2.10: Figure of summary of CNN architecture used in this work. W is the size/length of input and axes is the amount of axes/channels. k is the kernel size. n is the size of 1D input vector and m is the size of 1D output vector.

CNN layer to a vector with size of 10. In dense layer/fully connected layer, m is the size of input vector and n is the size of output vector. The last dense layer uses a sigmoid activation function while other layers use ReLu activation functions. The output from the network is the possibility of eating.

2.3.2.5 Other Settings

1) **Training Epoch:** One training epoch is defined as one pass to the whole training data set. This parameter control the times that the model is trained on the training set in order to reduce the loss of model. The maximum training epoch for the group model is 150 to replicate the setting in previous work [27]. The training epoch for individual models is set to be 20 for all experiments, since the losses in all training experiments can already converge without obvious changes within 20 epochs due to the small size of data set.

2) **Batch size:** During training, the whole training data set is divided into multiple batches with equal size. A model is trained on one batch and then updates its parameters based on the loss on that batch. If the batch size is too large, for example, when batch size equal to the size of training data set, then the update of model parameters is too slow and can also lead to software out of memory due to loading a very large data to memory from disk. However,

if the batch size is too small, like only one sample in one batch, then the update in model parameter can be noisy and hard to converge. Hence, when choosing batch size, the common way is to choose a mini-batch to train the model. The batch size selected in this work is 128 samples, which is suitable in this case without out of memory problem in software and make model training stable.

3) **Optimizer:** In order to update parameters in deep learning model, gradient descent based algorithm is usually utilized to compute the change of parameters and adjust the model's parameters. The basic gradient descent has the following equation:

$$\theta_j = \theta_j - \alpha \frac{\partial L(\theta)}{\partial \theta_j} \tag{2.12}$$

where θ is a set of learnable parameters in deep learning model and θ_j is the j^{th} parameter in that set. $L(\theta)$ is the loss of prediction from model, when using hyper-parameter θ. α is the learning rate that control the update rate of parameter. While there are many different optimization algorithms, the optimizer used in this work is called Adam optimizer, which is a popular optimizer and is computationally efficient, has little memory requirements,and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients [11]. The default learning rate in Adam optimizer is 0.001.

4) **Learning Rate:** learning rate controls the step or the size of change in parameter. As shown in figure 2.11, if the learning rate is too small, then the model need to longer training time in order to let loss converge to the local minimum. However, if the learning rate is too large, the step of change in parameter becomes to large, then the parameter will skip the best parameter corresponding to a local minimum of loss. Hence, a suitable learning rate is required. The learning rate is left to be default of 0.001 in this work since it already allows our models to converge quickly.

5) **Early Stopping:** Early stopping is a technique to avoid over-fitting during model training. Over-fitting is a phenomenon that a model performs very well on training set but performs worse on test set that the model has not seen before. During training the group model, early stopping is used to monitor the accuracy on validation set and save the model with the best

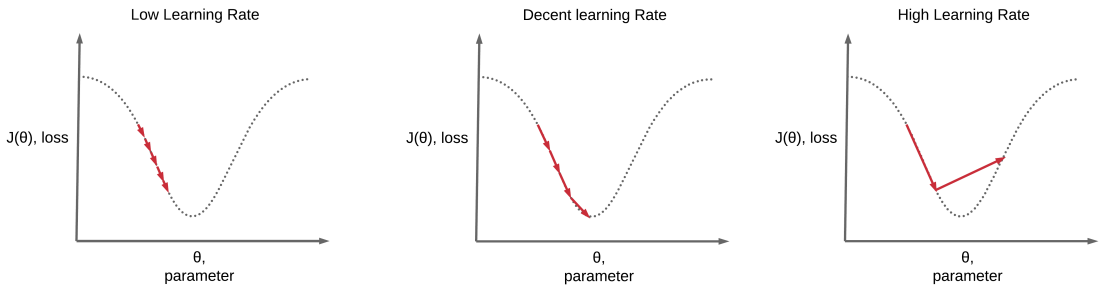

Figure 2.11: Figures of effect of different learning rates. The figures from left to right shows the effect of small, suitable and large learning rate on training respectively

validation accuracy so that the group model is not affected by over-fitting, as there is a risk of over-fitting when training the group model with 150 epochs from previous work [27].

6) **Hardware and Software:** All CNN models are constructed by using tensorflow2.3 in Python. The hardware platform used in this thesis is Palmetto Distribution Cluster from Clemson University with 4 CPUs and 62GB RAM and 2 GPUs of V100 with walltime of 24 hours.

2.4 Post-Processing and Hysteresis Threshold

As the task of eating detection is to detect meal episodes, it is necessary to segment the time series data in each day of data by finding the start boundary and the end boundary of each eating episode. There is a segmentation method for finding eating episode from our group's previous work [26, 27], called hysteresis threshold.

The basic idea behind hysteresis threshold is that for each day of time series data, it uses sliding window method to generate continuous samples and then tests CNN models on these samples to generate a sequence of possibility of eating $P(s_i)$ continuously. s_i is the i^{th} sample extracted from the window. Then when p(s_i) goes higher than T_s (start threshold), we mark the start of a detected eating episode. When p(s_i) goes less than T_e (end threshold), we mark the end of an eating episode. The use of two thresholds serves two purposes. First, eating can be more vigorous at the beginning of a meal, which can lead to a strong probability of eating. T_s starting threshold helps detect such vigorous behaviors in a meal and reduce false positive. Second, eating behavior can fade with satiety and becomes less vigorous. For example, a person usually slows down the eating speed and takes a rest at the end of a meal, leading to a lower possibility of eating. Hence a lower T_e ending threshold

can help detect the end boundary of a meal. T_e determines the length of the detected eating episode. Usually when T_e is higher and it is more likely to meet possibility lower than T_e and make the episode shorter.

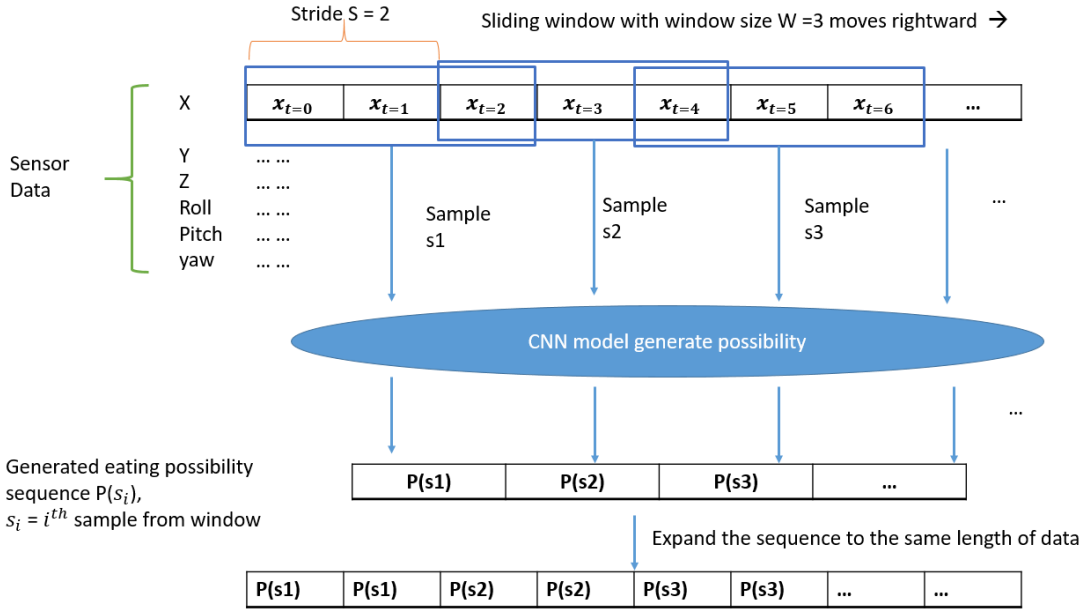

Figure 2.12: Example of generating possibility sequence using window with size of 3, stride of 2 on 6-axis data. Note that extending the possibility sequence to get original length is optional, since we can convert the index in possibility sequence back to time unit to get the segment as well.

For example, in figure 2.12, a sliding window with size of 3 data points and stride $S = 2$ is used to extracted the samples s_i and then CNN model generate the possibility that the sample belongs to eating behavior for such samples. The possibility sequence can be mapped to the same length as the length of time series data by extending each possibility $P(s_i)$ to the neighbor positions.

Figure 2.13 shows an example of generated possibility sequence and threshold result. The sequence of possibility of eating with 54000 data points from day 0 from one individual data is plotted in the top figure. X axis is about the index of data point along time. The grey curve represents the sequence of possibility of eating. $T_s = 0.8$ and $T_e = 0.4$ are used. The green region is a detected meal segmentation. The blue region is the ground truth label of meal episode reported by user. To better show the detected segmentation and the ground truth meal episode, the second figure and third figure are shown. The second figure (green curve) shows the prediction of eating episode from the model. The third figure (blue curve) displays the ground truth label of eating

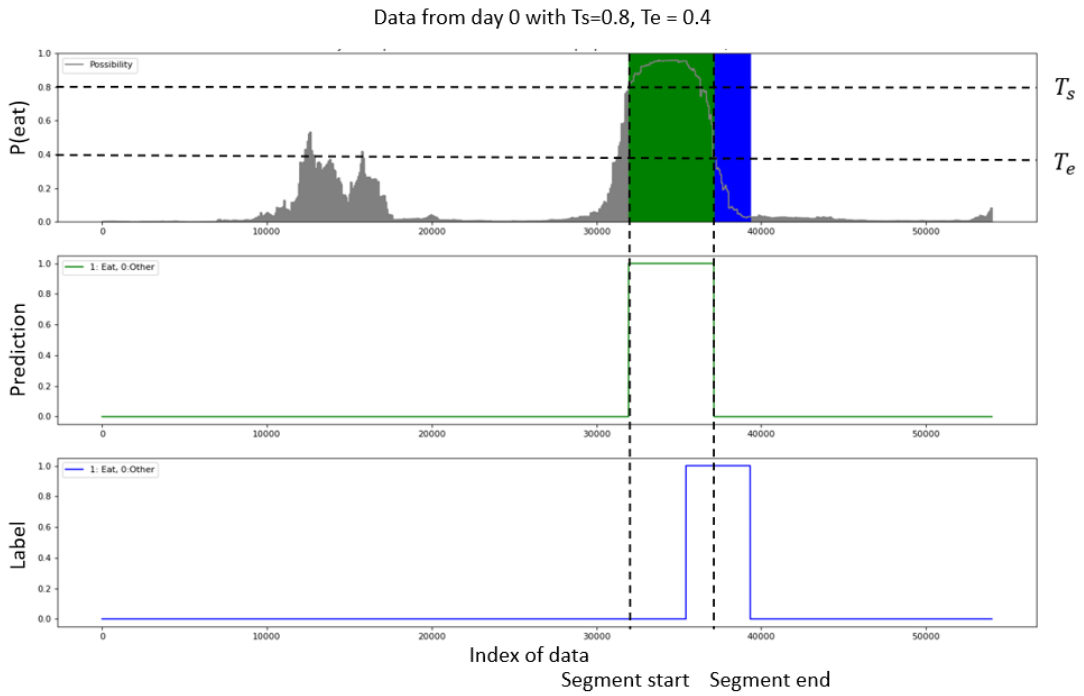

Figure 2.13: Figure of generated possibility sequence and detected segment and ground truth label. Grey curve is the sequence of possibility of eating from classifier. Green curve is the predicted segment after using hysteresis threshold. Blue curve is the ground truth label of eating episode. "1" means eating and "0" means non-eating.

episode. If the detected segment and the ground truth episode overlap, it means the model detects this meal successfully.

In this work, thresholds $T_s = 0.8$ and $T_e = 0.4$ from the work [27] are used in the group model. We call them the group thresholds. We first applied the group thresholds to individual models and the group model and compare their performance in episode metrics. Then we tuned T_s and T_e for individual models of each individual to improve individual models' performance to see if tuning individual parameters can improve performance of individual models. Details will be discussed in section 2.7.

2.5 Metrics Selection

This section discusses the metrics used to compare the performance of individual models and the group model and the formulas of these metrics.

2.5.1 Time Metric

Time metric measures the performance at labeling every moment of time or sample extracted from sliding window correctly as eating or non-eating throughout the day. Time metric is tested on window samples at some moments. The time metric used in this work is weighted accuracy (WAcc). In order to understand weighted accuracy, consider the confusion matrix [2] below:

		Ground Truth label	
		Eating	Non-Eating
Prediction	Eating (Positive)	True Positive (TP)	False Positive (FP)
	Non-Eating(Negative)	False Negative(FN)	True Negative(TN)

Table 2.2: Confusion Matrix of eating and non-eating

Ground truth label is the actual label reported in the data set while prediction is the prediction class from classifier. Since in the task of detecting meal, both ground truth and prediction have eating label "1" and non-eating label "0" only. A prediction is counted as a true positive (TP) when the prediction is eating and the actual label is also eating. A prediction is counted as a false positive (FP) when the prediction is eating but the actual label is non-eating. A prediction is counted as a false negative (FN) when prediction is non-eating, but the label is actually eating. A prediction is counted as true negative (TN) when prediction is non-eating and the label is also non-eating. Then we can compute the total amount of TP, TN, FP, FN to fill the confusion matrix. Then weighted accuracy is given by the formula:

$$WACC = \frac{W \times TP + TN}{W \times (TP + TN) + (FP + FN)} \tag{2.13}$$

where W is the balance ratio of each individual It can be calculated by equation 2.1. Balance ratio of each individual can be found in table 2.1. The balance ratio of CAD group data set is 20 according to previous work [27].

The reason why we use weighted accuracy as time metric is that eating detection data is usually significantly imbalance and the amount of non-eating samples is much greater than the amount of eating samples. For example, a person can record one day of data in 11 hours, but there is only 1 hour for eating and the remaining 10 hours are non-eating, leading to a problem that negative sample amount in data set is much greater than positive sample amount. Traditional accuracy in equation 2.14 can not reflect the real performance of classify. For example, if a classifier

can predict negative samples only and doesn't recognize positive samples well in imbalance data set that contains much more negative samples than positive samples, its accuracy is very high, but doesn't reflect the classifier's performance on positive sample. WAcc is a more reliable indicator of performance across data sets containing different balances.

$$ACC = \frac{TP + TN}{(TP + TN) + (FP + FN)} \tag{2.14}$$

2.5.2 Episode Metric

Episode metrics measure the performance of detecting meals or snack. The unit is a meal or snack. Episode metrics need to be calculated in a whole day of time series data rather than window samples at some moments. In episode metrics for eating detection, a meal with any amount of overlap with a detected segment is counted as a TP. A meal with no overlap with a detected segment is counted as a miss, false negative (FN). A detected segment not overlapping any meal is counted as a FP. In eating detection task, we only care the performance of model in detecting eating episodes, rather than non-eating periods, TP, FN and FP are considered only. True negative (TN) is not considered in episode metrics. Then the episode metrics are

$$TPR = \frac{TP}{TP + FN} \tag{2.15}$$

$$FP/TP = \frac{FP}{TP} \tag{2.16}$$

where TPR measures how sensitive the model is to the real meal episodes and FP/TP measures the ratio of false eating prediction over true eating prediction. When TPR is larger and FP/TP is smaller, the model performs better in detecting meals.

Figure 2.14 shows an example of 5 cases of detection that can happen in a time series data. The large rectangle frame represents one day of time series data between 8am and 8pm. The grey boxes are the ground truth labels of meal episodes and the blue boxes are the segments predicted by model. In case 1 to 3, actual meal episodes are detected by classifier, since those meal episodes overlap the detected segments from classifier (blue boxes). Hence there are 4 meals detected successfully and those meal episodes are labelled as TP, TP = 4. As for case 4, the meal is not

detected and there is one FN. For case 5, the classifier detected a non-eating period as eating episode and hence it is one FP. In this example, TPR = TP/(TP+FN) = 4/(4+1) = 0.8 and FP/TP = 1/4 = 0.25. Note that for some days of data, it may not contain a meal, leading to TP equal to 0. In this case, we compute the TP, FN, FP from all days of data of an individual and then sum them up before computing TPR and FP/TP.

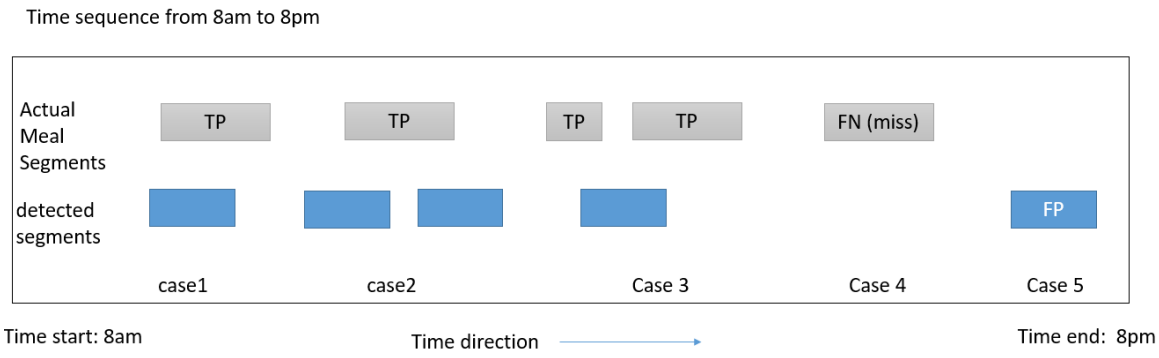

Figure 2.14: Example of episode metrics on detection on one day of time series data. The big recctangle frame represents time series data from 8am to 8pm. Grey boxes represent the actual meal episodes and blue boxes represent the segments detected by the classifier and hysteresis threshold

2.6 Model Training and Evaluation

2.6.1 Imbalance Data

Before training our models, we need to consider the imbalance problem in our data sets. Imbalance problem is a common problem in human activity data, in which some classes contain much more samples than other classes. In each individual data set, the ratio of non-eating hour over eating hour is around or above 10, which shows the serious imbalance problem between non-eating and eating classes. Such imbalance data can lead to a bias classifier that can predict much more samples as non-eating very easily and fewer samples as eating. This can make the classifier miss more eating episodes that we are interested in and decrease episode metrics performance (lower TPR and higher FP/TP).

In order to solve this problem, two simple ways can be considered: under-sampling and over-sampling. Under-sampling is to simply reduce the amount of the majority class samples by

randomly sampling part of them and let majority class have the same amount as the minority class. For example, if there is a data set with 10000 samples of non-eating class and only 2000 samples of eating class. Then we can randomly sample 2000 data from 10000 samples so that both non-eating class and eating class have 2000 samples. The second method oversampling has the opposite idea, which is to repeat sampling data from minority class to increase the amount of minority class samples. For example, in data set with 10000 samples of non-eating class and only 2000 samples of eating class, it repeats sampling data randomly with replacement from 2000 samples of eating class to increase the amount of eating class from 2000 to 10000. However, oversampling makes the training time longer and is time expensive. There are also another complex ways to balance data by generating new samples, such as using SMOTE, or generative adversarial network (GAN) [14, 16]. In this work, under-sampling method was used to balance the data set and avoid expensive training time.

2.6.2 Group Model Training

In group model training step, the CAD group data set was first smoothed by using Gaussian filter and then normalized by using z-normalization method as mentioned in section 2.2. After data processing, the CAD data set with 354 days was split into 80% of day ($\approx$ 283 days) as training set and 20 % of days ($\approx$ 71 days) as testing set, according to day. Then a sliding window with window size of 6 min (6min *60 sec/min * 15 Hz = 5400 data points) and stride of 15 seconds (15 sec * 15 Hz = 225 data points) was used to extract the window samples from training set and test set respectively. Window is labelled as eating if 50% of window fall within the reported eating period. By splitting data by days, it can avoid the data leakage problem that training set data have features overlap the testing set data. In order to solve the imbalance problem, the training set and testing set are first balanced by using under-sampling to reduce the amount of non-eating samples so that both non-eating class and eating class have the same amount.

We trained the group model on balanced training set and used the smaller balanced testing set as a validation set. Such small validation set is just used to monitor the performance of the group model and avoid over-fitting during training only. The reason of balancing the test set is that it can better reflect the model's ability of classifying eating class and non-eating class without being affected by the imbalance problem as mentioned in section 2.6.1 and this is equivalent to test the model on an imbalance data set using weighted accuracy. Early stopping method can monitor the

accuracy performance of model on this validation set and save the model with the best validation accuracy so that model is not over-fitting to the training set. Since our goal in this thesis is to compare the performance of this group model and individual models on individual data set, there is no final test on the CAD data set and this trained group model is used to compare with individual models for all experiments later.

2.6.3 K-fold Cross Validation Evaluation for individual models

K-fold cross validation is a model evaluation technique to estimate the performance of machine learning model. In cross validation, the data set is first divided into K independent subsets with equal or approximately equal size. Each subset is called one fold, hence there are K folds. Then one fold is chosen as a test set, called validation set, then remaining k-1 folds are used to train one model. After this model is trained, it is tested on the validation set to get the performance on this fold. This train-test process is repeated K iterations. In each iteration, a different fold is chosen as validation set and the model is re-initialized and re-trained on the remaining folds. Finally, there are K different models and K different performance results. Finally, these K different results are averaged to get the overall performance of this model. The process is shown in figure 2.15.

$$Final\ Performance = \frac{1}{k}\sum_{i=1}^{k} Performance_i \tag{2.17}$$

where $Performance_i$ is the performance result of the model in the i^{th} iteration, it can be accuracy, weighted accuracy, mean square error and so on. In this work, time metric and episode metrics are used to compute the performance.

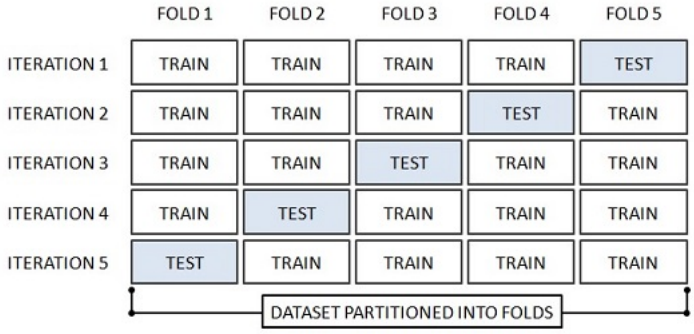

Figure 2.15: Example of 5-fold cross validation. Test set is selected differently in each iteration. Models are trained on the remaining folds.

For small data set, cross validation is a good way to estimate the performance of model on data set. For example, in small data set, like only 10 days of data, when 2 days are used as test set and 8 days are used as training set, since the test set is small, there is a case that there is 1 or 2 days contain novel eating behaviors that are quite different from eating behaviors from other days. Such days of data are used to test the model. Then model performance can be under-rated. Oppositely, there is also a case that there are some days containing eating behaviors that can be classified much easier than eating behaviors in other days and all these good days of data are used to train model, leading to overrated model performance. By applying cross validation to evaluate the small data set, it can reduce the effect of overrating and underrating and provides a more robust estimated performance on model.

In this work, we applied 5-fold cross validation for data of each individual. Assume one individual has 10 days of data, then each fold contains $10/5 = 2$ days. In each fold, 80% of days ($\approx$ 8 days) of data of one individual were used as training set and the remaining 20% of days ($\approx$ 2 days) were used as test set. A sliding window with window size of 6 minutes and stride of 5 seconds was used to extract samples from training set and testing set. As mentioned in section 2.2.3, stride of 5 seconds was to create more samples and diminish under-fitting effect, compared with the stride of 15 second used in CAD data set. Then under-sampling was used to balance the amount of non-eating and eating samples in training set right before training CNN models. In testing step, weighted accuracy was used as time metric to measure model's ability in classifying samples as eating or non-eating in imbalance test set. The overall time metric is the average of weighted accuracy of 5 folds. When measuring episode metrics, CNN model first generates possibility sequence on the time series data of each day in each testing fold and then hysteresis threshold is applied to generate TP, FP and FN. Since some days may not contain meals and TP can be 0, making FP/TP invalid and TPR become 0. To solve this problem, TP, FP and FN of all folds were first measured and then sum up together to get the TP, FP and FN for the whole data set for one individual. After that, TPR and FP/TP were calculated.

2.6.4 Performance Comparison

As the goal in this thesis is to compare the performance between the group model and individual models of each individual on the same individual data set, the group model need to be tested on the individual data set in the consistent way as how individual models are tested.

However, individual models were trained and evaluated in cross validation on individual data set, but the group model was trained on the CAD data set instead. In order to test the group model in individual data set in the consistent way, we tested the group model on each testing fold in 5-fold cross validation and then measured its WAcc and TP, FP, FN in each fold. Then the final WAcc of the group model is the average of WAcc of all folds and the TPR and FP/TP are computed after summing up all TP, FP, FN from all folds. The process is shown in figure 2.16. While there are 8 individual data sets, 5-fold cross validation is repeated 8 times, but there is only one group model trained to compare with all other individual models.

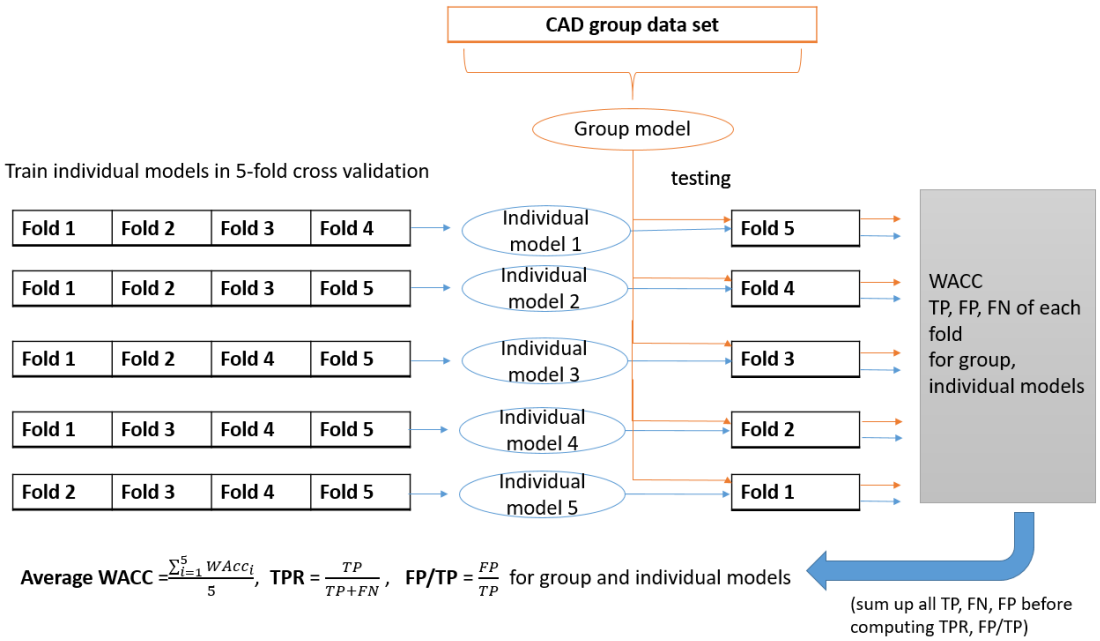

Figure 2.16: Example of performance comparison for the group model and individual models on data of one individual. The group model is trained on CAD group data set and then tested on each testing fold in cross validation in individual data of one individual. As for the individual models, they are trained and evaluated by 5 fold cross-validation

2.7 Hyper-Parameters Tuning

This work also investigates how tuning hyper-parameters can improve individual models' performance. There are 3 tuning hyper-parameters that can be investigated: window size, threshold T_s and threshold T_e in hysteresis threshold method.

In previous work of our group [26], window size effect was well investigated and the author

used cross validation method to validate the window effect and suggested to use a 6 minute window size in the group model. However, investigating the window size effect on individual models for each individual can be very time expensive. For example, training a single individual model can take M minutes and we want to use W different window sizes to investigate window effect. After applying 5-fold cross validation for N individuals, it can take $M \times 5 \times N \times W$ minutes. If there are 8 participants, $M = 15$ minutes and $W = 10$ different window sizes, then it requires 6000 minutes, around 4 days for testing. As this is very time-consuming, we don't investigate the effect of window size here, but leave it to future work.

When investigating the effect of T_s and T_e, we first let individual models generate the sequence of possibility of eating on all days of data. We then applied a grid search method that loop through all possible combinations between T_s and T_e based the set of T_s: [0.85,0.8, 0.75, 0.7, 0.65, 0.6, 0.5] and the set of T_e: [0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45]. As mentioned in section 2.4 about hysteresis threshold, T_s is to detect the higher possibility of eating and look for the start of eating episode while T_e looks for the lower possibility of eating and determine the end boundary of eating episode. Hence values in T_s set are higher than the values in T_e set. After looking for all combinations of T_s and T_e, we used three different strategies to search the thresholds that improve the episode metrics.

Algorithm 1 optimize TPR

```
input S_0 = the set of all T_s,T_e combinations with their TPR, FP/TP results
T_0 = TPR threshold
find set S_1 of all T_s,T_e pairs with TPR ≥ T_0 from S_0
if S_1 not empty then
  return the T_s,T_e pair with minimum FP/TP in S_1
else
  return the T_s,T_e pair with maximum TPR in S_0
end if
```

The first strategy focuses on TPR performance, called TPR method. The logic is that we first set a threshold T_0 for TPR and then find the set of pair of T_s and T_e with TPR greater or equal to T_0. Then we find the T_s and T_e pair in this set with minimum FP/TP value. If we don't find the set of pairs of T_s and T_e with TPR greater than or equal to T_0, then we simply return the T_s and T_e pair with maximum TPR, as shown in algorithm 1. This algorithm can have good TPR performance since it either returns TPR above the threshold T_0 or returns max TPR when the threshold condition is not met. TPR performance is given priority higher than FP/TP performance

in this algorithm. Note that the algorithm may return multiple T_s, T_e pairs when multiple T_s, T_e pairs have the same maximum TPR in S_0 or minimum FP/TP in S_1. In this case, we simply return the T_s and T_e pair with the smallest T_s. If T_s values are same in those results, then we return the T_s and T_e pair with the smallest T_e.

The second strategy emphasizes on the FP/TP performance, called FP/TP method. The logic is that we first set a threshold T_1 for FP/TP and then find the set ofT_s, T_e pairs with FP/TP smaller than or equal to T_1. Then we find the T_s, T_e pair with maximum TPR from this set. If the set is empty, we simply return the pair of T_s and T_e with minimum FP/TP directly, as shown in algorithm 2. This algorithm gives higher priority to FP/TP performance than TPR performance, since it first considers the T_s, T_e pair with FP/TP smaller than the threshold T_1. Only when it finds a set satisfying this condition, it then finds T_s, T_e with maximum TPR in this set. Otherwise, it only considers T_s and T_e with the minimum FP/TP performance. If algorithm 2 has multiple T_s, T_e pairs that have the same maximum TPR in S_1 or minimum FP/TP in S_0, we return the T_s and T_e pair with the smallest T_s. If T_s values are same in those results, we return the T_s and T_e pair with the smallest T_e.

Algorithm 2 optimize FP/TP

```
input S0 = the set of all Ts,Te combinations with their TPR, FP/TP results
T1 = FP/TP threshold
find set S1 of all Ts,Te pairs with FP/TP ≤ T1 from S0
if S1 not empty then
  return the Ts,Te pair with maximum TPR in S1
else
  return the Ts,Te pair with minimum FP/TP in S0
end if
```

The third strategy considers both TPR and FP/TP performance, called balance method. It first computes TP ratio in equation 2.18 for each T_s, T_e pair. It then returns the T_s, T_e pair with the maximum TP ratio. When a T_s, T_e pair leads to greater TP and smaller FP, FN, the TP ratio becomes larger and this pair is more likely to be selected. Hence this method finds the thresholds that balance the performance of TPR and FP/TP. If multiple T_s, T_e pairs have the same maximum TP ratio, the T_s and T_e pair with the smallest T_s is returned. If their T_s values are same, then we return the T_s and T_e pair with the smallest T_e instead.

$$TP\ ratio = \frac{TP}{TP + FP + FN} \tag{2.18}$$

With three strategies above, we obtained 3 different individual thresholds pairs T_s, T_e for each individual. We compared the episode metrics of using these individual thresholds with the episode metrics of using the group model threshold $T_s = 0.8$ and $T_e = 0.4$ from previous work [27].

In this work, we simply used $T_0 = 0.8$ for TPR method and $T_1 = 1$ for FP/TP method. The final strategy we choose is the third method since it provided the best average improvement in FP/TP and TPR, which is discussed in the next chapter. The research about the effect of T_0 and T_1 on TPR and FP/TP performance can be left to the future work. Note that TPR method and FP/TP method may not be able to improve TPR or FP/TP respectively in some cases when comparing individual models before and after applying such methods. For example, if individual models of one individual using the group thresholds T_s and T_e have TPR = 0.9 and FP/TP = 1.0. When applying TPR method of algorithm 1 by using threshold $T_0 = 0.8$, then it could return thresholds T_s and T_e with performance TPR = 0.85 and FP/TP = 0.1 as it returns the thresholds with minimum FP/TP when TPR ≥ 0.8. In this case, TPR actually decreases after using TPR method. Similarly, after using FP/TP method with $T_1 = 1.5$, it is also possibility that it returns thresholds with TPR = 0.95 and FP/TP = 1.3, leading to higher TPR but worse FP/TP. This depends on the thresholds T_0 and T_1. Hence the TPR method doesn't always guarantee TPR performance can be improved. FP/TP method also doesn't always guarantee FP/TP performance can be improved as well. It depends on T_0 and T_1 used. It could be a research question for the future work.

Chapter 3

Results

This chapter discusses the results about time metric (weighted accuracy) and episode metrics (TPR and FP/TP) measured from the individual models and the group model and the effect of T_s and T_e on the improvement of episode metrics. This chapters first discusses the distributions of possibility outputs from individual models and the group model and see how the outputs from individual models differ from the outputs from the group model in section 3.1. Section 3.2 discusses the results of time metric as well as the episode metrics measured from individual models and the group model. In section 3.3, we first illustrates the effect of tuning T_s and T_e on TPR and FP/TP and then discusses three methods of tuning thresholds T_s and T_e we used to improve TPR and FP/TP performance. The improvement results are shown in this section and appendices.

3.1 Distributions of Model Outputs

In 5-fold cross validation mentioned in section 2.6.4, we obtained the possibility sequence of each day in testing folds generated from individual models and the group model. For each individual data set, we first used these possibility sequences to visualize the eating class distribution and non-eating class distribution of the output from the group model and individual models on that data set. Figures 3.1, 3.2 and 3.3 show the distributions of the output possibilities from the group model and individual models within eating periods and non-eating periods on 8 individual data sets. The yellow regions represent the distributions of output possibility within self-reported non-eating periods. The blue regions represent the distributions of output possibility within self-reported eating

periods labelled by users. In each figure, X-axis is about the output values from models (possibility of eating) and Y-axis is about the probability of models' output values. In the left column of each figure, each plot shows the distributions of output from the group model trained and tested on each individual data set. The right column shows the distributions of output from individual models on the corresponding individual data set.

In figures 3.1, 3.2 and 3.3, we find that individual models trained and tested on individual data sets 1, 2, 4, 5, 6 and 8 respectively can well classify the eating and non-eating behaviors, since the eating distributions of these data sets focus on the output value of 1 and non-eating distributions focusing on the output value of 0, meaning that the models always output high possibilities in eating period reported by users, and low possibilities in non-eating period. For individual data sets with identities of 3 and 7, the eating distributions spread out and the probabilities of smaller output values are greater than the probabilities of the larger output values. For example, in individual models' prediction on data set 7, the probability of output value of 0.2 is much higher than probability of output value of 1. This means in the periods labelled as eating, most of output values from individual models trained on this data set are around 0.2. The individual models can not recognize the eating period on this data set very well.

In most individual data sets, the eating distribution and non-eating distribution from the group model overlap obviously. For example, in data set 3 in figure 3.1, the yellow region and the blue region overlap obviously. Both non-eating distribution and eating distribution focus on the range of output value around 0.2 to 0.4. This means the group model, when testing on data set 3, doesn't classify the eating behavior well, leading to more lower eating possibility in eating period and higher eating possibility in non-eating period. Compared with the individual models trained on data set 3, the group model shows worse classification performance. This also implies that the eating behavior patterns in data set 3 are very likely different from the eating behavior patterns in CAD group data set. Similarly, in data sets 1, 4, 5, 6 and 8, the performance of the group model may have worse classification ability than individual models as the eating distributions and non-eating distributions from the group model overlap more obviously.

In short, there is a trend that compared with the group model, individual models trained on individual data sets can output higher eating possibilities of eating in eating periods and lower eating possibilities in non-eating period more easily and better separate distributions of eating and non-eating. Comparison between models by different metrics will be discussed later.

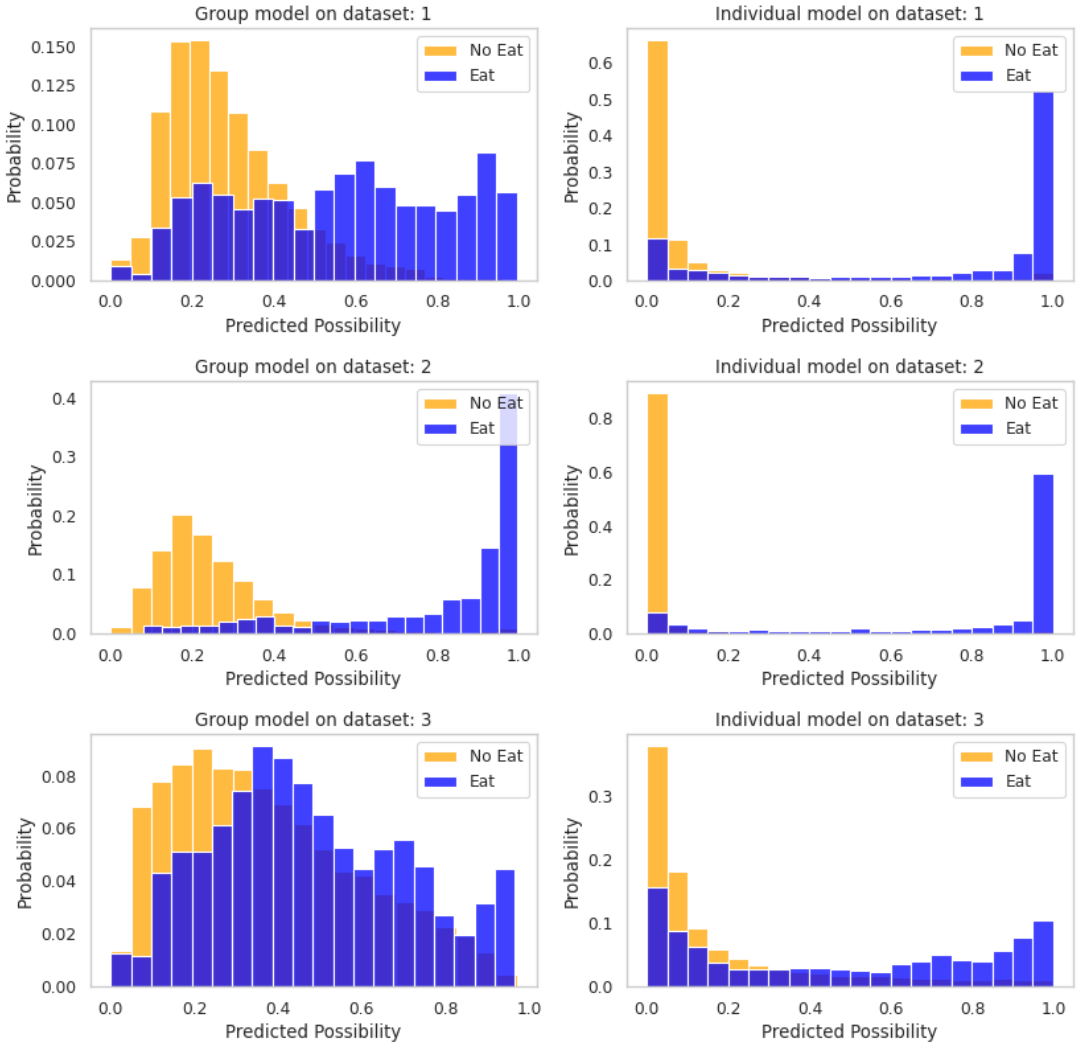

Figure 3.1: Figure of distributions of output probabilities on individual data sets 1, 2 and 3. In the left column, probabilities for the group model were calculated by training on the CAD dataset and testing on the individual. In the right column, probabilities for the individual models were calculated using 5-fold cross validation. Yellow histograms are distributions of models' output in ground truth non-eating periods. Blue histograms are distributions of models' output in ground truth eating periods.

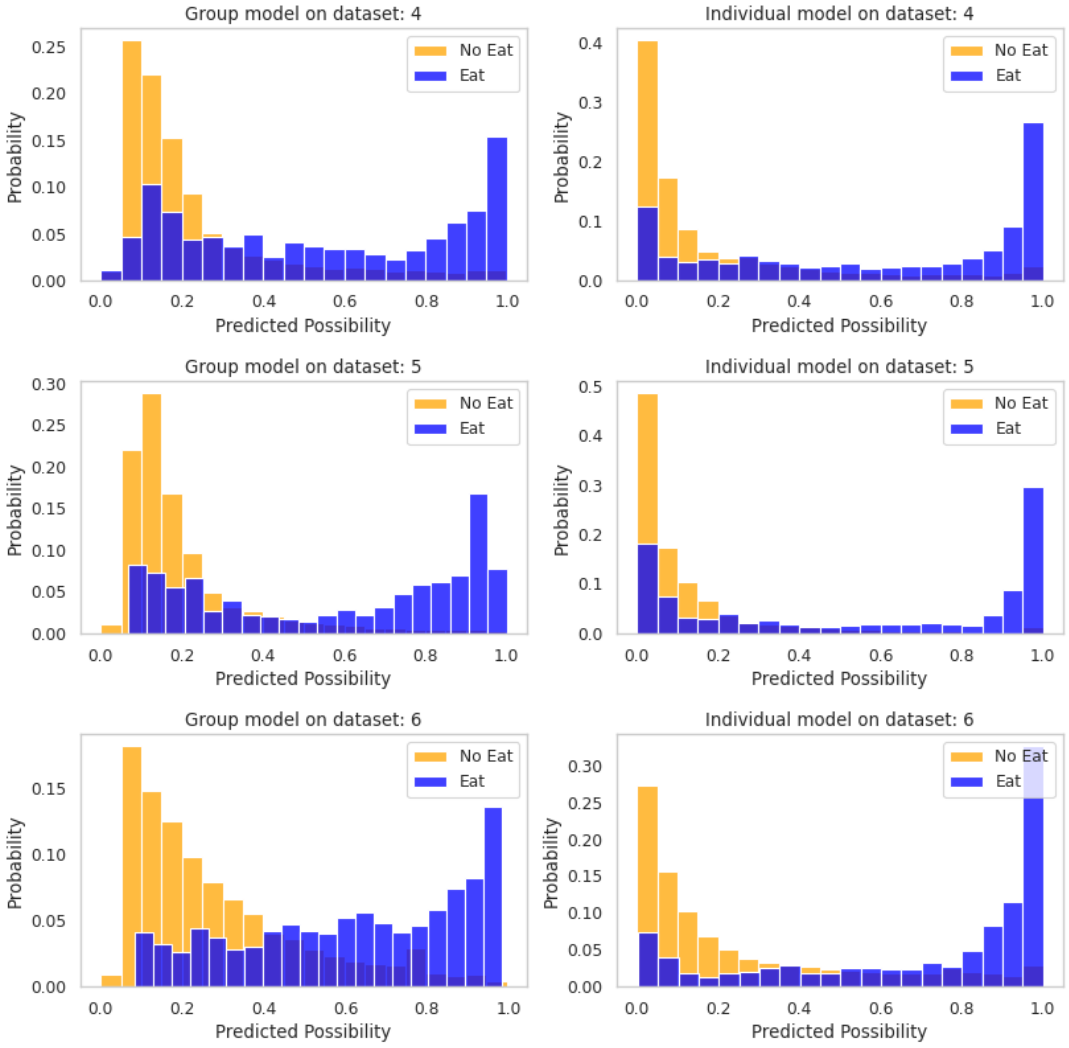

Figure 3.2: Figure of distributions of output probabilities on individual data sets 4, 5 and 6. In the left column, probabilities for the group model were calculated by training on the CAD dataset and testing on the individual. In the right column, probabilities for the individual models were calculated using 5-fold cross validation. Yellow histograms are distributions of models' output in ground truth non-eating periods. Blue histograms are distributions of models' output in ground truth eating periods.

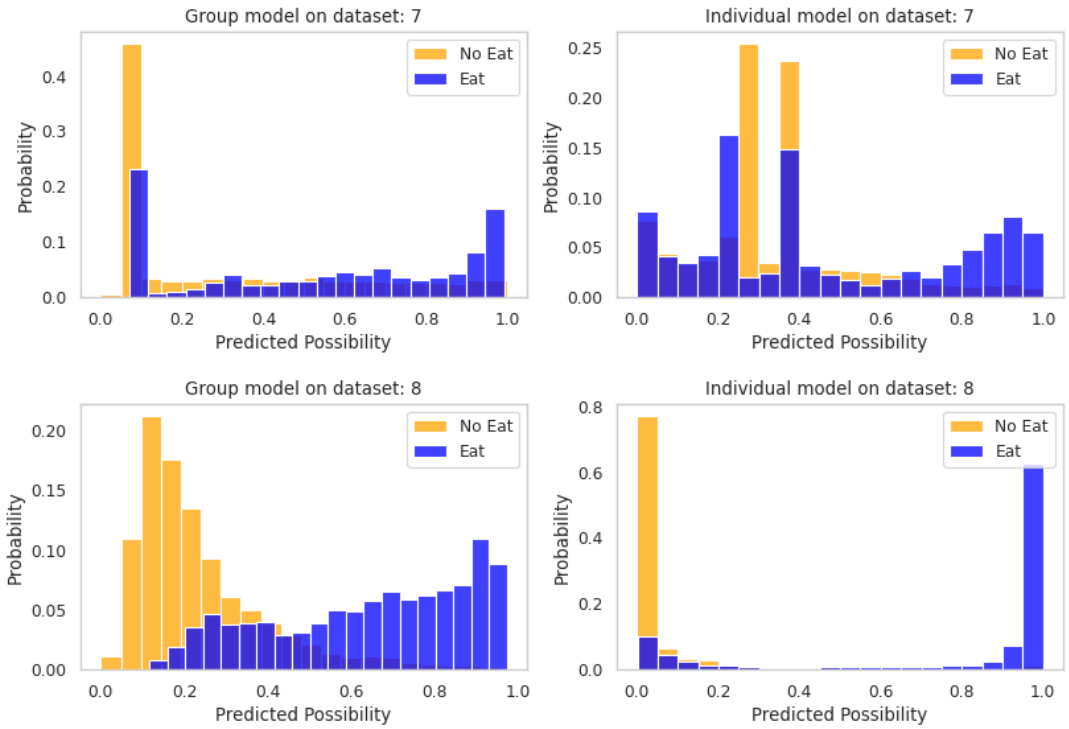

Figure 3.3: Figure of distributions of output probabilities on individual data sets 7 and 8. In the left column, probabilities for the group model were calculated by training on the CAD dataset and testing on the individual. In the right column, probabilities for the individual models were calculated using 5-fold cross validation. Yellow histograms are distributions of models' output in ground truth non-eating periods. Blue histograms are distributions of models' output in ground truth eating periods.

3.2 Model Comparison and Performance Analysis

This section discusses and compares the time metric and episode metrics performance of individual models and the group model.

3.2.1 Time Metric

In this work, weighted accuracy (WAcc) was chosen as the time metric, which is computed by equation 2.13. Each individual data set used its own balance ratio of non-eat/eat as the weight W in weighted accuracy. The ratio of non-eat/eat of each individual data set can be found in table 2.1. For each individual data set, 5-fold cross validation was used to train and test individual models for that data set and then we tested the group model on the same way as mentioned in section 2.6.3.

Table 3.1 shows the weighted accuracy on each individual data set. We calculate the average values of all WAcc values from the group models and from individual models respectively. We also calculate the improvement on individual models over the the group model on each individual data set by the formula:

$$WAcc\ improvement(\%) = \frac{WAcc_{individual} - WAcc_{group}}{WAcc_{group}} \times 100 \tag{3.1}$$

where $WAcc_{individual}$ is the weighted accuracy of individual models on one individual data set and $WAcc_{group}$ is the weighted accuracy of the group model on the same individual data set. Improvement is computed in percentage. Figure 3.4 visualizes the improvement on weighted accuracy of individual models.

In figure 3.4, it shows that compared with the group model, individual models on most data sets except the individual data set 7. Improvements on individual data sets 1, 3, 8 are more than 5%. It is are much more obvious than the improvements on data sets 4, 5 and 6. When looking back to the distributions of models' outputs in figures 3.1, 3.2 and 3.3, we find that in data sets 1, 3 and 8, the eating distributions and non-eating distributions from the group model overlap seriously. The group model shows less confidence in classifying the eating behaviors and non-eating behaviors, compared with the individual models. This explains why the group model performs so badly and the improvements from individual models on these three data sets are so distinct. This also implies that eating behaviors from those individuals are likely to be different from eating behaviors in CAD data

set. In addition, for individual data sets 2 and from 4 to 7, the WAcc performance of individual model are very close to WAcc performance of the group model and have slight or even no improvement. The group model separates the distributions of eating and non-eating on these data sets well and this is similar to the behaviors of individual models. This explains why the WAcc performance from the group model is close to WAcc performance from individual models on these data sets.

On average, individual models with WAcc of 81.9% is better than the group model with 78% WAcc. There is 5% improvement using individual models shown in figure 3.4. Most of individual models outperform the group model in weighted accuracy metric.

dataset	**WAcc**	
	Group Model	**Individual Model**
1	0.774	0.897
2	0.956	0.958
3	0.611	0.722
4	0.757	0.763
5	0.825	0.837
6	0.786	0.795
7	0.688	0.684
8	0.843	0.893
average	**0.780**	**0.819**

Table 3.1: Table of weighted accuracy performance of individual models and the group model. These results are measured from 5-fold cross validation as mentioned in 2.6.4

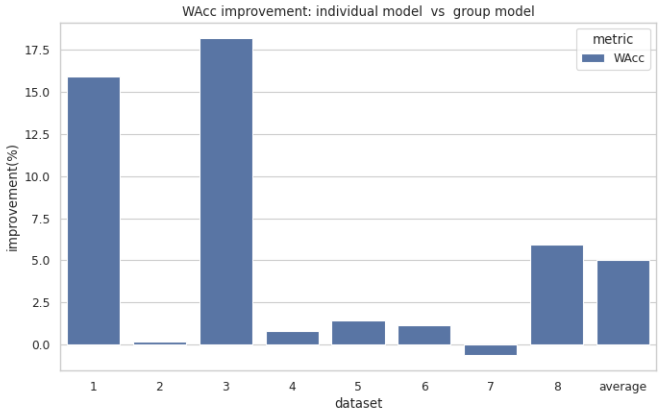

Figure 3.4: Improvement in percentage on weighted accuracy of individual models compared with the group model.

3.2.2 Episode Metric

We choose TPR and FP/TP as our episode metrics to measure the performance of models in detecting meal episodes. Table 3.2 shows the TPR and FP/TP performance of the group model and individual models when both of them using the group thresholds T_s=0.8 and T_e = 0.4 from previous work [27]. Figure 3.5 shows the improvements on TPR and FP/TP of individual models compared with the group model. The improvement on TPR and FP/TP of each individual data set are calculated using the following formulas:

$$\texttt{TPR improvement}(\%) = \frac{\texttt{TPR}_{individual} - \texttt{TPR}_{group}}{\texttt{TPR}_{group}} \times 100 \tag{3.2}$$

$$\texttt{FP/TP improvement}(\%) = -\frac{\texttt{FP/TP}_{individual} - \texttt{FP/TP}_{group}}{\texttt{FP/TP}_{group}} \times 100 \tag{3.3}$$

where $\texttt{TPR}_{individual}$ and $\texttt{FP/TP}_{individual}$ mean the TPR and FP/TP performance of individual models. $\texttt{TPR}_{group}$ and $\texttt{FP/TP}_{group}$ mean the TPR and FP/TP performance of the group model. Note that when FP/TP is smaller, model performs better. So we add a negative sign to equation 3.3 when calculating improvement of FP/TP.

Table 3.2 shows that in average, the TPR performance of individual models of 79.9% is better than the TPR performance of the group model of 73.6%, but the average FP/TP performance of individual models is 1.751, worse than average FP/TP performance of the group model 1.531. Figure 3.5 shows that the average improvement of TPR on individual models is around +10 % while the average improvement of FP/TP on individual models is around -10%. Most of individual models have better TPR and worse FP/TP than the group model.

The worse FP/TP results in individual models are caused by the setting that both individual models and the group model use the same group thresholds T_s and T_e. From figures 3.1, 3.2 and 3.3, we know that individual models of each individual data set can better separate the distribution of eating and the distribution of non-eating, leading to possibility output close to 1 in eating period and close to 0 in non-eating period. So it needs higher T_s value to reduce FP and lower T_e to detect the end of meal episode. However, the group model can not separate distribution of eating and distribution of non-eating well, leading to the group model's output values around or less than 0.8 in eating period and higher output values in non-eating period. Then it requires a lower T_s value

to detect more positive samples to increase TP and a higher T_e value to make the meal segment shorter. Therefore, T_s=0.8 is more suitable for the group model than the individual models. It is necessary to tune T_s and T_e thresholds for each individual data set and this is discussed in section 3.3.

dataset	**TPR**		**FP/TP**	
	GroupModel	**Individual-Models**	**GroupModel**	**Individual-Models**
1	0.594	0.969	0.368	0.613
2	0.923	0.923	0.375	0.417
3	0.333	0.667	2.8	1.2
4	0.763	0.763	0.897	1.759
5	0.75	0.6	1.4	2.25
6	0.885	0.923	1.739	2.083
7	0.773	0.636	3.882	4.786
8	0.864	0.909	0.789	0.9
average	**0.736**	**0.799**	**1.531**	**1.751**

Table 3.2: Episode metrics of individual models and the group model when both of them using the group thresholds: T_s=0.8, T_e=0.4. These results are also measured from 5-fold cross validation mentioned in 2.6.4.

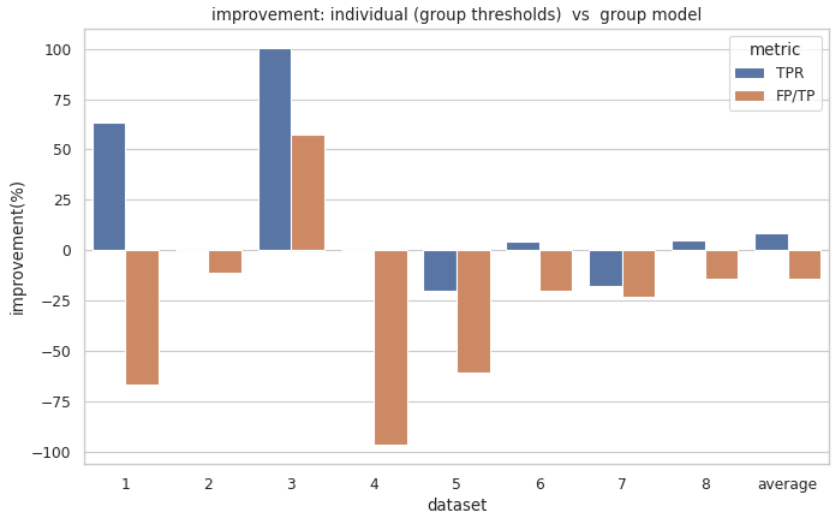

Figure 3.5: Figure of improvements of individual models on episode metrics compared with the group model when using T_s = 0.8 and T_e=0.4 for both individual models and the group model.

Based on results above, we find that most individual models have better weighted accuracy than the group model and the improvement of weighted accuracy from individual models can vary per individual as some of individual models can have improvement more than 10% and others have

improvements less than or around 5%. The improvement on episode metrics of individual models varies and it depends on individuals. There is a trade-off between TPR and FP/TP. This answers the first and the second questions in section 1.6.

3.3 Results of Tuning Hyper-Parameters

3.3.1 Effect of T_s and T_e on individual models

This section investigates the effect of T_s and T_e on the performance of TPR and FP/TP of individual models. We used the previously generated possibility sequence of each day during cross validation to investigate the effect of T_s by using a fixed the T_e value for all individual data sets and varying T_s to see the change of TPR and FP/TP. Similarly, we also fixed T_s for all individual models and then varied T_e to see the effect of T_e. Figures 3.6 and 3.7 show the changes of TPR and FP/TP performance when changing T_s and T_e. There are 8 different curves with different colors in each plot, representing the individual models for 8 participants respectively.

In figure 3.6, when T_e is fixed to be 0.1, there is a trend of descending in both TPR and FP/TP as T_s is increasing from 0.1 to 0.85. This is because when T_s is higher, output values greater than T_s become less and it decreases positive prediction. Hence TPR can decrease as well. When the decline rate of FP is faster than the decline rate of TP, FP/TP can decrease and get improved. Hence higher T_s can improve FP/TP and may decrease TPR. When T_s =0.85 and T_e is increasing, TPR and FP/TP of most individual models remain unchanged, but some of them change. For example, FP/TP for individual models of data set 7 increases when T_e increases in two figures. This implies that the effect of T_e on TPR and FP/TP depends on individual behaviors. T_e controls the end of eating segment. When T_e is higher, then it is easier to meet the possibility value smaller than T_e and make the segment end quicker, leading to shorter eating segment. Figure 3.7 resembles figure 3.6. The decrease of TPR and increase of FP/TP in individual data set 7 in figures 3.6 and 3.7 imply that for some individuals, shorter eating segments may be hard to overlap the ground truth eating episodes. This can decrease TP amount and lead to lower TPR and higher FP/TP.

Therefore, the results implies that higher T_s improves FP/TP but may also decrease TPR for some individuals. Suitable T_s for each individual is required. For some individuals, higher T_e may lead to higher FP/TP, but for others, T_e doesn't affect performance. In this case, lower T_e could be better.

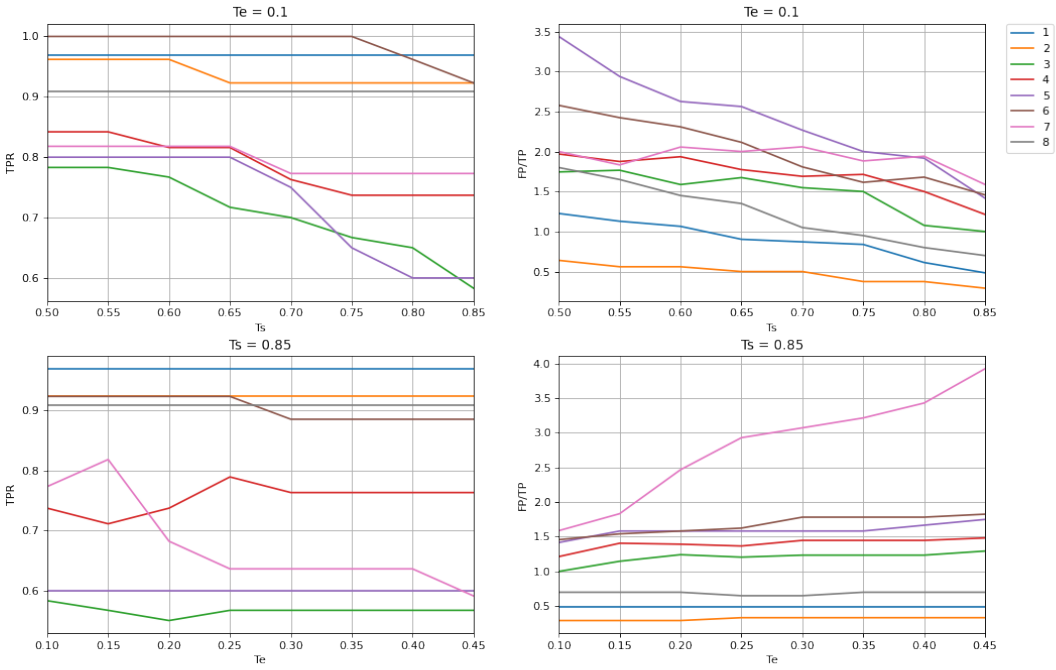

Figure 3.6: Effect of Ts and Te on TPR and FP/TP performance of individual models. The two figures on the top investigate the effect of T_s by fixing T_e=0.1. The two figures on the bottom investigate the effect of T_e by fixing T_s=0.85 and varying T_e

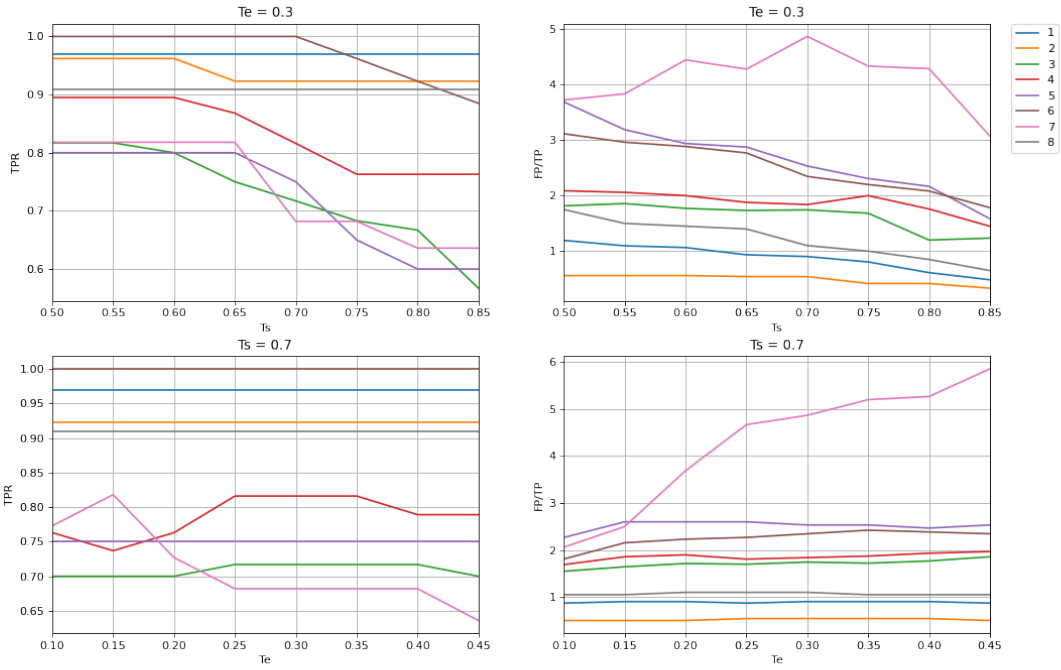

Figure 3.7: Effect of Ts and Te on TPR and FP/TP performance of individual models. The two figures on the top investigate the effect of T_s by fixing T_e=0.3. The two figures on the bottom investigate the effect of T_e by fixing T_s=0.7 and varying T_e

3.3.2 Improvement after Tuning T_s and T_e

To find thresholds T_s and T_e for each individual data set, we re-used the possibility sequences, generated by individual models, in testing folds from 5-fold cross validation as mentioned in 2.6.4, and defined a set of T_s : [0.85,0.8, 0.75, 0.7, 0.65,0.6, 0.5], a set of T_e : [0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45] and then used grid-search method to find TPR and FP/TP performance for all combinations of T_s and T_e. In those combinations, we tried three different strategies to look for the T_s, T_e that improve TPR and FP/TP. The improvements of TPR and FP/TP before and after applying these methods are shown in figure 3.8 and table 3.3.

method	**TPR:improvement(%)**	**FP/TP:improvement(%)**
Group Thresholds	8.6	-14.4
TPR Method	18.5	12.0
FP/TP Method	9.6	24.4
Balance Method	**10.1**	**33.2**

Table 3.3: Improvement of TPR and FP/TP of individual models compared with the group model after applying different methods to find T_s and T_e for individual models. Group Thresholds mean all individual models use T_s=0.8 and T_e =0.4 as same as the thresholds used in the group model. "TPR method", "FP/TP method" and "Balance method" means the T_s and T_e for models of each individual are searched by algorithms 1, 2 and equation 2.1 respectively

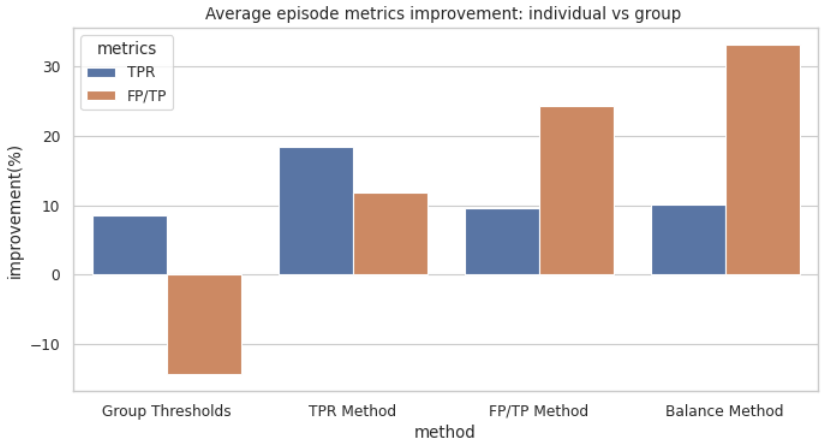

Figure 3.8: Average improvements of TPR and FP/TP on individual models before and after applying three different threshold tuning methods. This figure is the visualization of table 3.3

Table 3.3 summarizes the improvements before and after using three strategies for tuning T_s and T_e mentioned in section 2.7. Figure 3.8 visualizes the data in table 3.3 to give a better view for comparison. In table 3.3, "Group Thresholds" means both individual models and the group

model use the group thresholds: T_s =0.8 and T_e =0.4. "TPR Method" represents the algorithm 1 that gives higher priority to optimize TPR first and then optimize FP/TP. "FP/TP Method" is the algorithm 2 that gives higher priority to optimize FP/TP first and then optimzie TPR. "Balance Method" is the method that first calculates the TP ratio using equation 2.18 and then picks the T_s, T_e pair with the largest TP ratio. Since this method considers TP, FN and FP, it balances performance of TPR and FP/TP.

Figure 3.8 shows that three strategies provide improvements on both TPR and FP/TP. Balance method provides the best FP/TP improvement of 33.2% with TPR improvement of 10.1%, while the TPR methods provides the best TPR improvement of 18.5% and FP/TP improvement of 12%. Compared with TPR method, FP/TP method provides better FP/TP improvement of 24.4%.

We finally choose the balance method to select T_s and T_e for our individual models, when considering both TPR and FP/TP. The following tables and figures show the results from balance method. Table 3.4 shows the thresholds for each individual data set searched by the balance method. While most T_s values are equal to 0.85 and most T_e values are equal to 0.1, this method indicates that higher T_s and lower T_e are better to segment eating period on individual data sets. Table 3.5 shows the TPR and FP/TP performance of the group model and individual models with customized thresholds from table 3.4. Since figures of distributions in section 3.1 show that for individual models, the output possibilities are very high in eating episodes and very low in non-eating episodes, such high T_s and low T_e values from table 3.4 can better separate eating episodes from non-eating periods. Table 3.5 reveals that the average TPR of all individual models with customized thresholds is 81%, better than the average TPR of the group model, 73.6%. The average FP/TP of all individual models using customized thresholds is 1.022 and it is smaller than the average FP/TP of 1.531 from the group model.

dataset	**Ts**	**Te**
1	0.85	0.10
2	0.85	0.10
3	0.80	0.10
4	0.85	0.10
5	0.85	0.10
6	0.85	0.10
7	0.85	0.10
8	0.85	0.25

Table 3.4: Thresholds found by balance method

dataset	TPR		FP/TP	
	GroupModel	**Individual-Models**	**GroupModel**	**Individual-Models**
1	0.594	0.969	0.368	0.484
2	0.923	0.923	0.375	0.292
3	0.333	0.65	2.8	1.077
4	0.763	0.737	0.897	1.214
5	0.75	0.6	1.4	1.417
6	0.885	0.923	1.739	1.458
7	0.773	0.773	3.882	1.588
8	0.864	0.909	0.789	0.65
average	**0.736**	**0.81**	**1.531**	**1.022**

Table 3.5: Episode metrics of individual models using individual thresholds that maximize the ratio in equation 2.18 and episode metrics of the group model with thresholds T_s=0.8 and T_e=0.4

Figure 3.9 shows the improvements of individual models on TPR and FP/TP between using customized thresholds from balance method and using group threshold, the TPR and FP/TP performance between individual models using customized thresholds from balance method and the group model. The improvements of individual models on TPR and FP/TP after using customized thresholds from the balance method is calculated by formulas:

$$\texttt{TPR improvement}(\%) = \frac{\texttt{TPR}_{CTH} - \texttt{TPR}_{GTH}}{\texttt{TPR}_{GTH}} \times 100 \tag{3.4}$$

$$\texttt{FP/TP improvement}(\%) = -\frac{\texttt{FP/TP}_{CTH} - \texttt{FP/TP}_{GTH}}{\texttt{FP/TP}_{GTH}} \times 100 \tag{3.5}$$

where $\texttt{TPR}_{CTH}$ and $\texttt{FP/TP}_{CTH}$ are the TPR and FP/TP of individual models using customized thresholds (CTH). $\texttt{TPR}_{GTH}$ and $\texttt{FP/TP}_{GTH}$ are the TPR and FP/TP of individual models using the group model's thresholds (GTH), rather than customized thresholds. After applying the customized thresholds, figure 3.9 shows that the compared with using the group thresholds, improvement on FP/TP on each individual data set using balance method is around or more than 10% and the average improvement on FP/TP of all individual data sets is around 40%. TPR of individual data set 7 is improved by 20%, while the TPR values of individual models on data sets 3 and 4 drop slightly. Compared with the group model, the average improvement on TPR of all individual data sets is around 10 %. For completeness, the results of episode metrics of TPR method and FP/TP method are left in appendices A.1 and A.2.

From the results above, we learn that tuning the hyper-parameters, T_s and T_e, can improve

the average performance of TPR, FP/TP. Different tuning methods have different improvements on episode metrics. The improvement percentage of TPR and FP/TP can be various for each individual. Note that there could be a trade-off between improvement on TPR and improvement on FP/TP. This answers the third question in section 1.6.

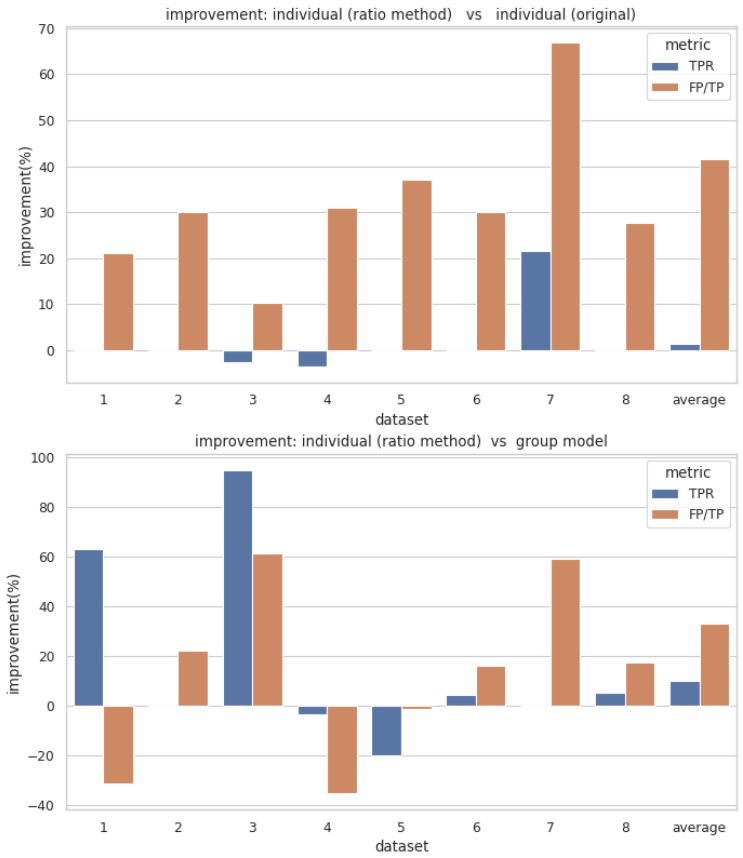

Figure 3.9: Figure of improvements on episode metric for individual models after tuning T_s,T_e using balance method with equation 2.1. The top image is the comparison of individual models before and after tuning T_s and T_e using balance method in equation 2.1. The bottom method is to compare individual models using balance method with the group model using T_s=0.8 and T_e=0.4

Chapter 4

Conclusion and Future Work

4.1 Conclusion

This thesis discusses the methods of training and evaluating the group model and individual models on the time metric, weighted accuracy, and episode metrics, TPR and FP/TP. We also discuss three methods of finding thresholds T_s and T_e for improving episode metrics of individual models. Based on the results, we can answers the questions in section 1.6:

1) Do individual models trained on small individual data sets perform better than the group model trained on large data from a group of people?

2) How much improvement do individual models get, compared with the group model? Do they vary depending on individual?

3) Does changing individual hyper-parameters help improve performance of individual models?

As for the first question, the individual models perform better than the group model when testing on the corresponding individual data set. In sections 3.2.1 and 3.2.2, the results show that most individual models perform better than the group model in weighted accuracy performance. Some of them have improvement more than 10% while the others have improvement less than or around 5%. As for the episode metrics, the average TPR performance of individual models using the group thresholds has about 10% improvement over the average performance of the group model. In figure 3.8, it also shows the episode metrics performance of individual models using different customized thresholds T_s and T_e can be better than the performance of the group model. So

individual models trained on small individual data set can outperform the group model.

The answer to the second question is that the improvement of individual models can vary per individual. For the weighted accuracy performance in figure 3.4, the improvement of weighted accuracy for some individual models can be more than 5%, but some of them, like data sets 2, 4, 5 and 6, show improvement less than 2%. In figure 3.5, when using the group thresholds, some individual models can have more than 50% improvement on TPR, but some of them have around -10% improvement on TPR. Hence such improvement can vary per individual. In addition, there is a trade-off between TPR and FP/TP. When individual models improve their TPR, they may have worse FP/TP performance than before. The main reason why some individual models got much better than others could be that we probably didn't have enough data for some of these individuals. Some people's eating behaviors may be consistent day to day and need 10 days of data or less to make individual models perform better, while others may need a lot more than 10 days to truly capture their individual behaviors. Then the amount of days of each individual data set could be a factor affecting performance of individual models. This factor can be investigated in the future work.

For the third question, the answer is that tuning hyper-parameters, T_s and T_e, do improve the performance of TPR and FP/TP. Figure 3.8 shows that using different strategies of tuning T_s and T_e for each individual data set can lead to different improvement on TPR and FP/TP. When using the balance method, it provides the best average improvement of above 30% on FP/TP. The TPR method from algorithm 1 provides the best TPR improvement of more than 20%. Moreover, the balance method of tuning T_s and T_e indicates that T_s =0.85 and T_e=0.1 provide better performance for most individual models. The reason why the tuned hyper-parameters T_s =0.85 and T_e=0.1 can be more selective is because the individual models better separated the distribution of eating from non-eating as seen in figures 3.1, 3.2 and 3.3. When individual models better separate the distribution of eating from non-eating, possibility of eating on eating episodes becomes much higher and possibility of eating on non-eating episodes becomes much lower. Hence higher T_s is better to detect TP and reduce FP and lower T_e is better to detect the start of non-eating period, which has very low eating possibility, and end the detected eating segment.

4.2 Future Work

This work investigates the performance of individual models for each individual data set and compare them with the group model to see how competitive the individual models are to the group model when testing on the individual data sets. This work also proposes three different strategies for looking for thresholds T_s and T_e to improve performance of individual models. There are still some limitations in this work and some interesting topics to explore in the future.

There are several limitations in this work. The collection of individual data sets is one limitation. Since there are only 8 individual data sets used to investigate the performance of individual models, it may be no enough. More individual data sets, such as 10 or even 20 participants, may help provide more convincing and robust results on time metrics and episode metrics. The amount of days of data per individual is also a factor affecting performance of individual models. More days of data for each individual can help capture the real distribution of individual eating behaviors. Hence we can also collect more days of data for each individual in the future. Additionally, it is better to make the amount of days of each individual data set same so that we can exclude the effect of the amount of days on the performance. In section 2.7, hyper-parameters tuning, we could also investigate the window size effect on individual models to see if window size can affect performance of individual models in the future. However, since doing such experiments to test window size effect on individual models is a very time-consuming process, this thesis doesn't investigate this, but leaves it to the future work.

Note that though training individual models can improve performance for detecting eating, collecting individual data for a new user requires the user to spend time in recording eating episode for each meal. This may bring some burdens to user. Then we need to consider if the improvement from individual models is worthy or not, when considering the cost in data collection and labelling. In fact, there is a trade-off between improvement and burden from data collection. If we emphasize the improvement from individual models, then it is worth collecting more data from that individual, despite that it leads to higher cost and more time for data collection and labelling. If we care more about the cost of data collection and labelling than the performance, then the improvement would be less worthy. We can simply collect fewer days to train the models and reduce the burdens. Hence such trade-off and worthiness of the improvement from individual models can depend on the needs from users. This is also a common problem in deep learning, which requires a large amount of data

to get better performance.

There are other interesting topics that can be extended from this work. For example, we can test how many days do each individual model require to be trained on in order to have better performance than the group model. By doing research about this, it can give us an insight about the approximate minimum amount of days required for one individual in order to train a good model, so that we can minimize the cost of data collection. We can also apply transfer learning in this research in the future to borrow the knowledge learned from the group model to each individual model so that each individual model can require less individual data to get better performance in eating detection.

Additionally, we can investigate subgroups of people. Could we look for subgroups of people and train the group model for each subgroup? We call the subgroups as phenotypes. In this case, we can consider person as a variable and cluster people into different phenotypes based on the similarity of their eating, non-eating behaviors and then investigate performance of models trained on these subgroups. For example, we can split people into different groups based on utensils they use, like fork, chopstick, etc, and train a group model on each subgroup. This allows us to investigate the differences between groups that affect eating behaviors.

The last topic for future work is about model architecture. As the group model has more data, could the group model use more parameters or use other model architectures to improve performance? It is reasonable that the group model trained on a very large data set may need more parameters to increase model capacity and learn more patterns. This could be also a good topic for the future work. However, in this work, we want to replicate the group model from the previous work [27] and compare that group model with individual models from this work. Moreover, the group model architecture from previous work was selected based upon maximizing accuracy on CAD data set already. So we don't change the amount of parameters in the group model. In addition, CNN model architecture, compared with other model architectures like recurrent neural network (RNN), LSTM, can save memory and have faster training speed due to the property of sharing parameters in convolution layer. Hence we utilized the same CNN model without changing model architecture or the amount of parameters in this work.

Appendices

Appendix A Results of two other T_s, T_e tuning methods

All T_s and T_e tuning methods were performed on the possibility sequence output from the testing step in 5-fold cross validation as mentioned in section 2.6.4.

A.1 Method focusing on TPR improvement

dataset	Ts	Te
1	0.85	0.10
2	0.85	0.10
3	0.50	0.40
4	0.65	0.10
5	0.65	0.10
6	0.85	0.10
7	0.55	0.10
8	0.85	0.25

Table 1: Table of thresholds T_s and T_e searched by algorithm 1 with TPR threshold T_0 =0.8

dataset	TPR		FP/TP	
	GroupModel	Individual-Models	GroupModel	Individual-Models
1	0.594	0.969	0.368	0.484
2	0.923	0.923	0.375	0.292
3	0.333	0.817	2.8	1.735
4	0.763	0.816	0.897	1.774
5	0.75	0.8	1.4	2.562
6	0.885	0.923	1.739	1.458
7	0.773	0.818	3.882	1.833
8	0.864	0.909	0.789	0.65
average	**0.736**	**0.872**	**1.531**	**1.348**

Table 2: Episode metrics of individual models using individual thresholds using algorithm 1, TPR method, that optimizes TPR and group model with thresholds T_s=0.8 and T_e=0.4

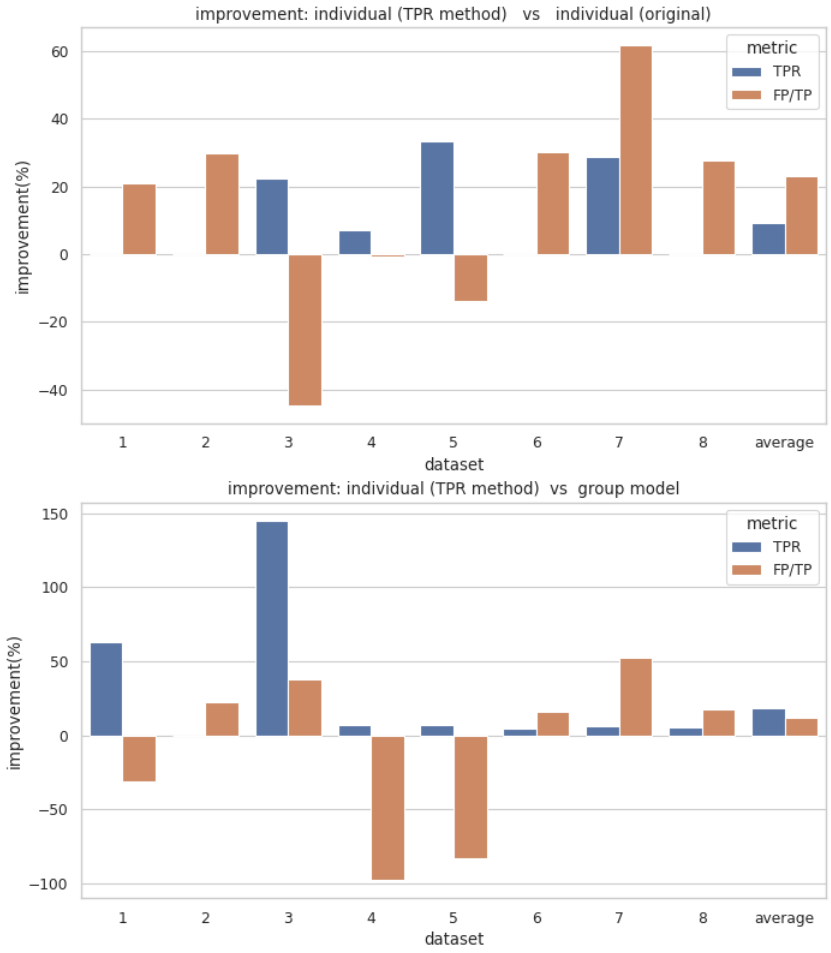

Figure 1: Improvements of episode metrics of individual models with thresholds searched from algorithm 1, TPR method with T_0=0.8. The top figure shows the improvements of individual models before and after applying TPR method. The bottom figure shows the improvements of individual models using thresholds from TPR method compared with the group model

A.2 Method focusing on FP/TP improvement

dataset	Ts	Te
1	0.55	0.4
2	0.50	0.1
3	0.85	0.1
4	0.85	0.1
5	0.85	0.1
6	0.85	0.1
7	0.85	0.1
8	0.75	0.1

Table 3: Table of thresholds T_s and T_e searched by algorithm 2 that focuses on optimizing FP/TP with FP/TP threshold T_1 =1

dataset	TPR		FP/TP	
	GroupModel	Individual-Models	GroupModel	Individual-Models
1	0.594	0.969	0.368	1.0
2	0.923	0.962	0.375	0.64
3	0.333	0.583	2.8	1.0
4	0.763	0.737	0.897	1.214
5	0.75	0.6	1.4	1.417
6	0.885	0.923	1.739	1.458
7	0.773	0.773	3.882	1.588
8	0.864	0.909	0.789	0.95
average	**0.736**	**0.807**	**1.531**	**1.158**

Table 4: Episode metrics of individual models using individual thresholds searched from algorithm 2, FP/TP method with $T_1 = 1$, that optimizes FP/TP and group model with thresholds T_s=0.8 and T_e=0.4

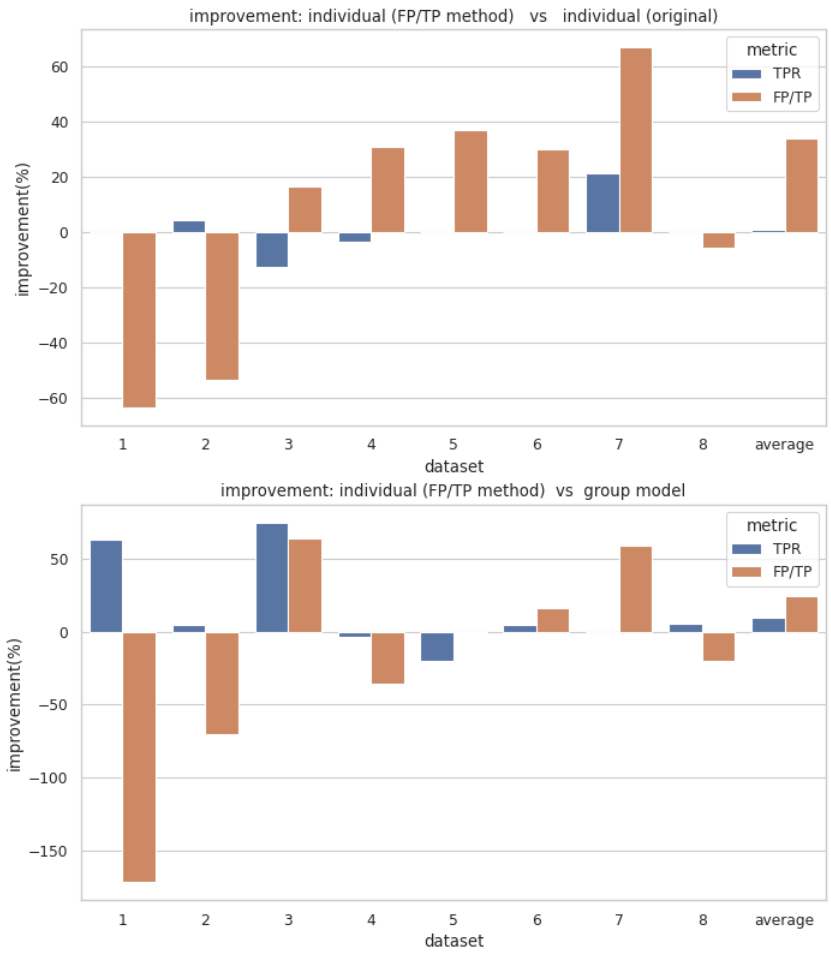

Figure 2: Figure of improvements of episode metrics of individual models with thresholds searched from algorithm 2, FP/TP method with T_1=1. The top figure shows the improvements of individual models before and after applying FP/TP method. The bottom figure shows the improvements of individual models using thresholds from FP/TP method compared with the group model

Bibliography

[1] D. Alimchandani. mhealth: Current opportunities and challenges. http://www.rxcomms.com/health-economics-news/mhealth-current-opportunities-challenges/deepa-alimchandani/, 2021.

[2] A. Bhandari. Everything you should know about confusion matrix for machine learning. https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/, 2020.

[3] A. Bhandari. Feature scaling for machine learning: Understanding the difference between normalization vs. standardization. https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/, 2020.

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

[5] Freedman ND et al Campbell PT, Newton CC. Excess body weight and the risk of primary liver cancer: an updated meta-analysis of prospective studies. *Cancer Research*, 76(20):6076–6083, 2016.

[6] C. D. Fryar C.M. Hales, M. D. Carroll and C. L. Ogden. Prevalence of obesity among adults and youth: United states 2015-2016. *Computer Networks*, 2017.

[7] Stanford University CS231n. Convolutional neural networks. https://cs231n.github.io/convolutional-networks/conv, 2020.

[8] Pegington M. Mattson M. P. Frystyk J. Dillon B. Evans G. Cuzick J. Jebb S. A. Martin B. Cutler R. G. Son T. G. Maudsley S. Carlson O. D. Egan J. M. Flyvbjerg A. Howell A Harvie, M. N. The effects of intermittent or continuous energy restriction on weight loss and metabolic disease risk markers: a randomized trial in young overweight women. *International journal of obesity*, 35(5):714–727, 2011.

[9] Y. Ho and S. Wookey. The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling. *IEEE Access*, 8:4806–4813, 2020.

[10] J. L. Scisco J. N. Salley and E. R. Muth. A comparison of user preferences and reported compliance with the bite counter and the 24-hour dietary recall. *in proc of Human Factors and Ergonomics Society Annual meeting*, 2012.

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *the 3rd International Conference for Learning Representations*.

[12] B. Lin. Machine learning and pedometers: An integration-based convolutional neural network for step counting and detection. Master's thesis, Clemson University, December 2020.

[13] Y. Luktuke and A. Hoover. Segmentation and recognition of eating gestures from wrist motion using deep learning. *IEEE Int'l Conf on Big Data*, 2020.

[14] Giovanni Mariani, Florian Scheidegger, Roxana Istrate, Costas Bekas, and A. Cristiano I. Malossi. Bagan: Data augmentation with balancing gan. 03 2018.

[15] MedicalNewsToday. How to naturally lose weight fast. https://www.medicalnewstoday.com/articles/322345takeaway, February, 2020.

[16] L. O. Hall W. P. Kegelmeyer N. V. Chawla, K. W. Bowyer. Smote: Synthetic minority oversampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[17] National Cancer Institute. Obesity and cancer. https://www.cancer.gov/about-cancer/causes-prevention/risk/obesity/obesity-fact-sheetwhat-is-known-about-the-relationship-between-obesity-and-cancer-, 2017.

[18] World Health Organization. Obesity and overweight. https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight, 2020.

[19] Abril E. P. Tracking myself: Assessing the contribution of mobile technologies for self-trackers of weight, diet, or exercise. *Journal of health communication*, 21(6):638–646, May 2016.

[20] M. Y. Park and T. Hastie. L1-regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):659677, 03 2007.

[21] Realtime Technologies Ltd. Shimmer user manual. https://bmslab.utwente.nl/wp-content/uploads/2019/12/Shimmer-User-manual.pdf, 2017.

[22] R. Rojas. *Neural Network: A Systematic Introduction*. 1996.

[23] O. Abdeljaber T. Ince M. Gabbouj D. J. Inman S. Kiranyaz, O. Avci. 1d convolutional neural networks and applications – a survey. 151:107398, 2021.

[24] A. Reiss M. Hanselmann R. Stiefelhagen S. M¨unzner, P. Schmidt and R. D urichen. "cnn-based sensor fusion techniques for multimodal human activity recognition". *in the Proceedings of the 2017 ACM International Symposium on Wearable Computers*, 29(9):158–165, 2017.

[25] United Kingdom National Health Service. Obesity - causes- nhs. https://www.nhs.uk/conditions/obesity/causes/, May 2019.

[26] S. Sharma and A. Hoover. "the challenge of metrics in automated dietary monitoring as analysis transitions from small data to big data". *IEEE Int'l Conf on Bioinformatics and Biomedicine*, pages 2647–2653, 2020.

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

[28] L. Tagliaferri. An introduction to machine learning. https://www.digitalocean.com/community/tutorials/an-introduction-to-machine-learning, September, 2017.

[29] Hoos H.H van Engelen, J.E. A survey on semi-supervised learning. *Mach Learn*, 109:373–440, 2020.

[30] J. Watson. Mems gyroscope provides precision inertial sensing in harsh, high temperature environments.
https://www.analog.com/en/technical-articles/mems-gyroscope-provides-precision-inertial-sensing.html, 2020.

[31] Wikipedia. mhealth.
https://en.wikipedia.org/wiki/MHealth, 2017.

[32] J. Wang Z. Yan Y. Chen, X. Wang and J. Luo. Excess body weight and the risk of primary liver cancer: an updated meta-analysis of prospective studies. *European Journal of Cancer*, 48(14):2137–2145, 2012.

[33] J. Scisco Y. Dong, A. Hoover and E. Muth. Detecting eating using a wrist mounted device during normal daily activities. *in Proceedings of the International Conference on Embedded Systems, Cyber-physical Systems, and Applications*, 18(4), 2011.

[34] M. Wilson E. Muth Y. Dong, J. Scisco and A. Hoover. Detecting periods of eating during free-living by tracking wrist motion. *IEEE journal of biomedical and health informatics*, 18(4):1253–1260, 2014.

[35] E. Muth Y. Shen and A. Hoover. Recognizing eating gestures using context dependent hidden markov models. pages 248–253. IEEE Press, 2016.

[36] M. Sun X. Zhu L. Zhao Y. Wang, H. Xue and Y. Yang. Prevention and control of obesity in china. *Computer Networks*, 9, 2019.

[37] X. Zhu and A. B. Goldberg. Introduction to semi-supervised learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1):1–130.