Detecting eating episodes from daily patterns of wrist motion using recurrent neural networks

---

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
Adam Patyk
August 2021

Accepted by:
Dr. Adam Hoover, Committee Chair
Dr. Harlan Russell
Dr. Yingjie Lao

Abstract

This thesis considers the problem of detecting when a person is eating during everyday life by examining daily patterns of wrist motion with recurrent neural networks. Our novelty is analyzing an entire day of data to classify and segment meals with a model we call the "daily pattern classifier". Previous research has only analyzed short windows on the order of seconds or minutes long that miss larger day-to-day patterns. The goal of this work is to utilize daily contextual indicators to improve eating episode detection and reduce false detections that occur throughout the day.

The wrist motion data used in this work was from the Clemson All-Day (CAD) dataset collected in previous work [32]. This dataset consists of 354 day-length recordings of wrist motion data from 351 participants for a total of 1,063 meals and 4,680 hours. Previous work used a sliding window approach and a convolutional neural network classifier to process this data and generate a continuous probability of eating, or $P(E)$, from a day-length recording [30]. We call this model the "windowed eating classifier". The day-level classifier proposed in this work operates on the $P(E)$ sequences, also called "daily samples". In order to train and evaluate the daily pattern classifier, we required a larger set of daily samples than the number of recordings in the dataset. For data augmentation, the windowed eating classifier was used repeatedly to generate a sizable set of daily samples for this purpose. Genuine noise from the volatility of the model differentiated samples from the same actual recording. To reduce the necessary model complexity and generation time, the daily samples were downsampled before further processing. After downsampling, the daily samples and the corresponding ground truth eating events were saved together to files for subsequent training and evaluation with the daily pattern classifier.

The daily pattern classifier proposed in this work utilizes a recurrent neural network (RNN) architecture. This was advantageous due to the memory attributes, masking abilities, and time series applications of RNNs. Training this model required all inputs to be the same length. Since

the daily samples varied in duration, they were all padded to the same size. These padded values were later masked out in the model, so they did not affect training. The daily pattern classifier was trained and evaluated using 5-fold cross validation. Before metrics were measured, a single-value thresholding algorithm was used for post-processing the output of the daily pattern classifier. Lastly, both time and episode evaluation metrics were measured to determine how well the classifier categorized individual timesteps as well as entire eating episodes respectively.

In our results, the daily pattern classifier significantly filtered background noise, which improved the separation between strong meal peaks and other noise in the $P(E)$ signal. Our approach achieved an eating episode true positive rate (TPR) of 85% with 0.8 false positives per true positive (FP/TP). This was a 4% decrease in episode TPR, but a 53% improvement in FP/TP when compared to the windowed eating classifier in previous work [30]. The time weighted accuracy of our approach was 85%, which is 5% higher than the windowed eating classifier indicating better overall datum-by-datum classification of eating and non-eating. In conclusion, we found evidence that a recurrent neural network can learn day-level contextual patterns and utilize them for better eating episode detection.

Dedication

This thesis is dedicated to my loving parents, family, and friends who have supported me unconditionally throughout my life and academic journey. I would like to specifically dedicate this work to my grandparents, Sue and Roy Munao and Eleanor and Sylvester Patyk, for showing me that anything is possible with hard work, dedication, and a positive attitude.

Acknowledgments

First, I would like to thank my research advisor and committee chair, Dr. Adam Hoover. His guidance, expertise, and patience over the past year has helped me grow as a student, professional, and as a person. I could not have made it this far without his help.

I would like to thank Dr. Harlan Russell for his encouragement to pursue graduate education. And, I would like to thank both Dr. Harlan Russell and Dr. Yingjie Lao for taking the time to serve on my defense committee. Furthermore, I am incredibly appreciative of all my professors at Clemson University and especially the Holcombe Department of Electrical and Computer Engineering for this wonderful opportunity.

Lastly, I would like to extend a huge thanks to my friends and members of our research group for their help and encouragement.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

This thesis considers the problem of detecting when a person is eating during everyday life by tracking their wrist motion using sensors found in a typical smartwatch. The motivation for this work is to help people track their energy intake, which is critical for weight loss and treating obesity. The following paragraphs outline the background for this work, which is covered in more detail in the rest of this chapter.

Obesity is an extremely challenging and widespread global health problem. Obesity can develop when energy intake (food consumption) is consistently higher than energy expenditure. Energy expenditure overall is much lower in the modern era since sedentary lifestyles are common and transportation is largely motorized. Thus, tracking energy intake is a valuable tool in fighting overweight and obesity. Traditionally, this is done with self-monitoring tools like food diaries, but these methods are tedious and subject to self-reporting bias and inaccuracy. A system that could automatically monitor energy intake would be more advantageous and effective.

Our group has been investigating methods involving wrist motion tracking to detect when a person eats [7, 8, 9, 21, 30, 37]. Previous work used a Bayesian classifier to segment episodes of eating from data collected with wrist-worn smartphones [9]. Later work used a convolutional neural network and a sliding window approach that improved eating detection accuracy by almost 10% [30]. A limitation of both these approaches is that they examined small windows of time (on the order of minutes) to determine if eating was occurring or not within that window.

Figure 1.1: Comparison of daily context (24 hours) and window context (6 minutes) to scale.

This thesis considers the possibility that additional context exists in a longer window, specifically an entire day, that can improve recognition accuracy. Figure 1.1 demonstrates our motivating principle. People tend to eat 3-4 meals a day at regularly spaced intervals. People also tend to eat right after a long period of rest (sleep). These patterns, as well as others in daily activities, could potentially be learned by a new classifier.

The remainder of this introductory chapter includes the background and context for this thesis. Section 1.2.1 explores the details and causes of the obesity epidemic facing much of the world population. Sections 1.2.2 and 1.2.3 discuss the field of mHealth and how it can impact personal health along with the concept of automated dietary monitoring (ADM). Section 1.2.4 briefly describes the wrist-worn motion tracking devices used to collect the original eating data. Section 1.2.5 includes background information in recurrent neural networks and deep learning, which are used to analyze the day-to-day patterns of eating behavior in this work. Next, section 1.3 looks at other relevant work in the field of eating detection and energy intake monitoring. Penultimately, section 1.4 encapsulates the motivation for this work and describes the use of daily contextual indicators for segmenting eating episodes. And lastly section 1.5 outlines the novelty of this thesis and the questions this work strives to answer.

1.2 Background

1.2.1 Obesity

Obesity is not just a challenging health problem; it is an epidemic. As of 2016, over 1.2 billion adults (18 and older) worldwide are overweight and over 650 million are obese [38]. In the United States, 42.4% of people over the age of 20 are obese with a body mass index (BMI) [kg/m^2] exceeding 30 according to a 2017-2018 report from the Centers for Disease Control and Prevention (CDC) [15]. It is estimated that by 2030, most of the American population will qualify as overweight

and nearly half the population will be obese [36]. Although many researchers debate the validity of BMI for measuring body fat percentage and obesity [12, 25, 28], the relative increase in these measures is alarming.

Worldwide obesity has tripled since 1975 and childhood obesity (ages 5-19) has increased ten-fold in that same time [26, 38]. This dramatic growth is attributed to poor diets high in fat and sugar, increased inactivity, and widespread sedentary lifestyles. Obesity is associated with increased risk of cardiovascular diseases (heart disease and stroke), diabetes, and some cancers, among other diseases and health conditions. More specifically, there is sufficient evidence of increased risk of colon, kidney, liver, and pancreatic cancer with obesity [20]. The recent COVID-19 pandemic has increased sedentary time, reduced physical activity, and increased unhealthy eating habits further amplifying the conditions that lead to unhealthy weight gain and obesity [23].

Nevertheless, obesity is almost entirely preventable [38]. It comes down to balancing energy intake and energy expenditure, both commonly measured in the unit of calories. If energy intake is greater than energy expenditure, the human body will store energy in the form of fat as a physiological response anticipating future food scarcity. If an imbalance exists, energy intake and energy expenditure could become balanced through greater energy expenditure in the form of physical activity or exercise. However, this is not as efficient or effective for weight loss as reducing energy intake [34]. Fundamentally, the easiest way to maintain a healthy energy balance is to reduce excessive energy intake and overconsumption of food. Although this may sound simple, drastic lifestyle changes are required if overconsumption has been the status quo for some time.

1.2.2 mHealth

Worldwide health and life expectancy have greatly improved in the last century. As a result, the most common causes of death have transitioned to chronic conditions and noncommunicable diseases (NCDs), or diseases that are not directly transmissible between people. According to a 2020 report by the World Health Organization (WHO), over 70% of deaths worldwide are caused by NCDs including cardiovascular diseases, cancer, and diabetes [39]. Most, although not all, of these NCDs are associated with certain lifestyle risk factors like unhealthy diet, overconsumption, and a deficiency of physical activity. This means that many NCDs are entirely preventable causes of death.

Figure 1.2: Apple Watch Series 6 smartwatch and fitness tracker [3].

Good personal health is paramount in reducing the risk of these diseases. Consequently, healthcare overall will need to shift from a paradigm of reactive care to one of proactive care. The rise of personal electronic devices like smartphones and wearables like fitness trackers and smartwatches has ushered in a new age of possibility for health tracking to assist with this shift. Using mobile devices and especially wearable devices to monitor personal health is known as mobile health or mHealth. And although electronic devices have contributed to a more sedentary lifestyle in some ways, the field of mHealth also offers ways for them to improve personal health in the battle against overweight and obesity.

The Apple Watch and Fitbit are among the most popular fitness trackers from the past 8 years [35]. These devices make tracking personal health and weight management approachable and straightforward. For example, the latest Apple Watch (shown in figure 1.2) is equipped with several biological instruments including a heart rate monitor, FDA-approved electro cardiogram (ECG), and blood oxygen sensor [3]. The device uses these various sensors to track caloric energy expenditure during workouts, resting heart rate over time, irregular heartbeat, and much more. This data can even be securely shared with an individual's doctor or physician if the user chooses to do so with the latest software [4]. This is an example of the shift to proactive healthcare that is needed to detect conditions before they become life threatening or even before they develop. Nonetheless, for weight loss, the smartwatches and fitness trackers on the market today only monitor energy expenditure.

1.2.3 Automatic Dietary Monitoring

There are several ways to monitor energy intake. Perhaps the easiest is through self-monitoring, where an individual records what foods they eat throughout the day and when they consume them. This may take the form of paper or electronic food diaries or even calorie logging smartphone apps. A literature review of 15 studies focused on dietary self-monitoring found a significant relationship between self-monitoring and weight loss [5]. A mixture of paper and electronic food diaries were used in the studies. It was noted that there was a decrease in the degree of self-monitoring completeness as the studies progress. Additionally, the individuals who maintained higher self-monitoring compliance throughout the studies experienced greater weight loss. This research demonstrates that the barrier to effective weight loss may be associated with the difficulty and tedium associated with maintaining dietary self-monitoring for a long period of time.

Furthermore, self-monitoring approaches are subject to human error and bias as they rely on an individual's ability to accurately report, or worse, recount, what they ate and how much they ate. According to a study, both common people and dietitians alike underreport the amount of calories they consume in a day [6]. The goal of the field of automated dietary monitoring (ADM) is to log energy intake automatically, so it can be accurately tracked [2]. Based on previous studies [5] this could aid individuals in weight loss and decrease the prevalence of obesity. Beyond supporting weight loss, ADM could also assist elderly individuals, especially those with Alzheimer's disease or dementia, maintain an accurate record of their meals.

Numerous devices already exist to accurately track energy expenditure, but a wearable device that can accurately track energy intake does not exist yet. We believe this is an important part of the puzzle. The goal of our research group is to develop a system that can be used to monitor food intake. Previous work has shown that it is possible to track ingestion events (bites) with wrist-worn devices [7]. However, these systems are quite sensitive to false detection events that resemble eating like brushing teeth or self grooming [30]. The goal of this work and others (see section 1.3) is to build a way to automatically recognize eating so the other, more sensitive bite counting systems can only be triggered when needed. This could be used to accurately predict the amount of food consumed by an individual in their everyday life and help with weight loss, eating disorders, and even eating speed [7].

Figure 1.3: Shimmer3 wearable IMU sensor device.

1.2.4 Shimmer3 Device

The wrist motion data for the CAD dataset was recorded on Shimmer3 devices [32]. This device, shown in figure 1.3, is a wearable piece of hardware that contains an inertial measurement unit (IMU) to measure and track motion. The IMU in each device was equipped with an accelerometer, gyroscope, and magnetometer (not used), to record a total of 6 axes of motion - linear acceleration and rotational acceleration with respect to the x, y, and z axes.

The large, circular, orange button on the front of the device was used as a way for the participant to record when they started and stopped eating. The LED lights on the front indicate the status of the device as well as different operational modes. The LEDs flashed in a regular pattern to notify the individual it was actively recording data. Data was recorded to an onboard microSD card. When data collection was completed for a day, the device was connected to a dock via the port on the bottom of the device (not shown). Data was imported from the device using the Shimmer Consensys software installed on a computer and then exported to a CSV file for subsequent processing. The entire procedure of processing the data for the CAD dataset is included in [30].

1.2.5 Neural Networks

Neural networks fall into the broad field of artificial intelligence (AI) that has exploded in the past decade. At a high level, an artificial neural network is a collection of neurons or nodes that each have an input, output, and perform a node function. There are additive biases for each

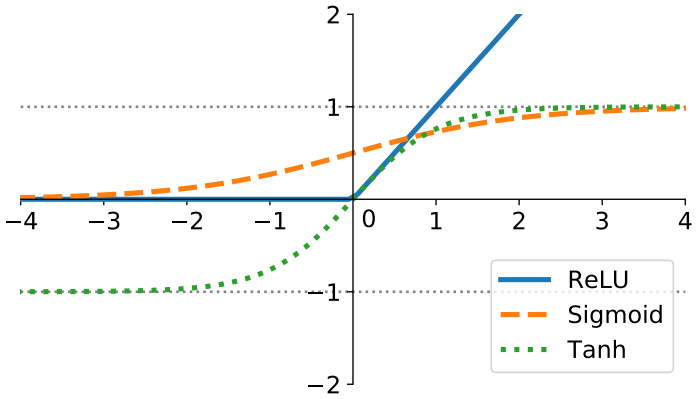


Figure 1.4: Rectified linear unit (ReLU), sigmoid, and tanh activation functions.

node and weighted connections between the nodes known as weights. A neural network is defined by these parameters (weights and biases) and the connections between nodes. A collection of nodes with the same input and output that are performing the same function are known as a layer. The node function mentioned previously is more commonly known as an activation function. Activation functions control the output of a node. There are many common activation functions used in neural networks, but we will only introduce the rectified linear unit (ReLU), tanh, and sigmoid functions since they are used in this work. The ReLU activation function serves to clip any negative values. The ReLU functions is defined as $\text{ReLU}(x) = \max(0, x)$, where all values less than 0 are clipped to 0. The tanh activation function is simply the hyperbolic tangent function, i.e. the hyperbolic analogue of the circular tangent function using in trigonometry, $\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$. The sigmoid activation function produces an output in the range from 0 to 1. It is defined as $\sigma(x) = \dfrac{1}{1 + e^{-x}}$. Plots of all three activation functions – ReLU, tanh, and sigmoid – are shown in figure 1.4

In order for a neural network to "learn" the best parameters for a specific problem, it undergoes training. For training, the network is initialized in some manner and then the input is provided. The network uses the weights and biases of the nodes to compute an output and this output is scored for accuracy. The accuracy of the output is used to determine how much the weights and biases should be adjusted. A complete cycle of training with all of the training data is called an epoch.

There are three types of training: supervised, unsupervised, and reinforcement learning. In supervised training, the data processed by the network is also labeled with a correct set of outputs known as the ground truth or GT. The network is then able to process the input to generate an output

and directly compare it to the desired results. Supervised learning is typically used for classification or regression tasks. When using supervised training with a neural network, it is important that the network is not trained and tested on the same data. If this were to occur, the model would likely achieve high accuracy but not transfer well to other data as it could just "memorize" the testing data. Unsupervised training is performed with data that has no labeled ground truth training data. This technique is used when ground truth data may not be available or the goal is to reveal latent features that are not apparent from the raw data. Lastly, reinforcement learning does not have labeled training data, but instead associates rewards or penalties with different actions. For example, reinforcement learning is used when training neural networks to play video games.

The error between the network output and the desired output is also known as the loss. The function that quantifies this value is the loss function. Basically, the loss function indicates how well the model performs with the current parameters. There are many different loss functions used in neural networks, but we will only focus on those used in this thesis. Others are beyond the scope of this work. The curious reader is encouraged to look to other literature or the Internet for more information on loss functions.

There are two classes of loss functions: regression and classification. Regression loss functions calculate the extent of inaccuracy when the network is predicting continuous values, while classification loss functions compute the error when predicting a discrete value like a category. For example, mean absolute error (MAE) and mean squared error (MSE) are regression loss functions. MAE is calculated as the average absolute value of the difference between the predictions (x_i) and ground truth values (y_i) and MSE is calculated as the average square of this difference (see equations 1.1 and 1.2). Classification loss functions include categorical cross-entropy and binary cross-entropy (discussed later in chapter 2).

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \tag{1.1}$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2 \tag{1.2}$$

Gradient descent is used to find the best set of parameters for a neural network. It is the process of calculating derivatives and slowly moving along a function to approach local or global minima (see figure 1.5). The amount to change the parameters with each set of training samples is specified by a parameter known as the learning rate η and the direction is found by calculating

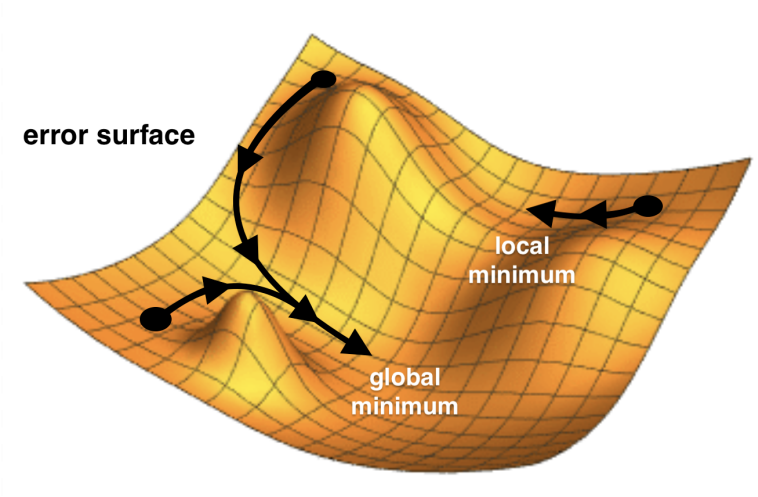


Figure 1.5: Example of gradient descent on a 3D surface with 3 different starting points.

the negative gradient of the error. The negative gradient is used since the positive gradient would indicate the direction of the steepest increase in error. The steepest *decrease* is needed instead because the goal is to reach the global minimum error and produce an output that most closely matches the correct output. The learning rate is a small number $0 < \eta < 1$ that regulates the size of the steps taken by gradient descent. The basic weight update equation for gradient descent is shown in equation 1.3, where w is a matrix of the network weights, η is the learning rate, and $\nabla E[w]$ is the gradient of the error.

$$w = w - \eta \nabla E[w] \tag{1.3}$$

There are three types of gradient descent: gradient descent, stochastic gradient descent, and mini-batch gradient descent. Standard (or batch) gradient descent updates the weights only after all of the training samples have been considered by summing the error for each. Stochastic gradient descent updates the weights after each individual training sample is considered and mini-batch gradient descent is a balance of both. Mini-batch gradient descent computes gradients and updates weights after groups or batches of training samples are considered. This is the preferred method because it reduces compute time and gives a good approximation of the overall gradient while mitigating noise in the data.

A simple fully-connected or dense neural network is shown in figure 1.6. A fully-connected network has an input layer, output layer, and one or more hidden layers. This is perhaps the most basic form of a neural network.

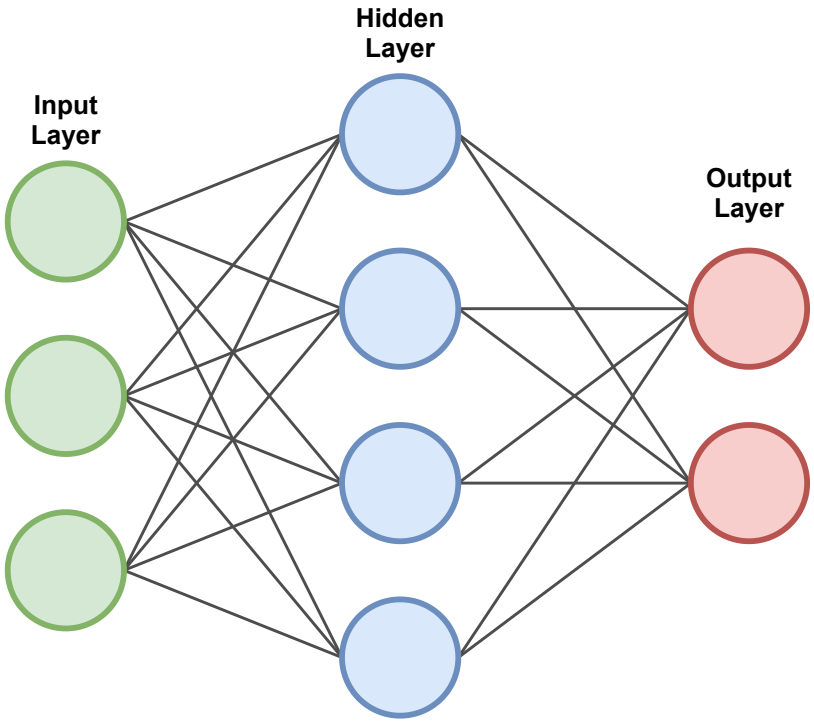


Figure 1.6: Simple fully-connected neural network architecture with one hidden layer.

Deep learning is a subset of machine learning and artificial intelligence that uses neural networks with many hidden layers known as deep neural networks. Training deep neural networks makes use of a technique called backpropagation to update the weights in the network. Backpropagation is needed since the desired values are not known for intermediate hidden layers in the network.

There are several different types of neural networks. First, there is the fully-connected neural network described earlier. In this type of network, every node is connected between two adjacent layers (see figure 1.6). Fully-connected networks can be made "deeper" by increasing the number of hidden layers. This concept applies to other types of neural networks as well.

A convolutional neural network (CNN) is a type of artificial neural network that uses convolution with filters learned during training to generate an output. CNNs are most common in image and speech processing, but they have applications in other one-dimensional (1D), two-dimensional (2D), and even problems with higher dimensionality. Since the convolutional filters or kernels that move across the input use shared weights, these networks usually require drastically fewer parameters. There are even some common, specialized architectures built using these techniques in tandem with other notable approaches like residual connections as in U-Net [27].

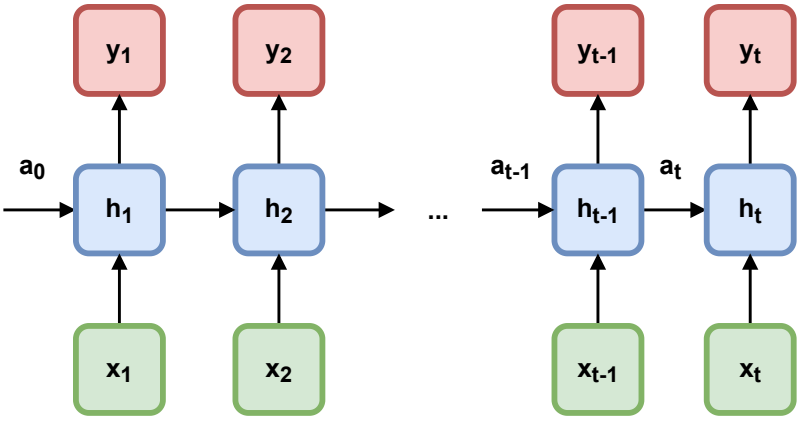


Figure 1.7: Simple recurrent neural network with inputs x_t, outputs y_t, hidden states h_t, and activations a_t for each timestep t.

For this work, the neural networks were built in a Python library called TensorFlow. TensorFlow is an open source machine learning platform that was developed by Google in 2015 [1]. Both TensorFlow and the high-level Keras application programming interface (API) built-in to TensorFlow were used.

1.2.6 Recurrent Neural Networks

A recurrent neural network (RNN) is a neural network where the neurons are arranged in a sequential manner and the input is parsed in the same fashion. Each neuron has an input and an output like normal neurons, but also a hidden state. The hidden state information from previous neurons is used as an input to subsequent neurons. A diagram that depicts the relationship between nodes in an RNN is shown in figure 1.7. There are several different categories of RNN. A one-to-many RNN has one timestep in the input and many timesteps in the output. Many-to-one and many-to-many RNNs are also named accordingly. The type of RNN used in this work is a many-to-many RNN (shown in figure 1.7) where both the input and output have multiple timesteps, which is a paradigm commonly used for machine translation. Moreover, an RNN can also be bi-directional (shown in figure 1.8). In this case, a forward pass of the input is completed first and then the data is flipped for a reverse pass. This allows the RNN to learn attributes of the sequence in both directions. It is also important to note that instead of vanilla backpropagation, backpropagation through time (BPTT) is needed to train an RNN which can be loosely explained as backpropagation back through each of the timesteps of the network.

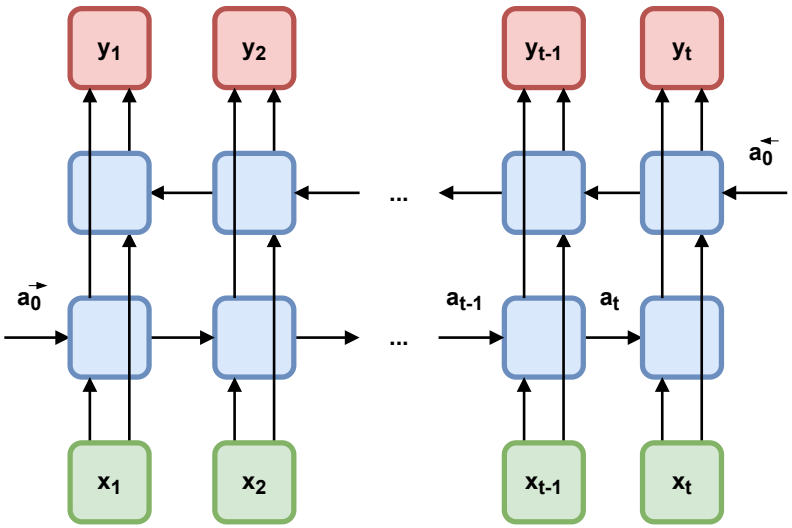


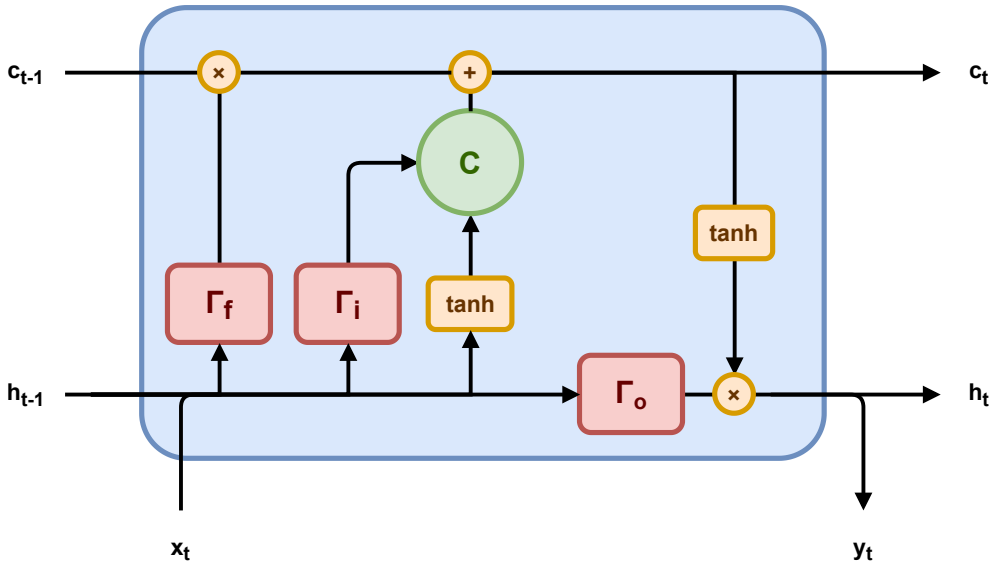
Figure 1.8: Bidirectional recurrent neural network with inputs x_t, outputs y_t, and activations a_t for each timestep t.

The weights and biases in an RNN are shared among all timesteps. This means that increasing the length of the input does not increase the number of parameters in the network. By sharing weights, the number of parameters is drastically reduced and the backpropagation process is simplified.

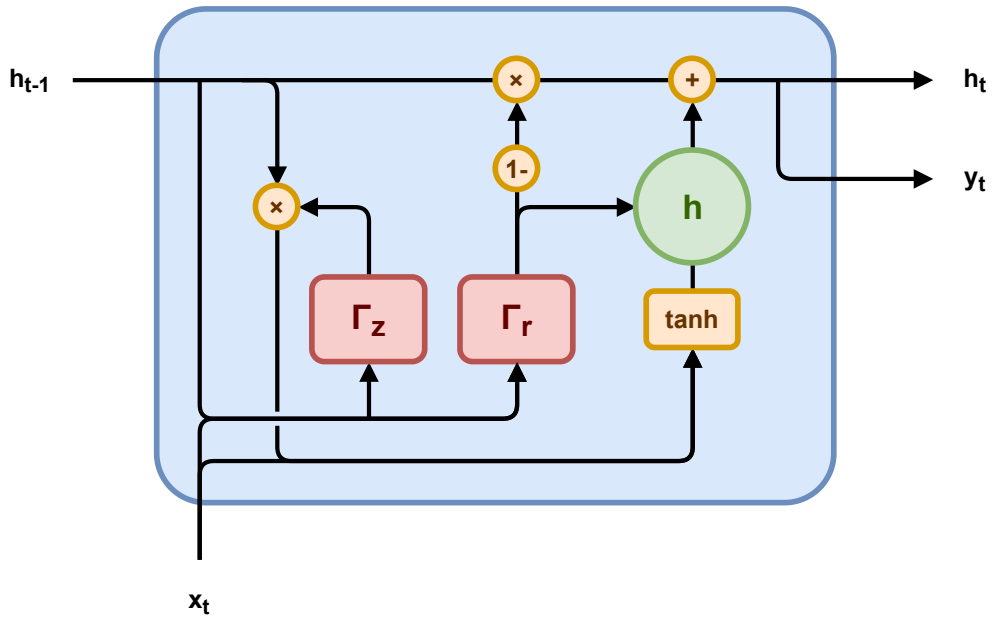
The two common types of neurons used in RNNs are long short-term memory units (LSTMs) and gated recurrent units (GRUs). An LSTM cell is a special type of neuron that includes 3 special gates. A diagram of an LSTM cell is shown in figure 1.9a. The 3 gates are the input gate, forget gate, and output gate. The input and output gates regulate whether the input or output is allowed to pass in or out of the cell respectively. The forget gate controls whether or not the value currently in the memory cell is erased. Each gate processes its designated inputs through a sigmoid function $\sigma(x)$ involving learned weights to produce an output. The general formula for a gate Γ is depicted in equation 1.4, where x_t is the input, h_{t-1} is the previous hidden state, W and U are weight matrices, and b is a bias matrix.

$$\Gamma = \sigma(Wx_t + Uh_{t-1} + b) \tag{1.4}$$

There is a large amount of complexity associated with a single LSTM neuron, so the GRU is often used instead. A GRU cell is a simplified version of the LSTM cell that merges the cell state and hidden state and combines the input and forget gates into an update gate. A reset gate is used instead of a forget gate, but the functions are similar concerning how much past information affects



(a) Long short-term memory (LSTM) cell with memory cell C, input gate Γ_i, output gate Γ_o, and forget gate Γ_f.



(b) Gated recurrent unit (GRU) with hidden state h, update gate Γ_z and reset gate Γ_r.

Figure 1.9: Types of recurrent neural network neurons with input x_t, hidden state h_t, cell state c_t, and output y_t.

the output. As such, there is no control of a memory cell and the full hidden state is exposed to the subsequent neuron. A diagram of a GRU is shown in figure 1.9b. Due to fewer gates and less complex structure, GRUs are more computationally efficient than LSTMs. GRUs have also been shown to outperform LSTMs for some tasks. More specifically, GRUs generally learn less prevalent patterns better, while LSTMs learn highly prevalent patterns better [14]. LSTMs also tend to perform better when deep understanding and long-term context is needed due to their memory. As a side note, in many deep learning frameworks the number of "units" in a cell refers to the dimensionality of the hidden state and cell state.

1.3 Related Work

A multitude of methods for tracking food consumption and ADM have been explored. Researchers have had success with placing sensors on the throat, neck, ears, wrist, and even eyeglasses to monitor energy intake. Amft et al. used an in-ear microphone to analyze chewing sounds of four different foods from four individuals [2]. This method enabled the authors to accurately classify when the individual was chewing and the type of food they were eating, but only from the four preselected foods. They later modified their approach to use a less invasive microphone placed outside the ear and achieved slightly lower accuracy due to environmental noise. Makeyev et al. proposed the use of a throat microphone to reduce ambient noise as they exploit vibrations on the surface of the skin instead of vibrations in the air [22]. The reported results for swallowing recognition were quite good (>95% accuracy) in a lab environment, but ultimately the method was dismissed due to the inconvenient sensor placement on the throat. Nguyen et al. approached the problem with a slightly different angle and utilized a recurrent neural network to detect and characterize eating using the number of times a person swallows [24]. Data was collected from 10 subjects in a controlled environment using a wearable necklace with two piezoelectric sensors and an IMU. The long short-term memory (LSTM) network developed for this task achieved a reported 74% F1 score for swallow detection.

More recently, Gao et al. developed an ADM system to detect eating episodes with the microphones in off-the-shelf Bluetooth headsets [13]. The authors used a traditional machine learning approach with a support vector machine (SVM) classifier as well as a deep learning approach. In a lab setting (N = 28), both approaches yielded 94-96% classification accuracy, but in a free-living

environment (N = 4) accuracy fell. The deep learning approach was still able to achieve 76% accuracy, showing better resilience to noise, but nonetheless a dramatic decrease. The researchers noted that ambient noise is the biggest hindrance for free-living acoustic eating detection.

Since acoustic methods for eating detection are limited by the presence of background noise, comfort, or socially awkward positioning of sensors and microphones, the form factor of eyeglasses has also been tested. Farooq and Sazonov designed a device worn on eyeglasses that incorporated a piezoelectric strain sensor positioned over the temporalis muscle and an accelerometer to detect food intake [11]. Their approach with SVM classifiers and a decision tree resulted in an average F1 score above 99%. Amft and Zhang later used a similar form factor equipped with an electromyography (EMG) sensor to detect chewing and reported an F1 score of 95% [40]. Cameras have also been used in some efforts to monitor energy intake. Doulah et al. used this approach with glasses equipped with an accelerometer, a strain sensor over the temporalis muscle, and a wide-angle camera for food image capture (N = 30) [10]. It was designed to only take pictures when the individual was detected to be eating and achieved an eating episode detection accuracy of 83%.

It has been shown that eating activity can also be detected by monitoring wrist motion with inertial measurement units (IMUs). Most IMUs include gyroscopes and accelerometers and some include magnetometers. When worn on an individual's wrist, these sensors provide information on the orientation and movement of the hand being monitored. Unlike acoustic methods that employ microphones or visual methods that employ cameras, wrist-worn devices do not threaten personal privacy. Moreover, the watch form factor is approachable and many people are already accustomed to wearing watches or fitness trackers. In fact, surveys have shown that a watch form factor is preferred for diet monitoring technology by a sizable margin [16].

Dong, Hoover, and Muth found that there is a characteristic rolling motion in the wrist that occurs when taking a bite of food while eating (see figure 1.10) [7]. The authors used this information to develop a rule-based algorithm to detect and count the number of bites taken. They collected wrist motion data from subjects in a controlled environment (N = 10) with an IMU device. The subjects were permitted to eat a meal of their choice with their desired utensils. With their method, the researchers reported a 91% true positive rate for bites detected serving as a proof of concept for wrist-motion-based eating detection.

Dong et al. were also the first to develop a method to detect periods of eating in normal, day-to-day life as opposed to a laboratory setting. First, the researchers used a wired wrist-worn

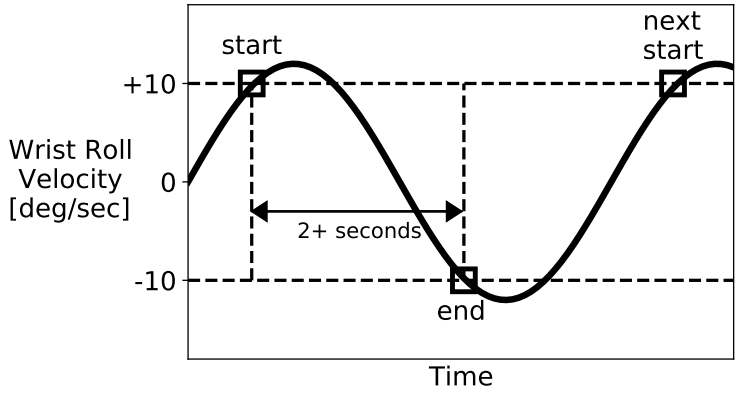


Figure 1.10: Characteristic rolling motion of the wrist corresponding to a bite, adapted from [7].

IMU device known as an InertiaCube3 connected to a laptop and a battery to track wrist motion (N = 4) [8]. An activity classification accuracy of 91% was reported with a rule-based algorithm. A state machine approach for eating detection was also developed with an 82% true positive rate and 70% precision. As a result of this work, the authors concluded that wrist motion could be used to segment eating episodes in natural, day-to-day life.

Another work of Dong et al. used wrist-worn smartphones (Apple iPhone 4) to record accelerometer and gyroscope data from free-living participants instead of the previous apparatus (N = 43, 449 hours, 116 eating events). Their method involved using periods of vigorous wrist motion to bookend periods of eating, which they found typically have less wrist motion. Periods of eating activity were segmented with a naïve Bayesian classifier that yielded a reported 81% accuracy. [9].

In recent years, deep learning has made advancements in many fields, including eating detection from wrist motion for natural daily living. Stankoski et al. used a combination of traditional machine learning and deep learning to process smartwatch IMU data from free-living participants (N = 12) [33]. Their research was focused on detecting eating segments rather than ingestion events (i.e. bites). The authors studied the relationship between model performance and cutlery type used for a meal as well as model performance with personalized models. The model performed better when the subject ate with utensils (as opposed to hands) and personalized models offered slight performance advantages on average. Overall, for their eating detection framework they reported a true positive rate of 81% and precision of 85%.

Luktuke used a deep learning classifier, more specifically a CNN with residual connections, to segment and categorize various eating gestures from wrist motion (N = 276) [21]. The model

architecture, which resembled that of U-Net [27], achieved correct classification of 79.6% of 'bite' and 80.7% of 'drink' gestures. Of all of the gestures in the publicly available Clemson Cafeteria Dataset that was used, 77.7% were correctly classified.

Kyritsis et al. proposed a bottom-up approach to automatically detect food consumption by amalgamating bites into meals [17, 18]. Using data from an off-the-shelf smartwatch (N = 12), the authors achieved a 79% weighted accuracy for eating episode detection with a neural network involving convolutional and recurrent layers [19]. Yet, the authors' latest approach was focused on detecting meals where a fork and/or spoon was the eating utensil of choice. This results in limitations based on the type of cutlery the individual uses (e.g. bare hands, chopsticks) and unpredictability if the user drinks outside of a meal. We believe these to be limitations of a bottom-up approach in general.

Sharma also used deep learning, but with a top-down approach to detecting eating on a much larger dataset [30]. Instead of analyzing eating episodes by grouping bites, eating episodes were segmented and classified based on overall wrist motion throughout the day, i.e. eating detection instead of bite detection. The Clemson All-Day (CAD) Dataset consisting of 4,680 hours of wrist motion data and 1,063 eating events collected from 351 participants was used for this research [32]. To our knowledge this is still the largest publicly available data set of all-day wrist motion data. A sliding window approach, CNN, and hysteresis-based detector were used to detect and segment eating episodes from the wrist motion data. The results of this work were 89% of all meals detected, 1.7 false detections for every true meal detected, and a time weighted accuracy of 80%. This thesis is built upon the work from Sharma, so further information is included in chapter 2.

Wei investigated training the model developed by Sharma [30] for individual participants [37]. The goal was to capture specialized individual-specific eating patterns to improve overall eating episode detection. To do so, a new dataset was collected with at least 10 days of data from 8 different subjects. With individualized models, an increase in average weighted accuracy was reported (82%), but the extent of the increase varied subject by subject.

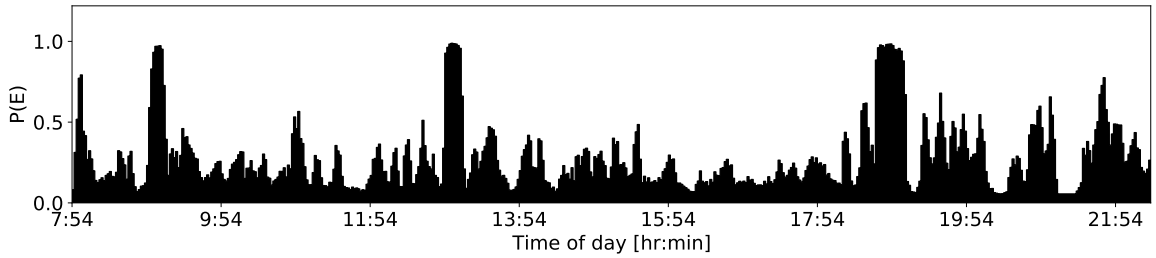


Figure 1.11: Probability of eating $P(E)$ throughout an entire day showing 3 strong peaks, all of which are actual meals, and low background noise.

1.4 Daily Context

Previous work from our research group produced a CNN classifier used to process wrist motion data and output a continuous probability of eating [30]. This model is referred to as the "windowed eating classifier" or "window-based eating classifier". The wrist motion data was from the publicly available Clemson All-Day (CAD) dataset consisting of 4,680 hours of wrist motion data and 1,063 eating events collected from 351 participants [32]. The probability of eating, or $P(E)$, output by the window-based classification model ranges from 0 to 1 throughout the day based on how likely an individual is to be eating based on their wrist movement. A higher $P(E)$ (closer to 1) corresponds to a higher likelihood of eating at that time. Figure 1.11 shows what the daily $P(E)$ looks like for one individual in the dataset. The goal of this work is to use the daily $P(E)$ data in a recurrent neural network model to achieve better overall eating episode detection and largely reduce the number of false detections. This model is referred to as the "daily pattern classifier". With their bottom-up approach, Kyritsis et al. used a base approximation window of 3.6 seconds [18]. The top-down approach Sharma used for the windowed eating classifier analyzed a sliding window of 6 minutes [30]. And, this work looks at an even wider window across an entire day (24 hours).

Within the $P(E)$ data from an entire day, we see various periods of high $P(E)$ indicating a high likelihood of eating. We call these "peaks". These may accurately correspond with an actual eating event or merely wrist movement that closely resembles eating motions. The types of individual peaks we tend to see in daily $P(E)$ sequences are shown in figure 1.12. The first type of peak (a) is flat, solid, and rectangular at values very close to 1.0. This variety of peak almost always corresponds with an actual eating event. The second category of peaks (b) is not as well-defined showing more dispersed $P(E)$ likely caused by secondary activities during eating like watching TV, using a smartphone, or talking with friends. Peak type (c) is a bifurcated peak that suggests the

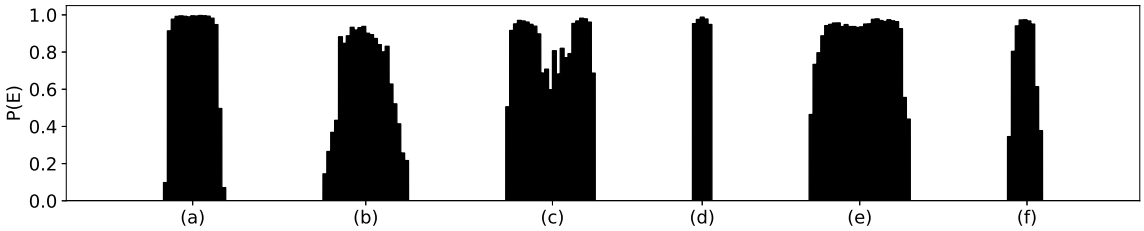


Figure 1.12: Types of individual peaks seen in daily $P(E)$ data: (a) obvious meal (b) fluctuating response (c) bifurcated (d) short meal (e) long meal (f) false detection resembling meal.

individual returned for seconds during a meal or rested between courses. Peak types (d) and (e) demonstrate the varying length of eating episodes. And finally peak type (f) shows a false detection of eating that resembles an authentic eating event. Even within these categories there is variability due to the daily routines, eating rate, or eating technique of different individuals. This suggests that a template matching approach would not be very useful. Thus, a neural network approach was chosen.

The motivating idea for this work is that the $P(E)$ from an entire day provides valuable insight about where eating occurs. On their own, the peaks do not contribute much information that can be used to conclude if they correspond to actual eating. However, this would be roughy the same amount of data analyzed by a windowed or convolutional approach. In short, this myopic approach is limited. We hypothesize that extended quotidian context could help improve classification of eating events by reducing the number of false detections. Moreover, a neural network model architecture designed to learn these temporal relationships and features could use them to accurately predict eating activity from the entire day in a post-hoc manner.

Figure 1.11 depicts that eating typically occurs at isolated periods of high $P(E)$. However, for this example the eating activity is very distinguishable since there is low background noise in the $P(E)$ signal. As mentioned earlier, there is also the possibility of false detections caused by gestures that resemble eating. A false detection or "false positive" would be where there is a period of high $P(E)$, but an actual eating event did not occur. For example, grooming activities that involve moving the eating hand to the face can cause false detections like fixing or brushing hair, adjusting glasses, or touching the face. This can also include instances like morning and bedtime routines where an individual may be brushing teeth, styling hair, shaving, or applying makeup.

We have observed a few contextual clues and patterns that help reduce the number of false detections when manually reviewing $P(E)$ sequences. We denote a period of time related to a pattern

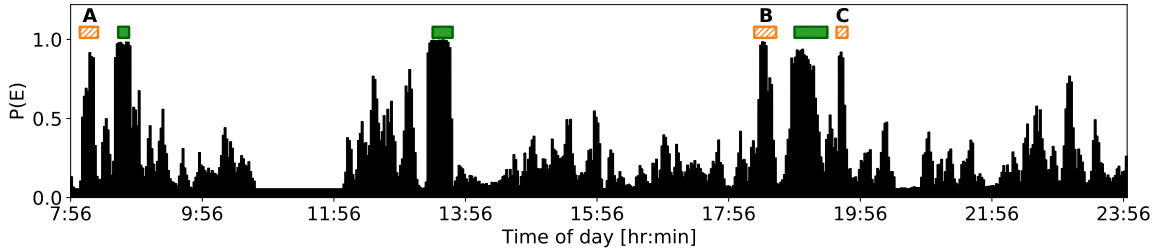
as an "event of interest". As a note, in this work meals that occur before 12:00 are referred to as "breakfast", meals that occur between 12:00 and 16:00 are "lunch", those that happen after 16:00 are denoted as "dinner", and all small meals interspersed throughout the day are "snacks".

First, a regularity of spacing of 4-6 hours between eating episodes is normal. Humans are less likely to eat several full-size meals in a short span of time. The daily meal schedule followed by most people adheres to this pattern as shown in figures 1.13a - 1.13c. In figure 1.13b, the regularity of meal spacing would even help ignore the false detection around 16:30 (B). Similarly, the $P(E)$ sample shown in figure 1.13c has many potential false detections early in the day (primarily B). Yet, these can be ignored by backtracking through the day and using the regularity of the strong meals as clues.

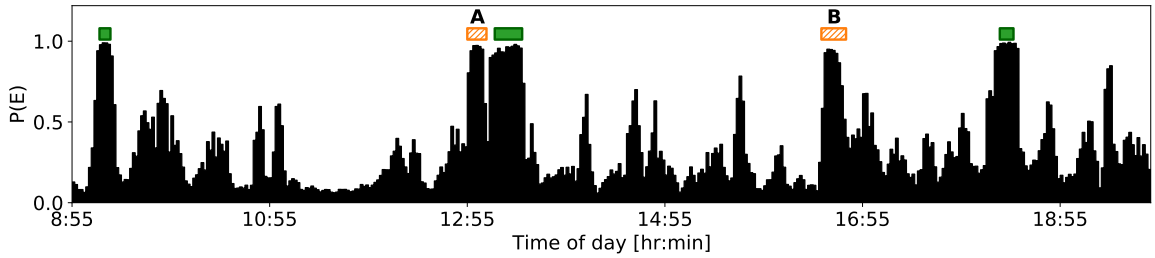
Second, an individual may snack on or taste food while preparing a meal or cooking, however this is not a true meal. This instance is usually indicated by two distinct peaks in close succession where the second peak is actually the meal. For example, the sequence shown in figure 1.13a demonstrates a case where the individual was possibly snacking while preparing dinner since there is a short interval of elevated $P(E)$ close to 18:00 (B). A similar pattern is seen after dinner (C) that may indicate a quick dessert or even seconds while cleaning up that did not constitute a full meal. Figure 1.13b shows a daily sequence with a very long, bifurcated peak around 12:55 (A) that indicates possible light snacking before lunch. In the report for this day the individual noted that lunch was at a restaurant with friends, so the high $P(E)$ may correspond to eating an appetizer or animated conversation.

Third, a morning routine generally occurs early in the day, so high $P(E)$ before the first real meal of the day (usually breakfast) could indicate a false detection. To illustrate this, figure 1.13a shows a daily $P(E)$ sequence the exhibits noticeable evidence of a morning routine with a $P(E)$ peak before breakfast around 8:00 (A). The beginning of the sequence in figure 1.13c likely demonstrates an extended morning routine from 6:45 to 7:15 (A) as well. This anomaly may even be present if the individual skipped breakfast. It would appear equally early in the recording, but without a meal following it.

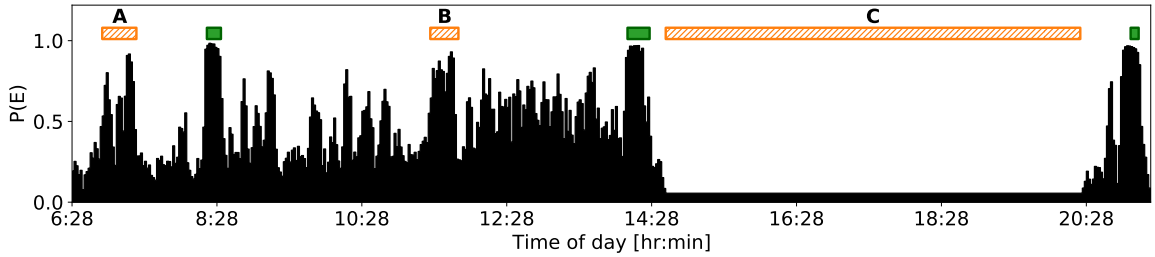
Lastly, periods of rest tend to occur before or after eating a meal. For instance, a person may take a midday nap or siesta in the period between lunch and dinner. In figure 1.13c, a pattern indicating this behavior can be seen as there are the two main strong meals occur preceding and succeeding a large period of rest in the afternoon (from 15:00 to 20:00 [C]). Overall, these important



(a) Daily sequence showing 6 strong peaks, of which 3 are actual meals and 3 are nearby transient responses. The transient responses are probably caused by a morning routine (A), food preparation (B), and cleanup (C), the latter two of which may include light snacking.



(b) Daily sequence showing 5 strong peaks, of which 3 are actual meals. The peak at A may be caused by snacking during food preparation. The peak at B is less likely to be an eating episode because of its proximity to two other peaks (eating episodes tend to be spaced multiple hours apart).



(c) Daily sequence showing 5 strong peaks, of which 3 are actual meals, and higher than typical background noise. Peak A is probably caused by a morning routine, peak B is too proximal to other peaks, and the large rest area (C) provides context that the peaks prior to C and subsequent to C are more likely to be actual eating.

Figure 1.13: Daily $P(E)$ sequences with actual eating episodes (green bars) and other events of interest (orange shaded bars) highlighted.

contextual indicators and patterns can be seen throughout the dataset. The ones presented here are merely the most common, perceptible ones we detected in our analysis. A neural network can be expected to extract less recognizable, latent features as well, hence our approach.

1.5 Novelty

The novelty of this work is applying neural networks to analyze an entire day of data and segment episodes of eating activity. Past work has explored windowed approaches to this problem with CNNs operating on accelerometer and gyroscope data from IMU devices. Using the output of such a model, we are able to analyze daily context for eating patterns in efforts to improve eating episode detection and reduce false detections. Furthermore, to our knowledge, very few works have investigated eating detection for free-living subjects with a dataset of this scale (N = 351). Overall, this work strives to answer the following questions:

1. Does analyzing the probability of eating in a daily context with a neural network improve eating episode classification?

2. Can this approach reduce the number of false detections in eating episode detection?

3. How do the results of this approach compare to those from a window-based classifier?

Chapter 2

Methods

An overview of the methods for this work is shown in the flowchart in figure 2.1. The methods of this study begin with the window-based eating classifier from previous work [30]. This model is used to process wrist motion data and produce continuous $P(E)$ sequences from a day-length recordings, also referred to as "daily samples". For data augmentation purposes, the window-based eating classifier is used repeatedly to generate a large number of daily samples. The daily samples are then saved with the corresponding ground truth, pre-processed by padding, and prepared for the daily pattern classifier. This data is used to train and test the daily pattern classifier with the paradigm of 5-fold cross validation. The output of the model is post-processed with a thresholding algorithm to generate a sequence of predictions from the model. Lastly, various evaluation metrics are measured. As a point of confusion, both the daily pattern classifier and the windowed eating classifier produce a probability of eating as their output. For greater clarity, $P(E_w)$ corresponds to the probability of eating from the windowed eating classifier and $P(E_d)$ refers to the probability of eating output by the daily pattern classifier for the remainder of this work.

2.1 Data Augmentation

2.1.1 Window-based Eating Classifier

To generate daily samples, first the windowed eating classifier was trained. Using a trained model, a sequence of model predictions were generated about the activity of an individual (eating/non-

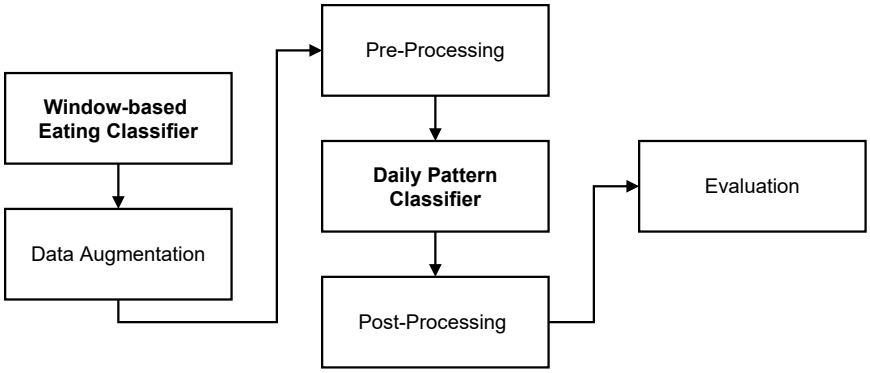


Figure 2.1: Overview of the methods for this work.

eating) that we call the $P(E_w)$. A brief recapitulation of the procedure for training this model is provided here. First, z-score normalized 15 Hz wrist motion data was separated into windows of length W minutes with s second stride between them. These "window samples" were used to train the windowed eating classifier. The ground truth used during training was based on a "majority vote" of the window. If more than half of a window contained eating, the window was marked as eating in the ground truth. Likewise for non-eating windows. Training was balanced with the same number of eating and non-eating windows to prevent classifier bias. After training, a $P(E_w)$ value ranging from 0 to 1 was output by the classifier indicating the likelihood that the provided input window sample contained eating. The $P(E_w)$ output was generated using model prediction, also

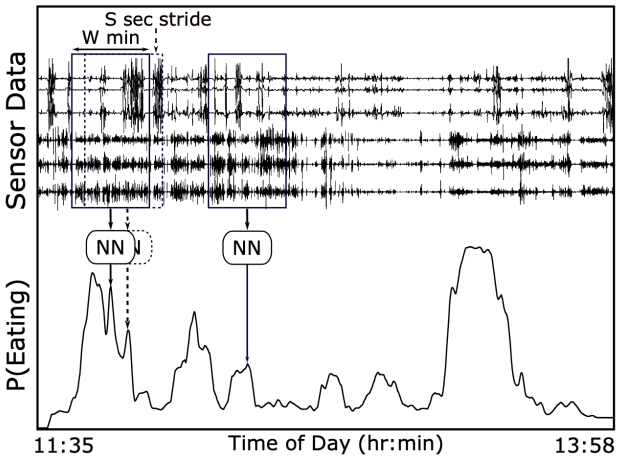


Figure 2.2: Overview of the windowed eating classifier method involving a sliding window of length W minutes and stride s seconds used to generate a continuous probability of eating for an entire day, adapted from [30].

known as model inference, where the output is predicted from an input using a trained deep learning model. By concatenating the outputs from adjacent windows, a continuous $P(E_w)$ sequence was generated for an entire day. This process is depicted in figure 2.2.

2.1.2 Day-Level Data Augmentation

The CAD dataset [32] only contained 354 recordings of an entire day. We required a much larger set of daily samples in order to be able to train and test a day-level classifier. Therefore, we used the CAD dataset to generate 565 different samples from each actual daily recording, yielding approximately 200,000 total daily samples. The data augmentation process is outlined in figure 2.3. To generate this day-level data, the windowed eating classifier was first trained on data from all 354 recordings. The standard training and prediction process described in the previous section was used to generate day-length $P(E_w)$ sequences for each subject in the dataset. An additional step

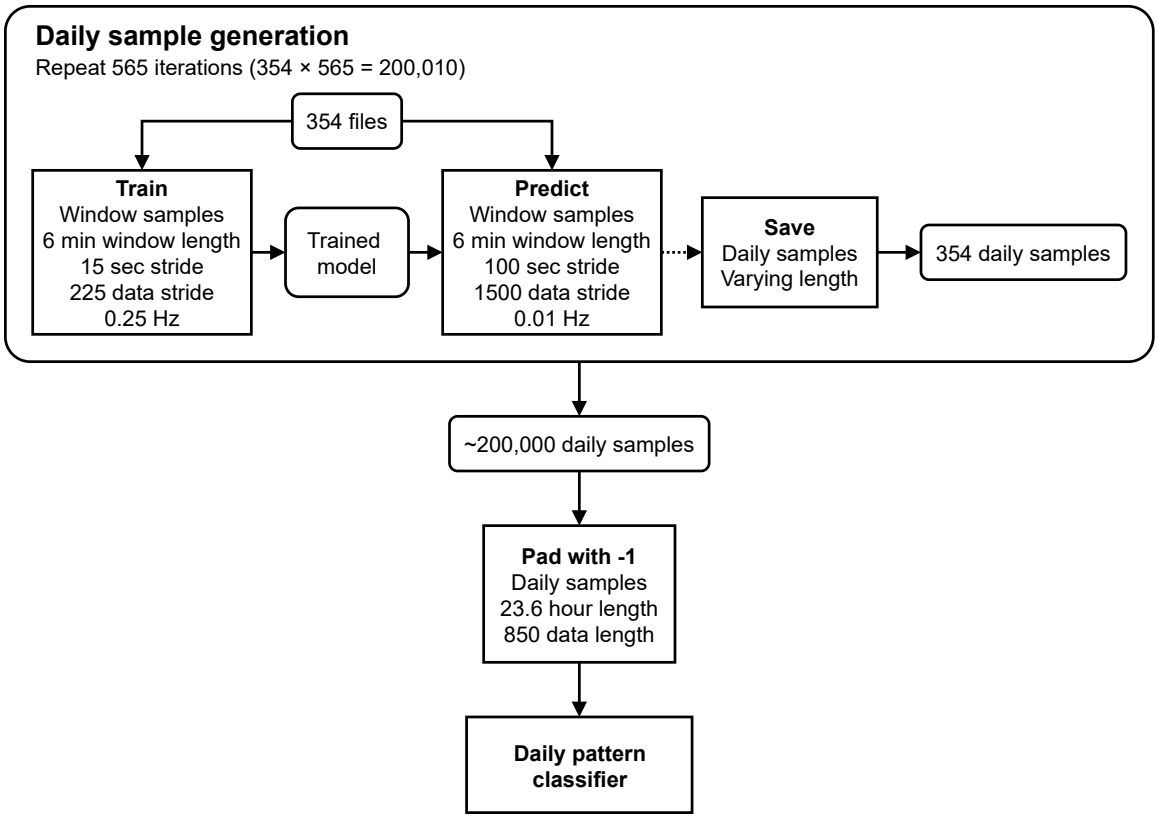


Figure 2.3: Flowchart of the data augmentation process to generate daily sample data for our daily pattern classifier.

...	0.11	0.29	0.58	0.76	0.89	0.95	0.96	0.96	0.94	0.57	0.27	0.15	0.18	0.2	...
...	0	0	0	1	1	1	1	1	1	1	0	0	0	0	...

Figure 2.4: Excerpt from a file containing a daily sample with $P(E)$ values on the top row and corresponding ground truth values on the bottom row.

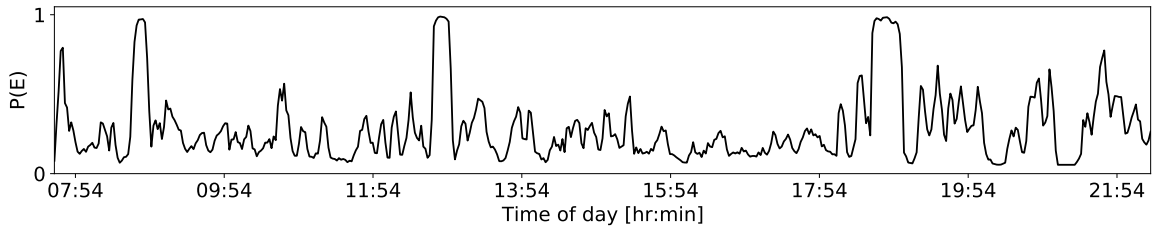


Figure 2.5: Plot of a daily sample showing the probability of eating throughout an entire day.

was added to save the full daily samples to a file with the corresponding ground truth (as shown in figure 2.4). The ground truth was defined as the raw label (eating or non-eating) for the point at the center of the window in the raw data. It is important to note that this varies from the ground truth labeling used for *training* the windowed eating classifier. However, this does emulate the ground truth labeling for *testing* the windowed eating classifier. The convention for labeling the ground truth was a value of 1 to indicate eating and a value of 0 to indicate non-eating. An example of a single daily sample is shown figure 2.5.

This three-part train, predict, and save procedure was repeated multiple times to achieve the desired quantity of generated samples. A total of 200,010 daily samples were generated to accommodate early estimates of the model size and complexity. For training, a window length W of 6 minutes (5400 data) and a stride length s of 15 seconds (225 data) were used as in previous work [30]. Each model was trained for 30 epochs and the model with the best training accuracy was saved. Training code from Sharma was used with minor modifications (e.g. multiple GPU optimization, repeated iterations) [30].

The potential drawback of this approach is that daily samples generated repeatedly from the same daily recording may not have much variability and consequently result in overfitting in our day-level classifier. To explore this issue, we visualized the variance in daily samples generated from the a single actual day-length recording. Figure 2.6 shows an example of the spread between the minimum and maximum values across all of the data generated from a single recording. Similar variance was observed across all actual daily recordings in the original dataset. Additionally, during

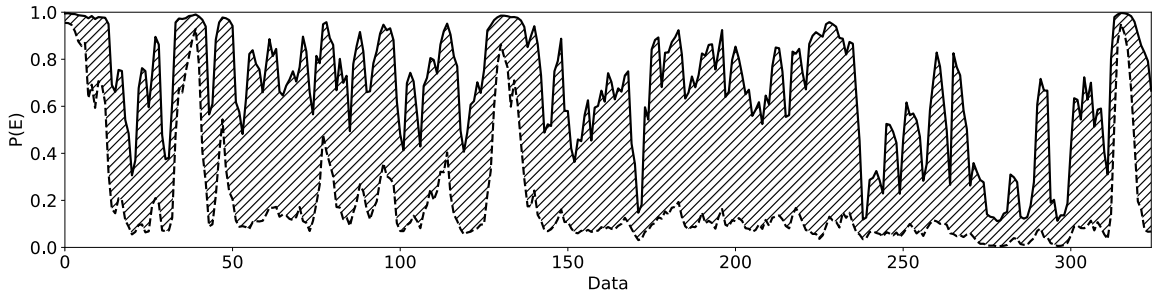


Figure 2.6: Difference (shading) between the minimum (lower dashed line) and maximum (upper solid line) values across all 565 samples generated from a single recording in the CAD dataset.

training and testing of the daily pattern classifier, we do not split the daily samples generated from the same actual day recording across the train/test boundary. That is, daily samples were split by original recording identifier and not by arbitrary indices for training and testing splits. As found in earlier experiments, the volatility of the windowed eating classifier produced substantial genuine noise to precipitate this phenomenon. The average standard deviation of evaluation metrics measured per subject varied 5-10% in subsequent model re-training runs. The full extent of this volatility is quantified and explored in appendix A.

2.1.3 Downsampling

A challenge in analyzing a day-length recording is that there is a large amount of data in a day. A classifier that considers all this data simultaneously would thus have a very large number of parameters and suffer from long training and inference times. Additionally, the time needed to generate the numerous 200,000+ samples had to be reasonable. In order to reduce this complexity, we downsample the day-length recording from 0.067 Hz (1 datum every 15 seconds) to 0.01 Hz (1 datum every 100 seconds or 1.67 minutes) during model inference. We seek to retain the overall daily pattern of $P(E)$ without modeling the small fluctuations that happen second to second.

We chose the downsampling factor using visual estimation. Figure 2.7 shows an example. In this figure daily samples are compared with different stride lengths. Each daily sample from (a) to (e) increases the stride length by an order of magnitude. Smaller increases did not substantially impact any of the prioritized objectives. Daily sample (a) was generated with a 1 datum stride, (b) with a 15 data stride (for a perfect 1 second), (c) with a 150 data stride, and so on. Daily samples (a) - (c) appear largely unchanged at the daily level. A stride length of 1500 data (d) is the first

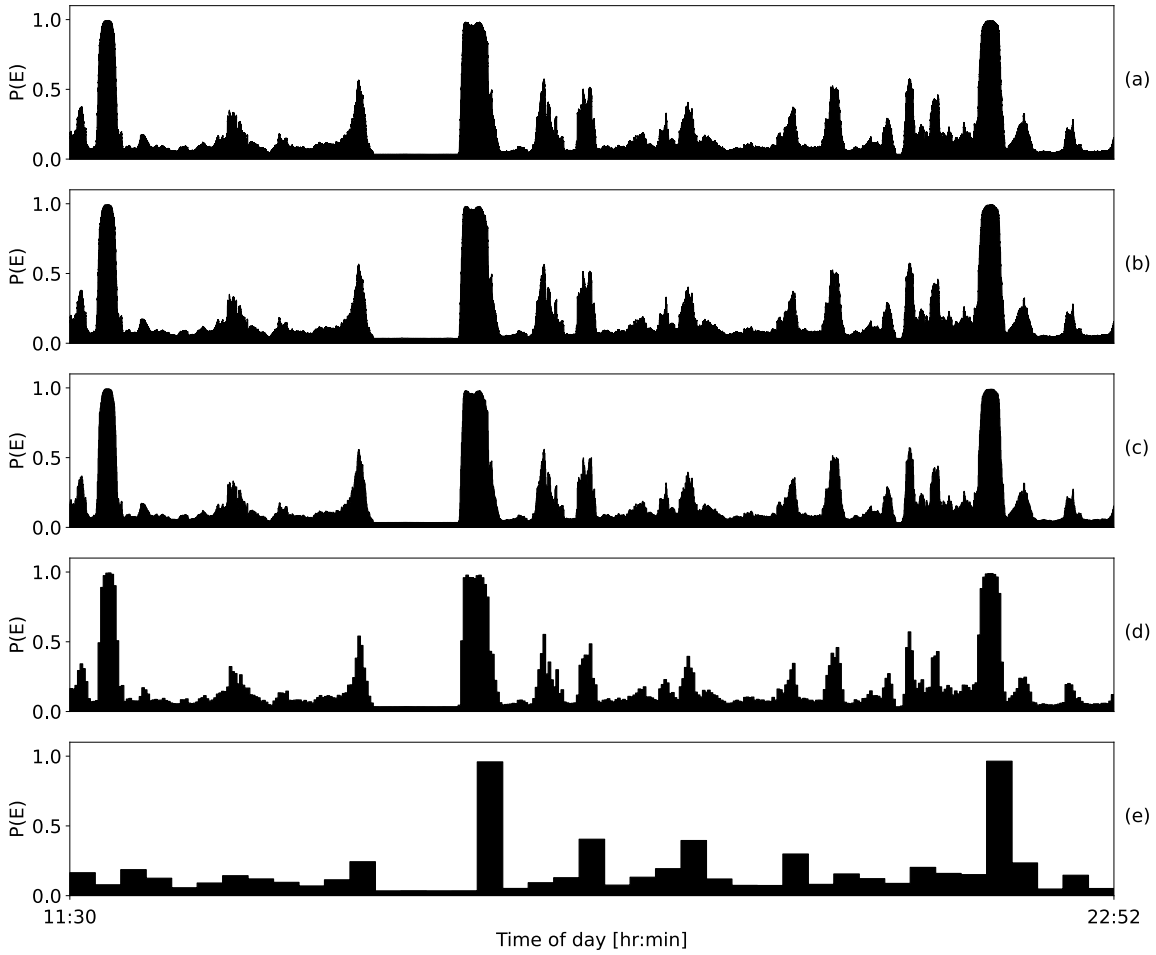


Figure 2.7: Effect of stride length s on resolution of a daily sample: (a) $s = 1$ datum ($\frac{1}{15}$ second), (b) $s = 15$ data (1 second), (c) $s = 150$ data (10 seconds), (d) $s = 1500$ data (100 seconds), (e) $s = 15000$ data (1000 seconds).

order of magnitude jump to show some downsampling. The next order of magnitude increase used to produce daily sample (e) is noticeably losing data. Thus, after scrutinizing this visual comparison of daily sample fidelity, a stride length of 1500 data was chosen because it offered downsampling without loss of the overall integrity of the signal. In summary, a 15 second (225 data) stride length was used for training and a 100 second (1500 data) stride length was used for prediction.

The nature of the windowed eating classifier allowed different stride lengths to be used for training and testing (or prediction). As the name implies, this ability was enabled by the fact that it worked on sliding windows of data. As long as the window length remained constant, changing the stride length only changed the number of windows that would be processed by the model and the resolution of the resulting $P(E)$ signal. This functionality was exploited in the original work to improve data granularity for testing (down to a 1 datum stride) [30].

2.2 Pre-Processing

The characteristics of $P(E)$ data from the windowed eating classifier already confined it to a range of [0, 1]. Consequently, no feature scaling or normalization was used on the daily sample data; raw daily samples were used as the input to the daily pattern classifier. Uniform and Gaussian smoothing were tested, but both degraded performance so they were not used. Since each daily sample was a different length, all samples were padded to the maximum length (850 data = 23.6 hours = $850 \times 100/(60 \times 60)$) with -1 values before processing. The value -1 was chosen because it was outside the range of the probability of eating. These values were masked out later in the model as described in section 2.3.1. The data was also reshaped to a 3D tensor with the shape [`batch`, `timesteps`, `features`] that is required by RNNs in TensorFlow before processing. For this data the shape was [-1, 850, 1], where the -1 value allowed TensorFlow to determine the number of batches at runtime.

2.3 Daily Pattern Classifier

A recurrent neural network design was used for the daily pattern classifier due to the strength of RNNs with variable-length time series data. The memory aspect of RNNs was also imperative to capture important indicators from the daily context. Other architectures were tested including a

fully-connected network and 1D convolutional neural network with residual connections (based on U-Net [27]). Both approaches had drawbacks. The fully-connected "dense" network was limited by the nature of its architecture. Dense architectures are designed to learn the relationship between a fixed-size input and a fixed-size output with both as aggregate units. We needed an architecture that could operate over a sequence datum-by-datum and learn to classify slices of that sequence. The 1D CNN architecture was also restrained as it fell back to a similar myopic view of the daily sequences. Appropriately, it demonstrated poor performance and limitation in learning attributes of the daily patterns. Furthermore, since our data had inconsistent lengths, it would need to be truncated, padded, or divided into fixed-length sliding windows for processing with these networks. Truncating would have removed useful data, padding would have added unnecessary data, and a sliding window approach is antithetical to the goal of capturing daily context. Therefore, an RNN was built for the task.

2.3.1 Architecture

The architecture of the daily pattern classifier includes a masking input layer, a single bidirectional GRU layer, and a time-distributed dense output layer. The bidirectional GRU layer uses tanh activation functions and the output dense layer uses sigmoid activations. A diagram of the model architecture is shown in figure 2.8. Overall, the model includes 1,841 trainable parameters. The classifier was designed, trained, and evaluated using TensorFlow 2.2.0 and the Keras API in Python 3.8.3.

2.3.1.1 Masking Layer

As described previously, the input was padded to with -1 values to a length of 850 data. The first layer of the network, a masking layer, was used to skip these invalid timesteps. That way, the computation of gradients and weights within the network was only dependent on real data and not padded zeros or other fabricated values.

2.3.1.2 Bidirectional GRU Layer

The next layer in the model architecture was a gated recurrent unit (GRU) layer enclosed in a bidirectional wrapper. The default GRU in Keras was used with the reset gate applied to the hidden state before matrix multiplication. A bidirectional wrapper was added around the GRU

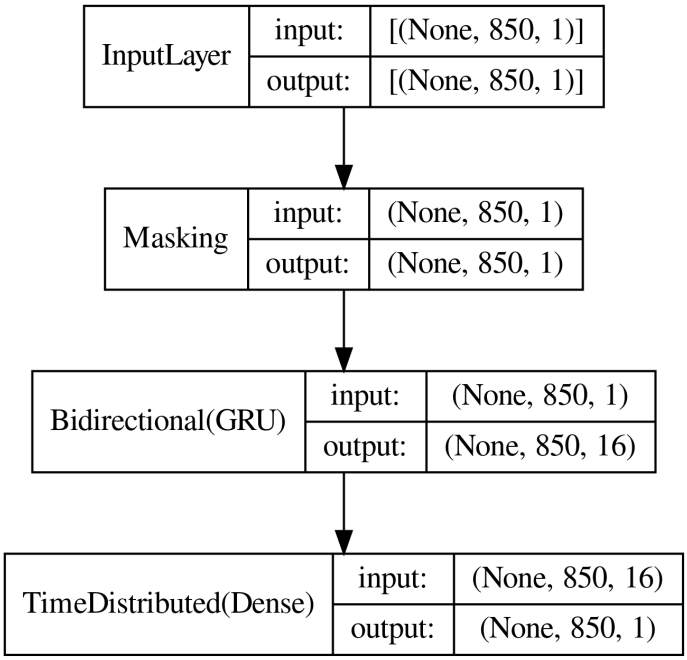


Figure 2.8: Recurrent neural network architecture for the daily pattern classifier ('?' is a placeholder for the number of batches).

layer to compute weights in a forward pass of the input first and then a flipped backward pass. The summation of the outputs from the forward and backward pass was computed instead of the default concatenation as this was found to offer slightly better performance. This was done by setting the `merge_mode` parameter to `sum` in Keras. Xavier normal initialization was used (`glorot_normal` kernel initializer in TensorFlow) for the weights of this layer because it yields better initial values that are related to the structure of the layer. The default activation functions were used. The hyperbolic tangent (tanh) function was used as the activation function for the layer and the sigmoid function was used as the recurrent activation function between recurrent steps within the layer. For the GRU layer, U units were used. Different values of U were tested including 8, 16, 32, 64, 128, and 256 (i.e. powers of 2 from 2^3 to 2^8). Both GRU and LSTM layers were also tested. The full results of this grid search for the best number and type of units is shown in section 3.1.

2.3.1.3 Time-distributed Dense Layer

Lastly, a single-unit dense layer was encapsulated in a time-distributed wrapper for the model output. The TimeDistributed wrapper in Keras applies the enclosed layer to each timestep of the input. This makes it possible to produce an output for each data point in the input when

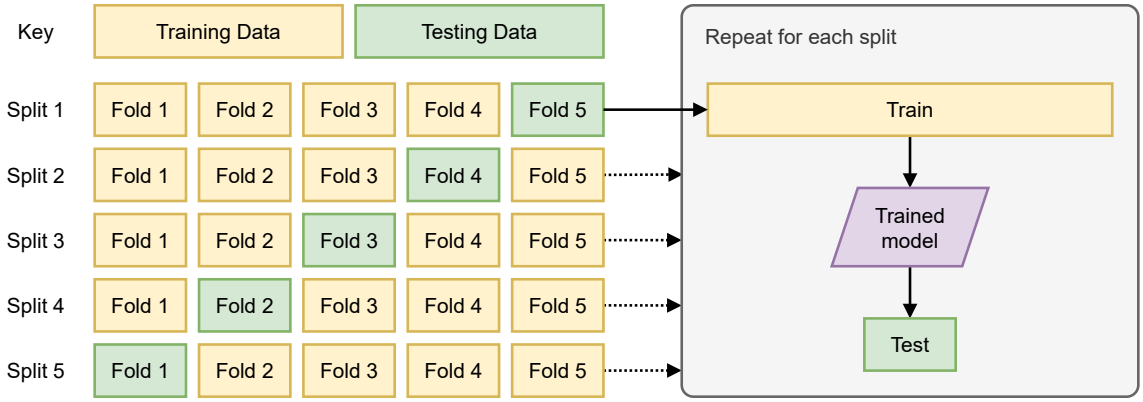


Figure 2.9: Overview of k-fold cross validation, $k = 5$.

combined with a 1-unit dense layer. Hence the input and output dimensions of the overall model are the same: 850 data. A sigmoid activation function was used on the output of the dense layer to limit the output to the range [0,1].

2.3.2 Training

Training was performed using k-fold cross validation. In general, k-fold cross validation is a technique to evaluate the performance of a neural network on all of the data in a dataset. The data is split into k different groups or "folds" and one fold is reserved for testing while the other folds are combined for training. The value for k is usually set to 3, 5, or 10; we will use $k = 5$. Figure 2.9 depicts 5-fold cross validation as it is used in this work. The folds were split by subject from the original CAD dataset [32], rather than by total data. This was done so there was no train/test overlap between samples that originated from the same recording. Each split included training data from 80% of the recordings (≈271 recordings) and testing data from 20% of the recordings (≈71 recordings). Accordingly, of the 200,000 available daily samples, each split involved training on ≈160,000 samples and testing on ≈40,000 samples. Model training was performed on the Clemson University Palmetto Cluster, a high-performance computing (HPC) system, on a compute node equipped with 40 cores, 370 GB of RAM, and 2 NVIDIA Tesla V100 graphics cards with a combined 32 GB VRAM.

2.3.3 Network Hyperparameters

Hyperparameters of a neural network are the adjustable controls outside the layers of the neural network architecture. There are several that need to be set to reliably and repeatably train a neural network like the number of training epochs and the batch size. The necessary hyperparameters for the daily pattern classifier considered in this work are defined here:

1. **Loss Function:** Binary cross entropy loss was used because the model was performing the binary classification task of determining whether each timestep was eating (1) or non-eating (0). The binary cross-entropy loss function is defined as follows, where N is the number of training samples, and $y_i \in \{0, 1\}$ is the target class:

$$\mathcal{L}_{\text{BCE}} = \frac{1}{N} \sum_{i=0}^{N} y_i \cdot \log(\mathbf{P}(y_i)) + (1 - y_i) \cdot \log(1 - \mathbf{P}(y_i)) \tag{2.1}$$

2. **Number of Training Epochs:** The daily pattern classifier was trained for 50 epochs and the model with the best training accuracy was saved. Training was tested up to 100 epochs, but upon inspecting the training accuracy and cross entropy loss on the training data over this range, we only noticed marginal improvement after 50 epochs. Furthermore, training for longer than 50 epochs did not result in any performance advantage on evaluation metrics.

3. **Optimizer:** The Adaptive Moment Estimation (Adam) optimizer was utilized for training. It builds on classical stochastic gradient descent by using adaptive learning rates for different parameters in the network. The Adam method incorporates elements from the RMSprop and Adadelta optimizers. As a result, it is quite popular and one of the best overall optimizers [29]. All parameters for the Adam optimizer were left set to the default values in TensorFlow.

4. **Learning Rate:** Closely related to the optimizer is the learning rate. The initial learning rate for the Adam optimizer was also left at the standard value of 0.001. Training with this value was predominantly stable. Training accuracy and loss were not erratic or slow to converge, so the learning rate was left consistent with TensorFlow defaults.

5. **Batch Size:** A batch size of 64 was used to balance training time per epoch and training accuracy. The batch size is commonly selected as a power of 2, seeing that this has been found to improve training time by best utilizing parallel processing on GPUs. The value of 64 was

the next step up from the default 32. We decided to increase the batch size by one notch to speed up training but avoid changing it too much and disrupting the results.

2.4 Post Processing

For post-processing, each sequence of $P(E_d)$ probabilities output by the daily pattern classifier was passed through a thresholding algorithm. A single threshold T was used for the algorithm, so if a datum in the sequence was above T, the datum was classified as eating (labeled 1). Likewise, if a datum was below T, it was classified as non-eating (labeled 0). This is shown mathematically in equation 2.2, where $f(x)$ is the thresholding function and x is the input $P(E_d)$ value.

$$f(x) = \begin{cases} 1 & \text{if } x > T \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Different values of T were tested. Other heuristics were also employed including merging detections within a half window length of each other and ignoring detections less than 1 minute entirely. The result was a binary sequence indicating the model predictions for where eating occurred. An example of a $P(E_d)$ sample processed by this thresholding algorithm is shown in figure 2.10. This prediction was then compared with the ground truth sequence to calculate the metrics described in section 2.5.

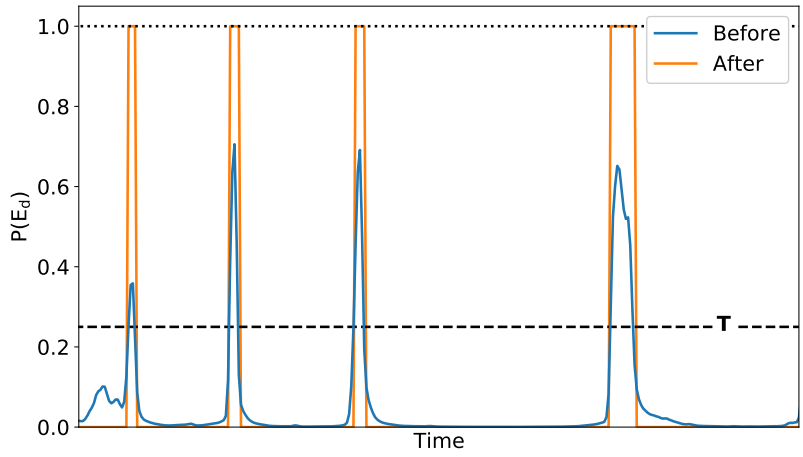


Figure 2.10: $P(E_d)$ signal before and after being processed with the single-value thresholding algorithm ($T = 0.25$).

		Predicted Class		
		Eating	**Non-Eating**	**Total**
Actual Class	**Eating**	True Positive (TP)	False Negative (FN)	Actual Condition Positive (P)
	Non-Eating	False Positive (FP)	True Negative (TN)	Actual Condition Negative (N)

Table 2.1: Eating classifier confusion matrix

For the window-based eating model a dual-threshold hysteresis approach was used for post-processing instead. In this approach the $P(E)$ signal would have to exceed a starting threshold T_S and fall below an ending threshold T_E to count as a detection. As this was used in previous work [30], this was the default for early testing. However, there was less noise in the $P(E_d)$ signal than the $P(E_w)$ signal, so we decided to use a single threshold algorithm instead.

2.5 Evaluation

There are two types of evaluation metrics associated with an eating detection classification model: time and episode metrics [31]. For this work, both sets of metrics were calculated from the binary prediction sequences after post-processing had occurred. First, time metrics were calculated by comparing the prediction output to the ground truth (GT) for each timestep. As mentioned earlier, the GT was constructed using the raw label value at the center of each window that produced a $P(E_w)$ data point. Essentially, this downsampled the GT to the stride length of the data for evaluation so it was consistent with the output of the model. All time metrics computed were based on the four categories of detection in a standard confusion matrix. Any eating classifier would have the same four values. A true positive (TP) occurred if the classifier predicted that a timestep was eating and it was also eating in the ground truth. A true negative (TN) was recorded if the classifier predicted non-eating for a timestep that was non-eating in the GT. A false positive (FP) was marked

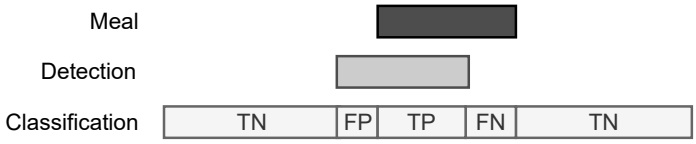


Figure 2.11: Labeling of eating time metrics between ground truth meal and model detection: true positive (TP), true negative (TN), false positive (FP), and false negative (FN)

if the classifier predicted eating when the data point was actually non-eating. And, a false negative (FN) was logged when the classifier missed an eating timestep and instead marked it as non-eating. The confusion matrix for these four detection counts is shown in table 2.1. Two additional quantities can also be computed: actual condition positive count ($P = TP + FN$) and actual condition negative count ($N = TN + FP$). The labeling convention for these types of detections is shown in figure 2.11.

Second, episode metrics were calculated by comparing the overlap of model predicted eating episodes and GT eating episodes. In this context, an eating episode is defined as a continuous interval of time classified as eating (i.e. a series of ones). If there was any overlap between the model prediction and the ground truth, the episode was counted as a true positive (TP). A false positive (FP) eating episode occurred when the model predicted an eating episode when there was not one in the GT data. And, a false negative (FN) was constituted by the classifier missing a ground truth event. There is no concept of a true negative (TN) eating episode for episode metrics with this data. The different types of episode detections are shown in figure 2.12.

The metrics for both categories were calculated from these detection categories. We considered the following time metrics: weighted accuracy (Acc_W), true positive rate (TPR), true negative rate (TNR), F_1 score, and precision (equations 2.3 - 2.7 on page 37). Weighted accuracy was considered instead of raw accuracy because of the class imbalance in the data. Eating was far less common than non-eating. For weighted accuracy, W is the weighted balance ratio of non-eating to eating calculated prior to this metric by fold. TPR is also known as sensitivity or recall and TNR is also

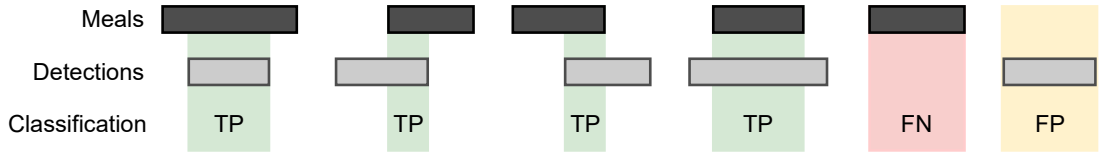


Figure 2.12: Labeling of eating episode metrics between ground truth meals and model detections: true positive (TP), false positive (FP), and false negative (FN)

known as specificity. The selected terms were chosen for clarity.

$$\mathrm{Acc}_W = \frac{W \cdot \mathrm{TP} + \mathrm{TN}}{W \cdot \mathrm{P} + \mathrm{N}} \tag{2.3}$$

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2.4}$$

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} \tag{2.5}$$

$$F_1 = \frac{\mathrm{TP}}{\mathrm{TP} + 1/2 \cdot (\mathrm{FP} + \mathrm{FN})} \tag{2.6}$$

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{2.7}$$

For episode metrics, we only considered TPR and F_1 score, each calculated in the same manner but with episode instead of time detection counts. We also evaluated the number of false positives per true positive (FP/TP) as an indicator of how much the classifier was triggering for other events throughout the day.

Evaluation metrics were calculated after summing up all of the time TPs, TNs, FPs, and FNs and the episode TPs, FPs, and FNs accordingly, not after processing each recording. Furthermore, evaluation metrics were only measured for data within the original recording length, not the entire padded sequence.

2.6 Runtime Considerations

2.6.1 Data Loading

A problem of this scale required specific implementation details to perform well without requiring an exorbitant amount of time. First, to import the 200,000 daily samples from individual text files, the files were read in parallel using the `multiprocessing` module in Python to decrease load times. However, the Python scripts for the daily pattern classifier needed to be run many times during development and testing, so additional runtime optimizations were added. The data was

loaded from all 200,000 text files only once and saved to binary NumPy files in the proprietary `.npy` format for subsequent usage. This dramatically improved load times from over 40 minutes to just 2 seconds (1200x faster). Additionally, the binary NumPy files provided greater storage efficiency requiring 67% less space (1.44 GB instead of 4.3 GB). Text files (`.txt`) and comma-separated values files (`.csv`) are some of the least efficient ways to store and load numerical data in Python. Both were tested during the process of optimization and found to be suboptimal.

2.6.2 Model Inference

Model inference or model prediction was needed to generate the daily samples as described in section 2.1.2. Since there were 200,000 samples overall, the 20% testing set for each fold of k-fold cross validation still included 40,000 samples. To improve model inference runtime, these samples were grouped into very large batches (e.g. 4,096) using the `batch_size` parameter on the `model.predict()` function. Unlike the batch size for training, the batch size for testing has no impact on performance since the model has already been trained. The only limit is the system memory, or more commonly GPU memory when training on a GPU.

Chapter 3

Results

This chapter includes the results of several experiments analyzed in this work. First, attribute testing for the type and number of memory units used in the RNN layers of the model architecture is evaluated. Second, the general performance of our day-level classifier is evaluated. This analysis is performed by comparing the time and episode metrics across different values of the threshold T in post-processing. Lastly, a detailed comparison is made between the results of the daily pattern classifier (with the values from previous experiments) and those from the windowed eating classifier.

3.1 Type and Quantity of Memory Units

A grid search was used to find the number of units U in each layer and the type of layer (LSTM or GRU) that offered the best performance. Performance was measured using the episode true positive rate (TPR) and time weighted accuracy. Powers of 2 from 8 to 256 were used for the grid search since these are common in RNN design. The effect of the number of units and the type of unit on the episode TPR is shown in figure 3.1. Similarly, the effect of these variables on the time weighted accuracy is shown in figure 3.2. The metrics in these plots were evaluated with $T = 0.4$ for post-processing.

In both figures there is a maximum point at 16 memory units – for both the LSTM and GRU – with a decline of 4-5% as the number of memory units increases to 256. Fewer units also offered slightly reduced performance. In both episode TPR and time weighted accuracy the

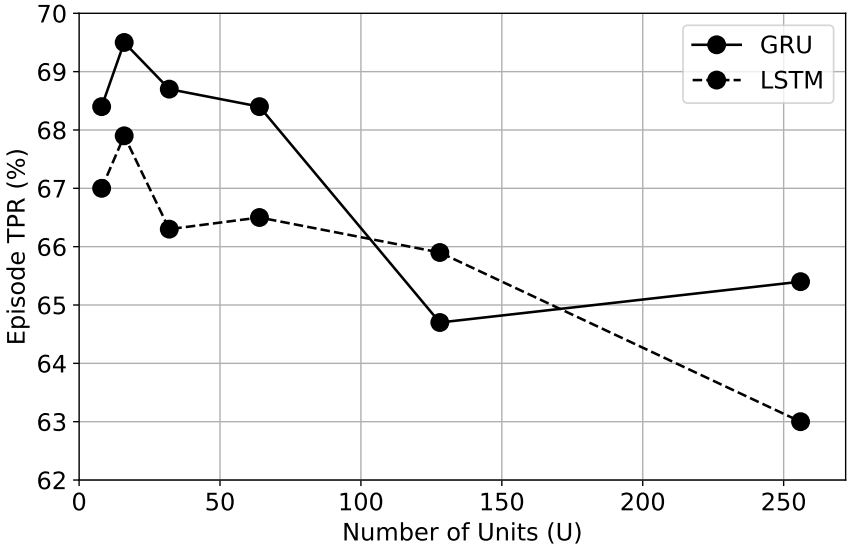


Figure 3.1: Effect of the number of units in each layer on TPR compared between LSTM and GRU cells, $T = 0.4$.

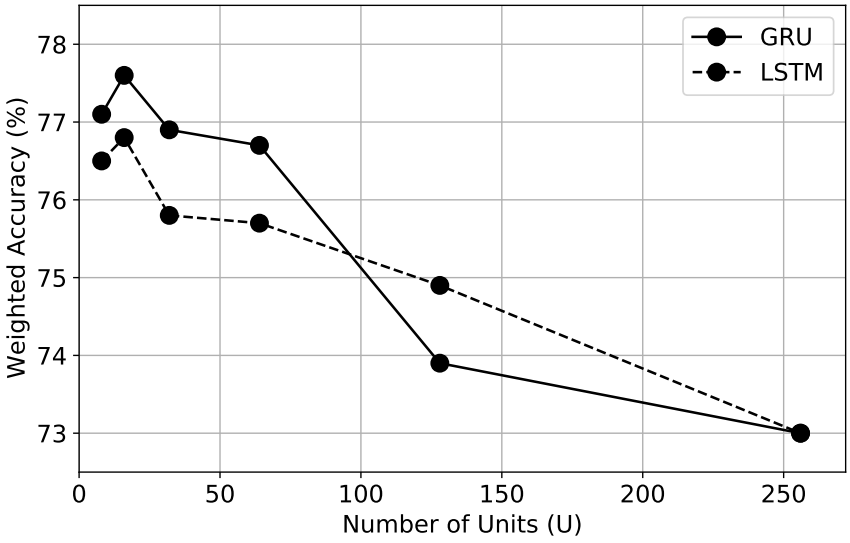
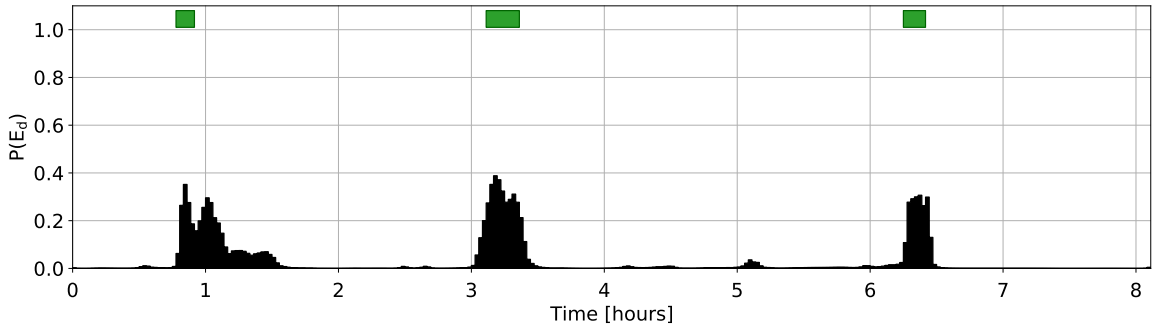
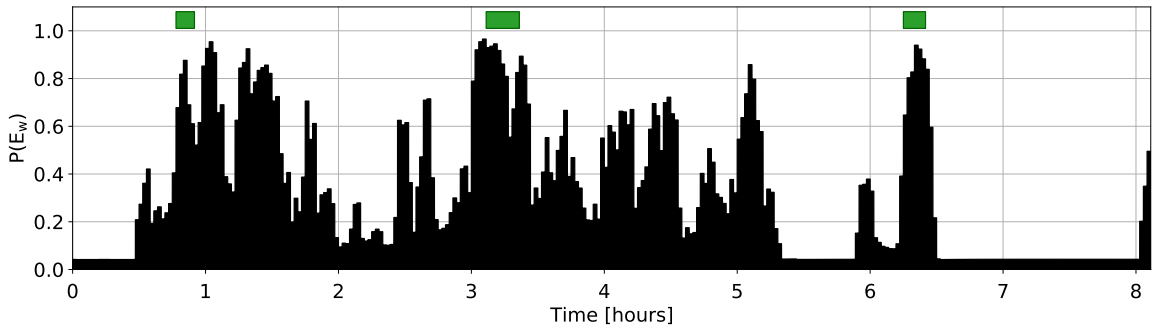


Figure 3.2: Effect of the number of units in each layer on time weighted accuracy compared between LSTM and GRU cells, $T = 0.4$.



(a) Daily pattern classifier, $P(E_d)$



(b) Windowed eating classifier, $P(E_w)$

Figure 3.3: Comparison between $P(E_d)$ and $P(E_w)$ for a sample with high background noise. Daily pattern classifier shows a subdued response with an input of this type. GT shown with green bars.

GRU layers outperformed the LSTM layers by 1-2% for every number of memory units except 128. GRUs have been shown to pick up on less prevalent patterns [14], so this aspect may explain this slight performance difference. Furthermore, LSTMs are more performant when extensive, long-term context is required. On the contrary, the data used with this classifier is relatively short (only hundreds of data long) and 1-dimensional. Based on these results, a value of $U = 16$ units was chosen with a GRU layer over an LSTM layer. All further results are reported with the parameter $U = 16$.

3.2 Performance Analysis

Figure 3.3a depicts an example output from our day-level classifier, for the input from the window-based classifier shown in figure 3.3b. It can seen that the background noise is much lower in $P(E_d)$ than in $P(E_w)$, demonstrating that the daily pattern classifier can mitigate many of the transient responses from the window-based eating classifier. The overall level of $P(E_d)$ is also lower

and more subdued than $P(E_w)$. Because of this, the threshold used for final segmentation on $P(E_d)$ needed to be much lower than the threshold used for $P(E_w)$ and even the standard cut-off of 0.5 used for a single-threshold approach.

A range of values for the threshold T were tested. To evaluate these thresholds, the episode true positive rate (TPR), number of false positives per true positive (FP/TP), and time weighted accuracy (Acc_W) were all computed at each value T. The goal for daily pattern classifier performance was an episode TPR exceeding 85% to roughly match performance reported in previous work [30] and less than 1.0 FP/TP.

The effect of T on the time weighted accuracy is shown in figure 3.4. At first, values of T decreasing from 0.8 to 0.1 in steps of 0.05 were tested since 0.8 was the value that offered the best performance with $P(E_w)$ [30]. Yet, for this classifier it led to the worst performance. Furthermore, the weighted accuracy continues to increase as T approaches 0.1. So, values decreasing from 0.1 to 0.01 in steps of 0.01 were tested as well. And, as depicted in figure 3.4, Acc_W reaches a maximum at $T = 0.05$ and then sharply drops off.

Likewise, the effect of T on the episode TPR and FP/TP is shown in figure 3.5. As T decreases, the episode TPR increases and the number of FP/TP increases too. Values of T between 0.2 and 0.05 are situated in the optimal region known as the"knee" of the curve. However, the goal of less than 1.0 FP/TP limited T to values over 0.08. Values of T below the maximum weighted accuracy seen in figure 3.4 at $T = 0.05$ exhibit higher episode TPR rates, but at the expense of weighted accuracy. This means that at these thresholds the model is classifying large portions of the recording as eating since T is so low. Ultimately, $T = 0.1$ was chosen to meet these objectives. All further results are reported using the threshold $T = 0.1$.

Figure 3.5 also shows the performance of the windowed eating model for various values of T_S while T_E was held constant at 0.3 reported in [30]. This figure gives a visual representation of the proximity of performance between the two classifiers in addition to the improvement in FP/TP discussed further in the next section.

3.3 Comparison to Previous Work

We compare our results to the CNN window-based eating classifier from previous work. All time evaluation metrics for the daily pattern classifier and the windowed eating classifier are shown

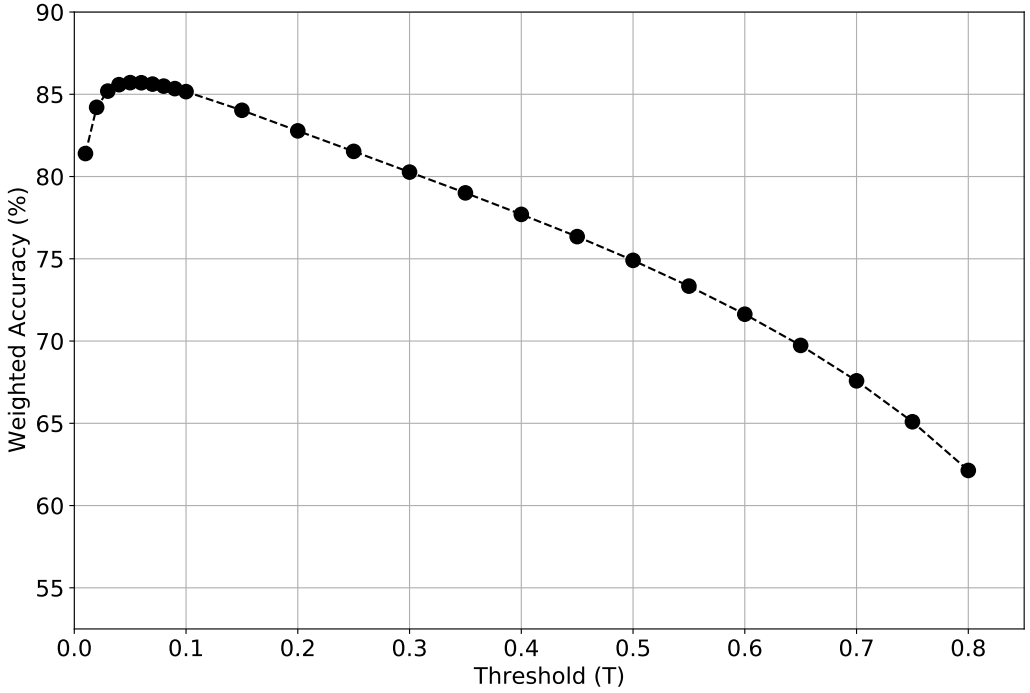


Figure 3.4: Effect of threshold T on time weighted accuracy for daily pattern classifier.

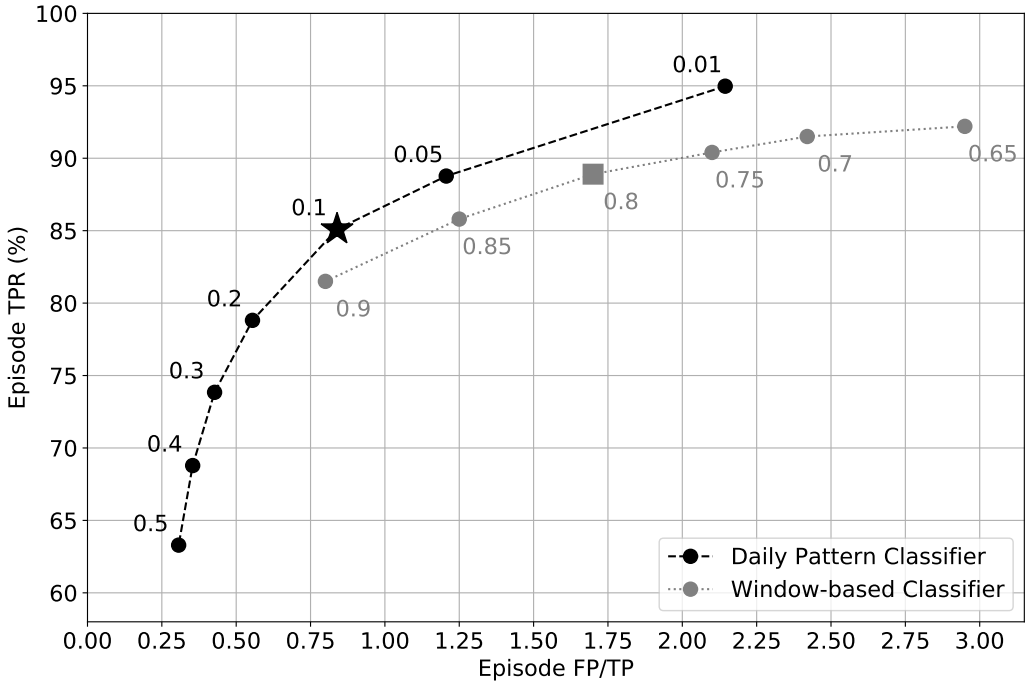


Figure 3.5: Effect of threshold T (number next to points) on episode TPR and FP/TP for the daily pattern model. The effect of threshold T_S (number next to points) on the window-based classifier with $T_E = 0.3$ reported in [30] is also shown for reference. ★, ■ indicate selected values.

Model	TPR (%)	TNR (%)	F_1 (%)	Precision (%)	Acc_W (%)
Window-based Classifier [30]	69	93	48	36	80
Daily Pattern Classifier	78	93	50	37	85

Table 3.1: Time metrics comparing windowed eating classifier and daily pattern classifier (this work). The latter shows a 5% increase in Acc_W and a 9% increase in TPR with other metrics remaining similar.

Model	TPR (%)	FP/TP
Window-based Classifier [30]	89	1.7
Daily Pattern Classifier	85	0.8

Table 3.2: The daily pattern classifier shows a 4% decrease in TPR, but a 53% decrease in the number of FP/TP. Eating episode metrics shown for the window-based eating classifier.

in table 3.1. Similarly, episode metrics for these two classifiers is shown in table 3.2. The results for the windowed eating classifier are those reported in previous work with $T_S = 0.8$ and $T_E = 0.4$ [30]. A comparison between the results reported in [30] and those obtained from replicating the experiment are included in appendix B. When we compare the evaluation measures of the day-level classifier and the window-based eating classifier we find that the time and episode metrics are comparable, while there is a drastic reduction in the number of FP/TP (i.e. false detections per true detection). With the chosen threshold of $T = 0.1$, the daily pattern classifier achieved an 85% true positive rate (TPR) for eating episodes with only 0.8 FP/TP and a time weighted accuracy of 85%. For the windowed eating model, an 89% episode TPR with 1.7 FP/TP and a time weighted accuracy of 80% was reported in previous work [30]. This is only a 4% decrease in episode TPR with the daily pattern classifier, but a 53% decrease in the number of FP/TP at the chosen $T = 0.1$. However, there is lower FP/TP values for all episode TPR values when compared. Full evaluation results are shown in table B.3 in appendix B.

For further analysis, we consider a signal processing perspective. With our day-length $P(E)$ signal, figures 3.6 - 3.10 demonstrate that our daily pattern classifier produces a signal with a much higher signal-to-noise ratio (SNR) than the windowed eating model. This means that the meal peaks in the $P(E_d)$ signal are more distinguishable from the other background noise. Conversely, the $P(E_w)$ signal for the window-based eating model has a much lower SNR. The lower SNR necessitated the more complex hysteresis thresholding method. Despite this post-processing approach, there was a much higher number of false detections. Our new daily-level classifier improves considerably in this

regard. Overall, the daily pattern classifier produced far fewer false detections than the windowed eating classifier from previous work.

Figures 3.6 - 3.10 show direct comparisons between the output of the windowed eating classifier and the output of the daily pattern classifier. Figures 3.6 - 3.9 compare the $P(E_d)$ and the $P(E_w)$ output for the same individuals shown in section 1.4. In each, the ground truth events are shown with green bars and detections for the respective classifiers are shown above the GT with blue bars.

In figure 3.6 there is a recording where the daily pattern classifier retains detection of all three eating episodes and reduces much of the background noise in the signal to zero. While there was no improvement in episode detection in this case, it can be seen that the overall output is more clean and refined.

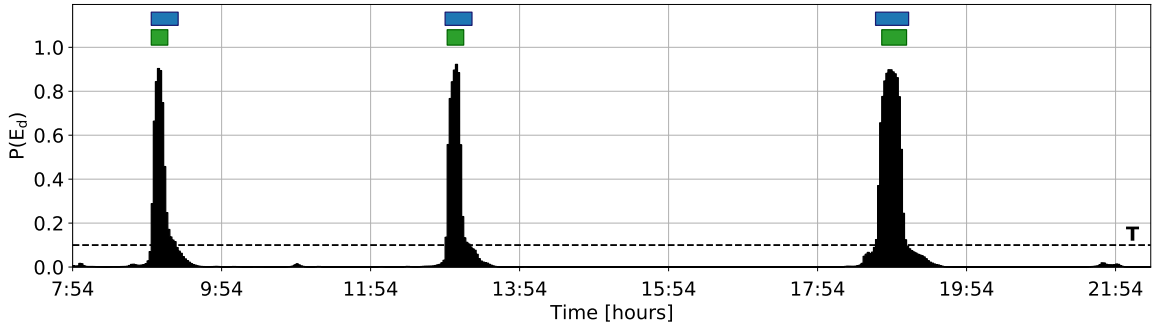
Figure 3.7 shows average performance of the daily pattern classifier with all three eating episodes detected and a 50% reduction in the number of false detections. The only false positive is triggered early in the recording, down from the two transient detections near the third strong peak. These brief responses likely indicate meal preparation/clean-up or light snacking before and after the last strong peak of the day in the $P(E_w)$. With our day-level classifier, they have been refined and melded into the strongest of the three peaks to make up the last detection in the $P(E_d)$.

The recording shown in figure 3.8 has a generally noisy $P(E_w)$ signal with several false detections throughout the day. The daily pattern classifier filtered many of them, but it did not substantially reduce the peak around 16:55, which triggered a false detection in the $P(E_d)$. This exemplifies excellent performance by the daily pattern classifier with an 80% reduction in the number of false detections. Other than the one false detection mentioned, the $P(E_d)$ signal for this recording is clean and all three meals are detected.

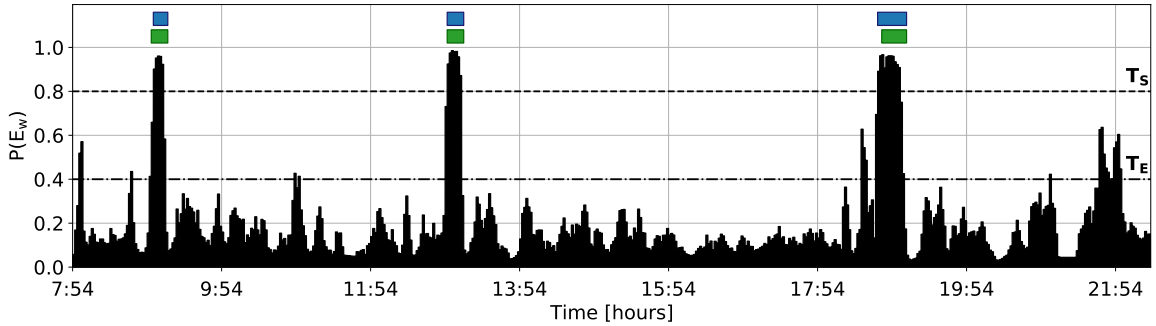
In figure 3.9 a $P(E_w)$ recording is shown with abundant background noise in the first half and a noticeable period of rest in the second half. The $P(E_d)$ generated from this recording has remarkably low noise with a large difference between the peaks and the ambient signal. All three meals were detected and the number of false detections was reduced from 5 to 0 demonstrating a best case scenario for the daily pattern classifier.

One final example in figure 3.10 is included to show that our daily pattern classifier cannot recover true positive eating episodes from false negatives in the $P(E_w)$ signal. If an eating episode is missed by the windowed eating classifier and it is consequently not recognizable as an eating episode

in the $P(E_w)$ sequence, it will very likely not be recovered. In the case of figure 3.10, the last ground truth meal is not recognized by either the windowed eating classifier or the daily pattern classifier. There are still 75% fewer false detections, but the false negative episode is not recovered as a true positive by the daily pattern classifier. This specific $P(E_w)$ recording was very tumultuous, which produced a less confident model output with subdued peaks around 0.3 in the $P(E_d)$ signal. As a result, the last meal in the recording was suppressed and did not qualify as a meal. A lower threshold T for this specific recording would perhaps register this detection but changing T would not be advantageous on the whole.

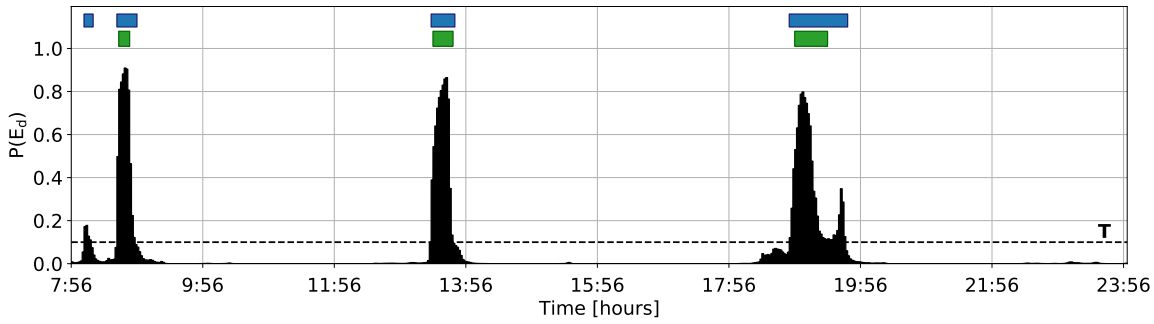


(a) Daily pattern classifier $P(E_d)$ with $T = 0.1$. 3 TP, 0 FP, 0 FN

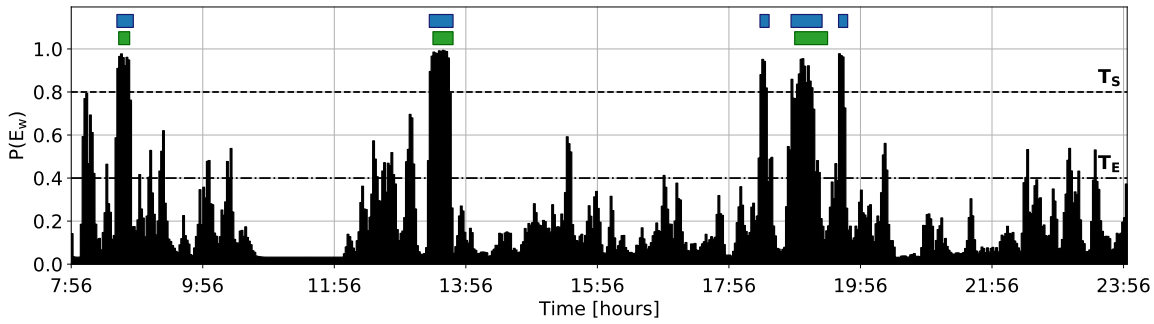


(b) Windowed eating classifier $P(E_w)$ with $T_S = 0.8$ and $T_E = 0.4$. 3 TP, 0 FP, 0 FN

Figure 3.6: Comparison between $P(E_d)$ and $P(E_w)$ showing noise reduction, but no marked improvement on a $P(E_w)$ recording with low background noise. Detections shown with blue bars (top) and GT shown with green bars (bottom).

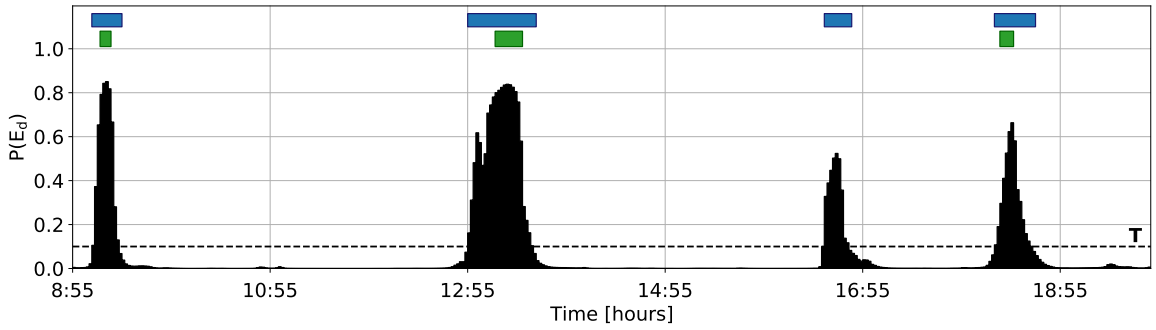


(a) Daily pattern classifier $P(E_d)$ with $T = 0.1$. 3 TP, 1 FP, 0 FN

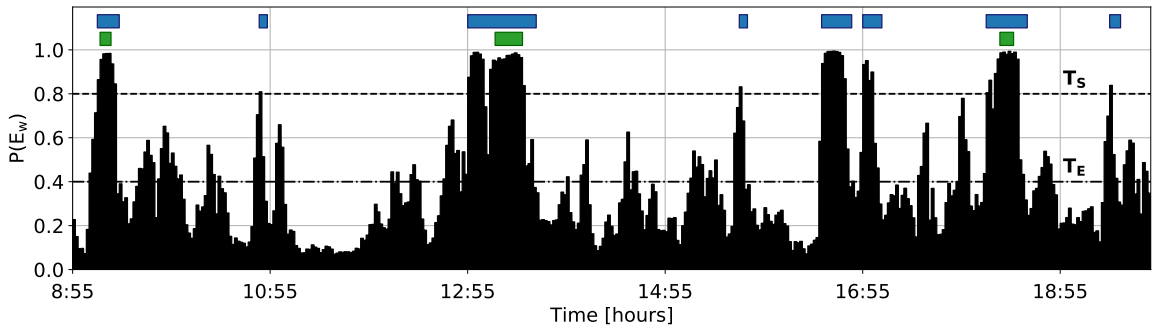


(b) Windowed eating classifier $P(E_w)$ with $T_S = 0.8$ and $T_E = 0.4$. 3 TP, 2 FP, 0 FN

Figure 3.7: Comparison between $P(E_d)$ and $P(E_w)$ showing decent performance of the daily pattern classifier with a 50% reduction in the number of false positives in the $P(E_d)$. Detections shown with blue bars (top) and GT shown with green bars (bottom).

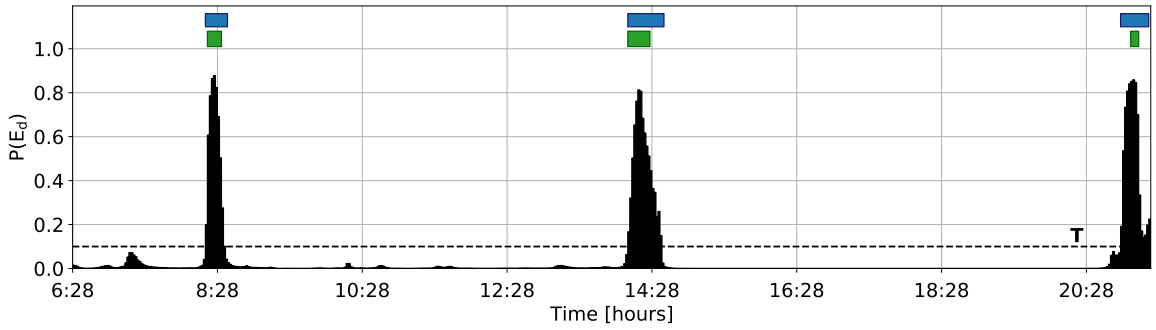


(a) Daily pattern classifier $P(E_d)$ with $T = 0.1$. 3 TP, 1 FP, 0 FN

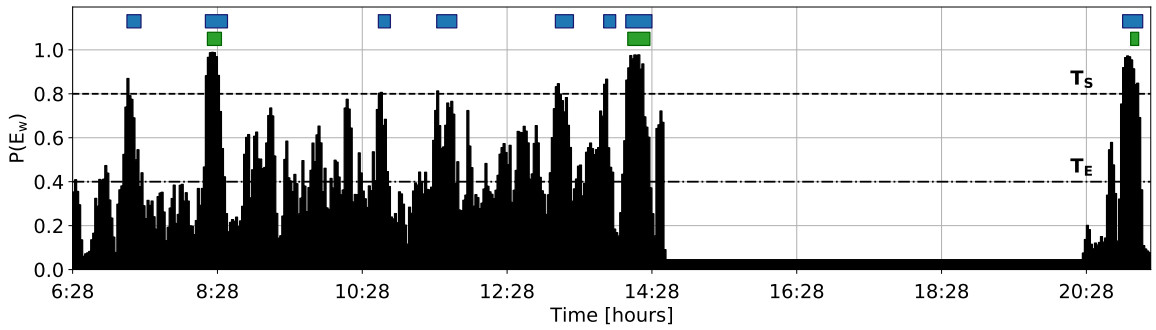


(b) Windowed eating classifier $P(E_w)$ with $T_S = 0.8$ and $T_E = 0.4$. 3 TP, 5 FP, 0 FN

Figure 3.8: Comparison between $P(E_d)$ and $P(E_w)$ showing an 80% decrease in the number of false positives in the $P(E_d)$. Detections shown with blue bars (top) and GT shown with green bars (bottom).

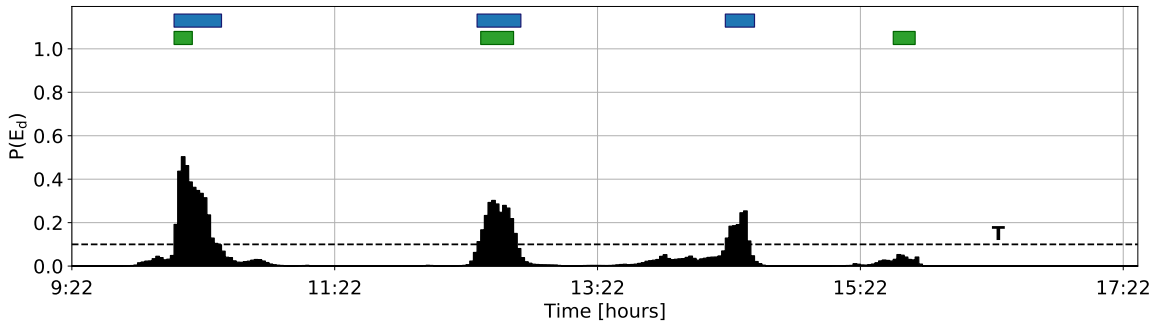


(a) Daily pattern classifier $P(E_d)$ with $T = 0.1$. 3 TP, 0 FP, 0 FN

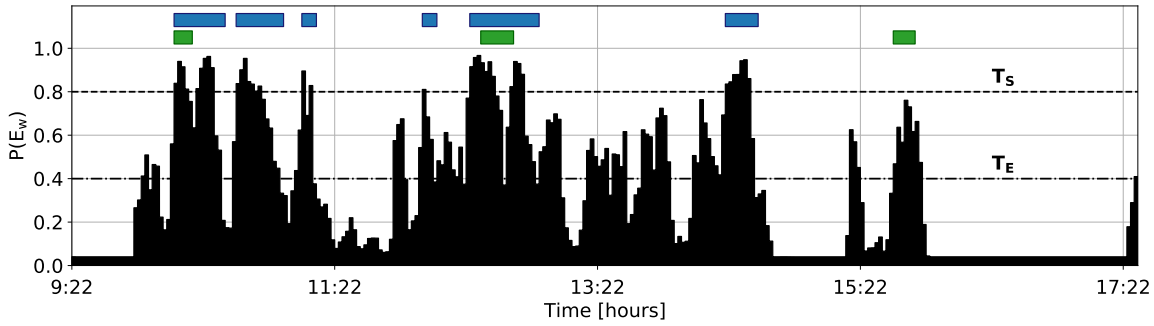


(b) Windowed eating classifier $P(E_w)$ with $T_S = 0.8$ and $T_E = 0.4$. 3 TP, 5 FP, 0 FN

Figure 3.9: Comparison between $P(E_d)$ and $P(E_w)$ showing a 100% reduction in the number of false positives in $P(E_d)$ from a $P(E_w)$ recording with high noise. Detections shown with blue bars (top) and GT shown with green bars (bottom).



(a) Daily pattern classifier $P(E_d)$ with $T = 0.1$. 2 TP, 1 FP, 1 FN



(b) Windowed eating classifier $P(E_w)$ with $T_S = 0.8$ and $T_E = 0.4$. 2 TP, 4 FP, 1 FN

Figure 3.10: Comparison between $P(E_d)$ and $P(E_w)$ where an FN episode (last ground truth event) is not recovered. Detections shown with blue bars (top) and GT shown with green bars (bottom).

Chapter 4

Conclusion

4.1 General Discussion

In this thesis, the novel concept of analyzing a day-length recording to detect episodes of eating was introduced as well as an RNN-based daily pattern classifier for this task. The results of the experiments in this work answer the three original questions:

1. Does analyzing the probability of eating in a daily context with a neural network improve eating episode classification?

   The 6-minute window signal from the window-based eating classifier had a low SNR and an abundance of noise due to the variety of gestures and wrist motion throughout the day that can resemble brief periods of eating. These gestures include grooming, shaving, brushing teeth, adjusting glasses, touching the face, and even food preparation, among a multitude of others. However, when the data was re-analyzed with a daily window this added more context and increased the SNR as shown in figures 3.6 - 3.10. The background response of the $P(E_w)$ signal was significantly filtered while retaining the peaks for meals, which caused improved SNR. This is evidence that a neural network can learn daily contextual clues and utilize them for better eating detection.

   Using an RNN for this task had many advantages. First, since the problem was inherently time series based, a recurrent neural network approach was needed for per-timestep classification. Second, the memory of RNN neurons enabled greater understanding of the daily context,

which was the focus of this work. Lastly, masking with time series data made it possible to only feed the classifier real data without creating unrealistic data with padded zeros or other values. Overall, an RNN-based classifier was ideal for the daily pattern classifier.

2. Can this approach reduce the number of false detections in eating episode detection?

   Our day-level classifier created more separation between background noise in the signal and data that corresponded to actual meals. This distinction enabled better detection of eating episodes by mitigating transient responses and leading to fewer false detections. Furthermore, this permitted a shift from a dual-threshold hysteresis approach to a single-threshold approach for post-processing.

3. How do the results of this approach compare to those from a window-based classifier?

   This approach exhibits similar or better results for every time and episode evaluation metric we measured when compared to the window-based classifier from [30]. Some performance metrics remained relatively unchanged, while others like the Acc_W, time TPR, and episode FP/TP increased by a larger margin. The largest improvement was a 53% reduction in the number of FP/TP eating episodes. The greatest decline was a 4% drop in episode TPR. All other metrics improved except time TNR, which remained constant between the two approaches.

4.2 Limitations

The most significant advantage of this approach is also its most significant limitation. The daily pattern classifier requires an entire day-length recording of data to operate. The daily frame of reference and contextual indicators are essential for this approach. As such, it can only work in a post-hoc fashion and is not designed for real-time use. The data used by the classifier is also not an end-to-end model that can process raw wrist motion IMU data. Data for this model must first be parsed with another model like the window-based eating model CNN approach from previous work [30] to generate a probability of eating signal. The daily pattern classifier and the windowed eating classifier are similar, but designed for two different approaches to the same problem. The windowed eating classifier is designed to be a real-time model and operate without the foresight of comprehensive patterns in the data. On the other hand, the daily pattern classifier leverages the latter for a model oriented for post-recording runtime.

Another limitation of this study is that the daily pattern classifier was only tested with the Clemson All-Day (CAD) Dataset [32] that includes over 350 day-length recordings from free-living participants. Albeit it is the largest dataset of its type known to us, the applicability to other datasets has not yet been investigated and may require changes to the model architecture or pre-processing.

4.3 Future Work

There are several opportunities for future research in this area. First and foremost, future work could look at incorporating both models (window-based and daily-context-based) into a single end-to-end encoder-decoder classifier. Although this model would be able to output a probability of eating from raw wrist motion data with the added benefit of daily context, it would also be an inherently post-hoc and not a real-time classifier. This is entirely due to the need for overall daily patterns that would not exist until an entire day of data existed as well. This classifier would also require more parameters, greater model complexity, and a much larger dataset to train end-to-end. A dataset of this size is not yet available.

The volatility of this model is also yet to be explored. Although only preliminary figures for quantifiable model volatility have been measured for the windowed eating classifier (see appendix A), it is apparent that there is a higher level of variability than expected. Whether this issue is arising from the amount of data, the nature of the classifier itself, how non-eating samples are selected for training, or another issue entirely remains to be seen. In any case, the volatility of this model should be evaluated more thoroughly. Furthermore, the same should be done for the daily pattern classifier as that is not investigated in this work. A comparison between the two would also be practical.

Finally, once the issue of model volatility is researched and controlled, grouping participants by eating behavior could be beneficial. These styles or behaviors of eating are also known as eating phenotypes. Currently, the problem is that model volatility is too high to use any metric to precisely measure performance difference between groups. A change in performance could be due to model volatility or better/worse grouping, but with current fluctuations there is no easy way to differentiate the two. Previous research has looked into using individual models [37] and large, full group models [30] for eating detection. This research would serve as a middle ground between the two by grouping people based on how they eat.

Appendices

Appendix A Quantifying Model Volatility

While conducting initial experiments for this work, it became apparent that the evaluation metrics changed drastically when the windowed eating classifier was retrained even when all other variables were held constant (e.g. random number generators). We realized that this issue arose because the windowed eating classifier was trained to optimize per-datum accuracy, where each datum was a 6-minute window of IMU data consisting of 5400 total data points from 6 axes. This caused variability in eating *episode* detection accuracy.

It is important to note that the thresholding algorithm used for the windowed eating classifier differs from that used with the daily pattern classifier. Instead of a single threshold, the method that was implemented used a dual-threshold hysteresis method with other heuristics. The 2 thresholds were the start threshold, T_S, and the end threshold, T_E. If the P(E) signal output by the model exceeded T_S, the start of an eating detection was marked. And, similarly when the probability signal fell below T_E, the end of an eating detection was marked. Based on previous work, T_S was set to 0.8 and T_E was set to 0.4 [30]. As for other heuristics, detections within a half window length of each other were merged and detections less than 1 minute were ignored entirely. The sequence of detections was recorded as a binary array used in subsequent time and episode metric evaluation.

We hypothesize that the variability can be attributed to the fact that the model was trained to optimize this per-window accuracy and therefore *weighted* accuracy through class balancing during training. The hyperparameters T_S and T_E used in the hysteresis thresholding method for eating episode detection were manually tuned to optimize per-episode accuracy. However, this left the model at a carefully-balanced point-of-inflection where any deviation in model training could result in a considerable difference in eating episode detection and consequently time and episode metrics. Figure A.1 shows an example situation where this could happen. If the model trained differently due to the model architecture, error surface, and gradient descent process, the probability of eating output by the model for a specific eating event could vary enough to no longer trigger T_S in the hysteresis thresholding method. Since the output of hysteresis thresholding is used to create the sequences compared for both time and episode metrics, this would adversely affect both. For example, as shown in figure A.2 if an interval that was classified as eating by one model due to narrowly exceeding the starting threshold, but fell below the threshold on another model the episode metrics would drastically vary.

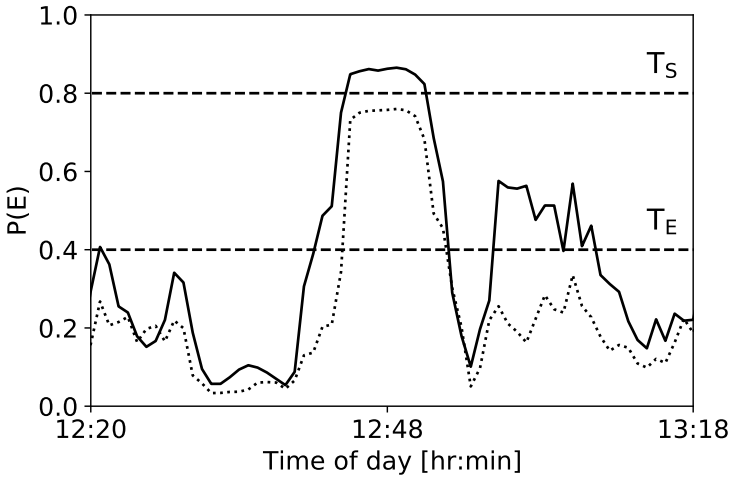
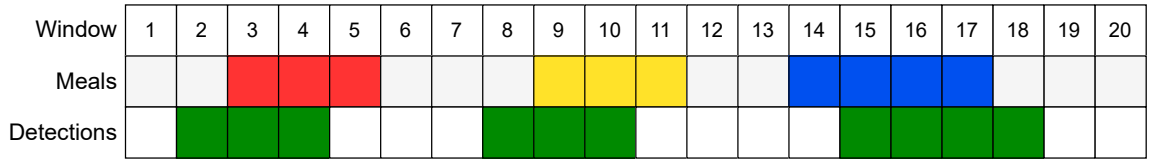
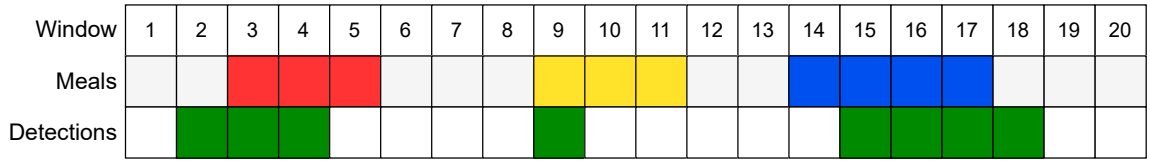


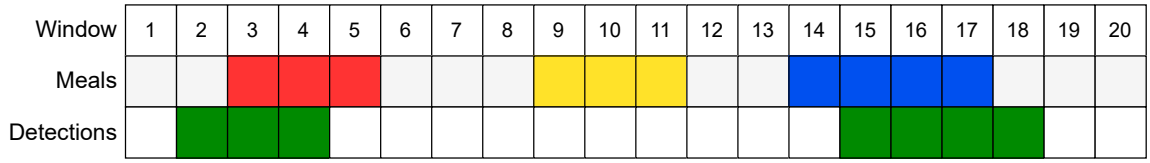
Figure A.1: Example of two P(E) results from the windowed eating model with hysteresis thresholds, T_S and T_E. The solid line P(E) would trigger an eating episode detection with the provided T_S and T_E, while the dotted line P(E) would not.



(a) All three meals detected and most windows correctly classified. Window TPR = 70% , episode TPR = 100%



(b) All three meals detected and only one window correctly classified in the second meal. Window TPR = 60%, episode TPR = 100%



(c) Second meal is missed entirely so only two meals detected. Window TPR = 50%, episode TPR = 70%

Figure A.2: Three examples indicating how window and episode metrics would change with varying amount of eating episode detection.

Metric	Average standard deviation per subject [%]
Window Acc_W	3.4
Time Acc_W	4.7
Time TPR	9.3
Time F_1 Score	9.1
Episode TPR	9.5
Episode F_1 Score	11.4

Table A.1: Windowed eating model volatility metrics for subsequent retraining.

To measure the volatility, the model was trained 30 times and a number of performance metrics were evaluated for each of the of 354 recordings in the CAD dataset. These metrics were: window weighted accuracy (Acc_W), time F_1 score, time true positive rate (TPR), time weighted accuracy (Acc_W), episode F_1 score, and episode TPR. The formulas for these metrics are defined in section 2.5. The fluctuation in these metrics was calculated by computing the average standard deviation per subject ("volatility") in the dataset after retraining 30 total times. The model was trained with 5-fold cross validation 30 times and then the aforementioned metrics were calculated on a subject-by-subject basis. For example, a trained model was used to calculate time F_1 score for subject 1, then subject 2, and so on. The standard deviation of each metric was calculated per subject and then averaged together for a final measure of volatility. These results are summarized in table A.1.

As seen in table A.1, the time weighted accuracy volatility was almost half the volatility of the other time and episode performance measures. Time and episode TPR and F_1 serve to explore the volatility in the other metrics that are evaluated for this dataset. Beyond time weighted accuracy, the volatility was higher than expected at around 10% regardless of the metric. This supports our hypothesis that the number of true positive eating episode detections is to blame since weighted accuracy is the only metric in this collection that considers all TPs, TNs, FPs, and FNs. In summary, the model has been optimized to a somewhat precarious point that causes widely differing results based on small changes during the stochastic training process.

There also appears to be further volatility coming from how the model is trained specifically. Window Acc_W variability was also measured and found to be 3.4%, which is much lower than other

metrics, but still higher than expected. We hypothesize that this per-subject volatility in window weighted accuracy arises from how non-eating and eating samples are selected for training, which leads to highly variable training and therefore volatility in time and episode metric measurement.

Appendix B Windowed Eating Model Results Replication

A replication experiment was performed for the windowed eating classification model from previous work [30]. The wrist motion data was sliced into $W = 6$ minute windows (5400 data) with $s = 15$ second (225 data) stride between them. Before processing, acceleration bias was removed with a trend filter, data was smoothed with a Gaussian filter with a window length of 15 data and $\sigma = 10.0$, and the data was z-score normalized with global trended standard deviations for each axis of data and all zero means. For training and testing, 5-fold cross validation was used. The classifier was trained for 30 epochs and the model with the best training accuracy was saved and used for testing. The windows of data used for testing were created with $W = 6$ minutes (5400 data) and $s = 1$ datum.

Time and episode metrics were measured after processing the model output with a dual-threshold hysteresis method. The approach was set up to mirror that in [30] and the two thresholds were set to $T_S = 0.8$ and $T_E = 0.4$ to match as well. The results of this experiment are reported in tables B.1 and B.2. Overall, the replicated time metrics were 1-4% lower than reported and the episode TPR was 2% lower. FP/TP also increased by 6%. Comparing the replicated results to the daily pattern classifier results there is only a 2% decline in episode TPR with a slightly larger 56% decrease in FP/TP at the chosen threshold $T = 0.1$.

All time and episode evaluation metrics measured for the daily pattern classifier are shown in

Model	TPR (%)	TNR (%)	F_1 (%)	Precision (%)	Acc_W (%)
Windowed Classifier (Reported in [30])	69	93	48	36	80
Windowed Classifier (Replicated)	68	92	44	33	80
Daily Pattern Classifier	78	93	50	37	85

Table B.1: Time evaluation metrics comparing reported and replicated results for the windowed eating classifier. Results for the daily pattern classifier (this work) are also shown for reference.

Model	TPR (%)	FP/TP
Windowed Classifier (Reported in [30])	89	1.7
Windowed Classifier (Replicated)	87	1.8
Daily Pattern Classifier	85	0.8

Table B.2: Episode evaluation metrics comparing reported and replicated results for the windowed eating classifier. Results for the daily pattern classifier are also shown for reference.

table B.3. Similarly, these results are shown for the windowed eating classifier replication experiment in table B.4.

The effect of various values of T_S was also investigated to compare with reported figures and the results of changing T with the daily pattern classifier. These results are shown in figure B.1. Overall, similar deviation can be seen between the reported and replicated results. The replicated results exhibit lower episode true positive rates for T_S values below 0.75 and higher FP/TP ratios for T_S values above 0.75. Still, for every episode TPR, the daily pattern classifier offers lower FP/TP than the windowed eating classifier. It is important to note that these values shown are at a T_E threshold of 0.3 to match those reported in a similar figure from [30]. The results from tables B.1 and B.2 on page 59 are with $T_S = 0.8$ and $T_E = 0.4$.

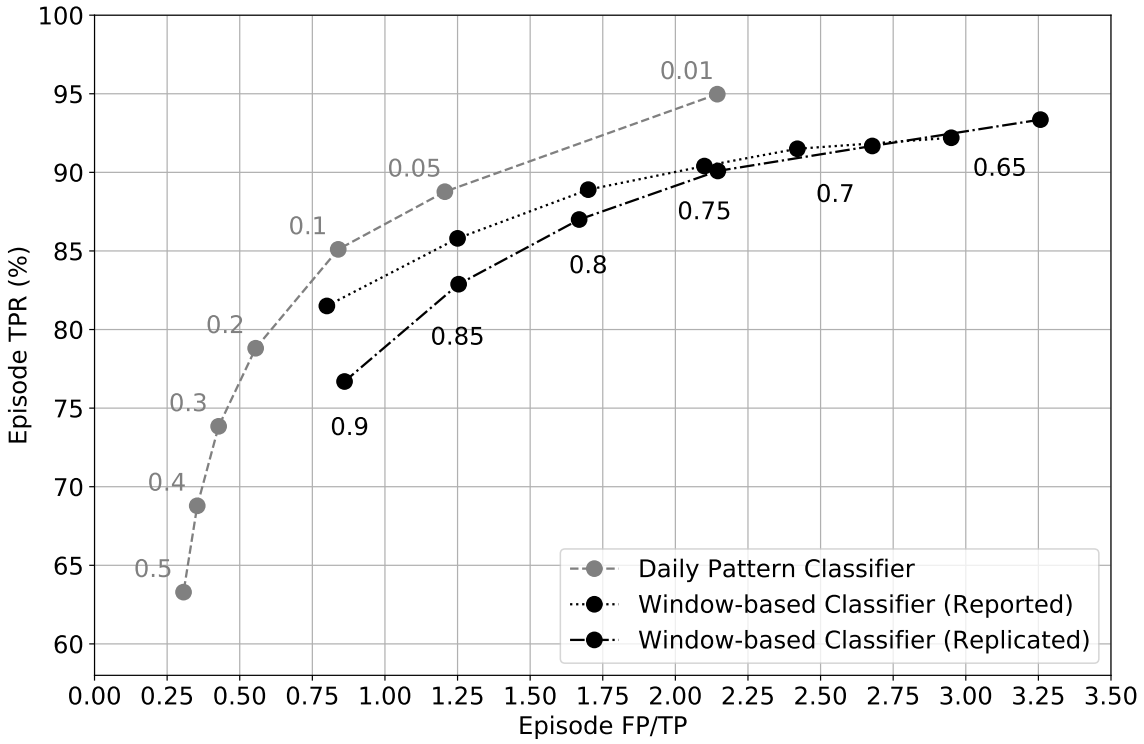


Figure B.1: Effect of threshold T_S (number below points) on the window-based classifier with T_E = 0.3 reported in [30] and replicated. The effect of threshold T (number next to points) on episode TPR and FP/TP for the daily pattern model is also shown for reference.

	Time Metrics (%)					Episode Metrics		
Threshold (T)	**Acc$_W$**	**TPR**	**TNR**	**F$_1$**	**Precision**	**TPR (%)**	**F$_1$ (%)**	**FP/TP**
0.01	81.40	90.89	71.90	26.34	15.42	94.97	47.76	2.14
0.02	84.21	88.00	80.42	32.80	20.20	92.78	52.05	1.77
0.03	85.19	85.98	84.40	37.09	23.69	91.17	55.43	1.52
0.04	85.58	84.36	86.80	40.23	26.47	89.89	58.04	1.34
0.05	85.71	82.97	88.45	42.69	28.80	88.77	60.10	1.21
0.06	85.70	81.74	89.66	44.69	30.82	87.81	61.80	1.10
0.07	85.62	80.63	90.61	46.36	32.60	86.94	63.22	1.02
0.08	85.50	79.62	91.37	47.78	34.20	86.15	64.43	0.95
0.09	85.34	78.69	92.00	49.00	35.65	85.42	65.46	0.89
0.1	85.16	77.79	92.53	50.06	36.99	84.71	66.33	0.84
0.15	84.03	73.71	94.35	53.73	42.36	81.55	69.33	0.66
0.2	82.78	70.11	95.44	55.81	46.45	78.81	70.93	0.55
0.25	81.53	66.86	96.20	57.03	49.81	76.33	71.76	0.48
0.3	80.27	63.77	96.77	57.66	52.71	73.84	72.00	0.43
0.35	79.01	60.78	97.23	57.88	55.33	71.37	71.86	0.39
0.4	77.70	57.79	97.61	57.71	57.73	68.79	71.34	0.35
0.45	76.35	54.76	97.94	57.21	59.98	66.10	70.50	0.33
0.5	74.90	51.58	98.23	56.33	62.11	63.29	69.38	0.31
0.55	73.34	48.19	98.48	55.02	64.17	60.28	67.96	0.29
0.6	71.63	44.54	98.72	53.23	66.19	56.99	66.17	0.27
0.65	69.73	40.54	98.93	50.82	68.16	53.26	63.88	0.26
0.7	67.58	36.03	99.14	47.60	70.17	48.97	60.94	0.24
0.75	65.09	30.86	99.33	43.25	72.33	43.94	57.16	0.23
0.8	62.13	24.75	99.52	37.16	74.53	37.57	51.69	0.21

Table B.3: Time and episode evaluation metrics for the daily pattern classifier at various threshold values T

Threshold		**Time Metrics (%)**					**Episode Metrics**		
T_S	T_E	**Acc**$_W$	**TPR**	**TNR**	**F**$_1$	**Precision**	**TPR (%)**	**F**$_1$ **(%)**	**FP/TP**
0.65	0.3	80.79	77.96	83.62	33.65	21.56	93.35	37.76	3.26
0.7	0.3	80.93	76.12	85.74	35.92	23.65	91.68	42.31	2.68
0.75	0.3	81.08	74.34	87.82	38.58	26.27	90.09	47.36	2.15
0.8	0.3	80.67	71.50	89.85	41.08	29.10	87.00	52.71	1.67
0.8	0.4	80.02	68.18	91.87	44.04	32.90	86.81	50.37	1.85
0.85	0.3	80.05	68.21	91.90	44.06	32.92	82.88	58.04	1.25
0.9	0.3	78.32	62.59	94.04	46.77	38.01	76.69	63.02	0.86

Table B.4: Time and episode evaluation metrics for the windowed eating classifier at various threshold values T_S, T_E

Appendix C Normalization Methods for Windowed Model

Early experiments for this work also examined the effect of various normalization methods for the raw wrist motion data on the windowed eating model [30] performance. Normalization methods tested included z-score standardization, min-max normalization, quantile normalization, and mean normalization. Normalization was performed per-axis. Additionally, variants of z-score standardization were tested including global standardization and per-file standardization. Because of the widely varying distribution of values and some extreme outliers, alternative minimum and maximum values were used for min-max normalization and mean normalization. The histograms shown in figure C.1 on page 65 give insight into how the modified values for the means and standard deviations were chosen. The exact values chosen are shown with the results in table C.1.

The equation for calculating each new value x' with z-score standardization is shown in equation C.1, where x is the original value and μ and σ are the mean and standard deviation of the original values respectively. Similarly, equation C.2 shows the equation for min-max normalization where min and max are the minimum and maximum of the original values respectively. Equation C.3 displays the formula for computing mean normalized values. And lastly, quantile normalization is a way of making two distributions have identical statistical properties. The `QuantileTransformer` function from the `preprocessing` module of the `sklearn` library was used to accomplish this in this experiment.

$$x' = \frac{x - \mu}{\sigma} \tag{C.1}$$

$$x' = \frac{x - \min}{\max - \min} \tag{C.2}$$

$$x' = \frac{x - \mu}{\max - \min} \tag{C.3}$$

The best training accuracy with each method and the loss at the end of training is shown in table C.1. These specific methods and values were chosen based on other experiments to find the best values for each normalization method. Overall, global z-score standardization yielded the best training accuracy and lowest training loss.

It is important to recognize that these are *training* metrics and this is a limitation of this preliminary investigation. Only select normalization methods were evaluated completely. The

Method	**Best Training Accuracy**	**Best Training Epoch**	**Training Loss After 30 Epochs**
Global standardization	82.6%	28	0.42
File standardization	82.5%	29	0.43
Quantile normalization (n = 10000)	81.7%	29	0.43
Mean normalization (±0.15, ±50)	81.5%	29	0.44
Min-max normalization (±0.15, ±20)	77.7%	30	0.49

Table C.1: Training results for various normalization methods. Numbers with ± indicate minimum and maximum values used for accelerometer and gyroscope axes in order of appearance.

time and episode evaluation metrics computed for these select few normalization methods showed a standard deviation of only around 1%. As a result of this limited experiment, z-score standardization was chosen as the the default normalization method.

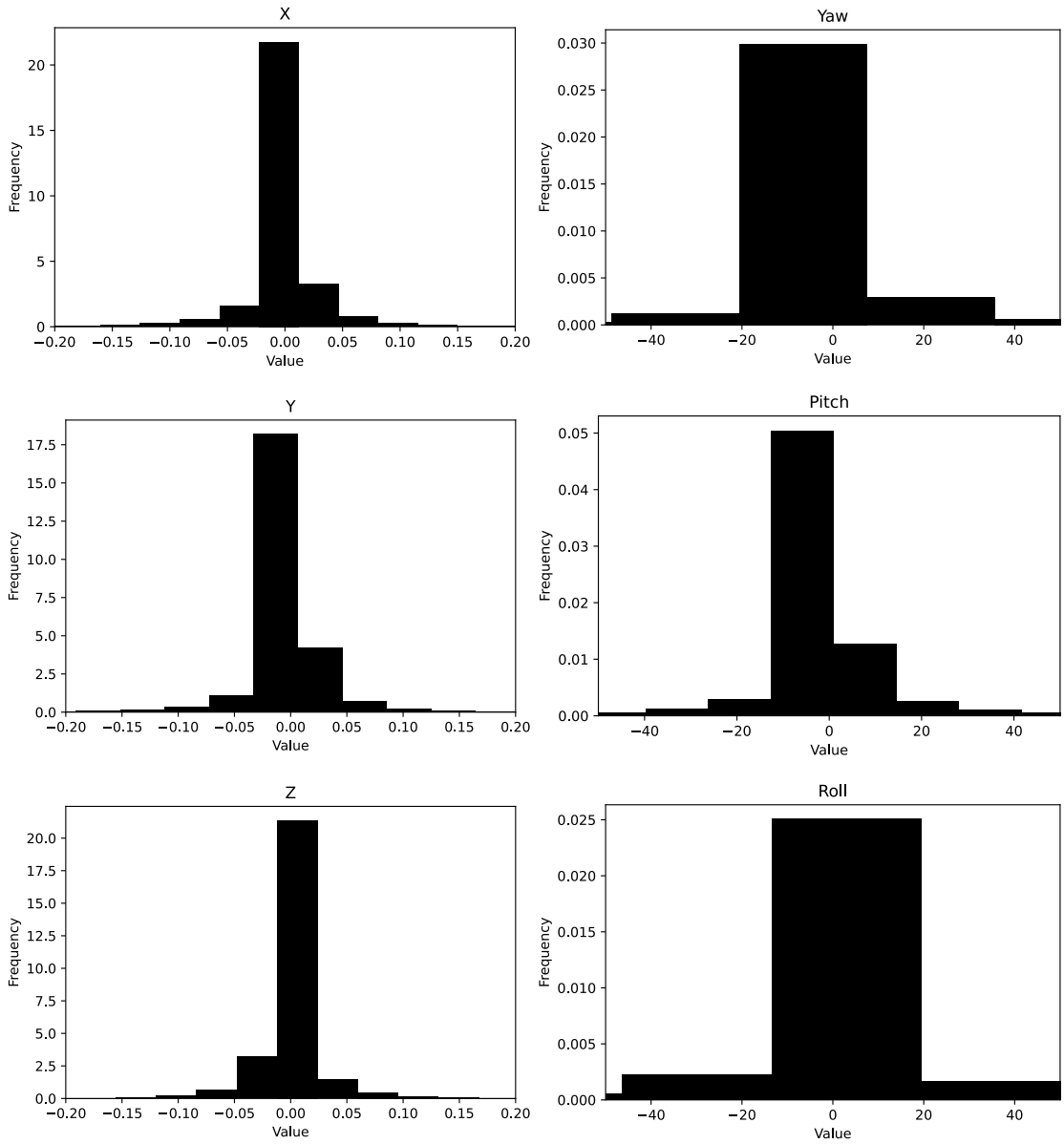


Figure C.1: Histograms of all values for each of the 6 axes of wrist motion data after acceleration trend filter was applied, $N = 100$.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, S. Ghemawat, I. Goodfellow, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] O. Amft, M. Stäger, P. Lukowicz, and G. Tröster. Analysis of chewing sounds for dietary monitoring. In *International Conference on Ubiquitous Computing*, pages 56–72. Springer, 2005.

[3] Apple. Apple Watch Series 6. https://www.apple.com/apple-watch-series-6/, 2020.

[4] Apple. watchOS 8 preview. https://www.apple.com/watchos/watchos-preview/, 2021.

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

[6] C. M. Champagne, G. A. Bray, A. A. Kurtz, J. B. R. Monteiro, E. Tucker, J. Volaufova, and J. P. Delany. Energy intake and energy expenditure: a controlled study comparing dietitians and non-dietitians. *Journal of the American Dietetic Association*, 102(10):1428–1432, 2002.

[7] Y. Dong, A. Hoover, and E. Muth. A device for detecting and counting bites of food taken by a person during eating. In *2009 IEEE International Conference on Bioinformatics and Biomedicine*, pages 265–268. IEEE, 2009.

[8] Y. Dong, A. Hoover, J. Scisco, 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 (ESCS)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2011.

[9] Y. Dong, J. Scisco, M. Wilson, E. Muth, and A. Hoover. Detecting periods of eating during free-living by tracking wrist motion. *IEEE Journal of Biomedical and Health Informatics*, 18(4):1253–1260, 2013.

[10] A. Doulah, T. Ghosh, D. Hossain, M. H. Imtiaz, and E. Sazonov. "Automatic ingestion monitor version 2"—a novel wearable device for automatic food intake detection and passive capture of food images. *IEEE Journal of Biomedical and Health Informatics*, 2020.

[11] M. Farooq and E. Sazonov. A novel wearable device for food intake and physical activity recognition. *Sensors*, 16(7):1067, 2016.

[12] D. S. Freedman and B. Sherry. The validity of BMI as an indicator of body fatness and risk among children. *Pediatrics*, 124(Supplement 1):S23–S34, 2009.

[13] Y. Gao, N. Zhang, H. Wang, X. Ding, X. Ye, G. Chen, and Y. Cao. iHear food: eating detection using commodity bluetooth headsets. In *2016 IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pages 163–172. IEEE, 2016.

[14] N. Gruber and A. Jockisch. Are GRU cells more specific and LSTM cells more sensitive in motive classification of text? *Frontiers in Artificial Intelligence*, 3(40):1–6, 2020.

[15] C. M. Hales, M. D. Carroll, C. D. Fryar, and C. L. Ogden. Prevalence of obesity and severe obesity among adults: United States, 2017–2018. *Hyattsville, MD: National Center for Health Statistics*, Feb 2020.

[16] H. Kalantarian, N. Alshurafa, and M. Sarrafzadeh. A survey of diet monitoring technology. *IEEE Pervasive Computing*, 16(1):57–65, 2017.

[17] K. Kyritsis, C. Diou, and A. Delopoulos. Food intake detection from inertial sensors using LSTM networks. In *International Conference on Image Analysis and Processing*, pages 411–418. Springer, 2017.

[18] K. Kyritsis, C. Diou, and A. Delopoulos. Modeling wrist micromovements to measure in-meal eating behavior from inertial sensor data. *IEEE journal of biomedical and health informatics*, 23(6):2325–2334, 2019.

[19] K. Kyritsis, C. Diou, and A. Delopoulos. A data driven end-to-end approach for in-the-wild monitoring of eating behavior using smartwatches. *IEEE Journal of Biomedical and Health Informatics*, 25(1):22–34, 2020.

[20] B. Lauby-Secretan, C. Scoccianti, D. Loomis, Y. Grosse, F. Bianchini, and K. Straif. Body fatness and cancer—viewpoint of the IARC working group. *New England Journal of Medicine*, 375(8):794–798, 2016.

[21] Y. Y. Luktuke. *Segmentation and recognition of eating gestures from wrist motion using deep learning*. PhD thesis, Clemson University, May 2020.

[22] O. Makeyev, E. Sazonov, S. Schuckers, P. Lopez-Meyer, T. Baidyk, E. Melanson, and M. Neuman. Recognition of swallowing sounds using time-frequency decomposition and limited receptive area neural classifier. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 33–46. Springer, 2008.

[23] M. Narici, G. De Vito, M. Franchi, A. Paoli, T. Moro, G. Marcolin, B. Grassi, G. Baldassarre, L. Zuccarelli, G. Biolo, et al. Impact of sedentarism due to the COVID-19 home confinement on neuromuscular, cardiovascular and metabolic health: Physiological and pathophysiological implications and recommendations for physical and nutritional countermeasures. *European journal of sport science*, pages 1–22, 2020.

[24] D. T. Nguyen, E. Cohen, M. Pourhomayoun, and N. Alshurafa. SwallowNet: Recurrent neural network detects and characterizes eating patterns. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 401–406. IEEE, 2017.

[25] F. Q. Nuttall. Body mass index: obesity, BMI, and health: a critical review. *Nutrition today*, 50(3):117, 2015.

[26] World Health Organization. *World Health Statistics 2018: Monitoring Health for the SDGs, Sustainable Development Goals*. Nonserial Publications. World Health Organization, 2018.

[27] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[28] K. J. Rothman. BMI-related errors in the measurement of obesity. *International journal of obesity*, 32(3):S56–S59, 2008.

[29] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

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

[31] S. P. Sharma and S. Hoover. The challenge of metrics in automated dietary monitoring as analysis transitions from small data to big data. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2647–2653. IEEE, 2020.

[32] S. P. Sharma, P. Jasper, E. Muth, and A. Hoover. The impact of walking and resting on wrist motion for automated detection of meals. *ACM Transactions on Computing for Healthcare*, 1(4):1–19, 2020.

[33] S. Stankoski, M. Jordan, H. Gjoreski, and M. Luštrek. Smartwatch-based eating detection: Data selection for machine learning from imbalanced data with imperfect labels. *Sensors*, 21(5):1902, 2021.

[34] P. A. Tataranni, I. T. Harper, S. Snitker, A. Del Parigi, B. Vozarova, J. Bunt, C. Bogardus, and E. Ravussin. Body weight gain in free-living Pima Indians: effect of energy intake vs expenditure. *International journal of obesity*, 27(12):1578–1583, 2003.

[35] J. Ubrani, M. Shirer, and R. Llamas. Consumer enthusiasm for wearable devices drives the market to 28.4% growth in 2020, according to IDC, Mar 2021.

[36] Y. Wang, M. A. Beydoun, J. Min, H. Xue, L. A. Kaminsky, and L. J. Cheskin. Has the prevalence of overweight, obesity and central obesity levelled off in the United States? Trends, patterns, disparities, and future projections for the obesity epidemic. *International journal of epidemiology*, 49(3):810–823, 2020.

[37] W. Wei. *Individualized wrist motion models for detecting eating episodes using deep learning.* PhD thesis, Clemson University, May 2021.

[38] World Health Organization. Obesity and overweight. https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight, Jun 2021.

[39] World Health Organization et al. *Noncommunicable diseases: Progress monitor 2020.* World Health Organization, 2020.

[40] R. Zhang and O. Amft. Free-living eating event spotting using EMG-monitoring eyeglasses. In *2018 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pages 128–132. IEEE, 2018.