Using Transfer Learning to Train Individualized Models to Detect Eating Episodes from Daily Wrist Motion

---

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
Cole Younginer
August 2021

Accepted by:
Dr. Adam Hoover, Committee Chair
Dr. Melissa Smith
Dr. Jon Calhoun

Abstract

This thesis considers the problem of detecting periods of eating in free-living conditions by analyzing wrist motion data collected using sensors embedded within a typical smartwatch. Previous work by our research group included the collection of a dataset containing 354 days of recorded wrist motion data from 351 different people (approximately one day of data per person) [42]. A machine learning model was then trained to classify this wrist motion data as either eating or non-eating [40]. We refer to this model as the group model. Subsequent work in our research group collected approximately ten days of data each for eight new individuals and trained a model for each person solely using their own data [51]. We refer to these models as individual models. It was observed that, in most cases, the individual models outperformed the group model when evaluating the data of their corresponding individual, but at the cost of requiring each individual to collect two weeks of additional data. The novelty of this work is using transfer learning to leverage features learned within the group model and apply them to new individual models to further increase performance and possibly reduce the amount of individual data needed.

Two datasets were used in this work. The first was the Clemson All Day (CAD) dataset, which contains 354 days of recorded wrist motion data from 351 different participants (approximately one day of data per participant). The CAD dataset includes a total of 4,680 hours of data, including 1,063 meals. The second dataset used was the Multiday dataset, which is comprised of at least ten days of free-living wrist motion data each for eight individuals. Both datasets were pre-processed using smoothing and normalization techniques. Training samples were then generated using a sliding window approach with a window size of six minutes.

All group, individual, and transfer learning models evaluated in this work utilized an identical convolutional neural network (CNN) architecture. For a given window, the classifier generated a value that represented the probability of eating ($P(E)$) in the window. Entire days of wrist motion

data were passed to the network to produce a continuous $P(E)$ sequence for an entire day. This sequence was processed using a dual thresholding technique to locate predicted segments of eating within the recording.

In our results, the transfer learning model achieved an eating episode true positive rate (TPR) of 81% with a false positive per true positive ratio (FP/TP) of 1.40. Compared to the individual model, this was a 6% decrease in episode TPR but a 43% improvement in FP/TP. The transfer learning model showed a time weighted accuracy (Acc_W) of 80%, which was only a 1% decrease relative to the individual model. After removing an outlier from the Multiday dataset and rerunning our experiments, the transfer learning model showed an episode TPR of 86% with an FP/TP of 1.34. Compared to the individual model, this was only a 3% decrease in TPR and a 46% improvement in FP/TP. By excluding the outlier, the transfer learning model also showed an 83% Acc_W, which was a 1% increase relative to the individual model. Furthermore, the transfer learning model was able to reduce training times by 12% compared to the individual model. In conclusion, we were able to find evidence that transfer learning could be utilized in order to improve individualized eating detection models by increasing weighted accuracy and decreasing false detections.

Dedication

This thesis is dedicated to my parents, Susan and Woody Younginer, who have shown me unconditional love and support throughout my life.

Acknowledgments

I would like to first thank my research advisor and committee chair, Dr. Adam Hoover. His professional knowledge and mentorship have helped me grow as an engineer and researcher. This work would not have been possible without his guidance and expertise.

I would next like to thank Dr. Melissa Smith for helping me start my graduate journey and for serving on my defense committee. I would also like to thank Dr. Jon Calhoun for serving on my defense committee.

I am also grateful to Dr. Harlan Russell for his guidance, as well as the other Clemson University faculty and staff who have contributed to my undergraduate and graduate education. Lastly, I would like to thank my friends and research group members who have provided invaluable assistance.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

This thesis considers the problem of detecting periods of eating by tracking wrist motion using sensors embedded within a typical smartwatch. Previous work by our research group included the collection of a dataset containing 354 days of recorded wrist motion data from 351 different people (approximately 1 day of data per person) [42]. A machine learning model was then trained to classify this wrist motion data as either eating or non-eating [40]. We refer to this model as the group model. Subsequent work in our research group collected approximately 10 days of data each for 8 new individuals and trained a model for each person solely using their own data [51]. We refer to these models as individual models. It was observed that, in most cases, the individual models outperformed the group model when evaluating the data of their corresponding individual.

This thesis considers the use of transfer learning to leverage features learned within the group model and apply them to new individual models to further increase performance. By using the group model as a baseline for continued training with individual data, we hypothesize that an improvement in eating episode detection could be achieved while reducing the number of false detections. We compare the performance of our transfer learning model against both the group model and individual models by evaluating each model on all individual datasets. We also compare the training time of our transfer learning models to those of the individual models in order to determine if transfer learning could be used to speed up the training process.

The remainder of this chapter first covers the background for this thesis. Section 1.2.1

examines the global obesity epidemic and its repercussions that are the primary motivations behind this work. Section 1.2.2 reviews the field of mobile health (mHealth) and its tools that allow people to track data related to their health. Section 1.3 describes the Shimmer3, a wearable wrist motion tracking device that was used to collect the datasets for the experiments outlined in this thesis. Section 1.2.4 reviews several topics related to deep learning, including neural networks, convolutional neural networks (CNNs), and transfer learning. After examining the background for this work, Section 1.3 explores related works in the field of eating detection and transfer learning. Finally, Section 1.4 briefly covers the novelty of this thesis and the questions that this work seeks to answer.

1.2 Background

1.2.1 Obesity

Obesity is a widespread and complex health issue. It is a medical disorder involving excessive fat accumulation that exposes a person to increased risk of diseases and other health problems such as heart disease, diabetes, and many cancers. Obesity is commonly diagnosed using the body mass index (BMI), which uses the ratio of a person's weight and the square of their height (kg/m^2). A person is considered overweight if their BMI exceeds 25 and obese if their BMI exceeds 30.

Global trends indicate an alarming increase in obesity, which has tripled since 1975. It was reported by the World Health Organization (WHO) that, as of 2016, more than 1.2 billion adults (ages 20 and older) worldwide were overweight and over 650 million were obese [55]. In addition, over 340 million children (ages 5-19) worldwide were overweight or obese in 2016 [55]. These figures have started to increase more rapidly with each passing year. In the United States, a recent data brief from the Centers for Disease Control and Prevention (CDC) highlighted the obesity epidemic, finding that, in 2018, 42.4% of all U.S. adults were obese [20]. Another recent study has predicted this progression in obesity rates will worsen, and that by the year 2030 approximately 50% of all adults living in the United States will be obese [50]. A plot illustrating the obesity rates in the United States for both children and adults over the last two decades is shown in Figure 1.1. In this short amount of time, it can be observed that adult obesity rates have climbed over 12% and childhood obesity rates have risen more than 5%. This disconcerting trend has the potential to severely impact public health initiatives and further drive up the costs of healthcare in our country.

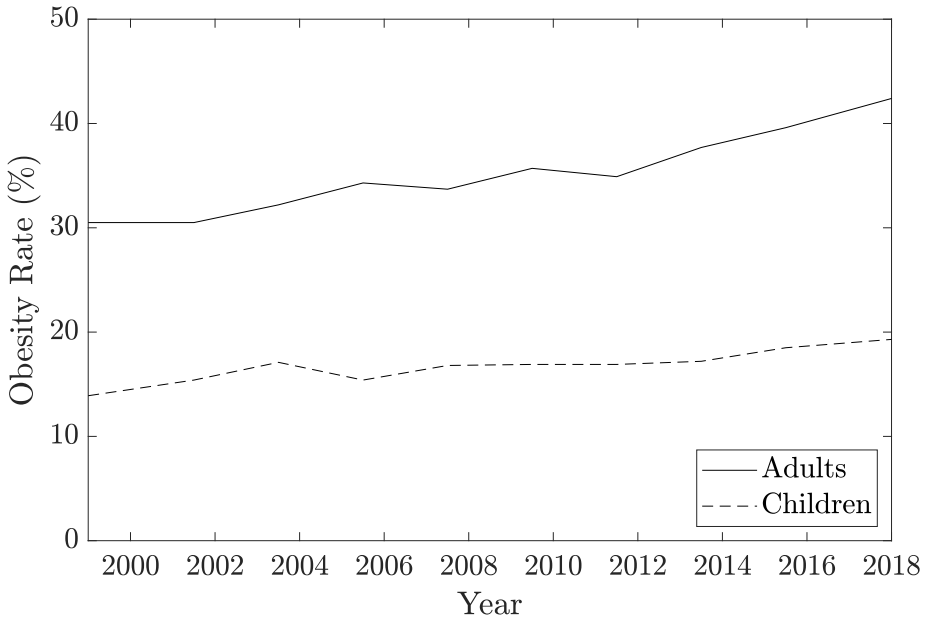


Figure 1.1: Obesity rates in the United States from 1999 to 2018, adapted from [20].

The primary factors associated with these increases in obesity rates are consumption of foods high in sugars and fat and increased sedentarism both at home and at work. Urbanization and the growth of public transportation have also led to decreases in everyday activity levels. The recent COVID-19 quarantines have compounded several of these factors. By forcing people to confine themselves within their homes, overeating and sedentarism have increased. In addition, the quarantines have limited access to conventional exercise venues such as public gyms [31]. It has also been shown that individuals who suffer from obesity and its associated diseases experience more severe symptoms after contracting COVID [10].

Previous studies have shown that genetic, physiologic, environmental, psychological, social, and economic factors can all have an influence on predisposition to obesity; however, obesity is highly preventable in most cases [56, 55]. Changes in weight occur due to an imbalance in energy intake and energy expenditure, typically measured in units of calories. If energy expenditure is higher than energy intake, the human body will burn stores of fat in order to keep up with energy demands. Conversely, if energy intake is higher than energy expenditure, the human body will convert the excess energy into fat to use in case of future energy shortages. Although physical activity and exercise can increase energy expenditure to help balance out a high energy intake,

the most important quantifiable aspect of effectively monitoring and ensuring weight loss is the measurement of calorie intake [48].

1.2.2 mHealth

Modern medicine has considerably improved both quality of life and life expectancy in recent history. With these improvements, noncommunicable diseases (NCDs) have now become the leading causes of death around the world. According to a 2021 survey by the WHO, NCDs are responsible for over 41 million deaths each year, which account for approximately 71% of all deaths worldwide [54]. The four most deadly NCDs were found to be cardiovascular diseases, cancers, respiratory diseases, and diabetes. Obesity has been proven to be a contributing risk factor to the development of all of these diseases [55]. This suggests that many cases of NCDs are preventable with a healthy lifestyle; however, modern healthcare is mainly reactive. Medical care is only administered when a patient seeks out a professional or when severe health concerns arise. In order to halt the onset of NCDs, more proactive methods of monitoring personal health are necessary. This is where the burgeoning field of mobile health, or mHealth, may provide assistance.

mHealth refers to the practice of medicine and monitoring of personal health using mobile devices such as smartphones and smartwatches. Two popular mHealth devices are the Apple Watch and Fitbit (shown in Figure 1.2), which together formed a market share of nearly 57% in North America in 2020 [7]. These devices make health monitoring more accessible to the average user by utilizing several automatic biometric sensors. For example, the Apple Watch includes a heart rate monitor that can automatically detect irregular rhythms, an electrocardiogram (ECG), and an IMU that can detect falls [4]. The Fibit also has several useful sensors that can automatically monitor blood oxygen saturation, skin temperature, and breathing rate [17]. These devices are most commonly used to monitor physical activity. By tracking activity, they are able to automatically provide energy expenditure estimations.

Although current mHealth devices are able to automatically track energy expenditure, they are unable to automatically track energy intake. This must be done manually using self-monitoring methods such as food journaling. An extensive review on self-monitoring found a strong correlation between self-monitoring and weight loss [6]. This same survey found that more frequent self-monitoring led to increased weight loss; however, a decrease in self-monitoring frequency was observed in longer studies. This suggests that long-term self-monitoring is difficult to maintain due

(a) Apple Watch [4] (b) Fitbit [17]

Figure 1.2: The Apple Watch and Fitbit, two of the most popular smartwatches at the time of writing.

to the tedious nature of self-reporting. In addition, self-monitoring can be prone to reporting bias. A previous study found that obese individuals are more likely to underreport energy intake and overreport physical activity [26].

The objective of the emerging field of automated dietary monitoring (ADM) is to produce a device capable of automatically tracking energy intake in order to mitigate the drawbacks of self-monitoring. Our research group has examined the use of wrist motion data to create an ADM system that can be integrated with a typical smartwatch [40, 28, 51, 34]. Our goal is to help with the treatment of obesity and eating disorders. A detailed analysis of our group's previous research and other works related to ADM is provided in Section 1.3.

1.2.3 Shimmer3 Device

The wrist motion data for the CAD and Multiday datasets were collected using the Shimmer3 wearable device, shown in Figure 1.3 [40, 51]. The Shimmer3 contains an inertial measurement unit, or IMU, which uses both an accelerometer and gyroscope in order to track motion. The accelerometer measures linear acceleration across the x, y, and z axes, while the gyroscope measures angular velocity about those same axes. This results in a multivariate time-series with six total axes of data. The Shimmer3 recorded data at a rate of 15.06 Hz, though this was downsampled to 15 Hz

Figure 1.3: The Shimmer3 wearable IMU device [44]

during post-processing.

Data collection using the Shimmer3 device was a straightforward process. First the device was turned on using the small switch on its side. This caused one of the LEDs on the device to turn green to indicate that it was running. The large orange button on the front of the device was then held down for approximately five seconds in order to begin the data collection process, which produced a regular flash on the second LED. Data was saved to an embedded micro SD card. In order to log start and end times for meals, the button on the front was quickly pressed to produce a timestamp in the data. This also produced a quick flash on the second LED. Once recording was finished, the button was held down again for 5 seconds in order to halt data collection. The Shimmer3 was then connected to a dock using a port located on the bottom of the device. Next, the dock was connected via USB to a computer to download the data in CSV format using the Consensys software package specifically developed for the Shimmer3 device. The complete data collection process for the CAD dataset and Multiday dataset are described in [42] and [51], respectively.

1.2.4 Deep Learning

Deep learning is a subset of the broad field of machine learning, which seeks to apply anatomical concepts of the human brain to mathematical models. Specifically, deep learning involves training these mathematical models, referred to as neural networks, to learn information provided

through examples. Many different deep learning architectures have been created, including deep neural networks (DNNs), recurrent neural networks (RNNs), and convolutional neural networks (CNNs). These architectures have made ground-breaking developments in many fields, including computer vision, machine translation, advertisement, financial forecasting, and healthcare.

There are four different categories of deep learning: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. In supervised learning, training samples that are provided to a neural network are labeled with their correct output, or ground truth. This allows the neural network to compare its output to the ground truth so it can better fit itself to the training data. Unsupervised learning takes the opposite approach and provides neural networks with training samples that have no ground truth. Unsupervised learning is used in order to reveal features or patterns within data that may be difficult to discern in their raw form. Semi-supervised learning combines supervised learning and unsupervised learning by using mostly unlabeled training data with a small number of labeled samples that can help identify important features. Finally, reinforcement learning is used to generate a set of instructions that maximize a reward function. For example, reinforcement learning can be used in order to train a neural network to play games such as chess.

The remainder of this section will focus on supervised learning for classification tasks, since that is what was used in the experiments of this thesis. This section first examines the basics of neural networks. Next, the convolutional neural network, or CNN, is reviewed. Finally, the concept of transfer learning is explored.

1.2.4.1 Neural Networks

Neural networks are mathematical models that are trained to learn information from examples. Neural networks consist of neurons, or nodes, that each have one or more inputs, an activation function, and a single output. When the output of one neuron is connected as an input to another neuron, this value is first multiplied by a value known as a weight. In addition, each neuron has a constant value added to its inputs called a bias. These bias values and weighted connections between neurons are what define a neural network. Once all weighted inputs and the bias value for a neuron have been added together, they are passed to the activation function to produce an output.

Mathematically, a neuron can be represented as:

$$y = f(\sum_{i=1}^{n} x_i w_i + b) \tag{1.1}$$

where n represents the number of inputs to the neuron, x represents the vector of input values, w represents the vector of weights, b represents the bias value, and $f(\cdot)$ represents the activation function of the neuron. The activation function mathematically relates the inputs of a neuron to its output. Many different activation functions have been developed for neural networks. The activation functions used in this thesis include the rectified linear unit (ReLU) and sigmoid functions, which are covered in more detail in Chapter 3.

A group of neurons that have the same input, activation function, and output are collectively referred to as a layer in the neural network. The size of a neural network is typically described in terms of layers rather than neurons. A neural network must contain an input layer and an output layer. Any layers placed between the input layer and output layer are referred to as hidden layers. Hidden layers can take many different forms. Examples include convolutional layers, pooling layers, and fully-connected layers. These types of hidden layers are examined more thoroughly in Section 1.2.4.2. Figure 1.4 shows a small neural network containing one fully-connected hidden layer. With deep learning, it is common to have many hidden layers present in a network.

Training a neural network is an iterative optimization process. During each training iteration, inputs are passed to the network to learn from. Each input is combined with the weights and biases in the network to produce an output. The error, or loss, between the network output and its desired output is then calculated using a function called the loss function to observe how well the network performed. The loss is then used to update all of the trainable parameters of the network, which include weights and biases. A lower loss value indicates that the network performed well and the trainable parameters do not need to be adjusted much. Conversely, a high loss value indicates that the network performed poorly and more drastic changes to the trainable parameters are necessary. Many different loss functions have been developed for machine learning. One such example is the cross-entropy loss function, which is commonly used in multi-class classification tasks. Cross-entropy can be mathematically represented as:

$$\mathcal{L} = -\sum_{c=1}^{C} y_c \cdot \log(\hat{y}) \tag{1.2}$$

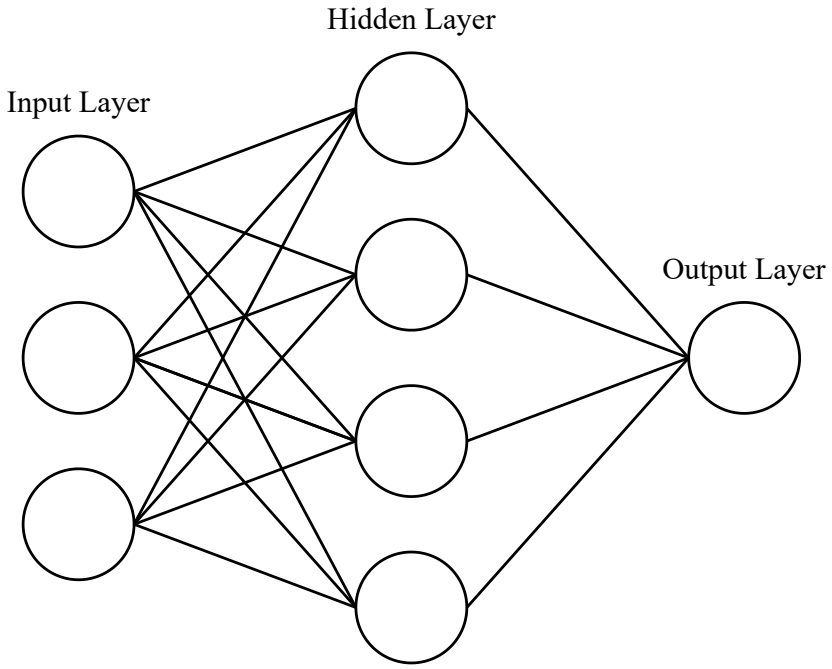
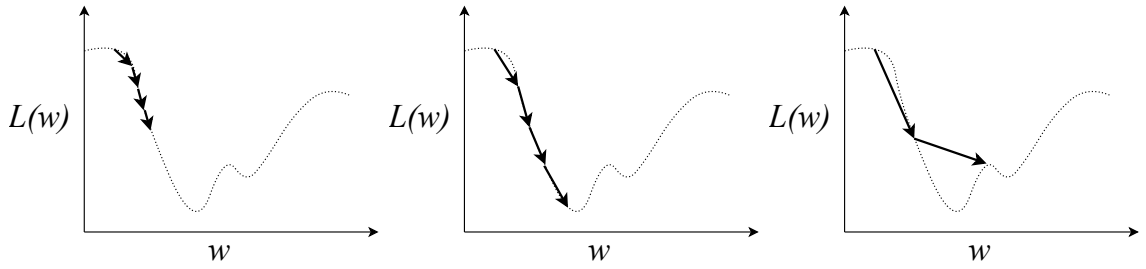


Figure 1.4: An example of a fully-connected neural network with an input layer containing three neurons, one hidden layer containing four neurons, and an output layer containing one neuron.

where L is the loss value that is supposed to be minimized, C is the number of classes, y is the vector of ground truth values, and $\hat{y}$ is the vector of output predictions from the neural network.

After the loss has been computed using the loss function, trainable parameters in the network are updated using a process known as gradient descent. During gradient descent, partial derivatives for a function are calculated in order to determine where minima are located within the function. With neural networks, we use gradient descent with respect to the loss function. Partial derivatives of the loss function are taken with respect to every trainable parameter in the network. This provides us with a vector that details the direction in which the loss function increases most rapidly. We then move along the loss function in the opposite direction, since our goal is to minimize loss. Computing the gradients provides us with a direction to move along the loss function; however, it does not tell us how far we should move. This is determined by an adjustable parameter called the learning rate.

Careful considerations must be made when selecting the learning rate for a given problem. Since the learning rate determines the speed at which the loss function is traversed, it is possible to miss minima in the loss function if the correct value is not chosen. A learning rate value that is too low will cause the loss to converge very slowly, meaning many training iterations will be needed



(a) Low learning rate (b) Good learning rate (c) High learning rate

Figure 1.5: An example showing how different learning rates can affect the gradient descent optimization task.

to find the minima. A low learning rate may also cause the gradient descent algorithm to settle at local minima rather than a global minima. A learning rate that is too high may skip over minima in the loss function entirely. Figure 1.5 illustrates how learning rate values can affect the gradient descent process.

When updating the trainable parameters in a network using the gradients and learning rate, it is important to remember that changes to lower layers will affect the performance of higher layers. Therefore, updates must occur in a top-down (last to first) fashion. This is accomplished using an algorithm called backpropagation. Once all network parameters have been updated, the gradient descent process is concluded and the network can begin another training iteration. Gradient descent can be expressed mathematically as:

$$w = w - \frac{\eta}{n} \sum_{i=1}^{n} \nabla L_i(w) \tag{1.3}$$

where w represents the network weights in matrix form, η represents the learning rate, n represents the number of training samples, and $\nabla L_i(\cdot)$ represents the gradient of the loss function for the i-th training sample.

There are three varieties of gradient descent: batch gradient descent, stochastic gradient descent, and mini-batch gradient descent. Batch gradient descent only updates network parameters after all training samples have been passed through the network. Batch gradient descent provides more accurate gradient estimations but can be taxing on certain hardware due to large memory requirements. Stochastic gradient descent updates network parameters after every training sample. This approach allows the network to train more quickly but can provide noisy gradient estimations.

Lastly, mini-batch gradient descent strikes a balance between the previous methods by updating network parameters after subsets, or batches, of the training data have been passed through the network. Regardless of what method is used, once all training samples have been examined, the network is said to have gone through one training epoch. It is common practice to perform many training epochs when fitting a neural network with data.

When performing network training for classification tasks, data is typically divided into training data and testing data. Training data, as the name suggests, is used to train the neural network. Testing data is used after the training process in order to determine how well the neural network performs when given data it has never observed before. It is imperative to make sure the training and testing sets are kept separate from one another. If a network were to test on the same data that it trained on, this could result in inflated network performance metrics. It is common when training a neural network to also select a small subset of the training data to use as a validation set. Validation sets act as smaller testing sets, but are evaluated after every training epoch in order to observe how a model is performing as it is training.

1.2.4.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a form of deep neural network that utilize several types of layers in order to extract identifying features from provided inputs. These layers include convolutional layers, pooling layers, and fully-connected layers.

CNNs derive their name from their use of convolutional layers. Convolutional layers operate by taking an input and performing a set of convolutions between the input and a set of kernels, or filters. These filters represent the trainable parameters, or weights, of the convolutional layer. Effectively, the filters gradually learn to recognize various input features during the training process. Ideally, each filter will learn a different feature from the inputs. The number of filters and the size of each filter are configurable for each convolutional layer. In addition, a stride value can be set for each convolutional layer. The stride value determines how each filter is moved across inputs when performing convolutions. A higher stride value will move the filters across inputs with larger steps. Deep CNNs typically utilize many convolutional layers at the beginning of their architectures. Lower convolutional layers are able to extract more general or generic features while higher convolutional layers extract more specific features. The combined set of features that a convolutional layer learns is referred to as a feature map.

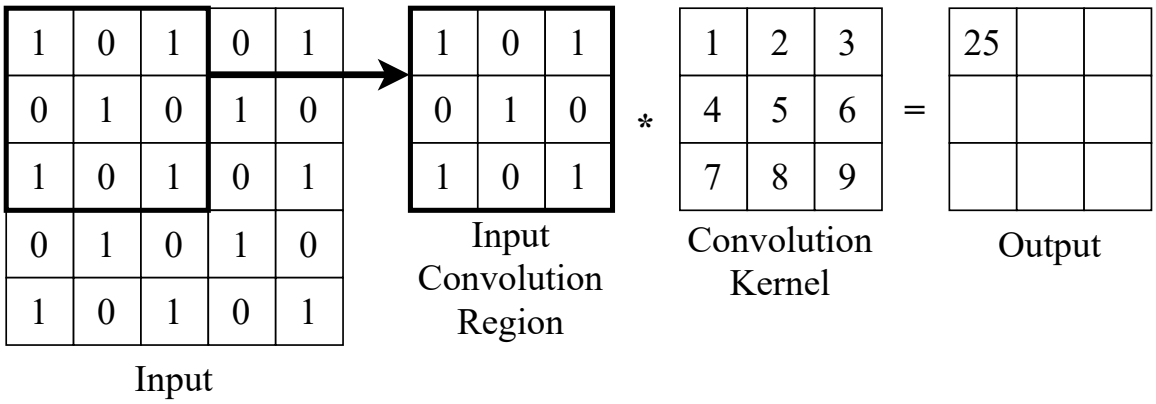


Figure 1.6: An example of two-dimensional convolution.

Convolutional layers are commonly used to extract many different features from their inputs. Pooling layers are then used in order to downsample these features. The two most popular pooling layer methods are average pooling and max pooling. In average pooling, several features are combined in order to form an average that is passed to the next layer. In max pooling, only the maximum value in a subset of features is selected to be passed to the next layer. Effectively, pooling layers allow a CNN to reduce its feature space while maintaining the features that are important for classification. Since pooling layers simply perform a mathematical operation, they contain no trainable parameters.

After extracting a final feature map using convolutional layers and pooling layers, CNNs use fully-connected, or dense, layers for classification. In fully-connected layers, all outputs from the previous layer are connected as inputs to all neurons within the fully-connected layer (see Figure 1.4). Put simply, these layers take features as inputs and output a final prediction for what class the features belong to. It is common to use several fully-connected layers for classification in CNNs.

CNNs are particularly useful for processing data with spatial or temporal dependencies. This makes them well-suited for computer vision tasks such as image classification. When training on images, CNNs perform two-dimensional convolutions. Filters are able to extract spatial features such as edges, colors, and textures from these images. Figure 1.6 shows an example of two-dimensional convolution.

In addition to images, CNNs are also useful for processing time-series data. With time-series classification, each sample within the time-series is treated as an input feature. For multivariate time-

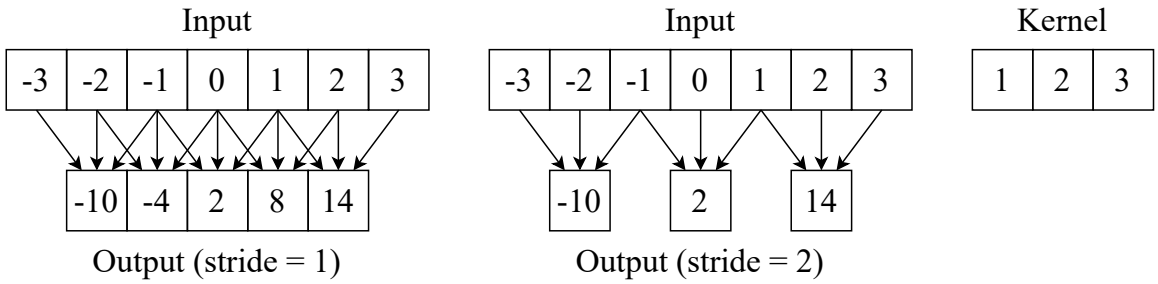


Figure 1.7: An example of one-dimensional convolution using different stride values.

series data, this can mean that inputs could have thousands or possibly millions of input features. CNNs are well suited for this type of data because they employ effective downsampling methods such as striding and pooling. Figure 1.7 shows an example of one-dimensional convolution using different stride values.

1.2.4.3 Transfer Learning

Transfer learning is a technique where knowledge gathered from one task is applied to another, related task. Consider, for example, the way we learn to use different programming languages. Everyone starts to program using one language, such as C, C++, Java, or Python. Although each of these languages has a unique syntax, they share many features. Loops, conditionals, and variables are fundamental concepts in most programming languages. Therefore, once we have become familiar with these concepts by learning our first language, it is easier for us to learn another language. This same technique can be applied to deep learning. With deep learning, knowledge can be transferred from one neural network to another. Knowledge, in this case, includes features or network weights. Consider another example, where a CNN has been trained as an image classifier that can differentiate between images of cats and dogs. Suppose now we want to train another image classifier that can discern different breeds of cats from one another. Instead of training a new network from scratch, we can use the pre-trained classifier as a base model for our new problem. This pre-trained model has already learned to recognize the general features of a cat, therefore once it is given new training images of different cat breeds, it will be able to converge more quickly.

Many neural networks developed for classification can be divided into feature extraction layers and classification layers. Feature extraction layers, such as convolutional layers and pooling

layers, are responsible for learning important features from their inputs. After learning a desired number of features, these features are passed to the classification layers. The classification layers are responsible for mapping their provided set of features into a predicted class label. Multiple layers used for classification are collectively referred to as a classification head. Fully-connected layers are often used as classification heads in CNNs. When using transfer learning, the feature extraction layers comprise the knowledge that is transferred to a new network. Conventionally, transfer learning for classification problems follows the following steps:

1. **Train a General Model**: First, a general or generic model is trained on a larger dataset. This general model should be comprised of both feature extraction layers and classification layers. Many popular deep learning models that have been trained on large datasets are publicly available. For example, the VGGNet [45] architecture that has been trained on ImageNet [37] is available online. TensorFlow, a popular machine learning library [1], even has a built-in function that loads this specific model.

2. **Transfer and Freeze Feature Extraction Layers**: Once the general model has been trained, its feature extraction layers are transferred to a new model. The weights in these feature extraction layers are then frozen, meaning they can no longer be updated. This is done so that the information stored in these weights is not overwritten when training the new model.

3. **Train a New Classification Head**: After adding the general feature extraction layers to a new model, a new classification head is added to the new model. This network is then trained using a new dataset related to the general dataset. The classification layers are the only layers that are updated during initial training epochs on the new model. Figure 1.8 illustrates this process.

4. **Fine-Tune Feature Extraction Layers**: During fine-tuning, the feature extraction layers are unfrozen and all trainable parameters within the network are allowed to learn. In order to avoid completely overwriting the weights of the feature extraction layers, fine-tuning is typically performed with a lower learning rate and a limited number of training epochs. For deep neural networks with many layers, some lower feature extraction layers may still be kept frozen during fine-tuning in order to retain knowledge of more general, low-level features.

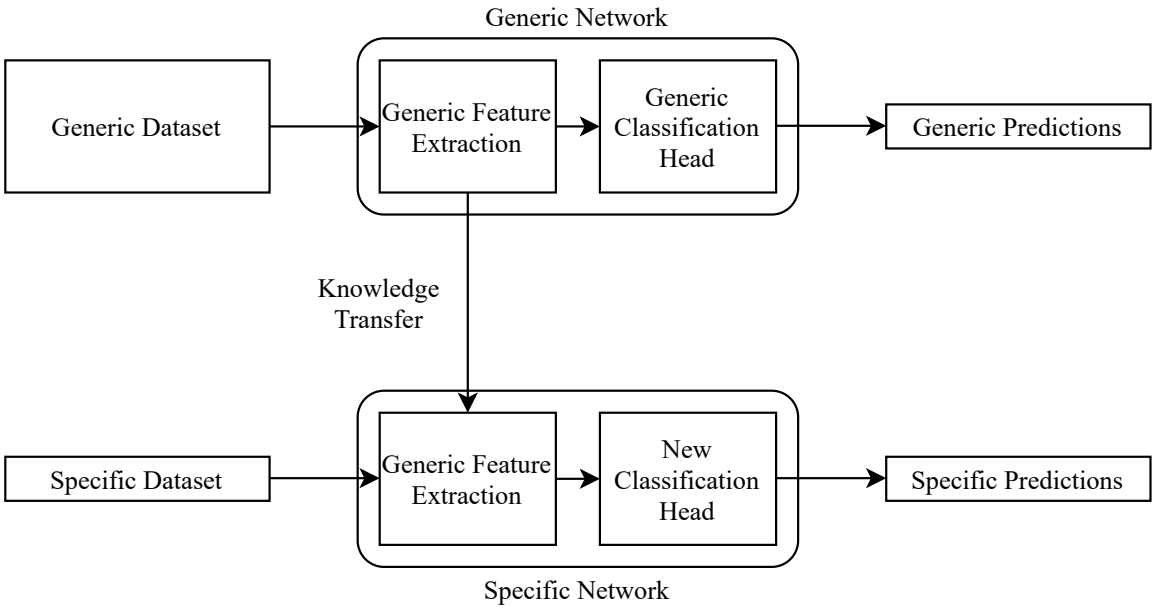


Figure 1.8: A flowchart illustrating the transfer learning technique for classification tasks.

Using transfer learning can provide many advantages. One major benefit of using transfer learning is that it can reduce the amount of data necessary for training an accurate classifier. Data collection can be a time-consuming and often expensive process. By utilizing feature extraction layers trained on a general or related dataset, fewer new training samples are needed in order to make a transfer learning model converge.

Another benefit of transfer learning is that it can help speed up the training process. By starting with a model trained on a similar problem, the model weights are initialized to values that will let them converge quickly. Even when training a new model on a dataset with a sufficient number of training samples, starting with a pre-trained model can reduce the number of training epochs needed to fit the model properly. This is especially useful for very deep neural networks that have many feature extraction layers and trainable parameters.

One more benefit that transfer learning can provide is improved model performance. In addition to helping models converge faster, pre-trained weights can also help models identify minima in their loss functions that they may have missed using only their own dataset for training. This is particularly advantageous when working with very small datasets.

1.3 Related Work

This chapter explores past literature related to the research of this thesis. First, methods developed for detecting periods of eating using ADM are analyzed, including the advantages and disadvantages of each implementation. Next, past transfer learning algorithms and applications are evaluated. The final section of this chapter includes a summary of all related works and a comparison of this literature to the work of this thesis.

1.3.1 Automated Dietary Monitoring

Studies in automated dietary monitoring (ADM) have employed a variety of different sensors and learning techniques. Early experiments in ADM by Amft et al. included the use of in-ear microphones that detected sounds of chewing [3]. The authors of this study determined that placement of the microphone in the ear would be unobtrusive and socially acceptable since other ear-mounted devices such as hearing aides and phone headsets were commonly seen in public. Four subjects were each given four different types of food to eat for data collection (N = 4). The microphone signal was segmented into individual chews using a low-pass filter and then classified using a simple decision tree classifier. Results showed that the classifier was able to recognize periods of chewing and determine what type of food was being chewed with an accuracy of 67-86% depending on the food type. Amft continued their work by designing a less invasive earpad sensor that rested outside the ear [2]. In this new study, three participants were given 19 different foods to chew (N = 3). A Bayes classifier was used to determine food type and showed an overall accuracy of 86%.

Makeyev et al. used a similar method to detect sounds of swallowing by placing a microphone over the throat [29]. By placing the microphone directly over the throat, the authors were able to reduce ambient noise. Data for this study was collected from a single subject (N = 1) that recorded 20 samples each for three classes of sounds (swallowing, talking, and head movement). A fourth class was artificially introduced containing random sounds of music in order to include a class with little similarity to the other three. By using a four-layer neural network, the authors were able to achieve a classification accuracy of 99%; however, this approach was eventually disregarded due to the intrusiveness of the throat sensor.

Gao et al. also used microphones to detect eating by using more commercially available, off-the-shelf Bluetooth headsets [19]. The motivation behind this was to make a practical ADM

system that was more acceptable to consumers. In this study, 28 participants (N = 28) were tasked with recording periods of eating, speaking, and drinking for data collection in a lab setting. Using both a support vector machine (SVM) and a deep neural network, a classification accuracy of 95-96% was attained. The authors of this study extended their work by evaluating their classifiers in free-living conditions as well; however, in this scenario the accuracy of the SVM classifier and deep neural network both dropped sharply to 65% and 77%, respectively. The main cause of this decrease in performance was attributed to excessive ambient noise. This study highlights the importance of testing ADM methods in free-living environments.

In addition to microphones, strain sensors have also been used to successfully detect periods of eating. Sasanov et al. used a piezoelectric strain gauge sensor worn below the outer ear to track movement of the lower jaw during food intake [39]. Data was collected from 20 volunteers (N = 20) that included periods of eating, resting, and speaking. By using an SVM, the authors reported an 81% classification accuracy on windows of 30 seconds. Nguyen et al. also used piezoelectric sensors along with an IMU to create an ADM device worn around the neck [32]. Both a random forest classifier and long short-term memory (LSTM) neural network were trained on data collected in a controlled environment (N = 10). Results showed that the LSTM was able to beat the random forest classifier in F1-score (76% to 67%).

Fontana et al. proposed a wearable sensor system containing a jaw motion strain sensor, hand gesture sensor, and accelerometer that was worn around the neck on a lanyard [18]. Information collected by the system was processed by a three-layer feedforward neural network in windows of 30 seconds for classification as eating or non-eating. Results for this sensor system showed an average classification accuracy of 90% on a dataset collected from 12 subjects (N = 12). Each subject recorded 24 hours of sensor data in a free-living environment. Farooq et al. continued this research by creating a sensor system with an accelerometer and temporalis muscle strain sensor that attached to a pair of eyeglasses [15]. Signals recorded by the sensors were divided into segments of 3 seconds for classification by a pair of SVMs and a decision tree (N = 10). An F1-score of 99% was reported for this learning technique. Recently, Doulah et al. continued this research even further [14] by adding a camera to the sensor system developed by Farooq et al. [15]. When eating is detected by the accelerometer and temporalis muscle sensor, the camera captures images of the food being eaten. Classification of sensor data was performed by an SVM using windows of ten seconds. Results showed a classification accuracy of 83% (N = 30). Bai et al. used a similar approach with a camera

and proximity sensor with comparable results [5].

The social considerations of wearable devices is an important aspect of ADM. Wearable sensors must be both comfortable and unobtrusive. A study by Kalantarian et al. [23] examined the efficacy and social acceptability of several different forms of wearable eating monitors, including microphones, piezoelectric strain sensors, cameras, and smartwatches that used IMUs. An online survey was used to determine social acceptance scores (N = 96). Out of all devices that were evaluated, the smartwatch was rated as the most socially acceptable device and also rated highly in terms of performance. Mercer et al. also examined the social acceptability of several different wearable activity monitors and determined that smartwatches were observed as the most tolerable form of device [30]. These studies show that smartwatches could serve as effective and marketable ADM devices.

Dong et al. experimented early with wrist motion-based eating detection by creating an IMU device that counted individual bites [11]. The authors discovered that there is a distinctive rolling motion of the wrist when bites of food are taken. This rolling motion is illustrated in Figure 1.9. Data was collected in a laboratory setting from ten subjects (N = 10) that were given a meal to eat with the utensils of their choice. A rule-based algorithm was then developed for detecting bites that showed a true positive rate (TPR) of 91%. Although the authors achieved a high TPR, it was noted that the bite counter did show a large number of false positives. Dong et al. continued their work by collecting an entire day of wrist data from four participants (N = 4) in free-living conditions [12]. Using an updated rule-based algorithm developed for detecting entire eating sessions, a classification accuracy of 82% was achieved. Dong et al. developed their research even further in 2013 by collecting a larger free-living wrist motion dataset with smartphones [13]. In this newer study, an entire day of wrist motion data was recorded from 43 volunteers (N = 43). One-second-long windows were used to detect periods of eating using a naive Bayes classifier. Results showed an 81% accuracy in eating episode detection. These studies showed that using wrist motion to automatically detect periods of eating in free-living environments was feasible.

A survey by Salley et al. compared the wrist-mounted bite counting device developed by Dong et al. [11] to a more traditional manual food journaling application in order to determine which was more appealing to users. [38]. Findings from the survey revealed that 76% of all participants (N = 83) preferred the bite counting device due to its ease of use. A later survey by Turner-McGrievy et al. also compared the same bite counting device [11] to another food journaling application to

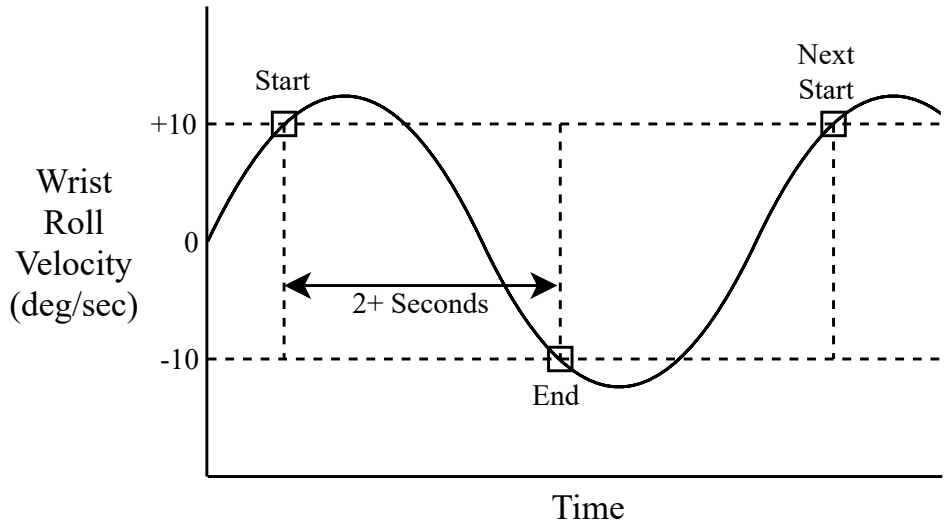


Figure 1.9: The distinctive rolling motion of the wrist during a bite, adapted from [11].

find which was more effective for weight loss [49]. Results showed that the participants who used the journaling application lost more weight than those that used the bite counter (N = 81). This was attributed to the fact that some bite counter users forgot to log some of their meals and were unable to make edits to their data. These surveys suggested that, although there was a high interest in a wrist-mounted ADM device, more work was required to make these devices accessible.

Shen et al. used hidden Markov models (HMMs), a type of statistical model, in order to recognize eating behavior in different demographics [43]. Wrist motion data for this study was collected in a cafeteria from 276 volunteers that each ate a single meal (N = 276). This dataset is now publicly available and known as the Clemson Cafeteria Dataset [21]. By using HMMs, it was shown that grouping subjects by age, gender, and utensil choice could all be used to increase gesture recognition accuracy.

Recent advancements in the field of deep learning have also been utilized to detect periods of eating from wrist motion. Kyritsis et al. used a hybrid CNN-LSTM network in order to classify wrist motion data collected from smartwatches (N = 12) [24, 25]. Individual bites were collected and merged in order to detect entire meals. Results from this study showed a weighted classification accuracy of 79%. Due to the large size of the network, training times for this architecture proved long, meaning integration with a standard smartwatch may not be feasible. In addition, the choice of eating utensils in this study gave varying results in network performance. This is a limitation of

a bottom-up approach to meal detection.

Stankoski et al. used a CNN architecture in order to detect entire meal segments from wrist motion data collected with smartwatches [46]. In order to accomplish this, the authors used a top-down approach with a large sliding window of 15 seconds to generate samples of eating and non-eating. 481 hours of free-living data was collected from 12 subjects (N = 12). Results showed an average true positive rate (TPR) of 81%. In addition to training a model on the entire dataset, the authors also created personal models for each test subject in order to observe if any performance could be gained by doing so. It was found that, on average, the personalized models offered similar or slightly improved performance compared to the group model, though this improvement varied depending on the subject. Lopez-Meyer et al. also explored the idea of using personalized models with similar results [27].

Luktuke used a CNN modeled after U-Net [36] to classify eating gestures from wrist motion [28]. The Clemson Cafeteria Dataset [21] was used to train this deep CNN architecture. Luktuke used a top-down procedure by training with windows of 30 seconds for classification. Results showed that 78% of all gestures were correctly classified on average.

Sharma also used a CNN for wrist motion classification of eating events with a large window size of six minutes [41, 40]. Sharma collected the Clemson All-Day (CAD) dataset for this approach, which consisted of 4,680 hours of data collected from 351 different volunteers (N = 351) [42]. Each volunteer recorded wrist motion in free-living conditions for approximately one day each. This is the largest known dataset of free-living wrist motion data. Using the CAD dataset, Sharma reported a weighted accuracy of 80% with their CNN architecture. They also developed a dual threshold hysteresis method to evaluate detection of entire eating events. Using this method, Sharma reported a TPR of 89% for meal detection with a false positive to true positive ratio of 1.7.

Patyk continued the work of Sharma [40] by developing a recurrent neural network (RNN) capable of applying daily context to detected meals [34]. This was accomplished by training the RNN on daily eating probability values provided by Sharma's CNN model. The CAD dataset was also used for these experiments [42]. Using this method, results showed an 85% TPR for meal detection with a false positive to true positive ratio of 0.8. Although this showed a decrease in TPR relative to Sharma's base model, the number of false detections decreased substantially. A limitation of this study was that entire days of data are required for classification to be possible.

Wei applied the CNN model developed by Sharma [40] to individual datasets containing

multiple days of data from the same subject [51]. The Multiday dataset was collected for this study, which consisted of at least ten days of data each for eight different subjects (N = 8). The objective of this study was to determine if a model trained solely on one individual's data could outperform a group model on the same data (an idea also examined by Stankoski et al. [46] and Lopez-Meyer et al. [27]). First, a group model was trained on the entire CAD dataset [42]. Next, an individualized model was trained for each subject in the Multiday dataset. Using the group model on each individual's data showed an average weighted accuracy of 78%. Using the individualized models showed an average weighted accuracy of 82% (a 4% increase); however, the indvidualized models were more prone to false detections. Improvements in weighted accuracy varied by subject. For some subjects, the group model was able to outperform the individual models. The work of this thesis draws largely from the work of Sharma [40] and Wei [51].

1.3.2 Transfer Learning

Several extensive surveys have been conducted in recent years that review the applications and benefits of transfer learning for many tasks related to machine learning [33, 52, 47]. One of the major benefits of transfer learning discussed in these surveys is that it can solve the problem of insufficient training data. By starting with a model trained on a related dataset, general features that were learned can accelerate learning on a new dataset that may have a limited number of samples. Not only can this improve the performance of the new model trained on the limited dataset, it can also reduce the training time needed for the new model to converge.

Transfer learning has been used for many different classification tasks, including image classification. Cheng et al. used transfer learning with CNNs in order to classify images of abdominal ultrasounds. [8]. Ultrasound images were rescaled to a lower resolution and then passed to two different CNNs based on VGGNet [45] and CaffeNet [22] that were first trained on ImageNet [37]. After this initial training period, the weights of all convolutional layers in each network were frozen in order to keep feature extraction intact. Next, several dense layers were added to the top of the network to act as a new classification head. Both CNN architectures were able to outperform a trained radiologist. Similar studies have used CNNs in conjunction with transfer learning techniques to improve image classification accuracy [35, 57].

Transfer learning has also been used for time-series classification. Fawaz et al. recently explored using transfer learning with CNNs to classify time-series data [16]. Using 85 different

publicly available time-series datasets, the authors trained a CNN with each dataset and then fine-tuned it with the others to observe performance changes. This resulted in the creation of 7,140 distinct CNNs. Results for this study showed improvements in classification accuracy for 71 of the 85 datasets, which demonstrates that transfer learning can indeed be beneficial for time-series data. Wen et al. also used transfer learning with time-series data for fault detection [53]. In this study, the authors trained a CNN based on U-Net [36] on a large time-series dataset containing many errors with corresponding ground truth labels. After training on this larger dataset, the authors fine-tuned their network using smaller synthetic datasets that were slightly corrupted to produce errors. By using transfer learning, the authors saw an average improvement in intersection over union (IoU) score of 20% compared to baseline models.

Cook et al. conducted an extensive survey on the applications of transfer learning to the field of human activity recognition [9]. One application the authors describe is that transfer learning can be used to transfer knowledge of one activity to a related activity. For example, a neural network trained to identify periods of walking could help train another network to identify periods of running. In addition, knowledge of an activity gathered from one type of sensor could be shared in order to identify the same activity with a different type of sensor.

1.4 Novelty

The novelty of this work is applying transfer learning to leverage features learned from a group eating detection model in order to improve the performance of individual eating detection models. We consider using the group model as a foundation for the creation of new individual models in order to reduce false detections and increase accuracy. This work specifically seeks to answer the following questions:

1. Can transfer learning be employed to increase the performance of individualized eating detection models by utilizing features learned from a group eating detection model?

2. How much performance can be gained by using transfer learning to create individualized models? Does this performance gain vary depending on the individual?

3. Can transfer learning speed up the training process for individualized models?

Chapter 2

Methods

This chapter outlines the methods used in the experiments of this thesis. We first describe the CAD [42] and Multiday [51] datasets that were used for our experiments in Section 2.1. Next, the data pre-processing techniques that we utilized on each of these datasets are examined in Section 2.2. The CNN architecture that was used for all group, individual, and transfer learning models is then reviewed in Section 2.3. Following this, Section 2.4 describes the training process for each type of model. The post-processing procedures used on each model are examined in Section 2.5. Finally, the evaluation metrics that were used to measure the performance of each model are provided in Section 2.6.

2.1 Datasets

Two datasets were used in the experiments of this work. The first was the Clemson All-Day (CAD) dataset, which contained 354 days of recorded wrist motion data from 351 different subjects. One subject recorded data for three days, another subject recorded for two days, and the rest recorded for one day. The CAD dataset contained a total of 4,680 hours of data, including 1,063 meals. Recording was performed in free-living conditions using the Shimmer3 IMU device. The Shimmer3 recorded six total axes of data (accelerometer x, y, and z as well as gyroscope pitch, roll, and yaw) at a rate of 15.06 Hz, though this was downsampled to 15 Hz after collecting the data from the Shimmer3. The CAD dataset was separated into 354 different files with the SHM (an abbreviation for Shimmer) file extension, where each file contained binary IMU data for one day

of recording. Each data file also had a corresponding text file that listed the ground truth eating events for the recording. The average length of each recording was approximately 13 hours. The entire data collection procedure for the CAD dataset is explained by Sharma in [42].

The second dataset used in this work was the Multiday dataset, which included at least 10 days of free-living wrist motion data each for eight individuals. Each of these individuals were referred to as subjects in the dataset and labeled using a three-digit number from 001 to 008. The Multiday dataset followed the same data collection procedure as the CAD dataset. Files were first separated by subject before being separated by day, however. The entire data collection procedure for the Multiday dataset is described by Wei in [51]. Some inconsistencies were observed in the Multiday dataset that Wei used in his experiments. In order to fix this, we manually edited the dataset in the following ways:

1. Subject 001 data included a duplicate data file for one day of recording. This file was removed.

2. Subject 001 data included two days of recording that were each split into two separate data files. The ground truth text files for these four recordings showed that each recording only contained data for a single meal. Since no non-eating data was also available for these recordings, they were removed from the dataset in order to maintain a class distribution (eating and non-eating) similar to other recordings in the dataset.

3. Wei excluded two days of recording for subject 003. Upon examination, these recordings contained valid data and were therefore added to our updated dataset.

4. One day of recording for subject 003 contained a ground truth event for teeth brushing. This event was removed from the ground truth file.

5. One day of recording for subject 003 contained a ground truth event for a single bite that only lasted for four seconds. This event was removed from the ground truth file.

6. Wei removed two days of recording for subject 006. Upon examination, these recordings contained valid data and were therefore added to our updated dataset.

7. Subject 007 data included two days of recording that each spanned three entire days. These two files were found to be duplicates. To fix this, each file was edited so that they only included the data for their particular day. In addition, the ground truth text files for these days were edited so that they only included eating events for their corresponding day.

Subject	Original Dataset				Triaged Dataset			
	Days	Meals	Total Hours	Eating Hours	Days	Meals	Total Hours	Eating Hours
001	17	32	129.2	7.7	12	26	115.1	6.2
002	14	26	125.6	3.3	14	26	125.6	3.3
003	23	60	167.2	7.6	24	59	170.7	7.6
004	13	38	134	9.3	13	38	134	9.3
005	14	20	127.8	5.9	14	20	127.8	5.9
006	12	26	144.8	7	14	30	174.6	8.6
007	10	22	120.4	5.9	10	19	111.1	5.0
008	12	22	115.5	10.8	12	22	115.5	10.8

Table 2.1: Comparison of the original Multiday dataset [51] to the triaged Multiday dataset

Overall, discrepancies were found in four out of eight subjects. Table 2.1 shows a comparison of the original Multiday dataset to the triaged version that was ultimately used.

2.2 Data Pre-Processing

Several pre-processing steps were taken in order to prepare each dataset for use in training machine learning models. Section 2.2.1 covers the technique we used to smooth raw IMU data. Section 2.2.2 examines the normalization method that was used on the smoothed IMU data. Next, section 2.2.3 details how we used a sliding window process to create training samples. Finally, section 2.2.4 examines how we balanced the classes of our training samples.

2.2.1 Smoothing

Raw data recorded using IMUs is prone to noise. An IMU worn on the wrist would further compound this issue, since people move their wrists during many different activities throughout the day. Even trying to hold the wrist completely still typically results in slight shaking motions. In order to help reduce the effects of noise in our data, we applied smoothing using a Gaussian filter as outlined in [40].

Each axis of the accelerometer (x, y, z) and gyroscope (yaw, pitch, roll) was filtered independently. The complete equation for the Gaussian filter is provided by:

$$S_t = \sum_{i=-N}^{0} R_{t+i} \frac{\exp(\frac{-t^2}{2\sigma^2})}{\sum_{x=0}^{N} \exp(\frac{-(x-N)^2}{2\sigma^2})} \tag{2.1}$$

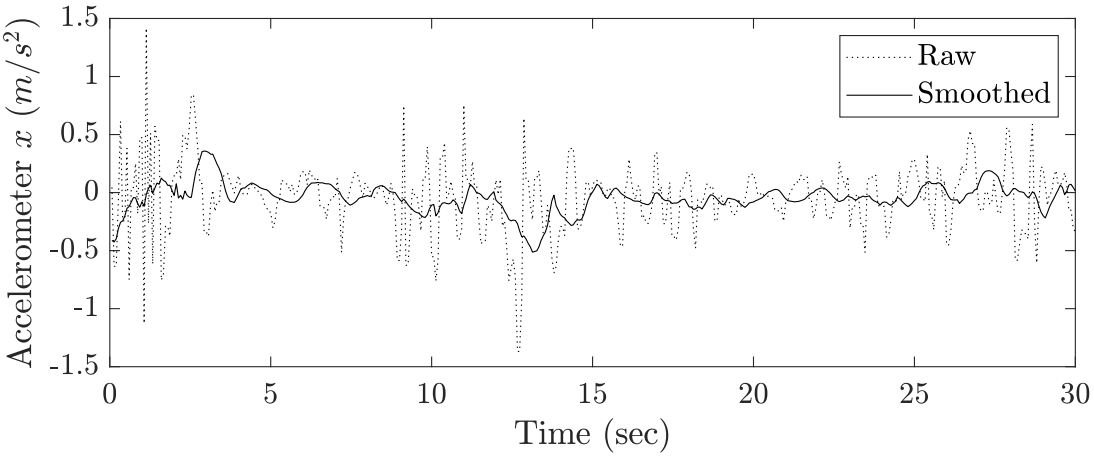


Figure 2.1: A sample of the Gaussian smoothing filter being used on the accelerometer x data axis.

where S_t represents each smoothed signal at time t, R_t represents each raw signal at time t, σ^2 is the variance of the filter, and N is the length of the filter. For all axes of data, we used a variance of $\sigma^2 = 10$ and a filter length of $N = 15$ in order to cover one second of past data. Figure 2.1 shows the Gaussian filter operating on a segment of data recorded from the accelerometer x channel of the Shimmer3 IMU device. It can be observed from this figure that the filter helps preserve trends in the data while eliminating unnecessary noise.

2.2.2 Normalization

Normalization is used when creating machine learning models so that the values of all features share a common scale. If one feature has higher values than the others, a model may become biased towards this feature during training. In our datasets, it was observed that the gyroscope measurements (measured in deg/sec) showed a larger range of values than the accelerometer measurements (measured in m/s^2). In order to rescale these values, we used Z-score normalization. Z-score normalization can be represented mathematically as:

$$z = \frac{x - \mu}{\sigma} \tag{2.2}$$

where x is the input data, μ is the mean of the data, and σ is the standard deviation of the data. By subtracting its mean and dividing by its standard deviation, the data is rescaled to have a new

mean close to zero and a new standard deviation close to one. Each IMU data axis was normalized independently using this method. Z-score normalization was used because it was shown to yield better results than other normalization methods such as min-max scaling [40].

As mentioned previously, it is important to keep training and testing sets separated from each other when training a neural network. When performing Z-score normalization, the means and standard deviations of only the training set are calculated. Both the training set and testing set are normalized using these values.

2.2.3 Sliding Window

Wrist motion data was recorded over the course of entire days. In order to reshape this data into a more efficient size for training purposes, a sliding window approach was used. With this approach, many shorter samples, or windows, are extracted from a larger sample. After extracting a window from our time-series data, the window was shifted along the recording by an amount called the stride. The number of time-series windows that could be collected from a day of recording can be calculated by:

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

where T is the length of the entire day of recording, W is the window size, and S is the stride value. The size of the windows that are extracted can have a large impact on neural network performance. For our datasets, we used a larger window size of $W = 6$ minutes in order to better detect entire meals. When collecting windows from the CAD dataset, we used a stride value of $S = 15$ seconds. A stride value of $S = 5$ seconds was used with the smaller Multiday dataset in order to extract more windows of eating data. Figure 2.2 shows how windows of data were collected from the smoothed and normalized IMU datasets.

Once data windows were collected from a recording, they were labeled using the corresponding ground truth file. We used a majority vote labeling scheme. If at least half of the data within an extracted window was self-reported as eating in the ground truth file, the window was labeled as eating (1). Otherwise, the window was labeled as non-eating (0).

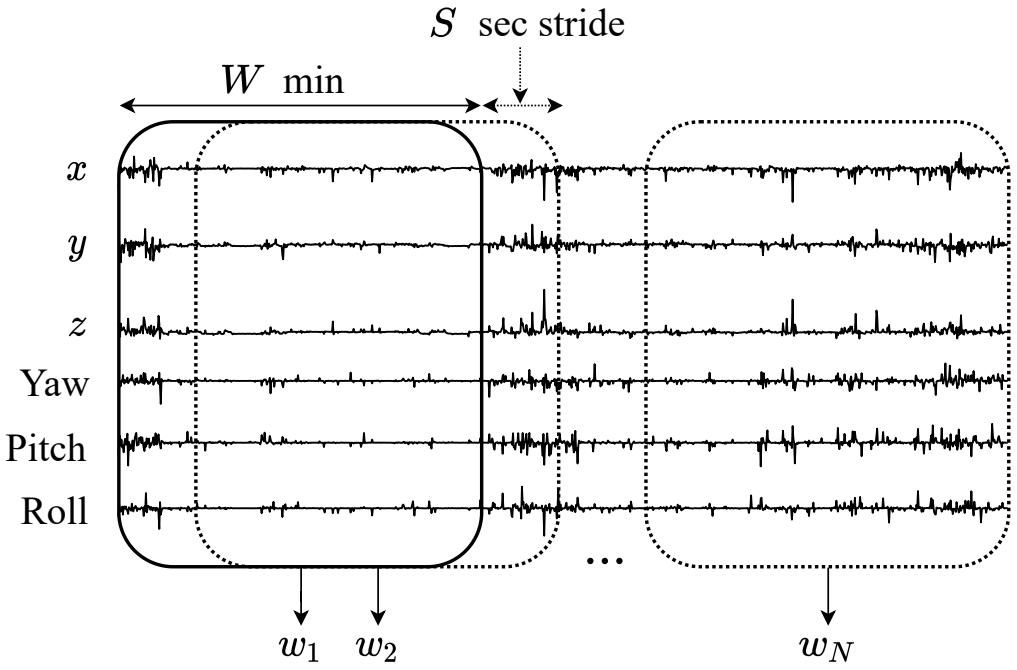


Figure 2.2: An illustration depicting the creation of multivariate time-series windows from IMU recordings using striding.

2.2.4 Data Imbalance

Determining whether windows of wrist motion data corresponded to eating or non-eating was a binary classification task. When creating a classifier, it is important that all classes are represented equally in the training data. The CAD and Multiday datasets, however, are imbalanced. This is because the average person spends much less time eating than not eating throughout the course of a day. If a neural network is trained using imbalanced data, it will learn to favor the most represented class.

Two methods of fixing the imbalanced data were available: undersampling and oversampling. With oversampling, samples from the minority class are randomly selected for duplication until the total number of minority class samples equals the number of majority class samples. Oversampling allows all samples from the majority class to be kept for training; however, by making exact copies of the minority class, overfitting becomes more likely. Another disadvantage of oversampling is that it can make training sets very large, resulting in long training times.

Undersampling takes the opposite approach. Samples from the majority class are randomly removed from the training set until the total number of majority class samples matches the number

of minority class samples. Undersampling discards potentially useful data from the majority class but is not prone to overfitting. Undersampling also allows for faster training times by shrinking the size of the training set.

We used an undersampling approach when training all models in our experiments. After windows of data were collected and labeled as described in Section 2.2.3, we randomly selected non-eating windows to be removed from the training set until the eating and non-eating classes were balanced. This approach was taken to reduce excessive training times.

2.3 Neural Network Architecture

This section examines the CNN architecture that was shared by all group, individual, and transfer learning models. Each layer of the architecture and its corresponding settings are described.

2.3.1 Input Layer

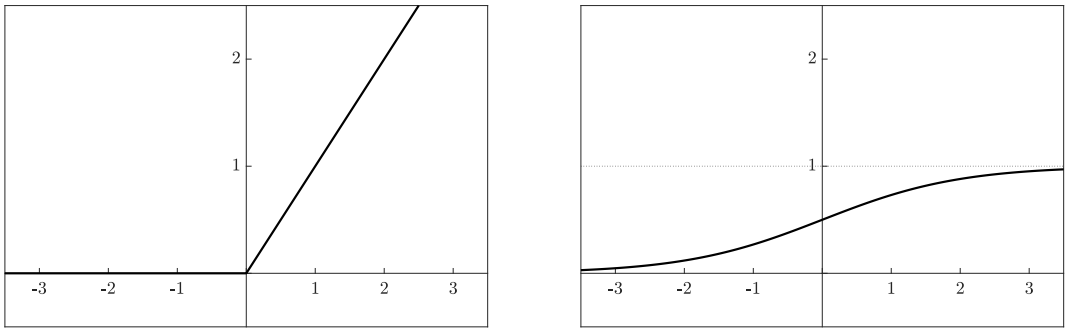
Windows of eating and non-eating were created from the CAD and Multiday datasets using the steps outlined in Section 2.2. Using a window size of $W = 6$ minutes (5400 data points at 15 Hz) and all six axes of data recorded from the Shimmer3 IMU device, each input passed through our CNN had a shape of (5400, 6). These inputs were fed in batches to the network for training.

2.3.2 Convolutional Layers

Input windows were first passed through a series of three one-dimensional convolutional layers in the CNN. These layers were responsible for learning identifying features from the multivariate time-series windows. Each convolutional layer was defined by their number of filters (f), the size of these filters (k), and a stride value (s). Each convolutional layer was passed an input of shape (L_i, c), where L_i was the length of the input and c was the number of channels, or axes, in each input. Using these variables, the number of trainable parameters in each convolutional layer was calculated using:

$$N = f \cdot k \cdot c + f \tag{2.4}$$

The total number of weights in each layer was represented by the $f \cdot k \cdot c$ term in this equation and the total number of biases was represented by f, since each filter had one bias value. The output



(a) Rectified linear unit (ReLU) function

(b) Sigmoid function

Figure 2.3: The rectified linear unit (ReLU) and sigmoid activation functions.

length of each convolutional layer was also calculated using these variables, and can be represented as:

$$L_o = \frac{L_i - k}{s} + 1 \tag{2.5}$$

Since each convolutional layer used f filters, the final output shape from these layers was represented as (L_o, f). Each convolutional layer was also assigned with an activation function. All three of the convolutional layers in our architecture used the rectified linear unit (ReLU) activation function. The ReLU function can be expressed mathematically as:

$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

where x represents an input value. The ReLU function clips any negative input values to 0, otherwise it returns its input. Gradient computations using the ReLU function are very efficient due to the function's simplicity. A graph of the ReLU function is provided in Figure 2.3a. Lastly, all three convolutional layers used an L1 regularization penalty of 0.01 in order to reduce the risk of overfitting [40].

The first convolutional layer in our CNN used $f = 10$ filters, a filter size of $k = 44$, and a stride of $s = 2$. Lower convolutional layers within a CNN are trained to recognize low-level features within input data. A filter size of $k = 44$ (approximately 3 seconds of data recorded at 15 Hz) was selected for this layer in order to identify wrist motion patterns associated with individual bites,

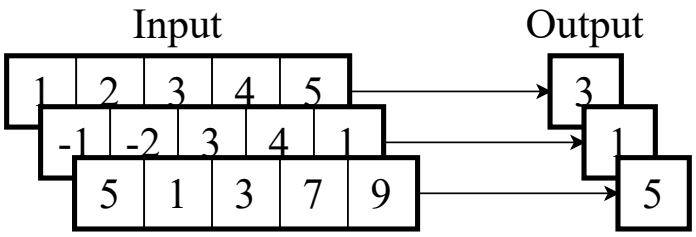


Figure 2.4: An illustration of one-dimensional global average pooling. Each one-dimensional input channel is reduced to the average of its values.

since previous work has shown the average bite lasts around 3 seconds [40]. This layer accepted inputs of shape (5400, 6) directly from the input layer. Using equations 2.4 and 2.5, this meant that the layer had $N = 2650$ trainable parameters and an output shape of (2679, 10).

The second convolutional layer of the CNN used $f = 10$ filters, a filter size of $k = 20$, and a stride of $s = 2$. This layer was designed to identify wrist motion patterns related to consecutive eating gestures. Using equations 2.4 and 2.5, this layer had $N = 2010$ trainable parameters and an output shape of (1310, 10).

The third and final convolutional layer of the CNN used $f = 10$ filters, a filter size of $k = 4$, and a stride of $s = 2$. This final convolutional layer was designed to detect high-level wrist motion patterns associated with entire meals. Using equations 2.4 and 2.5, this layer had $N = 410$ trainable parameters and an output shape of (664, 10).

2.3.3 Pooling Layer

A one-dimensional global average pooling layer was used after the three convolutional layers of our CNN in order to downsample the feature map provided by the last convolutional layer. Global average pooling reduces all values within an input channel to their average. In our CNN, the last convolutional layer had an output shape of (664, 10). The global average pooling layer averaged each channel of 664 values from this output to provide a new output shape of (10). Figure 2.4 illustrates how this process works by reducing an input of shape (5, 3) to an output of shape (3). Because pooling layers simply perform a mathematical operation in order to downsample, they contain no trainable parameters or associated variables. The global average pooling layer and convolutional layers together formed the feature extraction layers in our CNN architecture.

2.3.4 Fully-Connected Layers

Two fully-connected layers were used in our CNN architecture after the global average pooling layer. These fully-connected layers served as the classification head of our network. The number of trainable parameters in each fully-connected layer was calculated using:

$$N = L_i \cdot n + n \tag{2.7}$$

where L_i was the length of the input vector and n was the number of neurons in the fully-connected layer. The total number of weights in each fully-connected layer was represented by the $L_i \cdot n$ term in this equation and the total number of biases was represented by n, since each neuron had one bias value. Each neuron produced one output, meaning that the number of outputs from each fully-connected layer was equal to n.

The first fully-connected layer accepted an input of shape (10) from the one-dimensional global average pooling layer and used $n = 200$ neurons. This meant the layer produced an output of shape (200) and, using equation 2.7, had a total of $N = 2200$ trainable parameters. This layer used the ReLU activation function (see Figure 2.3a).

The second fully-connected layer contained a single neuron and functioned as the output layer in the CNN. Using equation 2.7, this layer had a total of $N = 201$ trainable parameters. The output layer used the sigmoid activation function. The sigmoid function can be represented mathematically as:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

where x represents an input value. The sigmoid function rescales all input values to the range $[0, 1]$. By using this activation function in the output layer, the CNN effectively provided the probability of eating ($P(E)$) for input windows of wrist motion data. A $P(E)$ greater than or equal to 0.5 represented eating and a $P(E)$ lower than 0.5 represented non-eating during classification. Figure 2.5 shows an example of a $P(E)$ curve produced from a full day of wrist motion data.

2.3.5 Architecture Summary

The full CNN architecture used for our experiments used three one-dimensional convolutional layers followed by a one-dimensional global average pooling layer for feature extraction. It

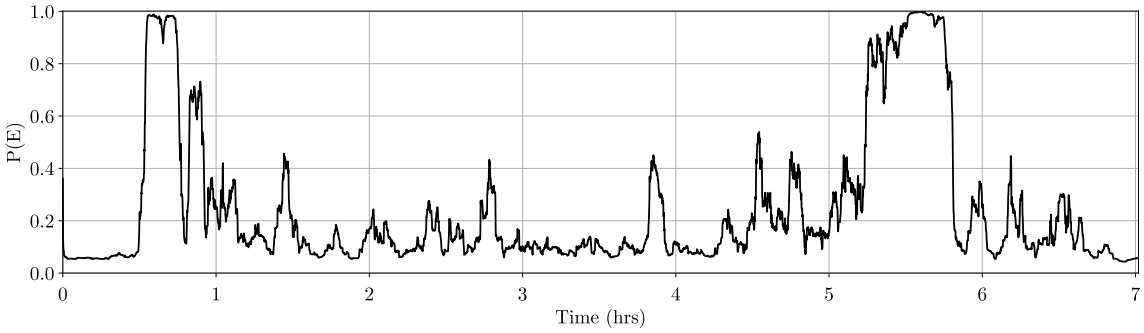


Figure 2.5: The probability of eating $P(E)$ throughout an entire day of recording.

Layer Type	**Output Shape**	**Parameters**
Input	(5400, 6)	0
1D Convolutional	(2679, 10)	2650
1D Convolutional	(1310, 10)	2010
1D Convolutional	(664, 10)	410
1D Global Average Pooling	(10)	0
Fully-Connected	(200)	2200
Fully-Connected	(1)	201

Table 2.2: Output shapes and number of trainable parameters for each layer in the CNN architecture.

then used two fully-connected layers for classification. The final fully-connected layer acted as an output layer. By using the sigmoid activation function, this layer provided a final output value that represented the probability of eating ($P(E)$) for each input window. Input windows passed to the network were of the shape (5400, 6), which represented $W = 6$ minutes of data recorded from an IMU. Table 2.2 shows a summary of each layer in the CNN, including the output shape and number of trainable parameters for each layer. Figure 2.6 shows the full architecture, including the activation function and variables used for each layer.

2.4 Network Training

Three types of models were used in our experiments: group models, individual models, and transfer learning models. This section examines the training process for each type of model as well as the hyperparameters used during training.

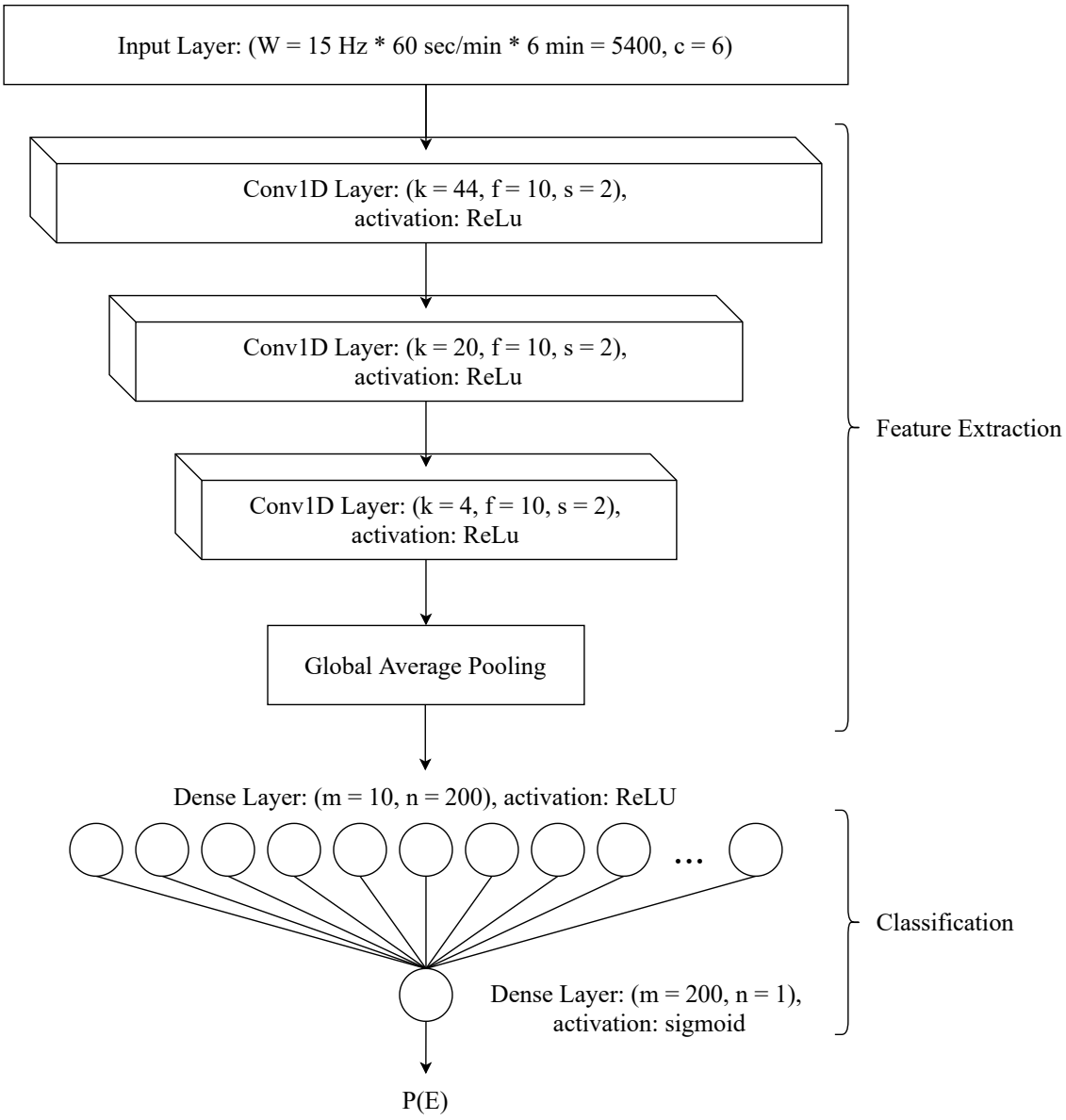


Figure 2.6: The complete CNN architecture used for all group, individual, and transfer learning models. In the input layer, W represents the window length in data and c represents the number of channels, or axes, in each input. In the convolutional layers, k represents filter length, f represents the number of filters, and s represents stride. In the fully-connected layers, n represents the number of neurons. The convolutional layers and pooling layer together formed the feature extraction layers in the network. The two fully-connected layers formed the classification head.

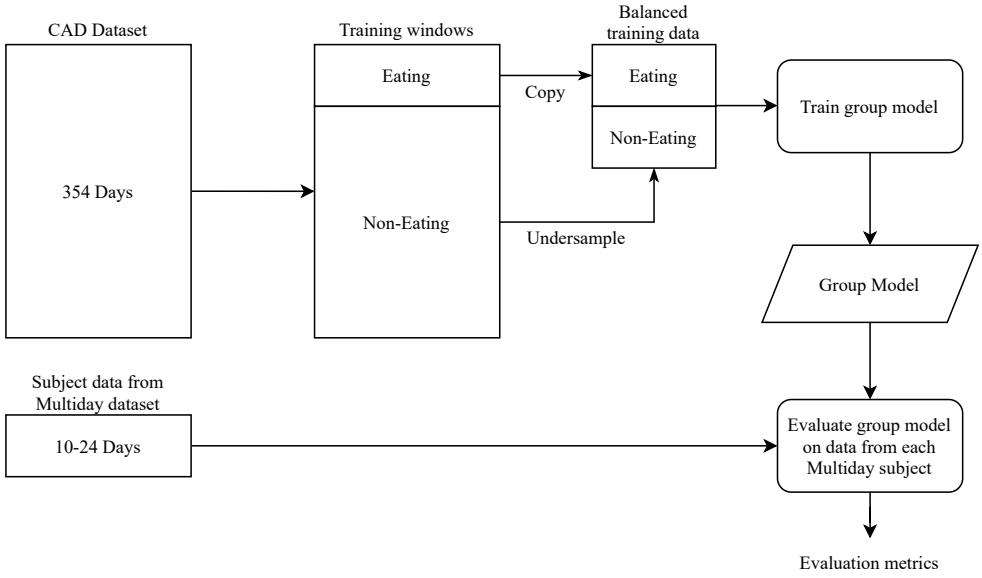


Figure 2.7: The process for training and evaluating a group model.

2.4.1 Group Model Training

Each group model used in our experiments was trained on the entirety of the CAD dataset (354 days of wrist motion data). All days of data were first split into windows. Next, windows labeled as non-eating were undersampled until the number of eating and non-eating windows matched (see Section 2.2). 10% of all training windows were used as a validation set when training each group model in order to observe model performance during training. Only the model that showed the lowest validation loss during training was saved and used for evaluation.

After training on the full CAD dataset, each group model was evaluated using the Multiday dataset. All days in the Multiday dataset were separated by subject and each model was evaluated on each of these eight individual datasets separately. It is important to observe here that no individual data was used when training the group models. Individual data was only used when evaluating the group models. Figure 2.7 shows the training process that each group model used.

We referred to the entire training and evaluation process for one model as a trial. A total of 30 group model trials were performed in our experiments. Because non-eating training samples were undersampled randomly to form training sets, the performance of each group model varied during evaluation with the Multiday dataset. Metrics were averaged across all trials in order to provide an unbiased estimate of the group model's overall performance.

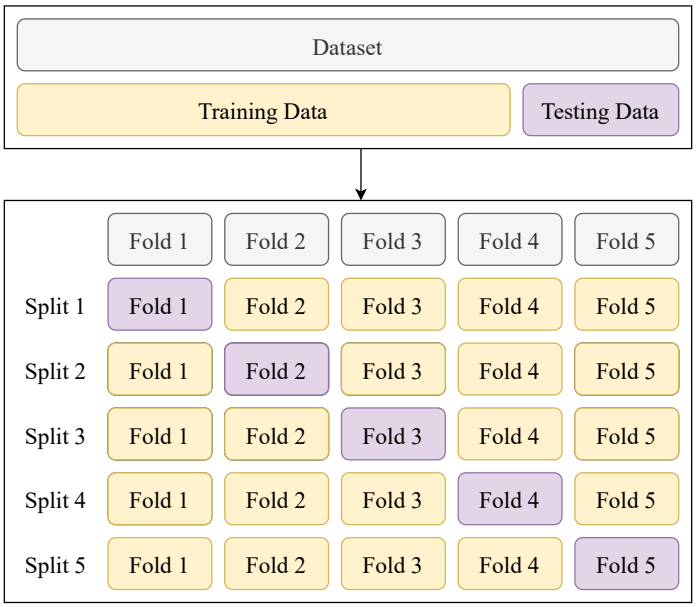


Figure 2.8: An illustration of k-fold cross validation where $k = 5$. Final performance metrics are evaluated by averaging the performance of each individual fold.

2.4.2 Individual Model Training

Individual models were trained for each subject from the Multiday dataset. The number of days recorded for each subject in the Multiday dataset ranged from 10-24. Due to the limited amount of data available for each subject, k-fold cross validation was used when training individual models. When using k-fold cross validation, all data for a given dataset is split into k equally sized partitions, or folds. One fold is withheld for testing while the others are used for training. This process is repeated for k iterations using a different fold for testing at each iteration. Final performance metrics are calculated by averaging across all k iterations. The individual model training process used 5-fold cross validation. Folds for each individual in the Multiday dataset were composed of entire days of data. This way, the training and testing sets for individual models did not contain any overlapping data from the same wrist motion recording. Figure 2.8 provides an illustration of 5-fold cross validation.

Days in each training set were split into windows and balanced as outlined in Section 2.2. 10% of all training windows were used as a validation set. Only models that showed the lowest validation loss during training were saved and used for evaluation, just as with the group models. 30 total trials were performed when creating individual models. Each trial used 5-fold cross validation

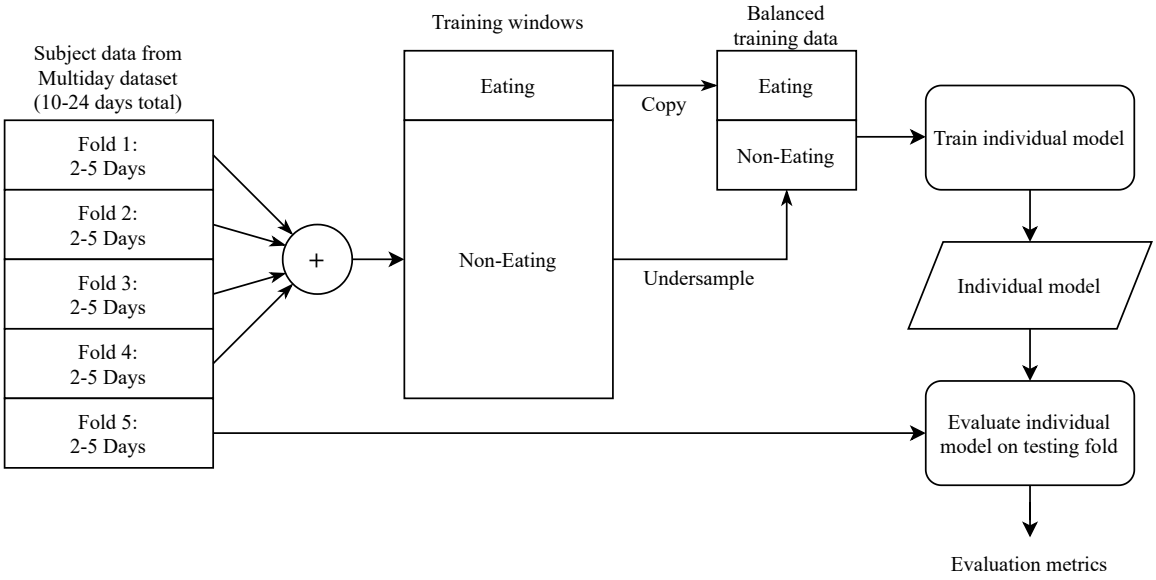


Figure 2.9: The process for training and evaluating an individual model.

for each subject in the Multiday dataset. This meant that each trial produced 40 different models and that 1200 individual models were trained and evaluated in total. Performance was first averaged across folds for each subject before being averaged across trials. Figure 2.9 illustrates the training procedure used for each individual model.

2.4.3 Transfer Learning Model Training

Transfer learning models were trained using the basic steps outlined in Section 1.2.4.3. First, a group model trained on the CAD dataset was loaded and its feature extraction layers were frozen. For our CNN architecture, these layers included three one-dimensional convolutional layers and a one-dimensional global average pooling layer (see Section 2.3). Next, the classification head consisting of two fully-connected layers was reset and trained using individual data from the Multiday dataset. After training the classification head, feature extraction layers were unfrozen and fine-tuned with individual data.

Four different types of transfer learning models were created by using different fine-tuning methods. The first type of transfer learning model was configured to fine-tune all of its convolutional layers. The second type of transfer learning model was configured to fine-tune only its top two (second and third) convolutional layers. The third type of transfer learning model was configured to

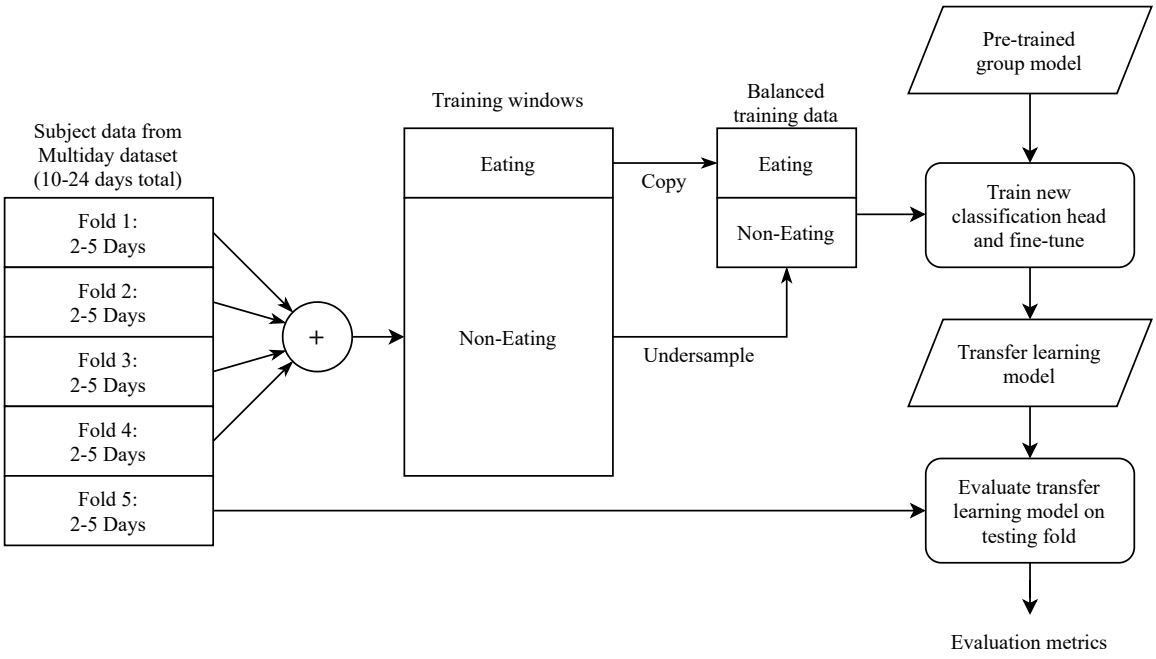


Figure 2.10: The process for training and evaluating a transfer learning model.

train only its top (third) convolutional layer. The final type of transfer learning model did not use fine-tuning at all, meaning that only its classification head underwent any training. This was done in order to determine what fine-tuning procedure was most optimal.

Transfer learning models were trained for each subject from the Multiday dataset. 5-fold cross validation was used by splitting subject data by day, just as with the individual models. Days in each training set were then split into windows and balanced as outlined in Section 2.2. 10% of all training windows were used as a validation set. Only models that showed the lowest validation loss during classification head training were used for fine-tuning. No model checkpointing was used during fine-tuning, meaning each model was saved after all fine-tuning epochs. 30 total trials were performed for each type of transfer learning model. During each trial, a different group model was loaded as a foundation for transfer learning. By using 5-fold cross validation, each trial produced 40 different models. This resulted in the creation of 1200 total models for each type of transfer learning model. Performance was first averaged across folds for each subject before being averaged across trials. Figure 2.10 illustrates the training procedure used for each transfer learning model.

2.4.4 Hyperparameters

Hyperparameters are configurable values that can be used to adjust the learning process of a neural network. The hyperparameters used in our experiments are provided:

1. **Loss Function**: Our experiments followed a binary classification paradigm. Time-series windows were labeled as either eating (1) or non-eating (0). Therefore, we used the binary cross-entropy loss function. Binary cross-entropy loss is a specific version of general cross-entropy loss (see equation 1.2). Binary cross-entropy loss can be represented mathematically as:

$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N} y_n \cdot \log(P(y_n)) + (1 - y_n) \cdot \log(1 - P(y_n)) \tag{2.9}$$

   where N is the number of training samples and $y_i \in \{0, 1\}$ is the ground truth class.

2. **Number of Training Epochs**: Both the group model and the individual models were trained for 30 epochs. Each transfer learning model was initially trained for 30 epochs with only their classification layers left unfrozen. Several higher values were tested for each model type; however, no further improvements in performance were observed beyond 30 epochs.

3. **Number of Fine-Tuning Epochs**: After initially training each transfer learning model for 30 epochs, they were fine-tuned for 10 epochs. During these epochs, the feature extraction layers were unfrozen and allowed to learn. Different feature extraction layers were either frozen or allowed to fine-tune depending on the transfer learning model type (see Section 2.4.3). By using a relatively low number of fine-tuning epochs, unfrozen feature extraction layers were able to better fit to individual data without completely overwriting their original weights learned from the group dataset.

4. **Optimizer**: The Adaptive Moment Estimation (Adam) optimizer was used to train all group, individual, and transfer learning models (this included fine-tuning transfer learning models). Adam uses stochastic gradient descent with a dynamic learning rate for different trainable parameters in the network.

5. **Learning Rate**: The default initial learning rate for the Adam optimizer is 0.001. This value was used when training all group and individual models. The default value was also used when training the classification head of each transfer learning model. When fine-tuning each

transfer learning model, the learning rate was decreased to 0.0001. Using this smaller learning rate for fine-tuning allowed the transfer learning models to successfully fit pre-trained feature extraction weights to individual training data without completely overwriting them.

6. **Batch Size**: We used mini-batch gradient descent in order to train all of our models. A smaller batch size has been shown to decrease both network training time and memory requirements, which can be useful if accelerators such as GPUs are available; however, a smaller batch size will also provide less accurate gradient estimations. Conversely, a batch size that is too large will lead to long training times and memory limitations that may prohibit use of certain hardware. Since the CAD and Multiday datasets were relatively small, a larger batch size of 256 was used when training all group, individual, and transfer learning models. This allowed each network to converge more consistently and reduced model volatility.

7. **Model Checkpointing**: Model checkpointing was used in order to save the best model during training epochs. Each model type used a 10% validation split when training. Only the model with the lowest validation loss was saved during training. Model checkpointing was only applied to classification head training for transfer learning models and was not used for fine-tuning.

2.4.5 Hardware and Software

All models were trained using TensorFlow, a software library built for machine learning [1]. TensorFlow version 2.2 for Python was used. All model training and evaluation was performed on Clemson University's Palmetto Cluster, a high performance computing (HPC) system. Compute nodes used on this system were supplied with 40 cores, 372 GB of RAM, and 2 NVIDIA Tesla V100 GPUs.

2.5 Post-Processing

After training, each model was evaluated using entire days of unseen data. For each day in a model's testing set, a sliding window procedure was used in order to generate a continuous series of testing windows. These windows were evaluated by their corresponding model in order to produce a sequence of values representing the probability of eating ($P(E)$) throughout the recording. Using

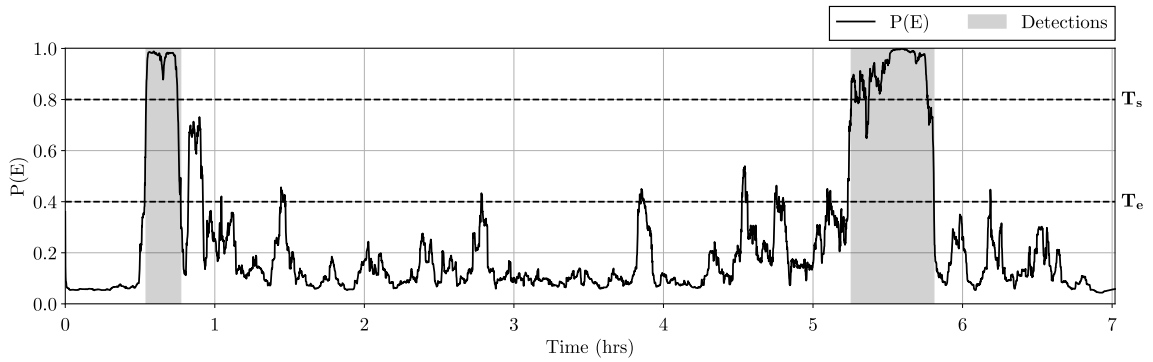


Figure 2.11: The probability of eating $P(E)$ throughout an entire day of recording with meal segments detected by the hysteresis thresholds. T_s indicates the starting threshold that the $P(E)$ curve must exceed to begin a meal detection. T_e indicates the ending threshold that the $P(E)$ curve must fall below in order to end a meal detection.

this $P(E)$ curve, a hysteresis thresholding method developed previously in our research group was used to detect periods of eating [40].

The hysteresis method used two different thresholds in order to detect meals from a daily $P(E)$ sequence. First, the $P(E)$ sequence for a recording had to exceed a starting threshold T_s in order to mark the start of an eating episode. Next, the $P(E)$ sequence had to fall below an ending threshold T_e in order to mark the end of an eating episode. This approach was taken because previous studies have found that eating behaviors and gestures tend to be more vigorous at the start of a meal. These gestures slowly become more subdued over the course of the meal. It was found in these previous works that a starting threshold of $T_s = 0.8$ and an ending threshold of $T_e = 0.4$ are optimal for accurately detecting meals while limiting false detections [40]. A stride of $S = 5$ seconds was used to evaluate days of data in order to avoid excessive computation times. Figure 2.11 shows how the hysteresis thresholds detect meal segments from a daily $P(E)$ curve. Once all meal detections have been found using the hysteresis method, evaluation metrics are calculated by comparing the detections to a ground truth.

2.6 Evaluation Metrics

Two types of evaluation metrics were used to assess the performance of each model: time metrics and episode metrics. This section examines both types of metrics as well as the measures we took to limit model volatility.

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

Table 2.3: Eating Classifier Confusion Matrix [34]

2.6.1 Time Metrics

Time metrics were calculated by comparing detections produced during post-processing to ground truth values at each time point where an evaluation window was produced. When performing evaluation, the ground truth label at the center point of each window was used as the label for the entire window. Time metrics were calculated using the four outcomes found in a confusion matrix. A true positive (TP) was identified if the classifier and ground truth both detected eating at the same time point. A true negative (TN) was produced if the classifier and ground truth both detected non-eating at the same time point. A false positive (FP) was recorded if the classifier predicted eating and the ground truth showed non-eating at the same point. Lastly, a false negative (FN) occurred if the classifier predicted non-eating at the same time point where the ground truth listed eating. The number of actual positive cases (P) can be calculated using the sum of all true positives and false negatives (P = TP + FN). Similarly, the total number of actual negative cases (N) can be calculated by summing all true negatives and false positives (N = TN + FP). Table 2.3 shows a confusion matrix with all possible classification outcomes. Figure 2.12 illustrates how ground truth meals and detected meals were compared to produce each time classification outcome.

The time metric that we used to quantify model performance was weighted accuracy (Acc_W).

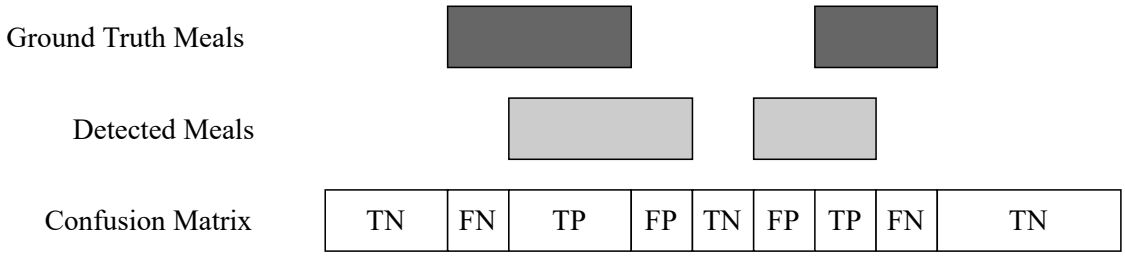


Figure 2.12: Labeling of eating time metrics using ground truth meals and model detections: true positive (TP), true negative (TN), false positive (FP), and false negative (FN).

Weighted accuracy was calculated using:

$$\text{Acc}_W = \frac{W \cdot \text{TP} + \text{TN}}{W \cdot (\text{TP} + \text{FN}) + (\text{TN} + \text{FP})} = \frac{W \cdot \text{TP} + \text{TN}}{W \cdot \text{P} + \text{N}} \tag{2.10}$$

where W represented the ratio of non-eating time points to eating time points in recordings. Weighted accuracy mitigated the class imbalance between eating and non-eating samples to provide an unbiased measurement of model performance.

2.6.2 Episode Metrics

Episode metrics were calculated by comparing the overlap between detected meals and ground truth meals. Episodes were first categorized using three of the four outcomes found in a confusion matrix. A true positive occurred if any overlap existed between a detected meal and a ground truth meal. A false positive was indicated by a detected meal that had no overlap with any ground truth meals. A false negative was identified if a ground truth meal did not overlap with any detected meals. True negatives were not considered when calculating episode metrics. Figure 2.13 shows how ground truth meals and detected meals were compared to produce each episode categorization.

Two episode metrics were used to quantify model performance in our experiments. The first was true positive rate (TPR), also known as sensitivity. TPR was calculated using:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.11}$$

TPR was used in order to determine what percentage of ground truth meals were successfully

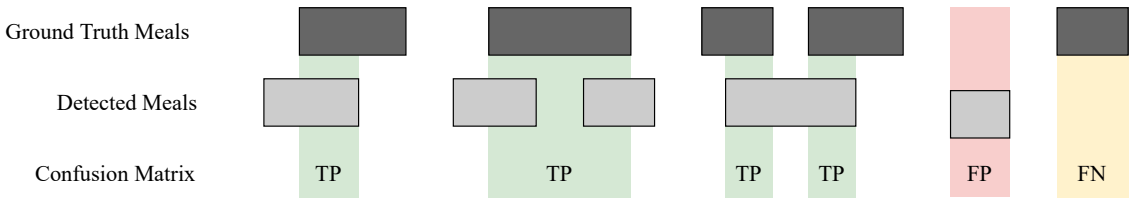


Figure 2.13: Labeling of eating episode metrics using ground truth meals and model detections: true positive (TP), false positive (FP), and false negative (FN).

detected. The second episode metric used was the false positive to true positive ratio (FP/TP). FP/TP was used in order to observe how prone a network was to false detections. There is typically a trade-off between TPR and FP/TP. As TPR increases, it is common for FP/TP to increase as well. Conversely, as TPR decreases, FP/TP will often decrease. Ideally, a balance between these two metrics should be sought that provides a relatively high TPR and low FP/TP.

2.6.3 Training Time

In order to measure the training time of each individual model and transfer learning model, a custom callback was written using TensorFlow [1]. This allowed the collection of accurate timing metrics for each model without factoring in overhead caused by model checkpointing. Total training time was calculated by summing the training times of each epoch. Timings for all individual models include classification head training time as well as fine-tuning time. Timing values for the group model are not considered in our experiments.

2.6.4 Model Volatility

Model volatility refers to the differences in performance that are observed when training and evaluating a model using the same data. One source of model volatility for all models was the selection of non-eating samples during undersampling. Another source of volatility for the transfer learning models and individual models was the selection of folds during 5-fold cross validation. In order to gather accurate performance metrics and mitigate model volatility, each type of model was trained and evaluated 30 times, as outlined in Section 2.4. To get a measure of model volatility, standard deviations for each performance metric were calculated across all runs.

Chapter 3

Results

This chapter examines the results gathered from our transfer learning experiments. First, the effects of fine-tuning different feature extraction layers are evaluated in Section 3.1 to determine the optimal transfer learning configuration. Next, the group model, individual model, and transfer learning model are compared in Section 3.2. Section 3.3 analyzes the performance differences per subject to see if transfer learning works equally well for all individuals. Finally, the volatility of each type of model is inspected in Section 3.4.

3.1 Fine-Tuning Transfer Learning Model Layers

Table 3.1 shows the average weighted accuracy (Acc_W), episode TPR, and episode FP/TP that were recorded for each fine-tuning configuration across all subjects from the Multiday dataset. Each row of the table shows the performance metrics for a different configuration. From the table, we observe that weighted accuracy and episode TPR both increase as more layers in the CNN are fine-tuned. These metrics improved most notably when the second convolutional layer was unfrozen, which can be seen by comparing the second and third rows of data in Table 3.1. Fine-tuning beyond two layers showed minor improvements to weighted accuracy and episode TPR. Using any amount of fine-tuning resulted in only a slight increase in FP/TP. We therefore conclude that fine-tuning all layers yields the best possible performance. Configuration metrics for each subject from the Multiday dataset are provided in the appendices for completeness.

Table 3.2 shows the average training time for each fine-tuning configuration across all sub-

Fine-Tuned Layers	Acc_W (%)	TPR (%)	FP/TP
0	73	63	1.20
1	75	69	1.42
2	80	80	1.43
3	80	81	1.40

Table 3.1: Average weighted accuracy (Acc_W), episode TPR, and episode FP/TP values measured across all subjects for each type of fine-tuning configuration.

Fine-Tuned Layers	Training Time (s)
0	53.4
1	70.9
2	72.4
3	86.6

Table 3.2: Average 5-fold cross validation training time measured across all subjects for each type of fine-tuning configuration.

jects from the Multiday dataset. We see from this table that the time spent training the classification head of the network (represented by 0 fine-tuned layers in the table) takes approximately one minute and is the largest contributor to training time. Subsequent rows in the table show that fine-tuning different layers in the network takes approximately 20-30 seconds, providing a small amount of additional training time. Despite the addition of this fine-tuning time, overall training time was low for all configurations. We therefore conclude that the cost of fine-tuning all layers is acceptable. The rest of the results shown in this chapter for transfer learning use fine-tuning for all three convolutional layers.

3.2 Group vs. Individual vs. Transfer Learning

Table 3.3 shows the average weighted accuracy (Acc_W), episode TPR, and episode FP/TP for the group model, individual model, and transfer learning model across all subjects from the Multiday dataset. We see from this table that the individual model shows an 8% higher Acc_W than the group model on average (81% vs. 73%), which demonstrates that learning wrist motion patterns by training on individualized data leads to improved classification performance. The transfer learning model reaches almost the same Acc_W as the invididual model on average (80% vs. 81%).

In terms of average episode metrics, the individual model achieves a much higher TPR than

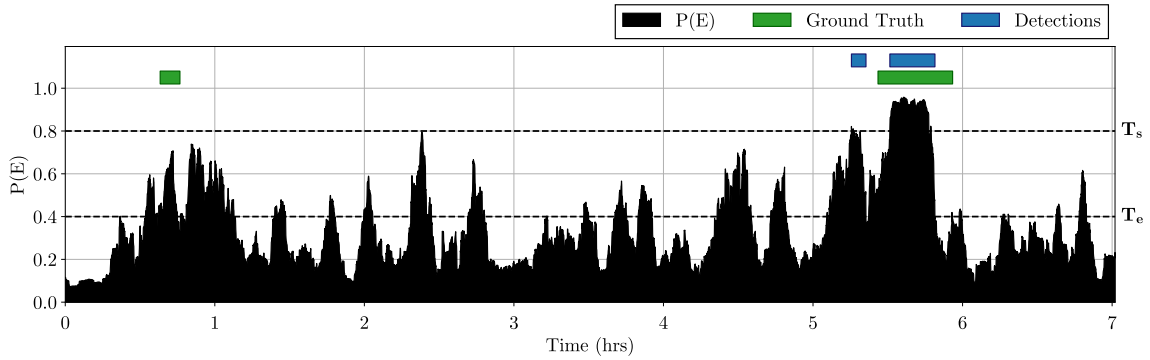
Model	Acc_W (%)	TPR (%)	FP/TP
Group	73	64	1.39
Individual	81	87	2.45
Transfer	80	81	1.40

Table 3.3: Average weighted accuracy (Acc_W), episode TPR, and episode FP/TP values measured across all subjects for each type of model.

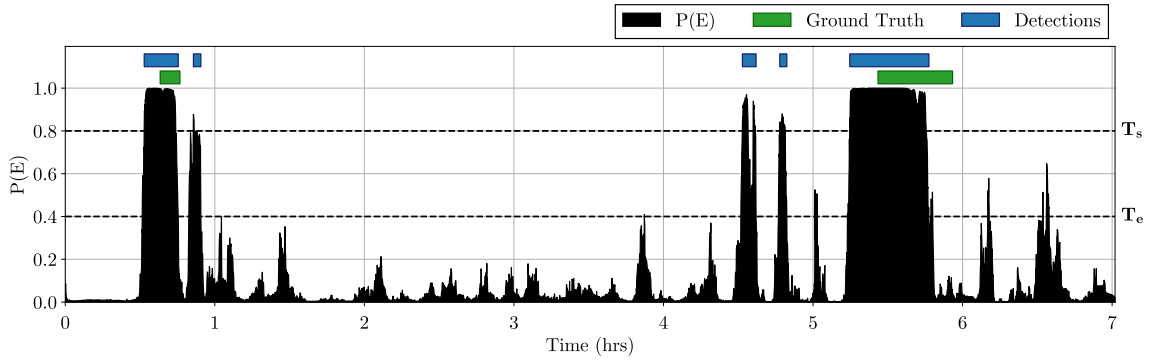
the group model (87% vs. 64%), but also yields an FP/TP approximately 76% higher than the group model (2.45 vs 1.39). The transfer learning model achieves nearly the same TPR as the individual model (81% vs 87%) but with an FP/TP comparable to the group model (1.40 vs 1.39). This may be because the individual model only has a limited number of days to learn all possible wrist motion patterns, whereas the transfer learning model can benefit from both the group data and individual data.

Figure 3.1 shows an example of the episode evaluation results for a full day of recording from the group model (3.1a), individual model (3.1b), and transfer learning model (3.1c). This day contained two meals reported in its ground truth file. It can be observed that the plot for the group model in this figure shows a large amount of noise, with $P(E)$ values commonly near 0.5. This shows that the group model had difficulty separating eating samples from non-eating samples. Overall, the group model correctly identified one meal, missed one meal, and had one false detection. Both the individual model and transfer learning model show less noise in their respective plots, with $P(E)$ values that mostly trend towards 1.0 and 0.0. This shows that the individual and transfer learning models were able to more confidently separate eating samples from non-eating samples in the recording. The individual model was able to correctly identify both meals but showed three false detections. The transfer learning model also correctly identified both meals and did not have any false detections.

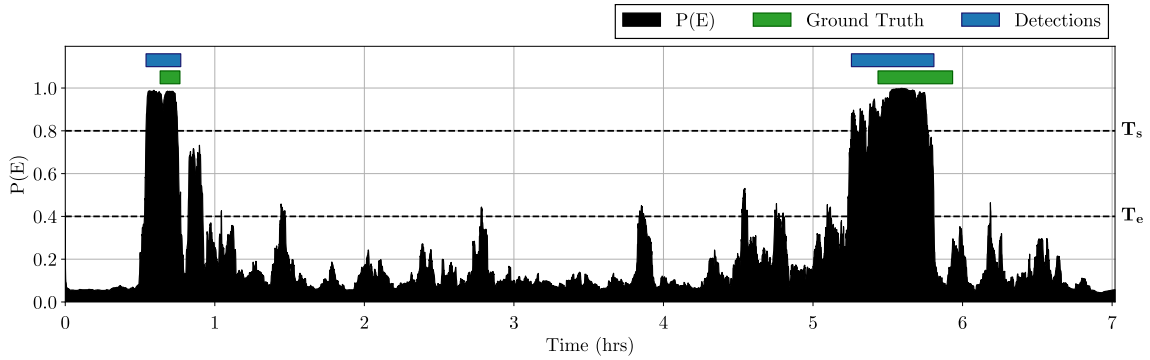
Figure 3.2 shows another example of episode evaluation on a full day of recording for each model. This day also contained two meals reported in its ground truth file. The group model (3.2a) showed low $P(E)$ values throughout the entire recording and was unable to detect either meal. The individual model (3.2b) was able to identify both meals but showed seven different false detections. The individual model was not able to detect most of the second meal approximately eight hours into the recording. This did not matter when calculating episode metrics, but would decrease time weighted accuracy. The transfer learning model (3.2c) was also able to detect both meals in the



(a) Group model episode evaluation: TP = 1, FP = 1, and FN = 1.

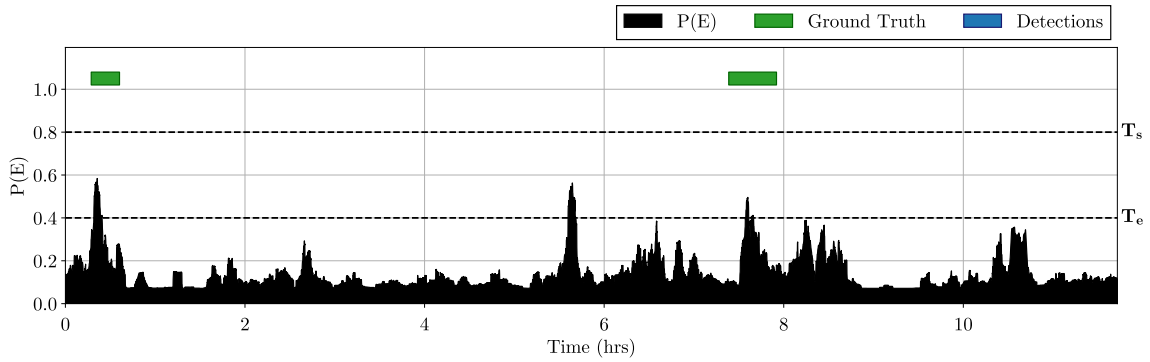


(b) Individual model episode evaluation: TP = 2, FP = 3, and FN = 0.

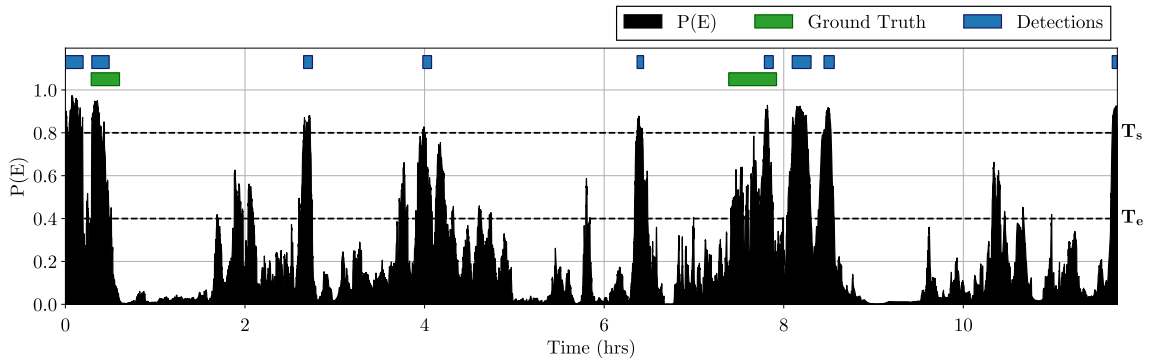


(c) Transfer learning model episode evaluation: TP = 2, FP = 0, and FN = 0.

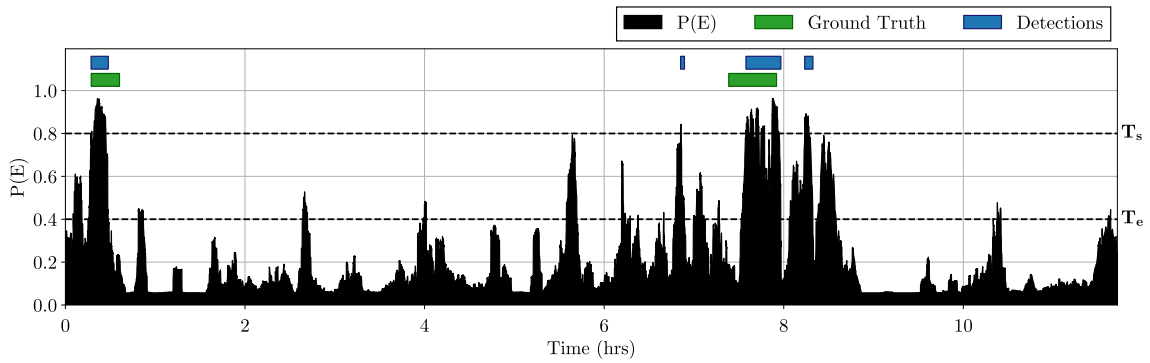
Figure 3.1: Example of episode evaluation performed on a full day of recording for each type of model.



(a) Group model episode evaluation: TP = 0, FP = 0, and FN = 2.



(b) Individual model episode evaluation: TP = 2, FP = 7, and FN = 0.



(c) Transfer learning model episode evaluation: TP = 2, FP = 2, and FN = 0.

Figure 3.2: Example of episode evaluation performed on another full day of recording for each type of model.

Subject	$\mathbf{Acc}_W$ (%)		
	Group	**Individual**	**Transfer**
001	82	90	90
002	90	93	94
003	54	71	62
004	60	75	74
005	83	82	83
006	63	82	83
007	73	67	69
008	77	89	89

Table 3.4: Average weighted accuracy (Acc_W) for all subjects from the Multiday dataset using each type of model.

recording and showed only two false positives. In addition, it can be observed that the transfer learning model was able to detect more of the second meal, which would lead to an increase in weighted accuracy.

In terms of training time, performing 5-fold cross validation took 98.9 seconds on average for the individual model. Thus, transfer learning saves approximately 12% training time while also providing knowledge shared by the group model. Training times could be reduced further by fine-tuning fewer convolutional layers but this would also lead to a decrease in performance.

3.3 Per Subject Variability

Table 3.4 shows the weighted accuracy across all three model types for each subject from the Multiday dataset. Similarly, Table 3.5 shows both episode metrics for each subject using all three model types. These tables allow us to observe which type of model performed best for each subject. From these tables, it can be observed that the transfer learning model had the highest weighted accuracy for most subjects (6/8). The individual model only outperformed the transfer learning model when evaluating subject 003 and 004. For subject 004, this difference was only 1%. For subject 003, however, this difference was 9%.

Upon closer examination of Table 3.4 and Table 3.5, it was observed that all model types performed poorly for subject 003. Most notably, the group model only detected 12% of all eating episodes for subject 003. Using the individual model for this subject, 76% of eating episodes were detected, but this dropped to 42% for the transfer learning model. This was especially unusual

Subject	TPR (%)			FP/TP		
	Group	**Individual**	**Transfer**	**Group**	**Individual**	**Transfer**
001	80	98	94	1.73	0.95	0.40
002	87	89	93	0.14	0.88	0.29
003	12	76	42	0.67	2.38	1.81
004	38	85	72	0.17	2.72	1.45
005	75	78	77	2.10	3.66	1.82
006	48	97	96	0.44	2.99	2.01
007	89	80	76	4.16	4.70	2.15
008	81	92	96	0.67	1.30	1.30

Table 3.5: Average episode metrics for all subjects from the Multiday dataset using each type of model.

since subject 003 had the most days of recorded data as well as the highest number of reported meals according to Table 2.1. Upon closer examination of the data for subject 003, it was discovered that many of their meals only lasted 1-5 minutes. It was shown in previous work that the CNN architecture used for all models had difficulty identifying meals less than its window length of $W = 6$ minutes [40]. In addition, it was shown in Section 2.1 that several meals had to be removed from the ground truth files of subject 003. This suggests that the participant may have misinterpreted the proper usage of the Shimmer3 IMU device. Another possibility is that the participant had highly irregular eating behaviors compared to the average group behaviors.

To study this further, we excluded subject 003 and recalculated the average performance metrics across the remaining seven subjects for all three model types. Table 3.6 shows our findings for this experiment. By excluding subject 003, the performance metrics of the group model and transfer learning model increase significantly compared to the original findings shown in Table 3.3. Most notably, the TPR of the group model increases from 64% to 71% and the TPR of the transfer learning model increases from 81% to 86%. The TPR of the individual model also increases slightly from 87% to 89%. Additionally, the transfer learning model shows the highest Acc_W of 83% and ties with the group model for the lowest FP/TP of 1.34. Although the transfer learning model receives a marked improvement in average performance metrics, this comparison shows that transfer learning may not be appropriate for all individuals. If personal eating behaviors fall far outside group behaviors, it may be better to exclude group data and only train an individual model.

Model	Acc_W (%)	TPR (%)	FP/TP
Group	75	71	1.34
Individual	82	89	2.46
Transfer	83	86	1.34

Table 3.6: Average weighted accuracy (Acc_W), episode TPR, and episode FP/TP values measured across all subjects excluding subject 003 for each type of model.

Model	Acc_W (%)	TPR (%)	FP/TP
Group	±3	±6	±0.66
Individual	±1	±3	±0.33
Transfer	±1	±4	±0.28

Table 3.7: Standard deviations for all evaluation metrics over 30 runs using each type of model.

3.4 Model Volatility

Table 3.7 shows the standard deviations for all evaluation metrics over 30 runs using each type of model. It can be seen from this table that the standard deviations for all group model metrics are noticeably higher than those of the individual model and transfer learning model. This is most likely because there is a much larger variety of wrist motion patterns represented in a group compared to an individual. The standard deviations for all transfer learning model metrics are comparable to those of the individual model. This shows that the fine-tuning process greatly reduces the volatility of the baseline group model during transfer learning.

Using our knowledge of model volatility, we compared the results of our experiments with the group model and individual model to those of previous work [51] in order to determine if previous results were replicated. Table 3.8 shows our findings. The table shows that the previous findings reported in [51] fall within 1-2 standard deviations of our findings but do not match our averages. This shows the importance of performing multiple runs when evaluating machine learning models, especially when data availability is limited.

	Acc_W (%)		TPR (%)		FP/TP	
	Group	Individual	Group	Individual	Group	Individual
Previous Work [51]	78	82	74	80	1.53	1.75
30 Runs Results	73±3	81±1	64±6	87±3	1.39±0.66	2.45±0.33

Table 3.8: Comparison of average evaluation metrics for group and individual models across 30 runs to results of previous work [51].

Chapter 4

Conclusion

This thesis explored the novel concept of utilizing transfer learning to improve individualized eating detection models. The results gathered from the experiments of this work answer the three questions posed in Section 1.4:

1. Can transfer learning be employed to increase the performance of individualized eating detection models by utilizing features learned from a group eating detection model?

   The transfer learning model showed a weighted accuracy (Acc_W) of 80%, an episode TPR of 81%, and an episode FP/TP of 1.40. Compared to the individual model, this was a 1% decrease in Acc_W, a 6% decrease in episode TPR, and a 43% improvement in episode FP/TP. Although the transfer learning model shows slight decreases in Acc_W and episode TPR, false detections were substantially reduced compared to the individual model.

2. How much performance can be gained by using transfer learning to create individualized models? Does this performance gain vary depending on the individual?

   It was shown in Section 3.3 that for most subjects in the Multiday dataset, the transfer learning model was able to achieve the highest weighted accuracy (Acc_W) while maintaining a high episode TPR and low episode FP/TP; however, there was a noticeable outlier in Subject 003. Group model and transfer learning model performance metrics for Subject 003 were much lower than individual model metrics. Upon closer examination, it was shown that Subject 003's data differed greatly from the data of other individuals. By excluding Subject 003's data and reevaluating, the transfer learning model showed a weighted accuracy of 83%, an episode TPR

of 86%, and an episode FP/TP of 1.34. Compared to the individual model, this was a 1% increase in Acc_W, a 3% decrease in episode TPR, and an improvement of 46% in episode FP/TP. This shows that, for most individuals, transfer learning can improve eating detection; however, some individuals may benefit from excluding group data and only using individual data.

3. Can transfer learning speed up the training process for individualized models?

   The transfer learning model configuration that utilized fine-tuning for all convolutional layers was able to decrease training time by approximately 12% compared to the individual model. As less layers were fine-tuned, training time was shown to decrease further. This shows that there is a trade-off between training time and evaluation performance when fine-tuning transfer learning models.

4.1 Limitations

Perhaps the largest advantage of the transfer learning model is also its greatest limitation: it uses a group model. Training a group model requires the collection of a large dataset, which can be both costly and time-consuming. Thus, transfer learning may be better suited for tasks that have large datasets that are readily available.

Another limitation of this study was the size of each dataset that was used. Although the CAD dataset contains data from 351 different people, having a larger dataset would be beneficial so that knowledge of more eating behaviors could be shared with the transfer learning model. Similarly, the Multiday dataset only contains wrist motion data for eight individuals. Collecting data from more participants could help quantify the robustness of the transfer learning model. More data could also be collected per individual so that an investigation could be performed examining how many days of data are required to create an accurate individual model. The limitations of this research do provide suggestions for future work, however.

4.2 Future Work

There are a number of additional experiments that could be performed as future work. One such experiment could examine adjusting network parameters to improve performance in individual

models. For example, the effect of window size (W) on individual accuracy could be examined. Similarly, changing the hysteresis thresholds T_s and T_e could be examined.

Another experiment related to transfer learning that could be explored is altering the amount of training data. Instead of performing cross validation, a set number of days could be used for training. This could help determine how many days of individual data are actually needed when the group model is already being used as a baseline. If this amount of data is found to be low, data could be collected more quickly from individuals.

An investigation into the causes of model volatility could also be performed. One of the most likely causes of model volatility is the choice of non-eating windows during undersampling. Future work could include the creation of a method that allows the most optimal non-eating windows to be selected for training. In addition to reducing model volatility, this could also improve network performance.

Another future study could examine different network architectures for the eating detector. Adding more convolutional layers to the CNN could result in better network performance. Other hidden layer types could also be used. For example, if an RNN was used, the effects of fine-tuning recurrent layers could be explored.

Appendices

Appendix A Per Subject Fine-Tuning Metrics

This appendix includes the evaluation metrics that were recorded for each subject when examining different fine-tuning configurations.

	Weighted Accuracy ($\mathbf{Acc}_W$)			
Subject	**Fine-Tune 0 Layers**	**Fine-Tune 1 Layer**	**Fine-Tune 2 Layers**	**Fine-Tune 3 Layers**
001	79	80	89	90
002	70	73	93	94
003	53	54	60	62
004	73	74	74	74
005	82	83	83	83
006	77	78	82	83
007	63	68	70	69
008	86	87	89	89
Average	**73**	**75**	**80**	**80**

Table 1: Weighted accuracy (Acc_W) values for all Multiday subjects using different fine-tuning configurations.

	Episode TPR (%)			
Subject	**Fine-Tune 0 Layers**	**Fine-Tune 1 Layer**	**Fine-Tune 2 Layers**	**Fine-Tune 3 Layers**
001	67	72	90	94
002	48	55	92	93
003	16	17	38	42
004	65	68	71	72
005	72	75	75	77
006	84	89	94	96
007	55	75	79	76
008	96	98	99	96
Average	**63**	**69**	**80**	**81**

Table 2: Episode TPR values for all Multiday subjects using different fine-tuning configurations.

Subject	Episode FP/TP			
	Fine-Tune 0 Layers	**Fine-Tune 1 Layer**	**Fine-Tune 2 Layers**	**Fine-Tune 3 Layers**
001	0.90	1.22	0.73	0.40
002	0.03	0.03	0.26	0.29
003	1.63	1.95	1.81	1.81
004	0.72	0.89	1.30	1.45
005	1.32	1.97	1.69	1.82
006	1.45	1.91	2.17	2.01
007	1.88	1.44	1.90	2.15
008	1.68	1.97	1.61	1.30
Average	**1.20**	**1.42**	**1.43**	**1.40**

Table 3: Episode FP/TP values for all Multiday subjects using different fine-tuning configurations.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] O. Amft. A wearable earpad sensor for chewing monitoring. In *SENSORS, 2010 IEEE*, pages 222–227, 2010.

[3] O. Amft, M. Stäger, P. Lukowicz, and G. Tröster. Analysis of chewing sounds for dietary monitoring. In *UbiComp 2005: Ubiquitous Computing*, pages 56–72, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[4] Apple. Healthcare on apple watch. https://www.apple.com/healthcare/apple-watch/, 2021.

[5] Y. Bai, W. Jia, Z. Mao, and M. Sun. Automatic eating detection using a proximity sensor. In *2014 40th Annual Northeast Bioengineering Conference (NEBEC)*, pages 1–2, 2014.

[6] 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.

[7] Canalys. North american wearables market q2 2020. https://canalys.com/newsroom/canalys-north-american-wearables-market-Q2-2020, 2020.

[8] P.M. Cheng and H.S. Malhi. Transfer learning with convolutional neural networks for classification of abdominal ultrasound images. *Journal of digital imaging*, 30(2):234, 2017.

[9] D. Cook, K.D. Feuz, and N.C. Krishnan. Transfer learning for activity recognition: A survey. *Knowledge and information systems*, 36(3):537–556, 2013.

[10] W. Dietz and C. Santos-Burgoa. Obesity and its implications for covid-19 mortality. *Obesity (Silver Spring)*, 28(6):1005, 2020.

[11] 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, 2009.

[12] 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.

[13] 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.

[14] 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*, 25(2):568–576, 2021.

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

[16] H.I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller. Transfer learning for time series classification. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1367–1376, 2018.

[17] Fitbit. Health metrics. https://www.fitbit.com/global/us/technology/health-metrics, 2021.

[18] J.M. Fontana, M. Farooq, and E. Sazonov. Automatic ingestion monitor: A novel wearable device for monitoring of ingestive behavior. *IEEE Transactions on Biomedical Engineering*, 61(6):1772–1779, 2014.

[19] 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.

[20] C.M. Hales, M.D. Carroll, C.D. Fryar, and C.L Ogden. Prevalence of obesity among adults and youth: United states, 2015–2016. 2017.

[21] A. Hoover. Clemson cafeteria dataset. http://cecas.clemson.edu/ ahoover/cafeteria/, 2020.

[22] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, et al. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

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

[24] 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.

[25] 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, 2021.

[26] S.W. Lichtman, K. Pisarska, E.R. Berman, M. Pestone, H. Dowling, et al. Discrepancy between self-reported and actual caloric intake and exercise in obese subjects. *New England Journal of Medicine*, 327(27):1893–1898, 1992.

[27] P. Lopez-Meyer, O. Makeyev, S. Schuckers, E.L. Melanson, M.R. Neuman, and E. Sazonov. Detection of food intake from swallowing sequences by supervised and unsupervised methods. *Annals of Biomedical Engineering*, 38(8):2766–2774, 2010.

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

[29] 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.

[30] K. Mercer, L. Giangregorio, E. Schneider, P. Chilana, M. Li, and K. Grindrod. Acceptance of commercially available wearable activity trackers among adults aged over 50 and with chronic illness: a mixed-methods evaluation. *JMIR mHealth and uHealth*, 4(1):e4225, 2016.

[31] M. Narici, G.D. Vito, M. Franchi, A. Paoli, T. Moro, 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*, 21(4):614–635, 2021.

[32] 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, 2017.

[33] S.J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[34] A. Patyk. Detecting eating episodes from daily patterns of wrist motion using recurrent neural networks. Master's thesis, Clemson University, July 2021.

[35] Ariadna Quattoni, Michael Collins, and Trevor Darrell. Transfer learning for image classification with sparse prototype representations. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[36] 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.

[37] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[38] J. Salley, J. Scisco, E. Muth, and A. Hoover. A comparison of user preferences and reported compliance with the bite counter and the 24-hour dietary recall. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 56(1):2177–2180, 2012.

[39] E.S. Sazonov and J.M. Fontana. A sensor system for automatic detection of food intake through non-invasive monitoring of chewing. *IEEE Sensors Journal*, 12(5):1340–1348, 2012.

[40] S. Sharma. *Detecting Periods of Eating in Everyday Life by Tracking Wrist Motion - What is a Meal?* PhD thesis, Clemson University, July 2020.

[41] S. Sharma and A. 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, 2020.

[42] S. 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.

[43] Y. Shen, E. Muth, and A. Hoover. Recognizing eating gestures using context dependent hidden markov models. In *2016 IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pages 248–253, 2016.

[44] Shimmer. Shimmer3 imu unit. https://www.shimmersensing.com/products/shimmer3-imu-sensor, 2021.

[45] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[46] 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.

[47] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

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

[49] G.M. Turner-McGrievy, S. Wilcox, A. Boutté, B.E. Hutto, C. Singletary, E. Muth, and A. Hoover. The dietary intervention to enhance tracking with mobile devices (diet mobile) study: a 6-month randomized weight loss trial. *Obesity*, 25(8):1336–1342, 2017.

[50] Y. Wang, M.A. Beydoun, J. Min, H. Xue, L.A. Kaminsky, et al. 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, 02 2020.

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

[52] K. Weiss, T.M. Khoshgoftaar, and D. Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[53] T. Wen and R. Keyes. Time series anomaly detection using convolutional neural networks and transfer learning. *arXiv preprint arXiv:1905.13628*, 2019.

[54] World Health Organization. Noncommunicable diseases. https://www.who.int/news-room/fact-sheets/detail/noncommunicable-diseases, April 2021.

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

[56] S.M. Wright and L.J. Aronne. Causes of obesity. *Abdominal Radiology*, 37(5):730–732, 2012.

[57] Y. Zhu, Y. Chen, Z. Lu, S.J. Pan, G. Xue, et al. Heterogeneous transfer learning for image classification. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.