Segmentation and recognition of eating gestures from wrist motion using deep learning

---

A Thesis
Presented to
the Graduate School of
Clemson University

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

---

by
Yadnyesh Yoginish Luktuke
May 2020

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

Abstract

This research considers training a deep learning neural network for segmenting and classifying eating related gestures from recordings of subjects eating unscripted meals in a cafeteria environment. It is inspired by the recent trend of success in deep learning for solving a wide variety of machine related tasks such as image annotation, classification and segmentation. Image segmentation is a particularly important inspiration, and this work proposes a novel deep learning classifier for segmenting time-series data based on the work done in [25] and [30]. While deep learning has established itself as the state-of-the-art approach in image segmentation, particularly in works such as [2],[25] and [31], very little work has been done for segmenting time-series data using deep learning models.

Wrist mounted IMU sensors such as accelerometers and gyroscopes can record activity from a subject in a free-living environment, while being encapsulated in a watch-like device and thus being inconspicuous. Such a device can be used to monitor eating related activities as well, and is thought to be useful for monitoring energy intake for healthy individuals as well as those afflicted with conditions such as being overweight or obese.

The data set that is used for this research study is known as the Clemson Cafeteria Dataset, available publicly at [14]. It contains data for 276 people eating a meal at the Harcombe Dining Hall at Clemson University, which is a large cafeteria environment. The data includes wrist motion measurements (accelerometer x, y, z; gyroscope yaw, pitch, roll) recorded when the subjects each ate an unscripted meal. Each meal consisted of 1-4 courses, of which 488 were used as part of this research. The ground truth labelings of gestures were created by a set of 18 trained human raters, and consist of labels such as 'bite' used to indicate when the subject starts to put food in their mouth, and later moves the hand away for more 'bites' or other activities. Other labels include 'drink' for liquid intake, 'rest' for stationary hands and 'utensiling' for actions such as cutting the food into

bite size pieces, stirring a liquid or dipping food in sauce among other things. All other activities are labeled as 'other' by the human raters. Previous work in our group focused on recognizing these gesture types from manually segmented data using hidden Markov models [24],[27]. This thesis builds on that work, by considering a deep learning classifier for automatically segmenting and recognizing gestures.

The neural network classifier proposed as part of this research performs satisfactorily well at recognizing intake gestures, with 79.6% of 'bite' and 80.7% of 'drink' gestures being recognized correctly on average per meal. Overall 77.7% of all gestures were recognized correctly on average per meal, indicating that a deep learning classifier can successfully be used to simultaneously segment and identify eating gestures from wrist motion measured through IMU sensors.

Acknowledgments

I would like to thank a lot of people for their help, support, guidance and motivation during my time at Clemson University. First of all my parents, Yoginish Luktuke and Yamini Luktuke without whom I would never have had the chance to pursue my education at Clemson University in the first place. Thanks are also due to my younger brother Yadvindra Luktuke, who has been a pillar of strength, motivation and support.

I would like to sincerely thank my advisor, Dr. Adam Hoover for the encouragement, guidance and motivation while pursuing my graduate education here. Dr. Hoover you are astounding, and it has been a great pleasure to work under your guidance. Thank you for encouraging me, having patience with me, pointing out my mistakes no matter how many times I continued to make them and helping me fix these each time. Most importantly however, I'd like to thank you for trusting me to do good research. I would like to thank my committee members, Dr. Richard Brooks and Dr. Richard Groff for taking time out of their schedule and agreeing to be on my committee so that I could defend my thesis as planned, especially in this strange time. Dr. Brooks was the first professor with whom I spoke before coming here, and it was due to his help that I was able to secure my first assistantship here. I would also like to thank the department of Electrical & Computer Engineering at Clemson University, especially Dr. Daniel Noneaker and Dr. Harlan Russell for the financial support provided to me while pursuing my education here. Thanks to Jennifer Gooch and Jeanine Hayes who've helped me navigate quite a lot of paperwork related obstacles carefully.

Thanks to my friend Surya Sharma, who has been the most supportive and encouraging friend I could've hoped to find here in Clemson. I would also like to thank Dr. Apoorva Kapadia, Dr. Yu Lu and Jonathan Oakley who helped me greatly when I was first getting settled with a lot of my work in Clemson. Last, but certainly not the least I'd like to thank all of my friends for their company during the good times and help during the bad ones.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This work addresses the problem of segmenting time-series data using deep neural networks for identifying different activities that occur during a meal. The data for this research was collected from a watch-like device worn by subjects on their hand while eating an unscripted meal. Examples of activities include eating related movements such as taking a bite of food, drinking, cutting food up into bite size pieces and masticating. Identifying these activities is challenging because different people eat their meals differently, and hence the proportion of each activity is different for each person. Moreover a meal is also associated with activities completely unrelated to eating, such as gesturing to a friend, checking one's phone or simply resting one's hand before taking the next bite.

In data analysis segmentation is the grouping of data into sets such that each set exhibits similar characteristics. Data is grouped based on some homogeneity property that all examples from a given set exhibit strongly among themselves, while examples from different sets exhibit poorly among each other. For example when grouping points that lie on a circle we can use the distance from the center of the circle as a grouping property, since points that lie on the circle will have the same radius. In the same example points that lie on different circles, or points that lie outside the circle will have values greater than the radius and hence will be separated out. In the context of activity tracking from wrist motion data, segmentation is the grouping of activities with similar wrist movement into each corresponding category or 'gesture' type.

Owing to the variety in wrist movements possible for a large set of people, it becomes very difficult to design good features that can be used to characterize each segment of recorded activity. Hence deep learning is considered important in this study. Deep neural networks can learn a set of

Figure 1.1: Example of a person wearing a recording device while eating a meal.

representative features from the data provided, and are hence particularly useful in a scenario such as the one considered in this study.

Deep neural networks are part of the family of machine learning models that are trained using supervised learning. This means that deep learning neural networks require data to be labeled or categorized before being used. The process of fitting a deep neural network to data is called as training, and is one of the many steps that needs to be carefully designed.

Figure 1.1 shows an example of using the device to record eating activity. In the figure, a subject is wearing a recording device on their right hand while eating. As it can be seen from the central panel, eating in humans is characterized by three distinct movements of the wrist. These are making a bite of food ready, turning the hand towards the mouth and moving the hand to the mouth for actually taking a bite from the utensil (fork or spoon). It must be noted here, that the same three distinct movements have been observed when eating without using utensils as well.

The rest of this chapter is organized as follows, section 1.1 explains why this research is needed today, while section 1.2 gives a background on the sensors used to record wrist motion. Section 1.3 describes related work, including that done by previous researchers in identifying and characterizing segments from the same dataset. Section 1.4 gives a brief overview on deep learning, and an example of how deep learning is used in image segmentation that is closely related to this research. This chapter is concluded with section 1.5 which describes the novel approach taken in this research study.

1.1 Motivation

As per the World Health Organization (WHO) overweight and obesity are caused by the abnormal and excessive accumulation of body fat in individuals [33]. The body mass index (BMI) of an individual can be used as an estimate or indicator of whether the person is overweight or obese. BMI is calculated as the weight of a person in kilograms divided by their height in meters [33]. Based on this, a person having a BMI greater than or equal to 25 is considered overweight, while that greater than 30 is considered obese. Both of these conditions have been linked with several chronic diseases including cancer, cardiovascular diseases and diabetes.

As per WHO, obesity has nearly tripled since 1975 [34]. As per the 2016 estimate, more than 1.9 billion people worldwide that are 18 years or older have been classified as overweight. Amongst these 650 million people were obese. Additionally 40 million children under the age of 5, were reported to be overweight or obese in 2018, while in the age group of 5-19, 340 million children or adolescents were overweight or obese as per estimates in 2016 [34].

In the United States of America, the Centers for Disease Control and Prevention (CDC), have reported that the prevalence of obesity was more than 42% in 2017-2018. In nearly two decades, from 1999-2000 to 2017-2018, the prevalence of obesity has increased from 30.5% to 42.4%, while severe obesity has almost doubled from 4.7% to 9.2% [4]. The estimated annual cost of treating obesity or obesity related disorders in 2008 was $ 148 billion, and the cost for people with obesity $1,429 higher than that for people with normal weight [4]. Figure 1.2 shows the percentage of people who reported themselves as overweight or obese in the United States of America during the year 2018, as reported by the CDC. On seeing the figure closely we see that all states and territories had more than 20% of adults with obesity, with 22 states and Puerto Rico having obesity percentage in adults between 30% - 35%, while 9 states having more than 35% of their adult population as being overweight or obese [3].

As per WHO, the leading cause of obesity is the imbalance between the daily calorific intake and expenditure in humans. This is mainly caused by the largely increasing sedentary forms of work, transportation and increase in urbanization [34]. This increase in physical inactivity is also accompanied by increased consumption of energy-dense foods high in fat and sugars.

While there has been an increase in the number of devices that can monitor physical activity such as fitness-trackers and smartwatches, there is relatively little work done in the area of energy-

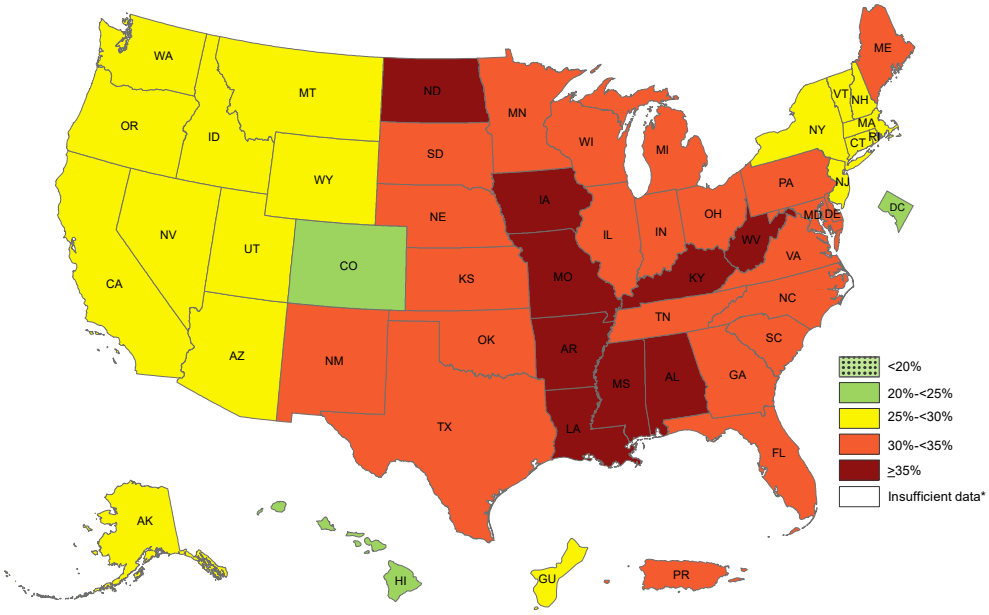


Figure 1.2: CDC: Self-reported obesity prevalence map (2018) [3].

intake monitoring. Conventional methods for monitoring energy intake, including self-reporting and 24-hour recalls such as those described in [12] and [26] are prone to under-reporting and under-estimation [27]. Additionally these methods are tedious, leading to non-compliance over time. All this makes monitoring energy intake over time a very challenging task.

Hence this research explores one of the many ways in which energy intake can be reliably tracked, over the duration of a meal. The foundational idea in our research group is to provide an end user an accurate estimate of the total calories consumed in a given period of time, with the additional goal of alerting people if they exceed their suggested daily intake. We believe that this will certainly push people to be more careful with their daily calorific intake and hopefully help to tackle the problem of growing obesity in the United States of America and even in other parts of the world.

1.2 Inertial Measurement Unit Sensors (IMU)

In order to record the wrist activity of a subject, a watch-like device needs to be designed using sensors capable of sensing motion such as wrist rotation, movement towards the food and back again to the mouth. This study considers the data recorded using accelerometers and gyroscopes,

which are themselves part of a large family of sensors fabricated using Micro-electromechanical technology (MEMS). The advantage of sensors fabricated using this technology, is low cost, small size and low power consumption [27]. Examples of other sensors from this family include magnetometers, microphones and optical sensors.

1.2.1 Accelerometers

An accelerometer is an electromechanical device that is capable of sensing acceleration that a body experiences due to an external force. This force experienced by the body can be static like gravitational force, or it could be dynamic such as the one experienced while moving one's wrist around in regular movement [27]. Based on their specific design, accelerometers can sense the acceleration, tilt and tilt angle, incline, rotation, vibration, collision or gravitational force acting on the body on which they are mounted [17].

Typically MEMS accelerometers are fabricated as capacitive devices, in which an inertial mass is attached to a spring and allowed to move along one axis. In addition to this mass, the sensor also contains fixed plates that are mounted along the axis of movement.

The basic principle of accelerometer sensing can be explained using Newton's second law of motion and Hooke's Law [27]. According to Newton's second law of motion, the force (F) experienced by a body is directly proportional to the mass of the body (m) and the acceleration (a) it experiences due to the external force.

$$F = m \times a \tag{1.1}$$

Based on Hooke's law, when a rigid mass attached to a fixed end with a flexible spring is displaced, the force experienced by the spring is directly proportional to the spring constant (k) and the actual displacement (x) experienced by the body. This can be expressed as:

$$F = -k \times x \tag{1.2}$$

The negative sign in Equation 1.2 indicates that the force experienced by the spring is in the direction opposite to that of the moving body. Combining Equations 1.1 and 1.2, we see that the acceleration

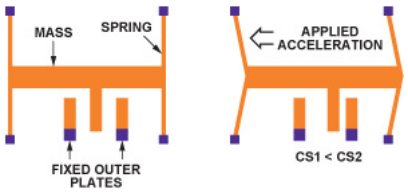


Figure 1.3: MEMS Accelerometer: Sensing principle [19].

is proportional to the displacement of the body, given by:

$$a = -\frac{k \times x}{m} \tag{1.3}$$

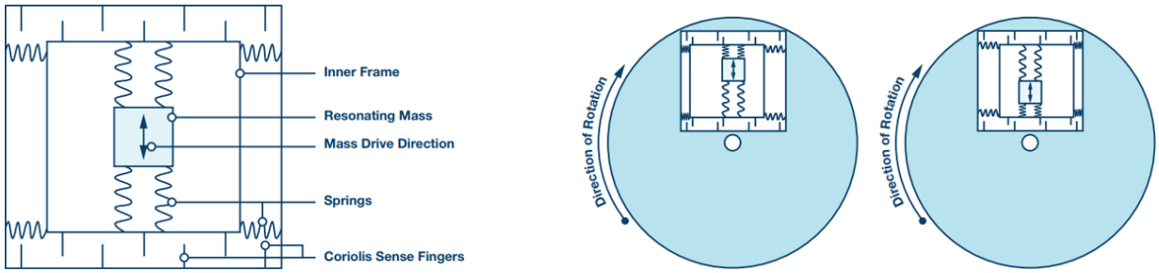
In practice, the acting forces displaces the inertial mass, causing capacitance change between the moving plate and the fixed plates. This change is sensed as a voltage (V) which can be converted to units such as meters/second^2 (m/s^2) or typically as a proportion of acceleration due to gravity $g = 9.81\, m/s^2$.

1.2.2 Gyroscopes

While accelerometers measure linear motion of a body along an axis, gyroscopes measure angular velocity of a body along an axis of rotation. This adds more degrees of freedom in order to accurately measure movement of a body in real-life [32]. The sensing principle of a gyroscope is based on the application of the Coriolis effect for a vibrating body that experiences angular motion. Specifically when a body having mass m oscillating with a velocity v, experiences an angular velocity Ω, the rotating platform exerts a force F that is directly proportional to the three quantities mentioned above. Corresponding to this, the sensing body experiences a reaction force given by [32],[27]:

$$F = -2m\Omega \times v \tag{1.4}$$

Similar to an accelerometer, gyroscopes sense angular velocity using differential changes in capacitance of sensing plates. The schematic of the sensing mass shown in Figure 1.4a, shows the oscillating mass tethered to the inner and outer substrates by compressible springs placed at 90 deg relative to each other. When the mass moves to the outer edges due to the Coriolis effect, the oscillating mass retains its motion, and experiences a force given by Equation 1.4. This compresses



(a) MEMS Gyroscope: Sensing mass.

(b) MEMS Gyroscope: Coriolis effect.

Figure 1.4: MEMS Gyroscope: Schematic and sensing principle [32].

the outer springs as shown in Figure 1.4b, which causes capacitance changes in the sensing teeth, which is converted to volts (V). Typically this sensor output needs to be converted to deg $/sec$

1.3 Related Work

Wearable sensors have been widely studied for monitoring energy intake in humans. This section describes a few of these sensors, and briefly explains their sensing mechanisms to the reader. For a comprehensive review, the interested reader is referred to chapter 1.4 of [27] as well as [22]. This section also describes why IMU sensors are thought to be the most suitable for the task of monitoring eating activities. It concludes by detailing some of the work done by our research group previously, and explains how this research builds on their foundational idea and takes it further.

1.3.1 Acoustic Sensors

As explained in [27], acoustic sensors, mainly microphones and piezoelectric strain-gauges are used for detecting sounds that are associated with eating activities such as chewing and swallowing. In [1], the authors considered a microphone located inside the ear canal to identify four different types of food based on their characteristic chowing sounds. However as explained in [27], such sensors suffer from environmental noise and hence reference microphones have been considered to eliminate environmental noise while detecting chewing in works such as [20] and [21]. In [9], the authors used a Bluetooth headset for sensing chewing sounds, and a Deep Boltzmann Machine for detecting periods of eating with an accuracy of 94.72% for in-the-field testing. This indicates that the idea of detecting eating activities through sounds in the ear canal or throat region is a viable idea. However acoustic sensor based systems still suffer from environmental and background noise,

limiting their use to laboratory studies or tests conducted under controlled environments [27].

1.3.2 Camera Based Systems

Unlike acoustic sensors which sense a proxy (chewing/swallowing) for detecting eating activities, camera based systems employ computer vision algorithms for detecting eating related activities and volume estimation using 3D techniques for estimating the energy intake in participants [27]. SenseCam originally developed in [13] for helping patients with Alzheimer's recollect their daily activities, was used in [10] for recording eating activity in a free-living environment. In this research study, 40 adult participants wore a SenseCam for 4 days (including 1 familiarization day) over a period of 15 days. The recorded data was stored, analyzed and reported to the subjects as a way of complimenting traditional self-reporting in free-living eating. In [29] the authors designed a wearable computer called the eButton, which could be worn inconspicuously by participants for monitoring daily activity such as eating and physical activity. The authors in [29] also proposed its application for other tasks such as understanding sedentary behavior in subjects, assisting blind and visually impaired people and helping older subjects suffering from dementia.

One concern regarding camera based systems, is the privacy of the individual using the device, as well as that of other individuals with whom he/she might interact. In addition camera based systems are cumbersome to wear, and are hence largely dependent on the compliance of the individual wearing these.

1.3.3 Smart Glasses For Monitoring Food Intake

In recent years, there has been an increase in the focus on using smart glasses for monitoring energy intake in participants. In [15] Huang et al., propose the idea of integrating an electromyography (EMG) sensor on glasses that can measure the muscle activity of the lower part of the temporalis muscle which touches the temple of the glasses worn by participants. This is one of the muscles associated with mastication. Hence activity in this muscle can be used to monitor food intake. The authors in [15] achieved 96% accuracy for counting the number of chewing cycles and up to 90.8% accuracy for classifying between five broadly different food types in a group of seven participants.

More recently the authors in [36] and [37] proposed a similar approach, an EMG sensor along with an accelerometer integrated into a 3D printed eyeglasses frame for monitoring chewing activity

and eating related episodes in a free-living environment. In [37] the authors considered a set of 10 participants for their study and their method achieved average food hardness classification accuracy of 94% and chewing cycle detection precision and recall of over 90% for their in-lab study. When tested in the field, the accuracy of their method was reasonable, 77% for 122 hours of recordings. Their eating detection algorithm revealed the 44 eating events they had considered with an average accuracy of over 95%, thus prompting the authors to conclude that smart glasses were suitable for monitoring chewing and eating in a free-living environment and could be used to analyze the wearer's natural chewing patterns.

Unlike camera based systems described in section 1.3.2, smart glasses based systems mitigate the concern of individual privacy. Since data that is recorded cannot be used to uniquely identify individuals by anyone except the researchers themselves, smart glasses offer a better alternative to monitoring eating activity as compared to both acoustic sensors described in section 1.3.1 and camera based systems 1.3.2. However these are limited to monitoring eating activities, as drinking a liquid produces no significant activity of the temporalis muscle which is used in these aforementioned works for monitoring dietary intake. In addition, not all individuals wear glasses. Hence a subject who wants to monitor his/her dietary pattern, but does not wear glasses regularly may not be very compliant with wearing a smart glasses based system for monitoring their eating activity.

1.3.4 Inertial Sensors

Based on the discussion in the previous sections for the different sensing modalities for detecting eating related activities in individuals, we realize the need for one sensor type that can be used by a larger group of people than those wearing eyeglasses. This sensing modality needs to be inconspicuous to use, without drawing attention to the user and needs to be small so that it can be worn on the body easily. Both of these factors would promote daily usage. It should be relatively free from environmental and background noise as compared to the acoustic sensors in section 1.3.1, and there should be no concern of individual privacy, which is major limiting factor in the use of camera based systems such as those described in section 1.3.2.

Wrist-worn IMU sensors thus seem like the most appropriate form of sensing modality for monitoring eating related activities in individuals. Unlike eyeglasses, a larger set of people wear a wristwatch. IMU sensors such as accelerometers and gyroscopes have already been integrated into devices such as smartwatches and fitness trackers. These can easily be used to sense wrist-motion

and thereby track eating activities in individuals who are already comfortable wearing these devices in their day-to-day lives. These sensors are relatively free from environmental noise, except for vibration and shock, and these can be considered as exceptions to the general rule of thumb when monitoring eating in individuals. There is no concern for individual privacy like that of camera based systems, except in rare cases when the data can be accessed by individuals other than researchers or medical care professionals administering these devices. As a good practice, researchers and medical care providers can ensure that participant privacy is always maintained by using encrypting and decrypting all data recorded by the device and also by never storing any information such as name, age, sex or any other personal information of the participant on the device.

Previous work in our group [6],[7],[8] has focused on characterizing the motion of the wrist, and designing an algorithm to count the number of times food or drink intake is detected. This work was also extended to counting the number of bites per minute based on the recording of the device called as the 'Bite Counter'.

More recently the work had shifted to detecting all different gestures that occurred during a meal, using hidden Markov models (HMM) [24],[27]. A Markov model is a stochastic model that can be used to predict the behavior of temporal or sequential data (data that can be ordered in a sequence as time evolves). Sequential data usually evolves while maintaining some dependency on its previous output as well as its state. A HMM is a special case of the Markov model where the states of the model are hidden, or cannot be exactly observed, while the output is completely observable. Additionally the stochastic process being monitored/tracked by a HMM is said to be a Markov process. This means that the probability of the stochastic process being in the current state conditioned on all previous states, is dependent only on the probability of the given state conditioned on the probability of the previous state and no state before that. This can be expressed as [27]:

$$P(X_n = x_n | X_{n-1} = x_{n-1}, ..., X_0 = x_0) = P(X_n = x_n | X_{n-1} = x_{n-1}) \tag{1.5}$$

Here X_n represents a state of the HMM or the stochastic process being modeled, while x_n represents its actual value and P is the discrete probability of the state, where $X_0 = x_0$ is the estimated initial state. It is important to stress that the set of states as well as outputs in a Hidden Markov Model are discrete, or limited to few possible values.

In [24] the authors considered three classifiers; a K-nearest neighbor classifier, a HMM that

captures sequential context of subgesture motions and a HMM that models intergesture sequential dependencies for detecting similar eating related gestures from a data set of 25 meals eaten by different subjects. Based on the higher accuracy of the HMM that captured the sequential dependencies between gestures; 96.5% as compared to 75.8% and 84.3% for the K-nearest neighbor classifier and the HMM capturing sub-gesture motions respectively; the authors demonstrated that eating gestures exhibit a sequential dependency among themselves. They suggested that this dependency can be exploited for improving the recognition accuracy of eating related gestures.

Building on this idea, the author in [27] considered three main variations of HMM's in order to model eating in the human subjects. HMM-S considered five different HMM's, one for each gesture type considered. This strategy modeled each gesture as a sequence of sub-gestures. For example a bite of food is often composed of activities such as raising food towards the mouth, ingestion, and returning the wrist to rest on the table [27]. HMM-N extended this idea to N previous gestures in order to improve the contextual information captured by a HMM in the task of gesture recognition. This was loosely based on the idea used in speech recognition HMM's.

Contextual varying HMM's were also considered in [27] in which each gesture was further categorized, in order to separate contextual information to better model the task of monitoring eating gestures. For example the gesture 'bite' was further divided into five variations: bite with fork, bite with spoon, bite with both hands, bite with single hand and bite with chopsticks. These sub-categories of 'bite' would certainly be preceded and followed by different gestures and sub-gestures for the large group of people that was considered. This was thought to improve the overall classification accuracy of the HMM.

Most recently in [18], the authors used a two step process for detecting food intake via wrist micromovements. In the first step a convolutional neural network (CNN) is used to learn the probability distribution of five specific wrist micromovements. This is fed into a recurrent neural network model known as a long-short term memory network (LSTM), which is used to capture the temporal evolution of the sequence and classify sequences of food intake cycles. They evaluate their model on the FIC dataset, which is available publicly at https://mug.ee.auth.gr/intake-cycle-detection/. This dataset was created using commercially available smartwatches and is hence very similar to the one being considered in this research. The authors in [18] compare their model against three state-of-the-art methods, including the one reported in [7]. Their proposed method achieves the highest F1 detection score of 0.913 in a leave-one-out crossvalidation approach, at detecting

periods of eating from inertial measurements. This work is really motivating because it suggests that deep learning can be used to learn features needed for classifying and identifying eating behavior from inertial measurements of wrist motion.

However one common feature of most of these works, is that they considered pre-segmented data. That is the data being considered in each of these, was divided by a human-expert into each of the five gesture categories described in detail later. In [18] the authors did consider a sliding window approach for generating continuous data used for training their neural network model. However they assumed that the start and end moments of the meal are known. This corresponds to a situation in which the user would manually start and stop the recording before starting a meal, and just as it was about to end respectively. Additionally the FIC dataset they considered is centered around eating using a fork, knife or spoon and other utensils such as chopsticks, are not taken into account and neither is eating with bare hands. Finally the data set does not include recordings for drinking a liquid, thus limiting the trained neural network at detecting only eating related gestures successfully.

The neural network considered in this research proposes a novel approach at monitoring eating activity. It builds on the foundational idea behind most of the previous work within our group, including [6],[7],[8],[24] and [27]. However it extends to data that is previously unsegmented and considers 'drink' related gestures as well. Hence the neural network classifier being considered in this research must be able to successfully identify different periods of eating and non-eating related gestures from data measured using IMU wrist mounted sensors, and it must simultaneously be able to classify each segment into one of the five categories of gestures considered. This is described in detail in section 1.5.

1.4 Deep Learning

A neural network is an inter-connection of computing nodes known as neurons, that are inspired by biological neurons present in human brains. When a set of neurons is arranged with each having a similar task, and being provided the same input, these neurons are said to be arranged in a layer. A neural network needs at least two layers, an input layer and an output layer. The input layer is of the same size as the input, and it works by simply feeding the input provided to any subsequent layers connected to it. The size and function of the output layer depend on the specific task that the neural network is being designed for.

In addition to these two layers, neural networks typically contain one or more hidden layers which learn a mapping from the given input space into a feature space which can be used for the specific task at hand. The mapping is itself dependent upon a non-linear function known as the activation function which is applied at the output of each neuron in a neural network, except for the neurons present in the input layer. The process of updating a neural network to do better at the task, is called learning, and it progresses by updating the mapping between layers till a fixed criterion is met. The mapping between layers is a function of the weights attached to the synaptic connections between different neurons. These weights get updated in the training of a neural network, and are eventually stored along with the actual architecture of the network itself for use.

When the input and associated targets of the task are available to the designer, training the neural network is known as supervised learning. Supervised learning is still the most common form of training neural networks, in spite of the additional task of generating targets or labels for individual elements of the input data. This is because the properties of supervised learning have been widely studied by the pattern recognition and machine learning communities, and there is a wide consensus regarding when a neural network is considered to be reliably trained.

Deep learning is a highly specialized area within neural networks. Deep learning typically makes use of multiple hidden layers, each having different activation functions in order to learn better mappings from input-space to feature-space. Deep learning has been steadily gaining a lot of momentum in the research community and in industrial settings for the last couple of years. As a result of this, there has been tremendous effort in researching, designing and implementing a wide variety of hidden layers, their activation functions etc. The main motivations behind using deep learning for machine related tasks are the versatility of designing models for a wide variety of applications using deep learning, and the accuracy with which these tasks can be performed by a carefully designed model.

Another advantage, and one that is often overlooked when using deep learning, is that of its re-usability. Typically neural network architectures used for classification follow the same hierarchical structure of mapping layers followed by downsampling layers arranged in a block. This block is repeated as many times as necessary, and is followed by one or more fully connected layers known as dense layers. Hence the architecture used for classifying one set of data such as a collection of street images, can very easily be re-used by resetting the weights and training the neural network with another set of images, say those of different clothes. Since the architecture does not differ much

between these two tasks on similar but distinct data, the neural network can be repeatedly re-used for both tasks. This functionality can be extended to other sets of images, as long as the data in the images and neural network architecture do not change much between different tasks.

A final advantage to using deep learning for tasks such as classification, sorting and labeling, is the concept of transfer learning. Typically owing to their hierarchical structure, deep learning networks learn the same features for similar data at the higher levels or in the first few layers, with task specific features or combinations of useful upper level features being learnt in the subsequent layers. This means that several upper level layers are multi-functional and can be used without modification for the similar task on different datasets. This greatly improves the versatility of neural networks, although the performance of the new network is not guaranteed to be the same as that of the original network. Current research in transfer learning is focusing on understanding the relation between upper level and lower level layers, and improving the mapping quality between these in order to enable neural networks to be used without much modification for several tasks.

At this point it must be noted that it is unwise to think that a deep learning neural network is always required or preferred for a machine related task such as classification, information-retrieval or labeling. While deep learning has produced state-of-the-art models in most of the tasks mentioned above, it has taken the researchers and practitioners that designed those models several years in order to collect the data that was required for reliably training the neural network within each model. While this was initially seen as a drawback of deep learning models, it is now becoming clear that acquiring the right kind of data to train a deep learning model, is often a simpler task than designing a model from start to end using classical approaches.

1.4.1 Deep Learning For Image Segmentation

Image segmentation is the task of labeling each pixel in an image based on some membership criteria with respect to a fixed number of categories. This is used to understand the image at a lower level than classification [30]. Typical approaches in computer vision and machine learning solve image segmentation using a matched filter, in which a window of data having certain properties such as sensitivity to angles, straight lines, shapes etc. is run through the entire image in a pixel-by-pixel manner. Based on the specific filter chosen, regions in the image having strong similarity to the matched filter produce a higher output than regions having weak similarity.

The output of the matched filter can hence be used to determine which regions most resemble

the desired shape, structure etc. that is present in the matched filter. This output may be used to train a classifier, over each image region, which would then be used to classify each pixel within the image region as belonging to one of the many classes chosen for the given classifier. Please note that this is a form of correlation based matching, in which the cross-correlation between the image region and the matched filter is used to determine locations having properties similar to that of the matched filter. In order to use a convolution based approach, the matched filter in question would have to be symmetrically rotated about its axis and then run over the image in the same pixel-by-pixel manner.

Recent advances in deep learning, have used a Convolutional Neural Network (CNN) trained on image patches or regions extracted from around each center pixel in order to obtain a class or score based on which category each image patch belonged to [5]. These networks comprise of convolutional layers that work on the same principle of correlation based matching for feature extraction. However the key difference is that the filter or window is learned iteratively in each layer within such networks.

As explained in [25] this strategy has two drawbacks. First training the network on each image patch is very slow and redundant as surrounding patches usually have only a few pixels that are different except for patches considered around the boundaries of objects within images. The second drawback of this approach is the tradeoff between localization accuracy and context. Larger patches require more max-pooling layers, those that aggregate information from previous layers, while smaller image patches do not allow the network to see much context [25].

Hence the approach followed in [25] and [30], differs significantly than the ones mentioned above. In these approaches a CNN is trained to produce an output image having the same size as the input image, with fewer or greater number of channels. Each channel represents a probability of the corresponding pixel belonging to each respective category. This can be pictorially explained using Figure 1.5.

A neural network, such as the one described in [25] takes as input a standard RGB image, in which intensity for each pixel is stored as a 3-dimensional vector. Each component of the vector corresponds to the intensity of the pixel with respect to the red, green and blue channels respectively. Other color formats are also used, but the RGB format is the most common type. In addition to the RGB image, the neural network is also provided with a target image, containing the class or category of each pixel as input. For example, the first pixel in the input image considered in Figure 1.5 belongs to class 2, the second one to class 0, the third to class 1 and so on.

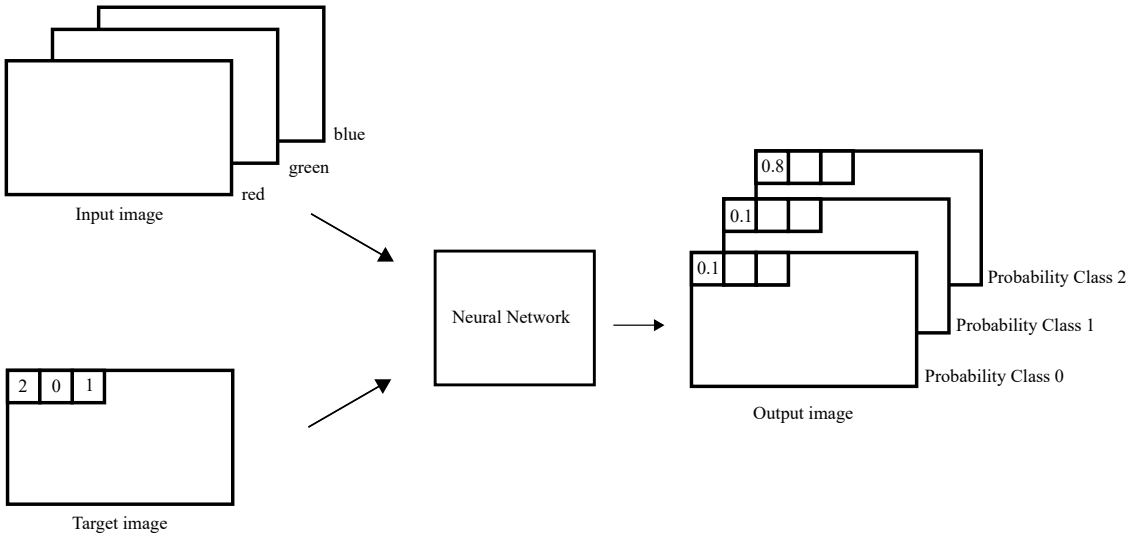


Figure 1.5: Image segmentation via pixel class prediction.

Based on these inputs, the neural network considered in [30], produces an output image having three channels corresponding to the number of classes. After successfully training the neural network, we can expect an output as shown in Figure 1.5. As it can be seen, the value for channel 2 in the output is higher than those in channels 0 and 1. This indicates that the probability of the first pixel belonging to class 2 is higher than that of the other classes, and hence it is classified as belonging to class 2, which is the correct output.

This basic idea for image segmentation was extended in [2], which used a deep fully convolutional neural network for semantic pixel-wise image segmentation. The novelty in their approach, known as SegNet, comes in the decoding phase, in which the decoder uses pooling indices computed in the max-pooling stage of the corresponding encoder block to first perform non-linear upsampling. These upsampling maps are sparse and are then convolved with trainable filters for producing dense feature maps, thereby eliminating the need for upsampling.

As observed in [31], although deep learning methods have achieved state-of-the art performance in medical image segmentation, they have not yet demonstrated the necessary accuracy or robustness needed for clinical use. Hence the authors in [31] propose an interactive segmentation framework by incorporating a CNN into a bounding box along with a scribble-based segmentation pipeline. This allows for image-specific fine tuning through the use of a weighted loss function allowing the model to learn and predict previously unseen object classes. As per the authors this

improves the robustness of the segmentation model and also improves image-specific segmentation accuracy.

Thus deep learning has been considered by the research community for a variety of image segmentation tasks, achieving state-of-the-art results in many of these applications. However it has not been applied for the task of segmenting time-series data, such as that recorded from IMU sensors. This is the main motivation behind considering a deep learning classifier for our research as explained in section 1.5.

1.5 Novelty Of Our Approach

The idea of image segmentation using deep learning as described in section 1.4.1 essentially casts image segmentation as a classification problem. The notable difference however is that instead of producing a classification score for the entire image, the network produces a score for each pixel. The weights learned by the neural network, can hence be considered as set of matched filters which produce locally strong responses in the image to structures contained within each matched filter. It is common practice for convolutional layers in a CNN to be succeeded by pooling layers. These successively downsample the response of the network after each convolutional layer. In this way, only the strongest set of responses are retained for each set of matched filters.

Since the network output needs to be of the same size as the input image, an additional category of layers is used, called as deconvolutional layers, or up-convolutional layers [25]. These produce an output that is successively bigger in each layer based on the choice of the filter window being considered in each layer. These are described in greater detail later on in this report. For now, the reader should appreciate the fact that a neural network can be trained using end-to-end learning for labeling each pixel in an image, based on its category or class.

Similar to the idea of using convolutional neural networks for classifying each pixel from a 2-dimensional image, this work considers using convolutional neural networks for segmenting a data set of time-series. Specifically the data available to us is measured using IMU sensors, specifically a 3-axis accelerometer and a 3-axis gyroscope fitted inside a watch-like device worn by subjects on their wrist. This device kept recording while the subjects each ate an unscripted meal for varying amounts of time. Following this, the data was labeled as belonging to one category from 'bite','utensiling','drink','rest' and 'other' by a set of human raters. For details regarding the inter-

rater agreement and reliability, the interested reader is referred to [27]. Each of these gesture-types can be considered as a category or class to which a given instance of time belongs to, in a particular meal.

Hence the novel research question being considered in this work, is whether a deep learning classifier can be trained to detect and automatically segment such eating related gestures from previously unsegmented IMU data. The rest of this report, provides more detail on how such a network was designed, trained, tested and evaluated. It also provides more information about the data set itself, the preprocessing steps required for converting the data into appropriate units for the accelerometer and gyroscope readings, as well as those required to make the data suitable for training for a convolutional neural network.

Chapter 2

Methods

This chapter provides a detailed description of the actual data used, analysis and design for the neural network architecture, data preprocessing as well as the evaluation metrics used and strategies followed for labeling each instance of time within a meal.

2.1 Data

The data for this research was collected from 276 participants eating a meal at the Harcombe Dining Hall at Clemson University, which is a large cafeteria setting. Each participant ate a meal that consisted of 1-4 courses. The data includes wrist motion measurements (accelerometer x, y, z; gyroscope yaw, pitch, roll) recorded using IMU sensors fitted inside a watch-like device that each participant wore on their hand. Of the total number of courses, 488 were retained as part of this research. This data is publicly available on the following website [14]. For demographics of the participants such as age, ethnicity, sex etc. as well as other information regarding the actual experimental setting, the interested reader is referred to chapter 2 of [27].

In order to measure the wrist-motion of each participant, a ST Microelectronics LIS344ALH accelerometer, and a ST Microelectronics LPY410AL gyroscope were used. As a reminder to the reader, an accelerometer when mounted on a person's wrist measures the orientation of the wrist with respect to a fixed axis, while a gyroscope measures the angular velocity when the wrist is turned. These particular devices were chosen as they have three degrees of freedom each, and can hence measure the wrist movement with a total of six degrees of freedom. In addition to the

measurements from these devices, the recordings also contain a measurement from OHAUS Scout Pro SP4001 scale. The last measurement is not a part of the training data used, but is mentioned for the sake of completeness. Hence each recording contains 7 columns of measurements, where the first 6 columns of measurements are from the 3 axes of the accelerometer and the yaw, pitch and roll measurement of the gyroscope respectively. As an additional note, all data was recorded at 15 Hz.

2.1.1 Converting A/D Voltages

As mentioned earlier, sensor readings are typically in volts (V) and hence must be converted into useful quantities when being used for practical applications. For the accelerometer data this can be expressed as follows:

$$G_{acc} = (V_{acc} - 1.65) \times \frac{5.0}{3.3} \tag{2.1}$$

Here, V_{acc} represents the voltage measured in each column of the accelerometer recording, while G_{acc} represents the converted value in gravities relative to the acceleration due to gravity $g = 9.81\, m/s^2$. For the gyroscope measurements, conversion can be expressed as:

$$D_{gyr} = (V_{gyr} - Zero_{gyr}) \times 400 \tag{2.2}$$

Where V_{gyr} represents the gyroscope output voltage in each column of the recording, and D_{gyr} represents the converted output in deg/s. For the gyroscope measurement conversion, we need to calculate an additional term denoted by the term $Zero_{gyr}$. This value is the average of the gyroscope measurements for each of the yaw, pitch and roll axes, and must be calculated separately for each meal/recording. It's typical value is around 1.25, and is needed to account for small voltage drifts for the 0 deg/s reference point as mentioned in [14].

2.1.2 Ground Truth Generation

As mentioned earlier, in a supervised learning setting, any classifier requires targets or labels associated with the data. The process of adapting a classifier to be more suitable for the task at hand, requires that the classification output be as close to the corresponding target as possible for each training example. This means that generating the correct ground truth is as important as

generating data in case of a supervised learning setting such as a deep learning classifier.

The process of generating these targets, is often called as preparing the ground truth for the machine learning model or classifier. This research uses the same ground truth labels as those mentioned in [27]. Since the primary task of this research is automatic segmentation of eating related activities, the number of intake gestures considered is higher than that of non-intake related gestures. These as 'bite', 'utensiling', 'drink', 'rest', 'other' and 'unlabeled'. The first five gestures are the same as those defined in [24] and [27]. The last gesture is unique to this research and is needed because each time instant needs to be labeled in order to use our approach of neural network based classification. Each of the other gestures can be defined with respect to the following:

1. the description of the activity;
2. the start time of the activity;
3. the end time of the activity;
4. any particular events that need to be included or excluded;

Bite

1. The subject puts food in their mouth.
2. The gesture starts when the hand or utensil starts moving towards the mouth.
3. It ends when the hand or utensil finishes moving away from the mouth.
4. A bite need not begin and end at a plate. Any motion towards and away from a mouth should define the gesture boundaries. In addition a bite may include multiple successive back-and-forth motions to and from the mouth, as long as each did not complete the hand motion away from the mouth was not completed and which were separated by less than 1 second.

Drink

1. The subject puts a beverage into their mouth.
2. The gesture starts when the cup or glass starts moving towards the mouth.
3. It ends when the cup or glass finishes moving away from the mouth.
4. For multiple sips, each should be treated as a separate drink.

Utensiling

1. The subject uses an utensil or their hand(s) to manipulate, stir, mix or prepare food for consumption.
2. The gesture starts when food begins to be manipulated.
3. It ends when the manipulation ends.
4. This includes moving food around the plate, cutting food into bite sized pieces, dipping food in sauce is also considered as a utensiling activity along with other similar activities.

Rest

1. The subject's dominant hand has little or no motion. The range of motion that may be considered rest differs from individual to individual. Different people have different levels of physiological tremor (motion that occurs in everyone and has no medical significance) and thus the threshold for maximum motion during rest will vary subject to subject.
2. Gesture starts when the instrumented hand stops moving.
3. It ends when a new intent becomes clear (instrumented hand starts moving again with clear intent for at least 1 second).
4. Only the instrumented hand must be considered when considered when determining rest. It may include the period when the subjects holds a morsel, drink or utensil still, remaining relatively motionless.

Other

1. All other actions such as reaching towards food (before an actual bite), gesturing while talking, cleaning the face with a napkin or moving a plate should be considered as other activities.
2. The subject's facial expression may be used to discern when the gesture type is ambiguous. For example slight movement in the instrumented hand when talking may be treated as other, instead of rest, since this is not a period of rest in between eating activities.

3. This gesture should be used when the definition between rest and other such as the one scenario mentioned above, or that between utensiling and other is unclear.

In addition to these five main gesture types similar to the ones defined in [27], an additional type, 'unlabeled' was considered when creating the data set for this research. This category marks all instances of time where no activity was labeled by the human annotator. As mentioned earlier, this gesture type needed to be defined for this research, because each instant of time needs to have a label in order to use a neural network based classifier for segmenting and identifying gestures from our data set. For generating the ground truth, a total of 18 raters manually labeled the meals based on the directions provided above. These raters were trained in several sessions to understand the process and the definitions of the gestures [27]. A total of 51,614 gestures were manually labeled, for a set consisting of 276 subjects. Of these the gestures for 264 subjects were retained, and are also publicly available on [14]. For details such as the assessment of inter-rater reliability, modifying ambiguous labels etc. the reader is encouraged to refer to chapter 3 of [27].

Figure 2.1 shows one recording of data and ground truth displayed on CafeView, the custom tool software built by our research group to visualize the data. The central green line in the figure marks the actual position being observed. The color codes on the left indicate the gesture type that was labeled by the human annotator, while the actual gesture appears below the actual measurement. These color codes are as follows, a bite is indicated with red, a drink with aqua, utensiling is labeled with orange, rest with black and the gesture type other is labeled with a gray. Each annotator marked the ground truth while watching a video of a subject eating a meal. This process is described in greater detail in [27].

2.1.3 Sliding Window Approach

Sequential data evolves over time, and hence maintains some relation with its past events. That is to say data measured previous to the current recording as well as data a few time-steps ahead is of vital importance in analyzing a given measurement of time. This means that a window of measurements needs to be considered when training a neural network based classifier for predicting on time-series data. This allows the temporal relationships between different time-instances to be seen as individual inputs by the neural network, thereby allowing it to learn a good set of features based on these temporal relationships. This is a key difference in analyzing images and sequences

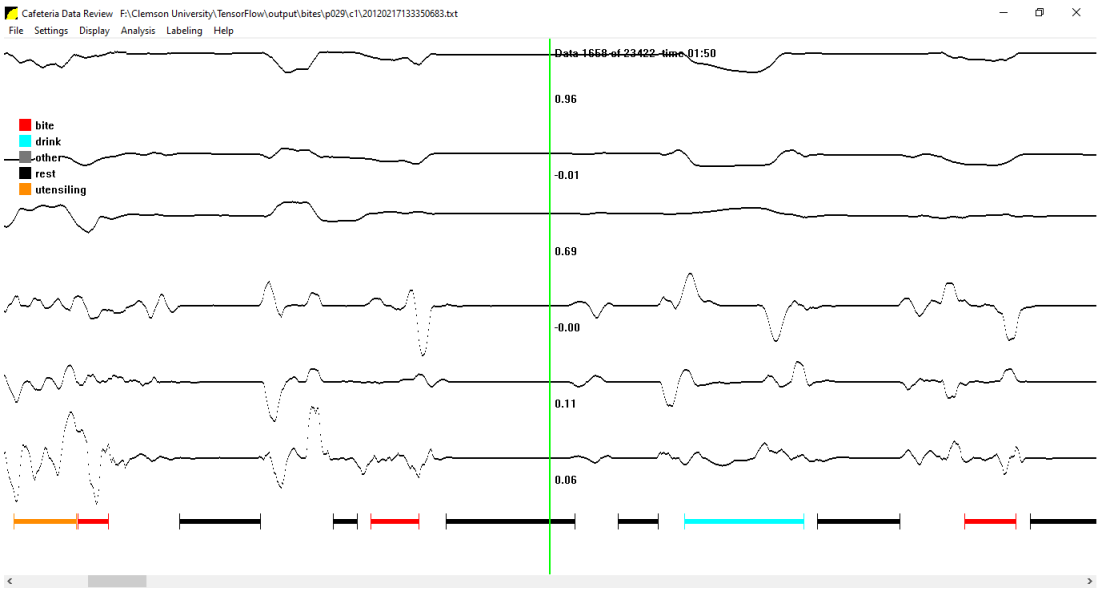


Figure 2.1: CafeView: Example of measurement data & ground truth for one meal. Gesture labels are red: bite, aqua: drink, gray: other, black: rest, orange: utensiling

using deep learning neural networks.

For this research a sliding window of 30 seconds duration was considered, meaning that a window of data contains 450 consecutive measurements from each of the 6 axes of data mentioned earlier. In addition the ground truth label considered for each window is a sequence of labels for each time-instant as identified in the ground truth file, or marked as unlabeled by a software program. The parameter 450 is known as the window length, while the period 30 seconds is known as the window duration. For this particular research, 30 seconds was thought to be an adequate window size, in terms of the number of measurements to reliably train a deep learning classifier.

In addition to the window length, another parameter of interest is the window stride, which determines how much the sliding window shifts in each iteration of data generation. This was chosen as 15 measurements, corresponding to 1 second of time. This means that once a sequence of 450 measurements is considered, the window shifts by 15 time-steps and the next sequence starting from the new start of the window, up to the next 450 consecutive measurements is considered in the new window.

This entire process is illustrated in the flowchart in Figure 2.2. Here M stands for the number of windows in each recording, and is a parameter that varies with each meal. Due to the sliding window, and because not all meals are guaranteed to be of the same duration, the data generation

algorithm needs to monitor that an out of data index is not encountered before proceeding in each iteration. In addition to this it also needs to monitor that the maximum number of windows allowed per recording is not exceeded. This value was set to 20,000, which is sufficiently high and can be used to extract data from all recordings present. In practice, the data acquisition part continued till the last window fitted just inside of the final data index indicated by the gesture ground truth file. Any indices beyond this were not considered for training the neural network. The exclusion can be justified by observing that most eating related activities would more likely occur towards the start or in the middle of each recording. The later part of each recording would most likely include periods of extended rest, or inactivity from the subjects.

As it can be seen in Figure 2.2, the gesture unlabeled was identified using the purple color. The instances of time in between the three gestures (bite, drink and rest) marked by a human annotator are all identified as unlabeled. In Figure 2.2, N represents the last index of the recording which fits just inside the last window M. Both these parameters vary from meal to meal, and the program used for generating the data set does take this into account. It also monitors that the end index of the last window does not go out of range from the last ground truth entry, and that the total number of windows does not exceed the maximum mentioned above.

2.1.4 Data Normalization

When dealing with neural networks, an additional step known as data normalization is needed, which is distinct from the step of normalizing units as described in section 2.1.1. This process is generally independent of the data generation steps and is hence performed once the entire data set is available but before the neural network is trained. In earlier experiments, our group found that neural networks are particularly sensitive to data normalization. We observed that this step was crucial, and without which a neural network did not learn any useful feature-mapping for a very simple case of classifying lines having positive and negative slope. The choice of data normalization is typically task dependent, and there is generally no way of knowing which normalization technique to use, prior to the actual experimental setting.

For this work, the strategy of min-max normalization was chosen, based on previous experience within the group, when this strategy worked well for training neural networks on sequential

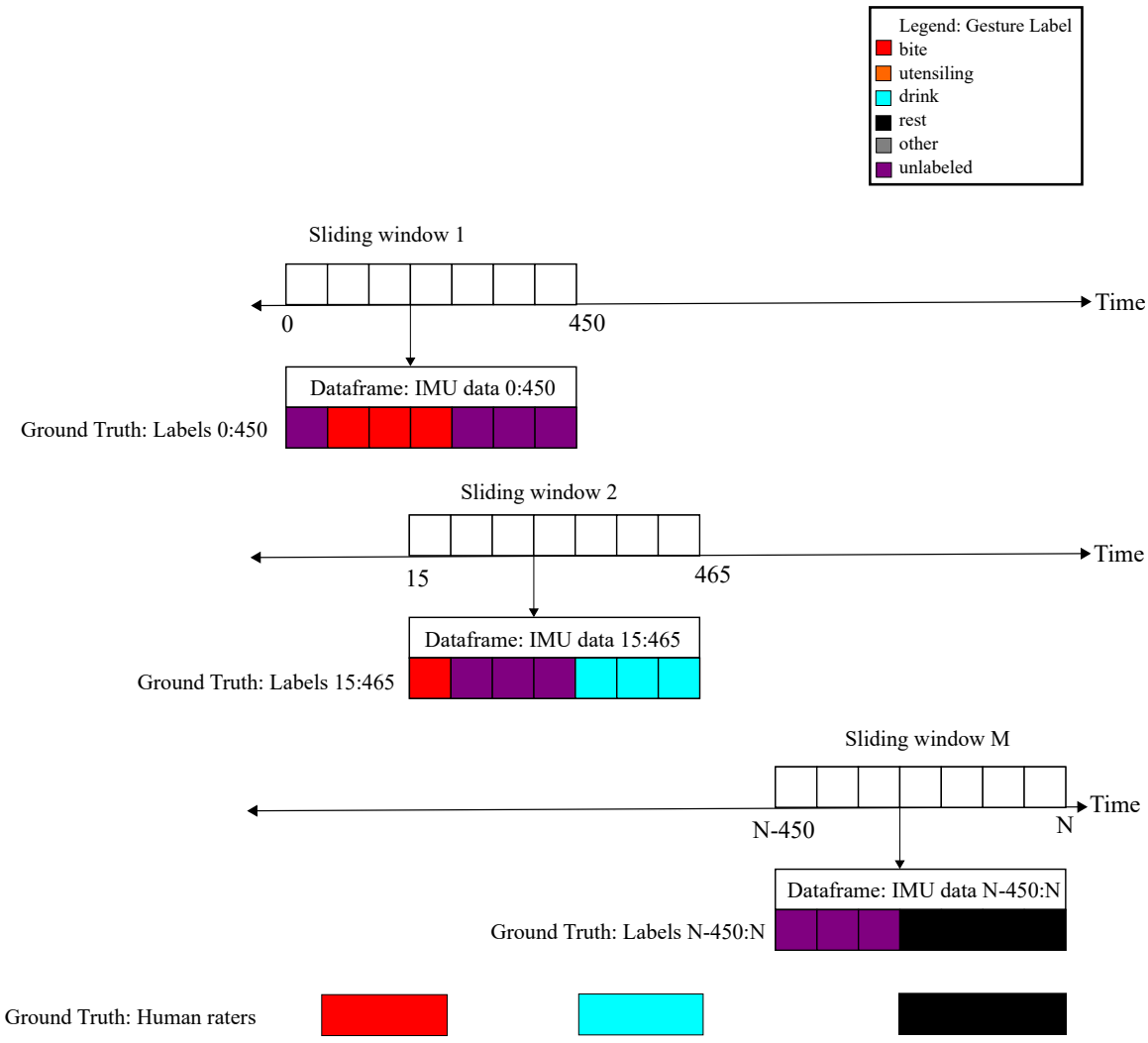


Figure 2.2: Creating data set and ground truth labels for each sliding window.

data. This can represented as follows:

$$x_{scaled} = \frac{x_{measured} - x_{min}}{x_{max} - x_{min}} \tag{2.3}$$

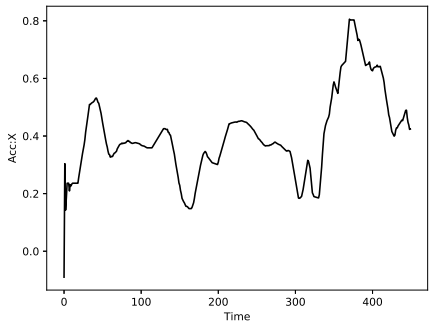
Here x_{min} and x_{max} represent the minimum and maximum value recorded for each of the 6 axes of data mentioned at the start of Section 2.1. These values were calculated separately for each recording, adding a small computation overhead on the overall algorithm. Correspondingly $x_{measured}$ and x_{scaled} represent the measured value and scaled value for each data entry respectively. Figure 2.3 displays the results after normalizing one sequence of input data for each of the 6 axes of input measurements.

2.2 Deep Learning Network

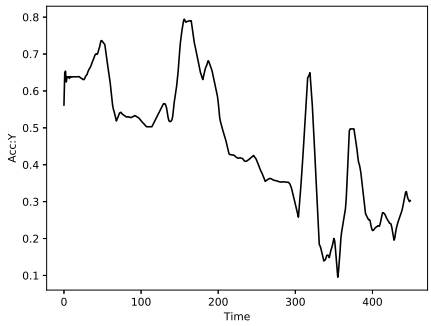
This section describes the main types of neural network layers that were used in developing the deep learning classifier and their role in the overall neural network architecture. Many of these layers are used more than once. However no two layers are the same, as each layer operates on data of different sizes. For example the first convolutional layer operates on a sequence length of 450, corresponding to the number of measurements in each window, whereas the next convolutional layer in the structure operates on a sequence length of 225. This is obtained by downsampling the output of the previous layer. In contrast the deconvolutional layers operate on sequences having smaller length. The layer operation produces a sequence having greater length by convolving it with a filter window after zero padding the input sequence. Each of these functions is described below:

2.2.1 Reshape Layer

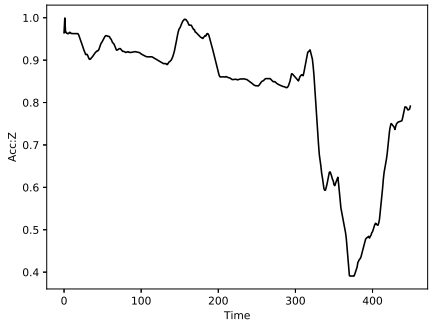
This layer is an additional layer from the one considered in [25] and [30] and is needed to reshape the data to be compatible for use with the convolutional layer used later in the neural network architecture. Figure 2.4 shows the general idea behind the reshape layer for reshaping one window of data. The input to this layer is a data array of size 450×6 which is reshaped to be a 3-dimensional array of size $1 \times 450 \times 6$ where each measurement from the IMU sensors mentioned earlier, is now treated as a channel of information. In terms of the machine learning data, this new array can be considered as a 2-dimensional image, consisting of 6 channels of information.



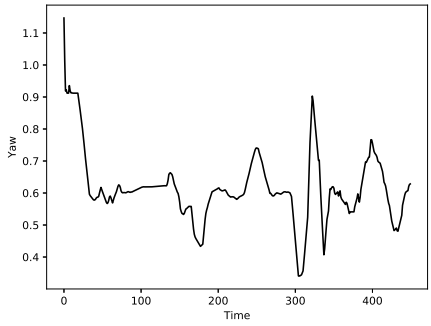
(a) Accelerometer: x.



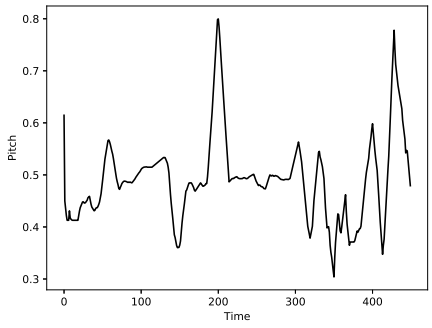
(b) Accelerometer: y.



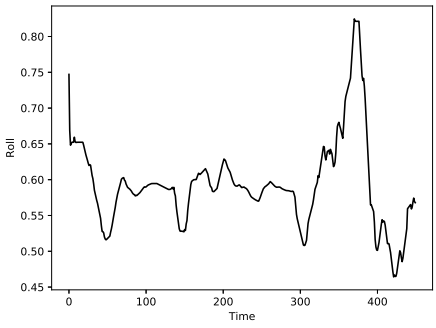
(c) Accelerometer: z.



(d) Gyroscope: yaw.



(e) Gyroscope: pitch.



(f) Gyroscope: roll.

Figure 2.3: Normalizing one sequence of data.

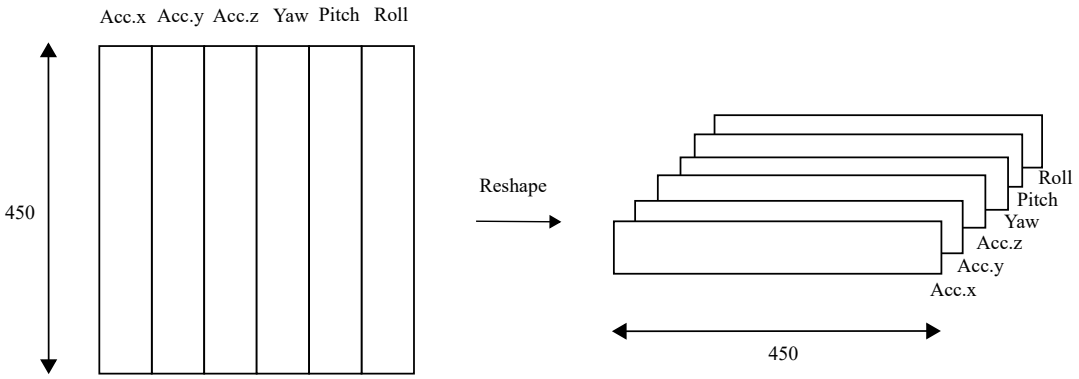


Figure 2.4: Reshaping input for compatibility with the neural network.

2.2.2 Convolutional Layer

This is one of the most important layers in the neural network, responsible for generating the feature maps from which the neural network learns representations of the input data. Convolutional layers are the core building blocks of CNN's, consisting of a set of learnable filters or weights. Although the size of each filter is usually very small as compared to the size of the image or in this case the sequence of data, each filter is repeatedly convolved throughout the entire length of the data for generating a useful representation.

As an additional feature, convolutional filters are said to share parameters, that is a single filter is used throughout the length of the sequence. This stems from the idea that an important pattern may occur more than once in a given sequence, separated by a finite temporal resolution. The number of filters in each convolutional layer is called its depth, and determines the dimensionality of the layer output, known as the activation map. It must be noted that a filter is also run along each channel of information separately. In this way a convolutional filter captures both the temporal information, as well as the information present each channel of its input.

Even in convolutional layers, the stride plays an important role as it did in the feature generation part. Stride determines the shift in each convolutional filter, as it runs over the length of the sequence. It can thus be used to control the size of the activation map as well. In TensorFlow, there are two options for choosing the parameter `stride`. These are `valid` and `same` with very distinct functions; choosing `stride = 'valid'` convolves the learnable filter along the data sequence without any zero padding. This results in an output sequence that is smaller than the input in the same way as convolution of a sequence of numbers with a function in classical signal analysis yields

a smaller output signal if no zero padding is used. Choosing `stride = 'same'`, however pads the input sequence with zeros on either side, so that that output sequence has the same length as that of the input. As a consequence, the latter was when designing the neural network for this research. Setting it this way allows us to control the size of the activation map very easily, and downsampling can be achieved easily as well, using the max-pooling layer described later.

Since the kernel size, the depth of each convolutional layer and its stride are so important when designing deep learning classifiers, these are often treated as the most important parameters of a CNN. Note that however, each of their function as well as that of other parameters must be carefully understood before designing a neural network.

Table 2.1 lists all filter size, stride and depth of each convolutional layer in the neural network. On seeing this, the reader should observe that the kernel size is a 1-dimensional window with the first parameter always set to 1. This is because of the earlier reshape operation mentioned in section 2.2.1 in which the data sequence was reshaped as a 3-dimensional image of size $1 \times 450 \times 6$. Another interesting observation is that the filter size kept reducing as we went deeper into the neural network architecture, whereas the number of filters that were learned kept on increasing, doubling actually at each stage. Such an approach was followed in [25], and was hence adopted for this research as well. The size of the first filter was chosen as 15 along the dimension corresponding to the axis running along the sequence length. This is because measurements have been recorded at 15 Hz as mentioned earlier, hence using this approach the neural network analyzes each sequence of input data, 1 second at a time. Owing to the max-pooling layer described later, the activation map of the convolutional layer is downsampled, and hence the filter size is also reduced by about half in each subsequent layer. This effectively means that the neural network analyzes the response of the first convolutional layer at a resolution smaller than a second in the subsequent convolutional layers.

The number of layers and filters chosen was very small as compared to [25]. This can be explained as follows; the size of each input to the neural network considered in [25] is 512×512. Such a dense array contains far more information for a convolutional layer to train on as compared to the array size of one input sequence of time-series data such as the one considered in this research. As mentioned earlier, the size of each individual input sequence to the neural network is 450×6, which is far smaller than the size of the input to the neural network considered in [25]. Hence it was decided to use a smaller set of output filters in each convolutional layer, and fewer convolutional layers as well.

Layer	Kernel size	Stride	Number of output filters
Convolutional Layer 1	(1,15)	1	8
Convolutional Layer 2	(1,7)	1	16
Convolutional Layer 3	(1,5)	1	32

Table 2.1: Convolutional layers: Filter size & output dimensionality.

2.2.3 Batch Normalization

Batch normalization was originally introduced in [16] as a way of improving the training in neural networks by reducing the shifts in internal covariance of a deep learning neural network. These shifts are caused by the change in the distribution of a layer's input due to changing parameters of the layer preceding it. Loosely explained, as the weights and biases in the previous layer change, so does the output of the layer, which is being fed directly or through an activation function (described in section 2.2.4) to the next layer. As explained in [16] this slows the training of neural network by requiring slower learning rates and careful parameter initialization. Using batch normalization the output of a preceding layer is normalized in a manner analogous to the data normalization described in section 2.1.4, with the key differences being that batch normalization is done repeatedly within a neural network, and that the parameters required for batch normalization are learned during the training phase of the neural network as explained in [16].

It can be expressed as follows; given a small set of m training examples, henceforth referred to as a mini-batch, the outputs of one intermediate layer (that precedes another) can be expressed as $\mathbb{B} = \{x_1, x_2, ...x_m\}$. The output can be modified using a transformation (batch normalization), expressed as $y_i = \mathrm{BN}_{\gamma,\beta}(x_i)$ where γ and β are parameters that are learned. The procedure for performing batch normalization during training is defined as follows:

1. For each mini-batch of layer outputs, $\mathbb{B} = \{x_1, x_2, ...x_m\}$:

   (a) Calculate mini-batch mean:

   $$\mu_{\mathbb{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.4}$$

   (b) Calculate mini-batch variance:

   $$\sigma^2_{\mathbb{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathbb{B}})^2 \tag{2.5}$$

(c) Normalize outputs using the above calculated values:

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathbb{B}}}{\sqrt{\sigma_{\mathbb{B}}^2 + \epsilon}} \tag{2.6}$$

Where ϵ is a small offset, added to avoid division by zero.

(d) Scale and shift normalized responses:

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}\left(x_i\right) \tag{2.7}$$

As a final note on batch normalization, we must note that the mini-batch statistics $\mu_{\mathbb{B}}$ and $\sigma_{\mathbb{B}}^2$ are computed only during the training phase of the neural network. As explained in [16], during testing or inference we want the output to depend only on the input deterministically. Hence we calculate moving averages of the mini-batch statistics, which can be expressed as:

1. For each activation map (x) within a given network, calculate the moving average of the mini-batch statistics $\mu_{\mathbb{B}}$ and $\sigma_{\mathbb{B}}^2$ as mentioned earlier:

   (a) Update the mini-batch statistics in each training epoch using the following:

$$E\left[x\right] \leftarrow E_{\mathbb{B}}\left[\mu_{\mathbb{B}}\right] \tag{2.8}$$

$$Var\left[x\right] \leftarrow \frac{m}{m-1} E_{\mathbb{B}}\left[\sigma_{\mathbb{B}}^2\right] \tag{2.9}$$

   Where m is the size of each mini-batch as mentioned earlier, Equation 2.9 represents the unbiased variance estimator and $E_{\mathbb{B}}$ is the expectation over training mini-batches of size m, having sample statistics as $\mu_{\mathbb{B}}$ and $\sigma_{\mathbb{B}}^2$ [16].

2. The linear transformation for batch normalization is changed from Equation 2.7 as:

$$y = \frac{\gamma}{\sqrt{Var\left[x\right] + \epsilon}} \cdot x + \left(\beta - \frac{\gamma \cdot E\left[x\right]}{\sqrt{Var\left[x\right] + \epsilon}}\right) \tag{2.10}$$

where γ and β are learned for the activation corresponding to each layer, as mentioned earlier.

Batch normalization was used after each convolutional and deconvolution layer for the neural

network considered in this research. The reader is alerted is to an on-going debate in the neural network community whether batch normalization is to be performed before or after the activation function, described next. There has been no general consensus regarding this subject, and a beginner in this field does find himself/herself feeling a little perplexed regarding the correct sequence of these two operations. For our research we decided on placing the activation function after the batch normalization operation, since this was the approach followed in the neural network known as `pix2pix` referenced in [30].

2.2.4 Activation Function

The activation function is the main non-linear component of a neural network architecture. Activation functions are used to transform the output of a neural network's layer into a range that depends on the specific activation function chosen. All layers in a neural network have an activation function, with the exception of the input layer and the reshape layer considered in this research, as these are used to simply broadcast the input to subsequent layers in dense networks or be operated upon by the first mapping layer in networks such as CNN's.

Covering the entire range of activation functions is beyond the scope of this research, and the interested reader is instead referred to a wide variety of articles and papers that are available online, and that offer a lot of theoretical background as well as practical examples of different activation functions used in different applications. For this work two activation functions were used, and these will be described in detail in this subsection. Each intermediate layer in the neural network designed as part of this research used a Rectified Linear Unit (ReLU) activation function, whereas the final output layer fed its response into a Softmax activation function.

2.2.4.1 Rectified Linear Unit (ReLU)

The rectifier activation function is defined as the positive part of its input argument, denoted as:

$$f(x) = x^{+} = max(0, x) \tag{2.11}$$

It was introduced in [11] as a better alternative training-wise to the logistic sigmoid and hyperbolic tangent activation functions which were widely used in neural networks prior to this. Currently it is one of the most popular activation function being used in the neural network community owing

to its ease of use, and because it allows very large networks with a variety of hidden layers to be effectively trained for many applications. A neural network unit employing the rectifier activation function is known as a rectified linear unit (ReLU).

2.2.4.2 Softmax

The softmax function is another activation function that has found widespread use in the neural network community, over other functions such as the binary sigmoid or the hyperbolic tangent function mentioned above. The main reason for its popularity, is because it allows us to convert the non-normalized output of a neural network into a probability distribution over the predicted output classes. It is mainly used in the final layer of a neural network, as follows. Let $\mathbf{z} = (z_1, z_2, ..., z_K) \in \mathbb{R}^K$ be the K outputs of a neural network classifier. The standard softmax function ($\sigma : \mathbb{R}^K \rightarrow \mathbb{R}^K$) is then defined as:

$$\sigma(\mathbf{z}_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \forall z_i \in \mathbf{z} \tag{2.12}$$

It can be easily verified that the sum of all transformed outputs will always be 1, and that each output would always be in the range [0,1], lending to the probabilistic representation of the network output. Hence the output of a neural network after applying a softmax activation function on its final output can be interpreted as an output probability predicted by the neural network with respect to each class of data considered.

2.2.5 Max-pooling Layers

These layers are used to retain the most significant response from the activation map of previous layers. This is thought be be similar to the operation of feature selection in classical machine learning models. These layers thus prevent the neural network from learning based on weaker features, which may not be useful in the classification task considered.

The max-pooling operation is a downsampling operation that can be done over any dimension of the input provided to a max-pooling layer. For this research only the dimension along the sequence length was considered for downsampling while all other dimensions were retained. Hence the size of the filter in each max-pooling layer was chosen as `(1,2)` which instructs the software to only consider the stronger response from two consecutive responses of the previous layer, and upon doing so move to the next two consecutive responses without operating on the first two again.

2.2.6 Transposed Convolutional Layers

Transposed convolutional layers, sometimes called deconvolutional layers were first introduced in [35] for the task of image analysis and synthesis via learned mid-level representations. These were used in [25] and [30] for the task of image segmentation in the latter half of the model known as the decoding phase.

As explained in [23], these layers produce an activation map or output of a fixed resolution, based on an input that may be a downsampled version of an earlier activation map of the same resolution. In order to do this the software pads the input with zeros along each dimension and performs a convolution similar to that in convolutional layers. The kernel or the filter is also learned similar to that in convolutional layers. Hence the only difference between these two layers is in the zero-padding. While zero-padding is an option in convolutional layers, it is the default first step in a deconvolutional layer.

Owing to their same functionality, both convolutional and deconvolutional layers are followed by the same operations of batch normalization and activation function. However the step of max-pooling the response of a deconvolutional Layer is discarded, since we need an output of a bigger size than that provided as input to each deconvolutional layer. This process is referred to as up-sampling in [30], although there are some key differences from this operation and up-sampling as it is explained in the context of classical digital signal processing.

Similar to convolutional layers in section 2.2.2, the kernel size, stride and number of output filters are important parameters in deconvolutional layers. For the neural network considered in this research, these are listed in Table 2.2. Notice that unlike the convolutional layers in Table 2.1 the number of output filters reduces as we increase the number of deconvolutional layers. This structure is also adapted from the neural network in [25] which use a decreasing number of filters in each deconvolutional layer, till the depth of the last deconvolutional layer is equal to the desired number of output classes. The parameter `stride` is of particular importance in deconvolutional layers. Choosing `stride = (1,2)` tells the software to pad one zero on both sides of the axis corresponding to the sequence length of the input to each deconvolutional layer. Hence the deconvolution operation increases the size of its input along the axis of the sequence length, creating sequences of increasing length in each stage of deconvolution. The kernel sizes for each deconvolutional layer are the same as those of their corresponding convolutional layer, and this choice was made for the sake of maintaining

Layer	Kernel Size	Stride	Number Of Output Filters
Deconvolutional layer 1	(1,5)	(1,2)	32
Deconvolutional layer 2	(1,7)	(1,2)	16
Deconvolutional layer 3	(1,3)	(1,2)	6

Table 2.2: Deconvolutional layers: Filter size & output dimensionality.

a symmetrical structure in the model. The kernel size in the final deconvolutional layer was chosen as (1,3) in order for the neural network to generate locally strong responses over shorter temporal lengths while generating the final output of the classifier. Note that a 1×1 kernel size also exists in neural network literature, but was not thought to be of much use, as this would not consider any temporal relationship between an element of the output from the penultimate deconvolutional layer and its immediate neighbors.

2.2.7 Stacking Multiple Layers Into Blocks

Another common feature in deep learning is the arrangement of multiple functional layers in a sequential manner, known as a block. A neural network can have many such blocks arranged repeatedly in a hierarchical structure. The neural networks in [25] and [30] have two distinct functional blocks. The blocks consisting of convolutional layers are known as encoder blocks, since they are used to learn features from the input data. The first convolutional layer is used to learn features directly from input data. Later convolutional layers are then used to learn useful combinations of the features learned by upper convolutional layers, thus encoding information available from the input data.

In the opposite manner, deconvolutional layers are used to decode information from the previous deconvolutional layer and a convolutional layer from the encoder phase into a more useful representation such as a segmentation map. Hence these blocks are known as decoder blocks, since they learn a reverse mapping from feature-space to output-space. A common feature of deconvolutional blocks in the neural network architectures used in [25] and [30] is the presence of a concatenate layer. This layer merges information from a previous deconvolutional block and a convolutional block that occurs much earlier in the model architecture. It is thought to be useful, because it merges information from the encoding and decoding parts of the model. This helps the neural network in reconstructing information into a useful representation of the input data.

The neural network model used in this research was adapted from those present in [25] and

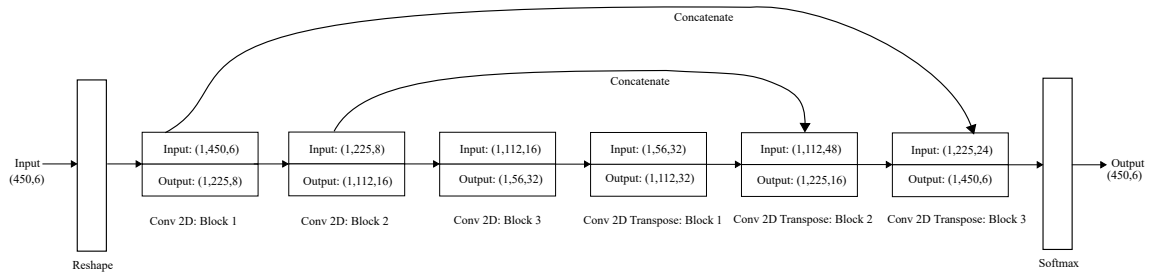


Figure 2.5: Model architecture: Convolutional and deconcovolutional blocks stacked in an encoder-decoder structure.

[30]. Modifications were made to the data such as those described in section 2.2.1 in order to make the data compatible with the neural network architecture. Other modifications include those made to the number of convolutional and deconvolutional layers, as well as the number of output filters in each layer as described in section 2.2.2.

The overall model architecture is shown in Figure 2.5. The resulting neural network model used in this research is a stack of 3 convolutional blocks, followed by 3 deconvolutional blocks arranged in an encoder-decoder fashion. At its output a softmax activation function is used to convert the response of the neural network into a probability distribution over the number of classes as explained in section 2.2.4.2. The convolutional blocks used in this research are all identical, whereas the deconvolutional blocks used differ from each other with respect to an additional layer or a missing layer. These differences are described next.

Convolutional Block

A convolutional block consists of a convolutional layer, followed by a batch normalization layer, a rectifier activation function and a max-pooling layer. As mentioned in section 2.2.2, each convolutional layer pads its input with zeros and produced an activation map having the same sequence length as its input. This can be seen in Figure 2.6. Note that the output dimensions of the convolutional layer differ from those of its input only in the final axis, corresponding to the number of information channels. Each convolutional layer transforms its input, encoding it information with the number of channels as described in Table 2.1.

Batch normalization and the rectifier activation function have no effect on the dimension of the input and hence these operations do not change the dimension of their input within a con-

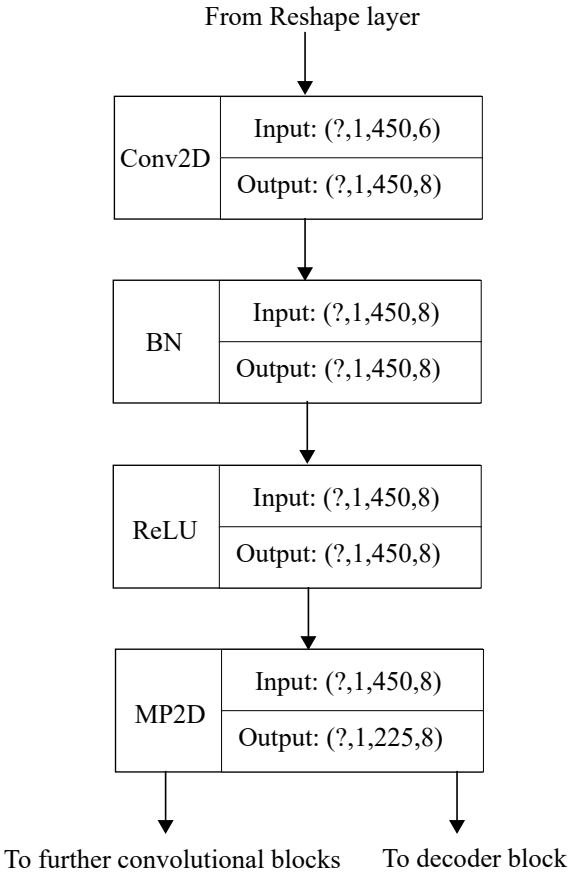


Figure 2.6: Convolutional block 1: The layers from top to bottom are, Conv2D: convolutional layer with 8 output filters, BN: batch normalization, ReLU: rectifier activation layer, MP2D: 2-dimensional maxpooling.

volutional block as seen in Figure 2.6. The final layer in a convolutional block is the max-pooling layer which downsamples the input along the axis corresponding to the sequence length as described in section 2.2.5. The output of the max-pooling layer is fed to the next convolutional block in the model architecture and the corresponding deconvolutional block, which will occur after the encoding phase of the neural network. All 3 convolutional blocks used in our neural network were identical in structure, and each operated on input that was successively smaller owing to the max-pooling layer in the previous convolutional block. The only difference in the third convolutional block was that its input was fed only to the first deconvolutional block, as there were no more convolutional blocks after it.

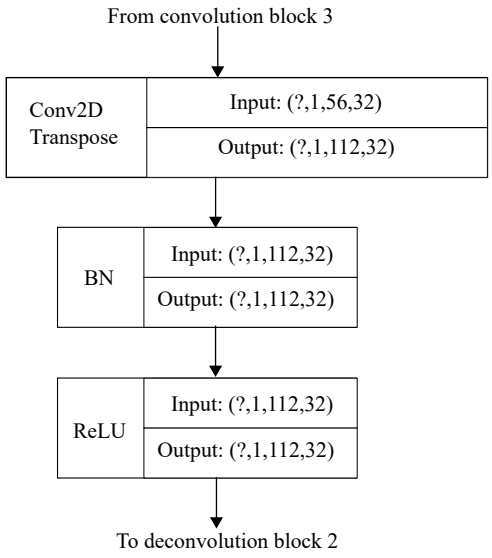


Figure 2.7: Deconvolutional block 1: The layers are from top to bottom, Conv2D Transpose: deconvolutional layer with 32 output filters, BN: batch normalization and ReLU: rectifier activation layer.

Deconvolutional Block

In contrast to the convolutional blocks, each deconvolutional block in the neural network architecture is different from each other. However each block is similar with regards to its core function. Each deconvolutional block decodes information from one or more preceding blocks in order to produce the final segmentation map required at the output of the neural network. The difference in the 3 blocks comes from the way input is fed into each block and how the output of a block is used in the subsequent stages of the neural network.

The first deconvolutional block is connected to the third and last convolutional block, and hence does not need a concatenate layer at its input as seen in Figure 2.7. It operates on a sequence length of 56, zero pads both sides of each element and performs a filtering operation as discussed in section 2.2.6. This increases the output sequence length to 112, and is passed to the subsequent layers in the block. Batch normalization is used to reduce the internal covariance shift of the output as discussed in section 2.2.3, and this response is passed to the rectifier activation before being finally passed to the second deconvolutional block.

The second deconvolutional block is the first block in our neural network architecture to use a concatenate layer at its input. This network merges information from deconvolutional block 1,

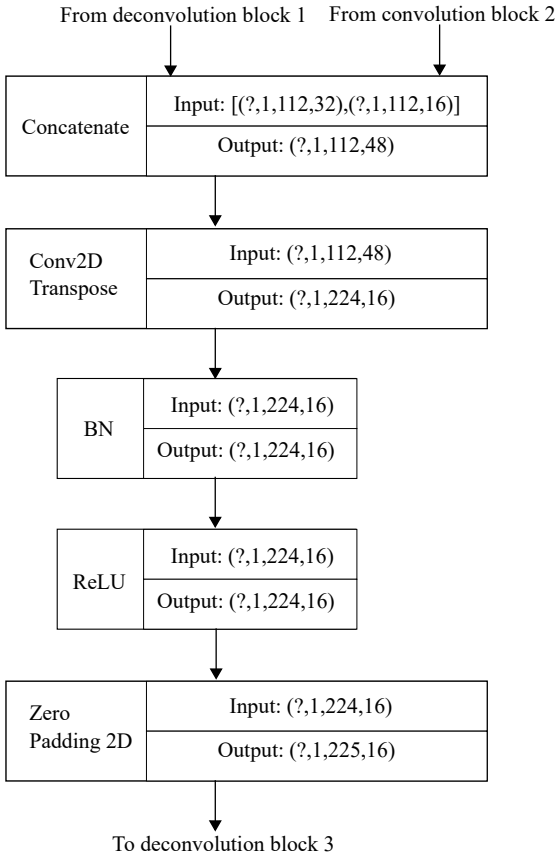


Figure 2.8: Deconvolutional block 2: The layers are from top to bottom, Concatenate: concatenates output from Deconvolutional block 1 and Convolutional block 2, Conv2D Transpose: deconvolutional layer with 16 output filters, BN: batch normalization, ReLU: rectifier activation layer and Zero Padding 2D: needed to increase output sequence length from 224 to 225.

and convolutional block 2 from the encoding phase. The merged input is passed through a deconvolutional layer similar to that of deconvolutional block 1, and then through a batch normalization layer and rectifier activation function in order to produce the second stage of the decoded output. At the end of deconvolutional block 2 in Figure 2.8, we see a zero padding layer. This is used to pad the response after the rectifier operation with an additional zero along the sequence length, and is required since we desire that the output of the neural network be a sequence of predictions of length 450, equal to that of each input sequence.

The third and final deconvolutional block also has a concatenate layer at its input which merges information from the deconvolutional block 2 and convolutional block 1. However in this block, once the deconvolutional layer has operated upon the merged input, it is not passed through

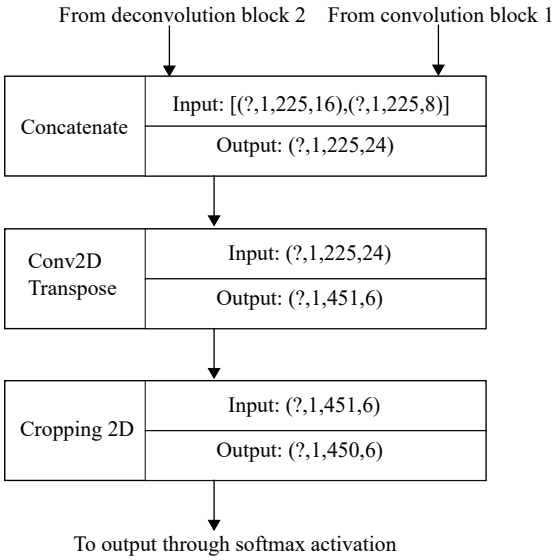


Figure 2.9: Deconvolutional block 3: The layers are from top to bottom, Concatenate: concatenates output from Deconvolutional block 2 and Convolutional block 1, Conv2D Transpose: second deconvolutional layer with 6 output filters and Cropping 2D: needed to clip superfluous response from earlier zero padding operation.

a batch normalization layer or the rectifier activation function. This is because the third deconvolutional block consists of the final set of layers in the neural network architecture. Hence its output needs to be normalized using the softmax activation and used for producing the final segmentation map. A batch normalization layer is thus not needed at this stage, and the rectifier operation is to be replaced with the softmax activation function. However before passing this response to the softmax activation function, we need to clip the superfluous response which occurs due to the earlier zero padding operation. This is done using the cropping layer as seen in Figure 2.9. After clipping this response, the neural network response is finally fed through the softmax activation function and used as the final output of the neural network. This final output of the neural network is a set of probabilities corresponding to each category of gestures. Thus the neural network produces a probability distribution over the gesture categories for each time-instant at its output. This is explained in detail later.

It should be noted that the '?' in Figures 2.6, 2.7, 2.8 and 2.9 indicates that the mini-batch of training examples is allowed to change.

2.2.8 Training A Neural Network

All the earlier discussions relate to the design of a neural network, and choosing the layer, activation function or normalization strategy best suited for a particular task. Once the final output of a neural network is available to us, parameters and functions that are related to how the neural network updates its weights and biases become important. Weights and biases are key components in a neural network, since these store the feature mappings learned in layers such as convolutional layers. Other mappings such as useful recombinations of features, and even decoder mappings in deconvolutional layers are stored as the set of weights and biases in a neural network.

Here the reader is alerted to the fact that, while the choice of a correct loss or metric is always important for a neural network, all neural networks parameters are optimized via gradient descent. The method of gradient descent is an iterative procedure in which the parameter updates follow a path indicated by the opposite of the gradient of the loss function chosen. This is because the gradient always follows the path of highest increase, and choosing the opposite indicates that we are interested in following the path which reduces the loss or error of our chosen model.

The interested reader is again referred to texts other than this report for a comprehensive review of the set of loss functions possible, since this list is also too big to cover completely in this report. While a few of these loss functions were experimented upon in the initial stages of this project, cross-entropy was chosen to be the most suitable for the task of classifying instances of time as belonging to one of many classes. This is described next:

2.2.8.1 Categorical Cross-Entropy

In machine learning, when calculating model errors between 0 and 1, categorical cross-entropy is the same as the multi-class log loss expressed as:

$$f\left(\mathbf{y}_o, \mathbf{p_o}\right) = -\sum_{c=1}^{M} y_{o,c} \cdot log(p_{o,c}) \tag{2.13}$$

Where $\mathbf{y_o}$ is the vector of network targets (ground truth labels) and $\mathbf{p_o}$ is the vector of network outputs. The negative sign is used because the positive logarithm of numbers smaller than 1 results in negative numbers and can be confusing to work with, especially when comparing machine learning models. Setting the loss function in this way also builds on the probabilistic representation of network outputs as described in section 2.2.4.2.

However there is an additional step that needs to be done, when using the provided labels with a cross-entropy loss function. Each component of the vector of network targets is further transformed into a one-hot vector representation, in which one indicates true value and zero indicates false. For example, consider a vector from a 6 class classification problem, denoted by $\mathbf{y} = (0, 1, 2, 3, 4, 5)^T$ which indicates that the class of the first training example is 0, that of the second class is 1 and so on. This is transformed into its one-hot vector representation as follows; $\mathbf{y_{o,1}} = (1, 0, 0, 0, 0, 0)$, $\mathbf{y_{o,2}} = (0, 1, 0, 0, 0, 0)$ all the way up to $\mathbf{y_{o,6}} = (0, 0, 0, 0, 0, 1)$ for the final class label, where $\mathbf{y_o}$ is the vector representation to be used in Equation 2.13. The one-hot vector representation can also be interpreted as a probability over the ground truth labels. However in this case, since the actual class of the training example is known to us, only one component of the vector will be 1, and all other components will be 0.

2.2.8.2 Network Hyperparameters

In addition to the correct loss function, other network parameters such as the size of the mini-batch of training examples, the learning rate, choice of optimizer and its settings, the number of epochs the network is trained for and the total number of training examples, play an important part in the reliably training a deep learning neural network. These are collectively known as the hyperparameters of the network. For the neural network considered these are described as follows:

1. **Size of mini-batch of training examples:** This controls how fast or slow the loss function of the neural network converges during the training process, and consecutively the weights and biases get updated to their final values. Choosing a mini-batch size to be very big results in slow convergence, as the gradient descent updates need to be made over a larger set of training examples. On the other hand choosing the value to be very low (close to one), results in a training curve that is very noisy, as the loss oscillates due to frequent updates to the weights and biases. Typically the value is chosen as a multiple of 2, since this has been found to improve the training time on modern GPUs using parallel processing. In this research the mini-batch value chosen was 32, which is the default setting in TensorFlow.

2. **Choice Of Optimizer:** An optimizer is the algorithm used to calculate the updates to the weights and biases of the neural network during each step of gradient descent. Owing to the popularity of deep learning, the choice of optimizers available in most software packages has

seen a steady increase in the past couple of years. For this network the `adam` optimizer, is used with parameters `beta_1` and `beta_2`, which are momentum parameters set to their default values (0.9 and 0.999 in TensorFlow- keras respectively).

3. **Learning Rate:** This parameter controls how much of the update calculated by the optimizer is used to change each weight and bias within a neural network. Like the choice of the mini-batch size, this parameter also has an immediate effect on the speed with which the neural network converges. A parameter close to 1 results in very quick convergence, however the weights and biases fluctuate in each training epoch, due to the updates made by the optimizer. Choosing a value too small slows the convergence greatly, but generally results in a smoother loss curve, and slow convergence of the weights and biases. For this research the learning rate was also kept at its default value of 0.001.

4. **Number Of Training Epochs:** An epoch is a pass over the entire training set of a neural network, whereas a pass over a mini-batch of training examples is known as an iteration. The number of epochs thus directly controls the training time of the network. Typically deep learning neural networks require a large amount of time to train reliably. However this should be monitored closely, and if the model does not show any improvement in convergence for a large number of consecutive epochs, then training should be halted and only be resumed when the hyperparameters or the model architecture has been reasonably changed. For this network, the number of training epochs was varied between 10 to 200, with the network showing better learning at 100 epochs improving steadily till the number of epochs was increased to 200, and no significant improvement beyond that. Hence 200 epochs was chosen as the total number of epochs for this research.

2.2.9 Cross-Validation

When dealing with a training data of fixed size, it is often advantageous to consider training the network via cross-validation. With this technique, the entire data set is repeatedly divided into training data and a small portion which is known as the testing data. The latter is used to assess the model's capacity for solving it's task on general data (generalization), since this part of the data set is unseen for the model in each iteration of cross-validation. It is important to note that a new model must be initiated for each iteration of cross-validation and that each separation of the data

set as training set and testing set must be distinct or mutually exclusive, to prevent a false promise of model generalization.

Cross validation is useful because it provides a way of using sufficient data for training a model, when data is limited. In addition it also provides a way to assess model generalization, when the amount of data is limited, cross-validation also allows us to be confident of the model's performance in a real-life scenario. Since a new model is initiated, trained and tested on different data each time, if the model performance is good on average, it indicates that such a model would be of good use in practice when trained with the entire data set. It is important to note that the training data in each fold is further sub-divided into train and validation data sets, which are used to actually train the model and validate against overfitting during the training of the model in each fold. This process is described in detail later.

Typically cross-validation is used as a k-fold cross-validation in which the data set is divided into k parts, where $k-1$ parts are used as training data and the k^{th} part is used as testing data. For this research 5-fold cross-validation was used, in order to partition the available data such that both training and testing data were adequately large for the chosen model.

2.3 Post-Processing

Once a neural network is trained by tuning the hyperparameters and cross-validating over the entire dataset, it can then be used to predict on an input data sequence as shown in Figure 2.10. Notice that in a manner similar to the image segmentation example in section 1.4.1, the network prediction is a probability distribution over the class labels for each given input sample. In a manner analogous to earlier, the output probability of the class corresponding to the ground truth is significantly higher as compared to the other probabilities, which helps us to distinguish between the total set of output probabilities for each given sample within a sequence of data.

2.3.1 Max-Voting Strategy

As mentioned in section 2.1.3, each meal is divided into multiple windows of input data having a fixed size, which are used as input when training the neural network. The neural network in turn produces a prediction score corresponding to each of the six classes of gestures considered, for each time instant within each window. Except for the first 15 time-instants, all instances appear

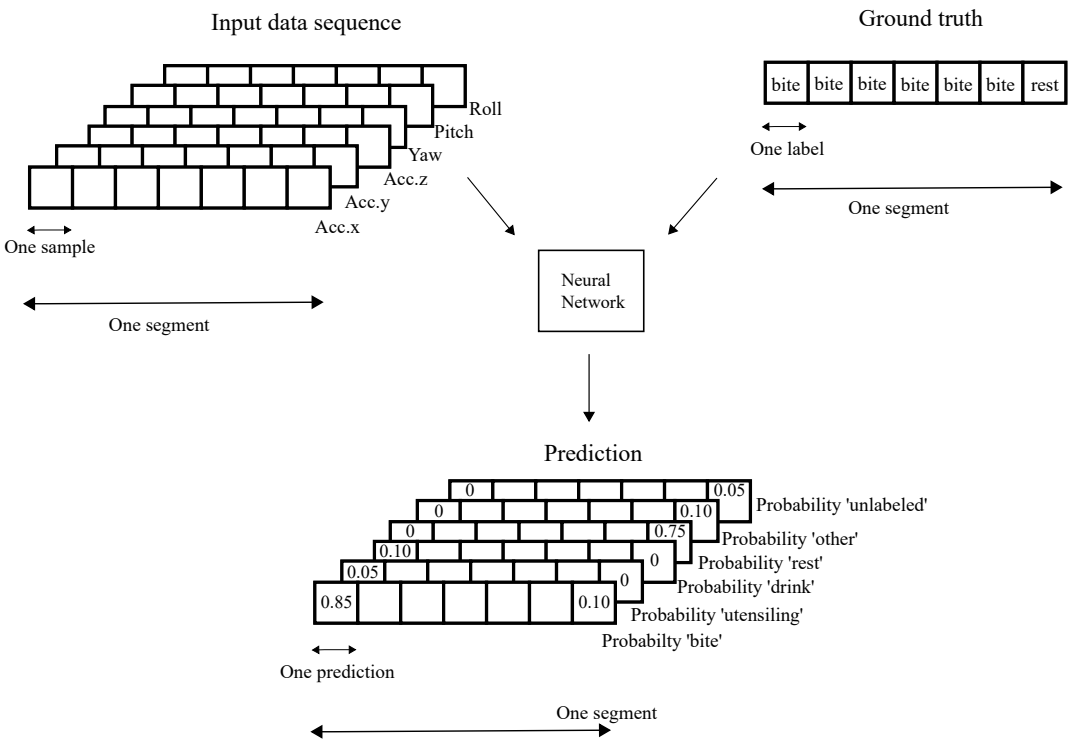


Figure 2.10: Generating class labels for one segment of data.

in more than one window. Hence there is a possibility that some instances may be labeled differently in different windows, and a strategy needs to be designed in order to judge the most likely label of a time-instant.

The strategy of max-voting was used in this research, and is explained pictorially in Figure 2.11. Using this strategy, the software keeps a count for each gesture type for each time-instant labeled by the neural network. Time-instants that occur in more than one window, will thus have different counts for each gesture type. Following this the gesture type with the maximum count is retained as the final label for a time-instant or datum. In Figure 2.11, this is shown for the boundary of two gestures, 'bite' and 'utensiling'. The time-instant 7 gets labeled as a 'bite' in Prediction window 1, but gets labeled as 'utensiling' in each of the other two windows in which it occurs, and hence 'utensiling' is retained as the final label for this time-instant. Note that for the scenario considered in Figure 2.11, each window shifts by three time-instants in each iteration, and hence this time-instant would not occur in the fourth prediction window. For time-instant 8, which would occur in the fourth window, the label type is unambiguous, as it has been labeled as 'utensiling' in each of the previous three windows, and hence its label would remain unchanged.

This ambiguity between gestures is often seen at data corresponding to boundaries between different ground truth labels. This is to be expected, since a datum within each window which occurs at such boundary locations has different temporal relationships with the other datums from the two windows respectively. The class label of such datum, would thus also be dependent on the specific relationship or feature-mapping that is learned by the neural network.

If a certain datum had equal counts for two or more gestures after applying the max-voting strategy as described above, the datum was assumed to be part of the previous gesture type and the last known gesture label was applied to the datum. Figure 2.12 shows a flowchart for the entire process of generating predictions from a meal using the neural network, as described in this section. In it we see that all instances of time that occurred outside the final window of predictions are identified as unlabeled by the prediction algorithm. This is consistent with our approach of labeling during the data generation phase. Since eating related gestures are more likely to occur at the start and the middle of each meal, gestures outside the final window of predictions can be treated as unlabeled instances.

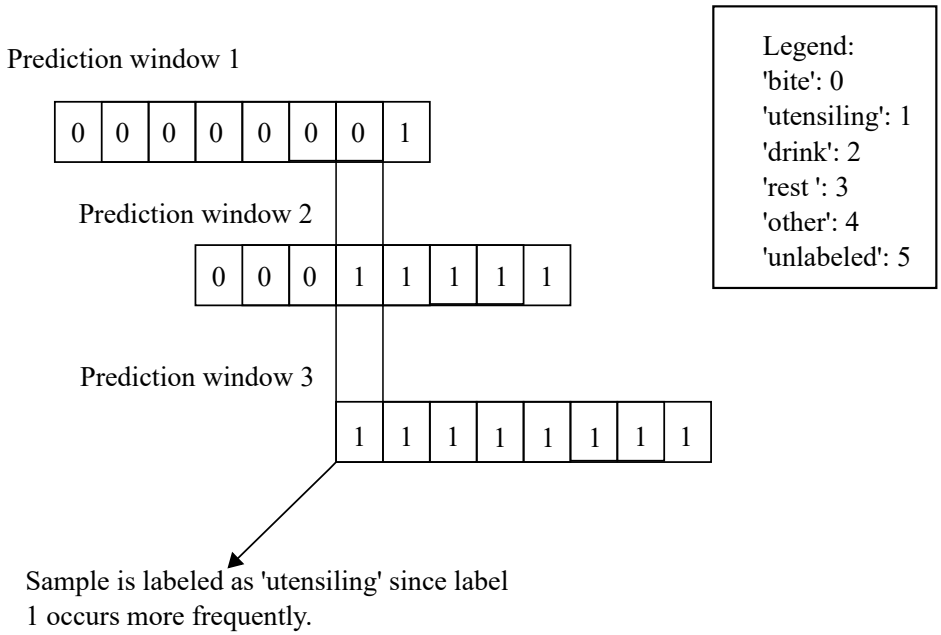


Figure 2.11: Max-voting strategy: Retain most frequent label.

2.3.2 Final Output Of Classifier

After following all the steps listed above, the classifier prediction for all time-instances in a meal, were stored in a single array. Following this, the software began tracking the label for each instance for the purpose of printing out each gesture type to a file which would be stored on disk. This was a simple implementation, by initializing two variables start and end at 0 each, and increasing end by 1 each time the gesture type of the current instance was the same as that of the next instance immediately succeeding it. When the two instances were not the same, the program printed out the current gesture label, the corresponding start and end, and reset the variables start and end to the index of the succeeding gesture and the counter started anew. This process was repeated for all time-instants labeled by the classifier, and the final output was stored as an ASCII text file containing the gesture name, followed by the start and end of the gesture as labeled by the classifier.

The reader is reminded that the class 'unlabeled' was not part of the original ground truth file, but instead created since the output of the network needed to be the same size as the input. Hence this class of labels was never printed out to the file stored on disk, and neither were the start and end dates corresponding to each unique gesture identified as 'unlabeled' by the classifier. Instead

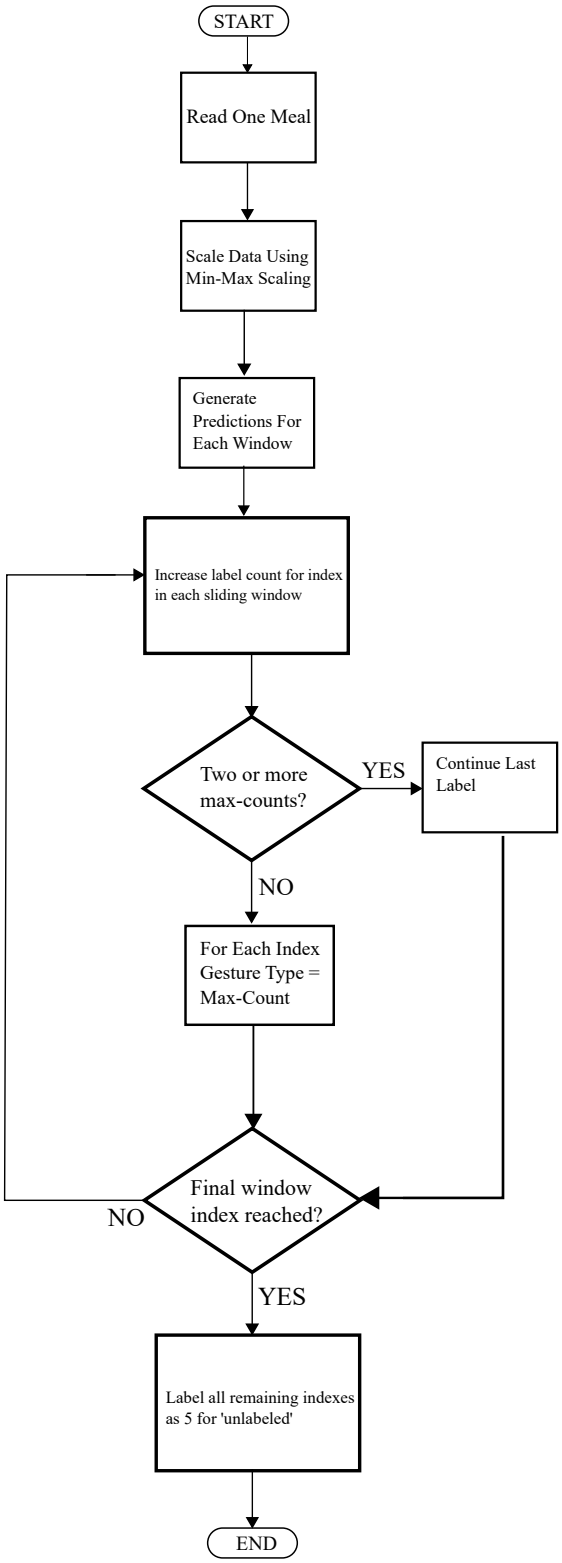


Figure 2.12: Generating network predictions.

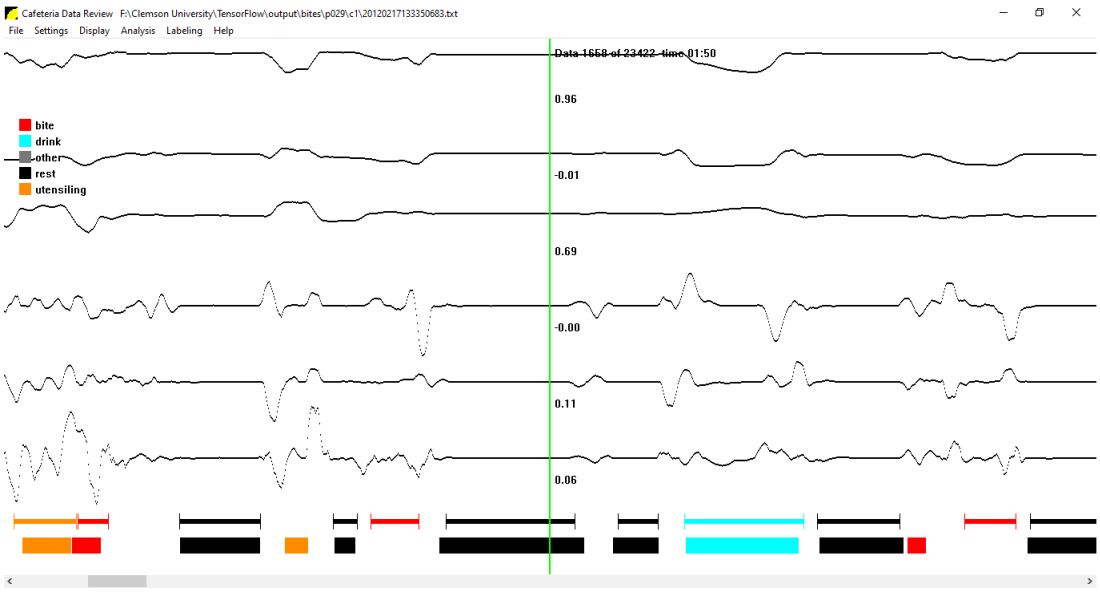


Figure 2.13: CafeView: Viewing ground truth (top) against classifier output (bottom). Gesture labels are red: bite, aqua: drink, gray: other, black: rest, orange: utensiling

the count for start and end was simply reset each time a unique gesture identified as 'unlabeled' was encountered by the program.

Figure 2.13 displays the classifier output viewed against the ground truth label for the same meal shown in Figure 2.1. We observe that early into the labeling process, the classifier output agrees closely with the labels present in the ground truth, which is a good sign when designing neural networks.

2.4 Evaluation Metrics

Once a given meal had been labeled by the neural network following the different strategies described in section 2.3, it was necessary to determine the efficiency of the neural network, and assess how well the generated labels agreed with the ground truth labels provided by the human-raters. The ideas described in this section build on those described under inter-rater reliability in [27]. However their approach was used to resolve discrepancies between two ground truth labels made by two different human raters. This was important because there rarely is perfect agreement even among humans trained by the same operator, and observing the same data. For our approach, however we assumed that the ground truth labels were accurate, and only the classifier output

needed to be evaluated in order to assess the classifier accuracy. This was done in two different ways, as described in this section.

2.4.1 Counting Number Of Indices Agreed Upon

This was the first stage of assessing the effectiveness of the neural network, and the simplest way of understanding whether a trained classifier was good for labeling different instances of time present in the data set. Using this approach, the software kept a count of the number of indices when the label generated by the classifier (after post-processing) matched that present in the ground truth file. This process was repeated in the opposite direction, and the number of indices when the ground truth label matched that present in the classifier label were counted as a way of assessing the bidirectional accuracy of the model.

Bidirectional accuracy in terms of the number of indices correctly identified is important when designing a classifier, because we are interested in knowing how many indices labeled by the classifier are correct. However we are also interested in knowing how many ground truth indices are correctly identified by the classifier as well. This serves as a sanity check for the classifier, and helps us evaluate its performance better.

Consider two examples as explained next. Let us assume that a classifier segments and correctly identifies only two gestures from a meal consisting of ten gestures. In this case the percentage of classifier indices agreed upon is close to 1.0, while the percentage of ground truth indices agreed upon is relatively very low. In contrast consider that a classifier segments and classifies all 10 gestures correctly, but produces superfluous gestures in between each of the 10 ground truth gestures. These superfluous gestures are indicative of the classifier being unable to identify the unlabeled category which occurs between gestures such as bite-drink and utensiling-bite for example. The percentage agreement among ground truth indices would be very high in this case (close to 1.0), while that among classifier indices would be very low.

Both these scenarios are undesired when segmenting our data set. Bidirectional evaluation of indices thus serves as a way for us to understand how closely the classifier output resembles the ground truth, and vice-versa.

2.4.2 Counting Agreement Among Gestures

While counting the number of indices agreed upon by the classifier and ground truth provides a way of assessing the overall accuracy of the classifier, it does not provide a way to determine the gesture level accuracy of the classifier. In other words, there is no way of knowing from the percent agreement among indices how many gestures the classifier labels accurately. Hence an alternative strategy was needed as part of this research, which is described below.

Gestures from ground truth and the classifier output were compared based on their overlap. As mentioned earlier, the ground truth was accepted as the true label of each datum. According to their overlap with respect to the ground truth, classifier gestures were distinguished as being part of one of the following:

1. **Agreement:** One or more classifier gesture matches the ground truth gesture with more than 50% overlap among their indices, and they each have the same label. Depending on the network output, there are three possible ways in which a classifier gesture is categorized under agreement. These are as follows:

   (a) Partial agreement: The start index and/or end index of the classifier output do not match those of the ground truth, but the overlap among the labels is more than 50%. This is shown in Figure 2.14 (a) and (b). Note that an overlap in either direction is treated as agreement among gestures.

   (b) Partial agreement: The classifier identifies multiple unique gestures within the start and end indices of the gesture label, and these have the same label as that of the ground truth label, while the individual start and end indices of the gestures may not match, shown in Figure 2.14 (c).

2. **Missed:** The classifier identifies no gesture within the start and end indices of the ground truth. This indicates that the actual label of the data instances is completely missed by the classifier, and is the first example of erroneous output of the classifier. It is shown in Figure 2.14 (d).

3. **Mislabeled:** The classifier identifies one unique gesture with at least 50% overlap with the ground truth label, but the label of the classifier output does not match that of the ground

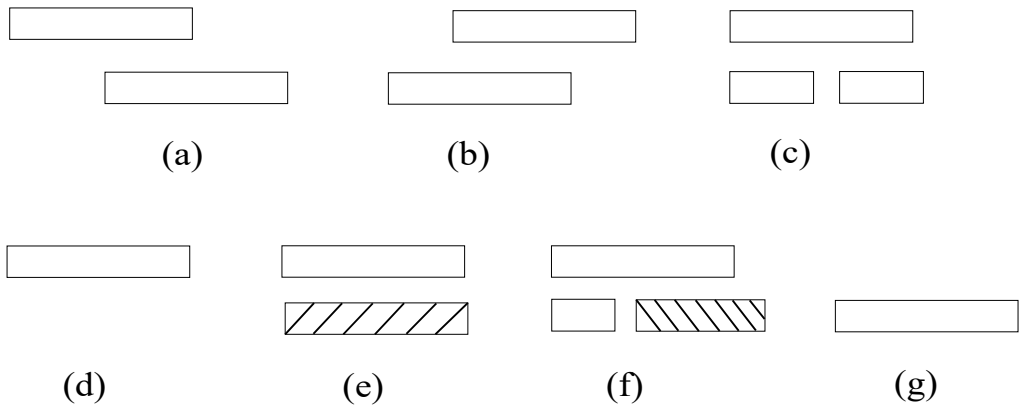


Figure 2.14: Gesture comparison with ground truth (top) and classifier output (bottom). Top row, (a), (b) and (c) represents three ways in which Agreement is identified. In the bottom row we see, from left to right Missed (d), Mislabeled (e), Mangled (f) and False Positive (g).

truth label. This is termed as the classifier mislabeling the instances of time corresponding to this ground truth label, shown in Figure 2.14 (e).

4. **Mangled:** The classifier identifies two or more unique gestures within the start and end indices of the ground truth label, but the label of at least one output gesture does not match that of the ground truth, shown in Figure 2.14 (f).

5. **False Positive:** The classifier identifies a segment of time as a particular gesture, but no label exists within the ground truth file, shown in Figure 2.14 (g).

As mentioned earlier, these tests were adapted from similar tests in [27]. However the author in [27] used these to measure the inter-rater reliability between two human raters labeling the ground truth. For our case we modified these for comparing the ground truth labels that were generated previously, and assumed to be true, against the classifier output.

Chapter 3

Results

This chapter explains the results of training the neural network using the cross-validation technique as described in section 2.2.9, as well the experimental results obtained after evaluating the accuracy of the classifier, both at the index level and the gesture level as described in section 2.4.

3.1 Deep Learning Accuracy

The metrics described in section 2.4 provide a way of assessing a trained neural network, for determining its accuracy in terms of automatically segmenting and recognizing gestures. Prior to this, a neural network also needs to be evaluated during the training process, to ensure that a phenomenon known as overfitting is not occurring.

Overfitting is the process in which a machine learning model adapts itself to produce a perfect or near perfect classification score on the training data set, while completely losing on its ability to classify examples from an unseen test data set. The performance of a classifier on unseen data is known as the generalization performance of the classifier. It is always desired, that the generalization performance of the classifier be as good as possible. Hence overfitting is an unwanted phenomenon when training a classifier. Overfitting typically occurs due to a smaller size of the training data set, a very large and complex model with a lot of free parameters, inadequate size of the testing data set (to determine if overfitting has occurred) or a combination of some or all of these three phenomena.

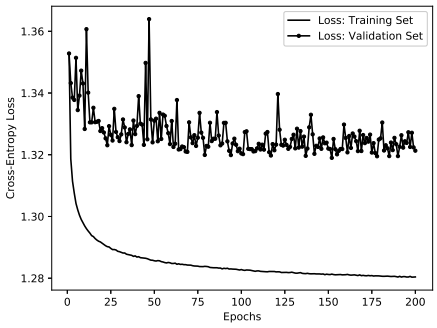
Hence any machine learning model is typically monitored as it trains, by plotting the loss

or error incurred on the training data set, against that incurred on an independent data set known as the validation data set. The validation data set is created by randomly shuffling the training data set, and reserving a small percentage of it, usually 20% as validation data. A model is said to overfit to the training data set, if the loss incurred on this set continually reduces as the learning progresses, while that incurred on the validation data increases or stays the same across multiple training epochs.

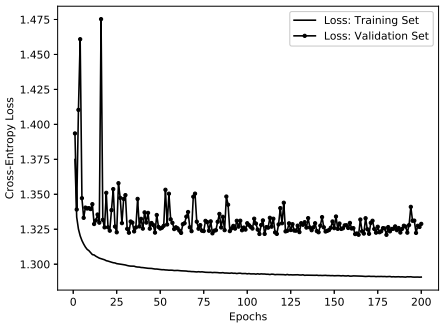
Techniques to prevent overfitting include reducing the model complexity by pruning the number of free parameters, increasing the data set size by data augmentation techniques and usage of a dropout, whereby a fixed percentage of neurons within a layer are turned off (input weights are set to zero), in order to force the leftover neurons to adapt their weights in response to the new activation maps and targets. In addition as described in section 2.2.9, cross-validation can also be used to prevent a model from overfitting to a given training data set. By repeatedly partitioning the available data as training data and testing data, the model learns from different data sets each time, while providing an estimate of its performance on unseen data during the testing phase. Since a large part of the available data is used as training data, the model avoids overfitting to this particular data set.

For our research, the technique of 5-fold cross-validation was used, and the loss-curve was plotted for the train and validation data sets in each fold along with the average of the loss curve for all 5 folds. Figure 3.1 shows the results obtained for each of the five folds from Figure 3.1a to 3.1e, while the Figure 3.1f displays the results obtained after averaging the loss for all five folds respectively.

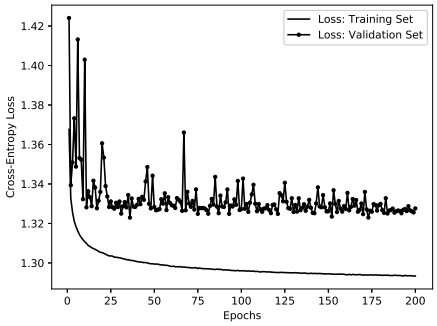
Once each model was trained for 200 epochs, the absolute difference between the validation loss and training loss was computed for each model trained in each fold. These values were 0.0409, 0.0380, 0.0342, 0.0271 and 0.002 for each model, from model 1 to model 5 respectively. The mean and standard deviation of these values was calculated as 0.0286 and 0.0137. Hence we can see that the maximum absolute difference between validation loss and training loss after the model has been trained occurs for model 1, where the value is 0.0409. This value is within one standard deviation of the mean value, and is hence within an acceptable range for the absolute difference between validation loss and training loss for a trained model. This indicates that no model within any of the cross-validation folds has overfit to the training data set, and hence the training of the neural network was successful.



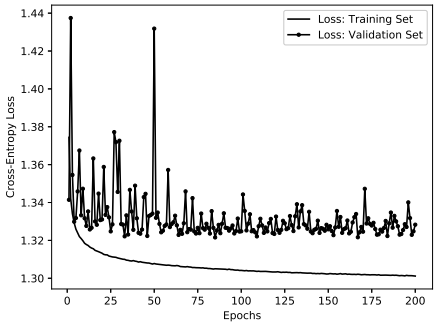
(a) Loss curve for fold 1.



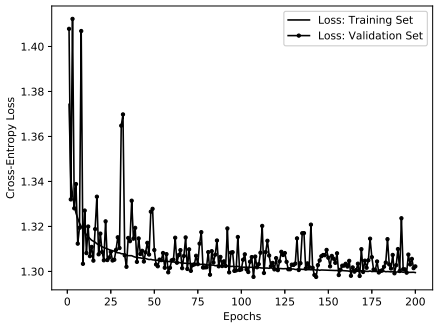
(b) Loss curve for fold 2.



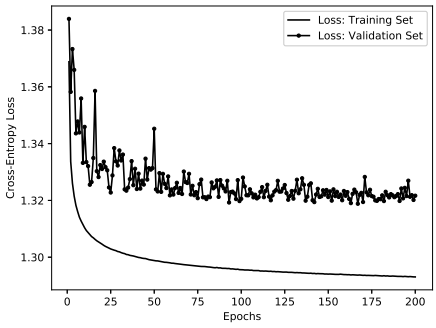
(c) Loss curve for fold 3.



(d) Loss curve for fold 4.



(e) Loss curve for fold 5.



(f) Average loss curve.

Figure 3.1: 5-fold cross-validation for evaluating model learning.

3.2 Evaluation Accuracy

As mentioned in section 2.2.9, cross-validation produces a distinct model for each fold of the data set in addition to a distinct test data set. Hence each model can be evaluated on the unique test set created during each fold, and similar to the earlier discussion in section 3.1, the results can be averaged over each fold to provide a reliable estimate of model performance on unseen data.

First, the total number of unique ground truth gestures and classifier gestures were identified for each meal. Please note that while doing so, classifier gestures outside of the start of the first ground truth gesture and the end of the final ground truth gestures were not considered in the evaluation phase. These gestures are false positives and do not contribute to any other category from those considered in section 2.4.2. In addition there is no way to determine if these are actual false positives, or if a gesture is actually present without having to re-do the entire ground truth process. The raters in this case would need to identify a unique label for each time-instant within a meal, and that was considered to be outside the scope of this research. In addition to removing out of bound gestures before evaluating gesture matching, gestures smaller than 1 second were pruned as well. This is because gestures smaller than 1 second do not occur in the ground truth labels, and would hence negatively impact the evaluation process, just like the out of bound gestures.

Once we have defined the boundaries, we can count accuracy both at the index level and the gesture level as mentioned in section 2.4. It should be noted that the percentage of agreement, missed, mislabeled and mangled was calculated with respect to the ground truth gestures assumed to be correct, while the percentage of false positives was calculated with respect to the classifier output respectively.

3.2.1 Correct Identification Of Indices

As mentioned in section 2.4.1, this is the first level of determining the usefulness of the neural network classifier for the task of automatic segmentation of time-instants. The average percentage of ground truth indices matching those of the classifier output, and vice-versa must both be sufficiently high for the neural network to be considered useful. This indicates that the classifier is able to correctly label a large number of time-instants from all the meals considered. At the same time, the standard deviation of the percentage must be low, which would mean that the amount of indices correctly labeled does not change much from meal to meal.

Metric	Ground truth	Classifier
Average	73.12	79.80
Standard deviation	14.19	9.98

Table 3.1: Percentage of indices correctly identified per recording (meal course).

Table 3.1 lists the average percentage of ground truth indices matching the classifier output, as well as the average percentage of classifier indices matching the ground truth along with the standard deviation of the percentages per meal course/recording. As it can be seen the average percentage of correctly identified indices is high in both cases, while the standard deviation is reasonably low, indicating that the neural network can correctly identify and label individual instances of time for the given data set.

Another thing we observe in Table 3.1, is that the percentage of classifier indices that are correct does not equal the percentage of ground truth indices. This is because the percentages are calculated as the number of time data in agreement divided by the number of total data labeled. Since the ground truth and classifier output may have different numbers of total data labeled, the ratios can be different. As mentioned in section 2.4, both of these percentages are important for evaluating the usefulness of the classifier.

3.2.2 Correct Identification Of Gestures

In order to compare the classifier at the gesture level, the percentage of gestures that was correctly identified, missed, mislabeled, mangled and were considered as false-positives were identified from each meal. Following this the average percentage of each type and its standard deviation was calculated from all meals. This is listed in Table 3.2.

Like earlier, we want that the percentage of correctly identified gestures be as high as possible, while the standard deviation of correctly identified gestures should be sufficiently low. This indicates that the neural network can correctly identify a large number of gestures from each meal, while the number of correctly identified gestures would not vary from meal to meal. On observing the first column in Table 3.2, we see that the numbers appear to be reasonable. On average the neural network identifies close to 78% of gestures correctly from each meal, and the spread of correctly identification for gestures is about 14%. This was seen as satisfactory performance from a neural network for this data set.

In contrast to correctly identified gestures from each meal, the average percentage of gestures

Metric	Correct	Missed	Mislabeled	Mangled	False Positive
Average	77.77	11.27	5.08	5.86	16.37
Standard deviation	13.79	8.62	5.80	5.00	11.28

Table 3.2: Percentage of various mappings between gestures per recording (meal course).

Metric	Bite	Drink	Utensiling	Rest	Other
Average	79.61	80.71	78.71	80.97	0
Standard deviation	19.47	26.78	19.09	18.02	0

Table 3.3: Percentage of individual gestures correctly identified per recording (meal course).

missed, mislabeled, mangled or falsely identified by the classifier must be sufficiently low to prevent erroneous output from the classifier in real-life. In addition the standard deviation of these incorrect gesture level metrics must be lower than that deemed acceptable for the correctly identified gestures. On seeing columns 2-5 in Table 3.2, we see that this is indeed the case. The average percentage of gestures from the classifier output that are mismatched against those from the ground truth is sufficiently low in each case, and the standard deviation in each case also appears to be lower compared to the standard deviation for gestures correctly matched between the classifier output and ground truth. This is further indication of the classifier being useful for the task of automatic segmentation of different gesture types.

3.2.3 Determining Accuracy Of Each Gesture Type

Since the primary goal of this research is to identify eating related activities that occur during a meal, intake gestures viz. bite and drink were the most important gestures among those considered when training our model. Hence in addition to total gesture recognition accuracy, we were also interested in determining how well our model performed at detecting intake related gestures from each meal.

In addition the classifier must also be able to correctly identify other gesture types, in order for the overall accuracy to be sufficiently high. Since we are considering correctly identified gestures, the average percentage of correct detection over all meals should be high, while the standard deviation of the percentage must be reasonably low.

Table 3.3 displays the accuracy of our neural network model for identifying each distinct gesture type. The percentage of correctly identified gestures belonging to each category was calculated for each meal, and then the average and standard deviation were calculated in the same way

Meal ID	Bite	Drink	Utensiling	Rest
215/c3	88.46	88.88	71.42	80.00
274/c1	100	100	97.77	80.00
326/c2	100	100	100	100

Table 3.4: Percentage of individual gestures correctly identified in displayed meals.

as earlier. In Table 3.3, we see that the average percentage of each gesture from 'bite', 'drink', 'utensiling' and 'rest' is sufficiently high, close to 80% in each case. The standard deviation of correctly identified gestures from each type is reasonably low as well, close to 20% in each case. This means that the chosen classifier is good at identifying these gestures from the recorded meals in our data set.

However we observe that gestures belonging to the category 'other' were not identified at all by the classifier. This can be explained by considering how this category of gestures was labeled in the first case as explained in section 2.1.2. The annotators were instructed to label all activities except those related to eating as 'other', including all ambiguous activity that the annotator was not entirely sure of. This meant that the recorded signals from the IMU sensors would have a lot of variability for the period of time corresponding to this gesture type. In addition this type of gesture occurred most infrequently in our data set. Of all the meals retained for training the neural network classifier, only 123 meals have segments of time identified as belonging to the gesture category 'other'. Even in these meals this category occurs very infrequently, typically less than ten times. This means that the frequency of the data corresponding to this gesture type is very low. This is a case of imbalanced data, and typically machine learning models such as neural networks do not perform well on such type of data.

Figures 3.2, 3.3 and 3.4 display the CafeView results for the meals 215/c3, 274/c1 and 326/c2. We see that the classifier is able to identify and segment gestures successfully in each of these three meals. Table 3.4 contains the percentage of individual gesture types correctly identified by the model in each of these three meals. Note that neither of these three meals contained any gestures identified as 'other' in either the ground truth or the classifier output. We see that the percentage of correctly identified gestures is very high in each of these three cases, which corresponds correctly with the displayed results.

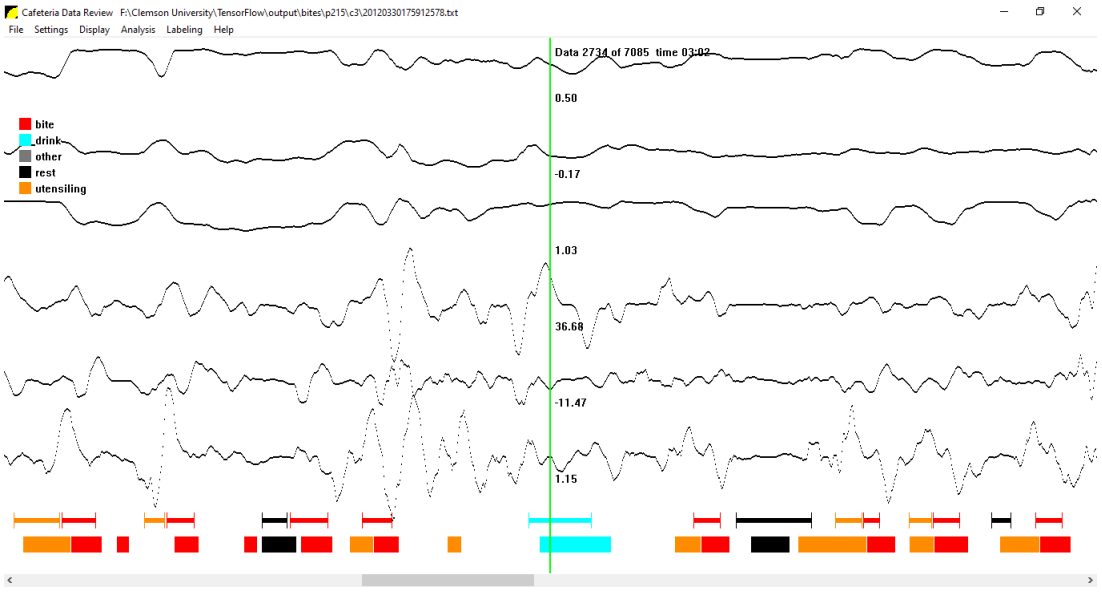


Figure 3.2: CafeView: Comparing ground truth (top) against model output (bottom) for the meal 215/c3. Gesture labels are red: bite, aqua: drink, gray: other, black: rest, orange: utensiling

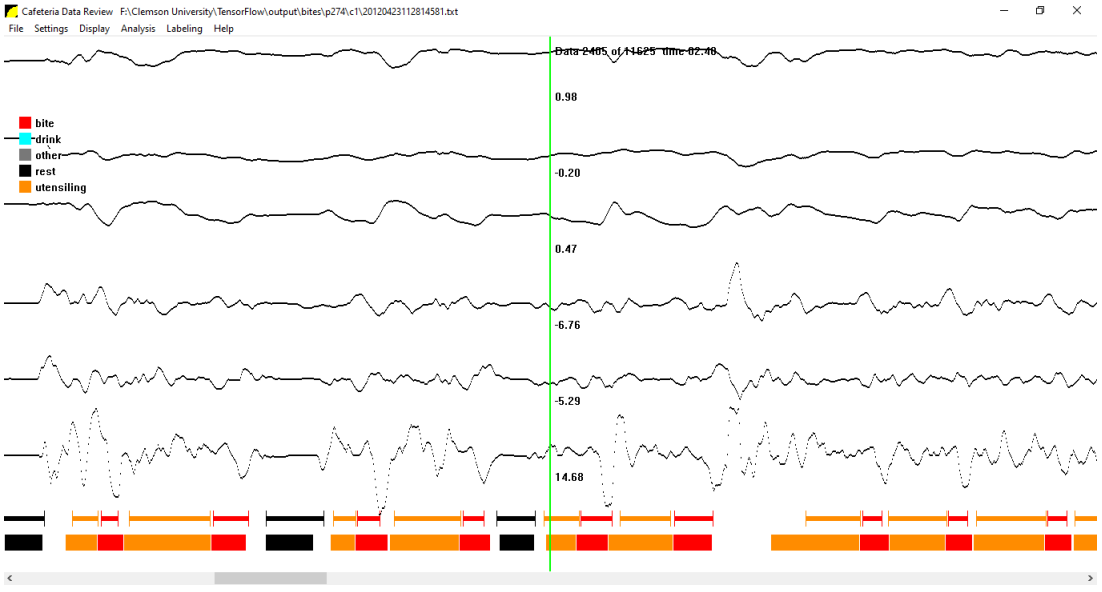


Figure 3.3: CafeView: Comparing ground truth (top) against model output (bottom) for the meal 274/c1. Gesture labels are red: bite, aqua: drink, gray: other, black: rest, orange: utensiling

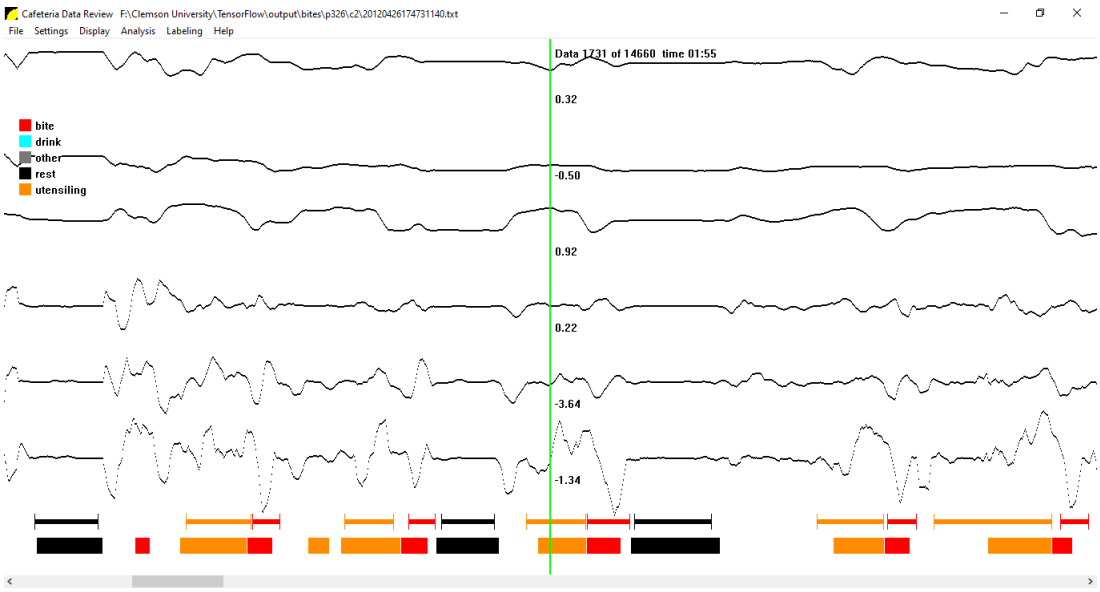


Figure 3.4: CafeView: Comparing ground truth (top) against model output (bottom) for the meal 326/c2. Gesture labels are red: bite, aqua: drink, gray: other, black: rest, orange: utensiling

3.3 Identifying Outliers

In order to understand the performance of the classifier for correctly segmenting gestures from meals, the distribution of the percentage of correctly identified gestures was plotted as shown in the histogram in Figure 3.5. On seeing this plot, we realize that the distribution is long tailed and the percentage of correct classification of gestures drops off drastically 3 standard deviations away from the mean.

On plotting a histogram for the percentage of 'bite' and 'drink' gestures detected correctly, as shown in Figures 3.6 and 3.7 respectively, we see a similar phenomenon. A long tailed distribution falling sharply beyond 3 standard deviations away from the mean. This indicates that some meals are most likely outliers within the data set. Hence it is important to understand the data contained within such meals, in order to understand why the neural network would perform poorly for these meals. This section describes a few important observations regarding outlier analysis, and attempts to explain why the classifier would perform poorly on such data.

Two meals from among the set of outliers were identified as `p170/c1` and `117/c4`. The overall percentage of correctly recognized gestures in these two meals was 37.5% and 21.2%. The percentage of bite and drink gestures correctly recognized in in `p170/c1` was 18.5% and 50% respectively while

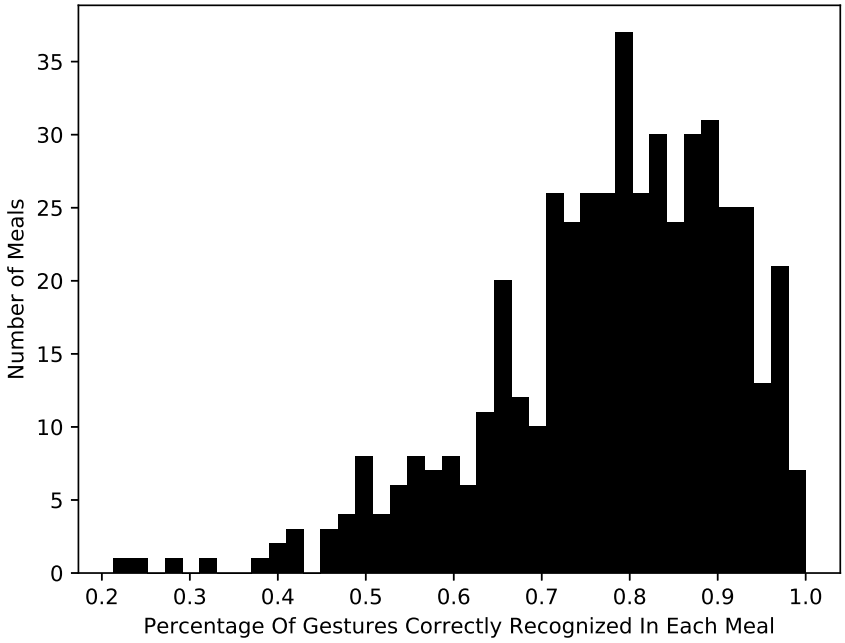


Figure 3.5: Plotting histogram to identify meals with lower total gesture agreement.

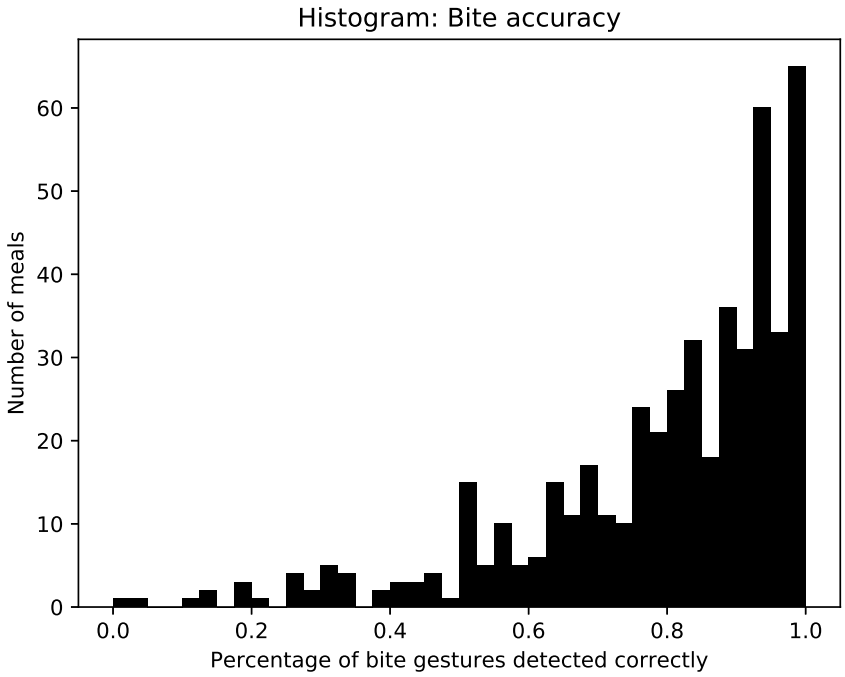


Figure 3.6: Histogram for percentage of bites correctly identified.

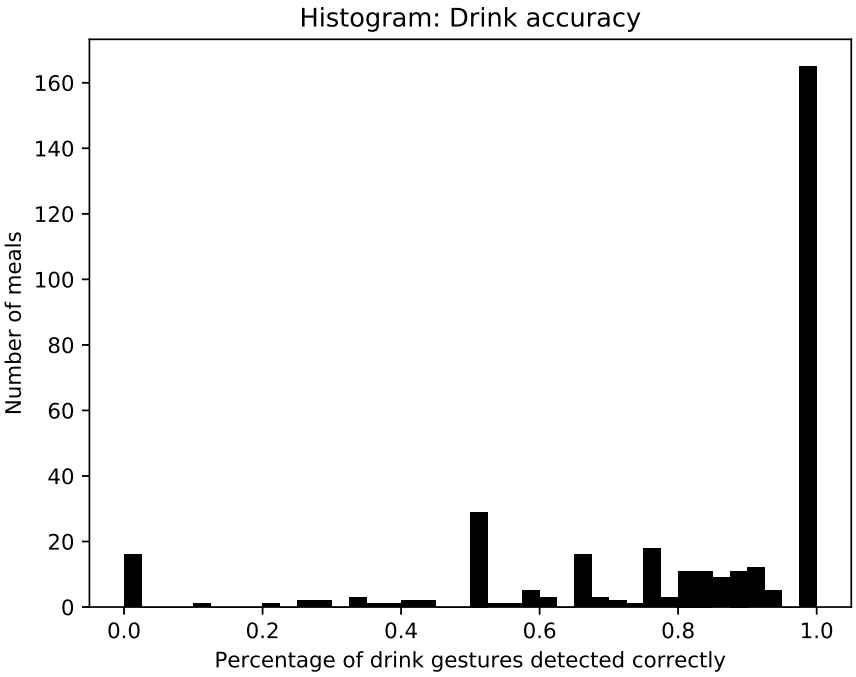


Figure 3.7: Histogram for percentage of drinks correctly identified.

the percentage of correctly recognized bite gestures in `117/c4` was 13.3%. There were no drink gestures in the ground truth of `117/c4` and no gestures identified by the classifier as well.

The classifier output is compared against the recorded data and ground truth labels for these meals shown in Figures 3.8 and 3.9 respectively. In Figure 3.8, observe the two bite gestures, incorrectly identified as rest by the classifier towards the right of the figure. On observing the signals recorded by the IMU sensors corresponding to the two bite gestures, we see that there is no activity present, and the signal values do not show any change in their measurements for a sufficient period of time. This is typical of a recording corresponding to rest, and is seen at the farther end of the same figure.

This phenomenon is seen in Figure 3.9 as well, with multiple gestures belonging to 'bite' and 'utensiling' showing no significant activity measured by the IMU sensors. This indicates that the subject ate their meal with an uninstrumented hand. As explained in [27] the ground truth labels were created by raters, while watching a video of the subject eating their meals. It is most likely that the rater failed to observe the uninstrumented hand being used for food or drink intake, while marking the start and end of the ground truth gesture. As the type of recording does not match with its intended ground truth label, the classifier would not be able to correctly identify such gestures and would eventually get a low percentage of total gestures and intake gestures correct in such meals.

However it is known that motions conducted by one arm/wrist/hand tend to cause related motions in the other arm/wrist/hand. Since the goal of this research is to identify all eating related activities, we consider it important to build a classifier that can detect eating related gestures from an instrumented hand as well as an uninstrumented hand. Hence we hope that our classifier can learn to recognize the motions in the instrumented hand that are related to eating gestures by the other hand. This is discussed further in chapter 4.

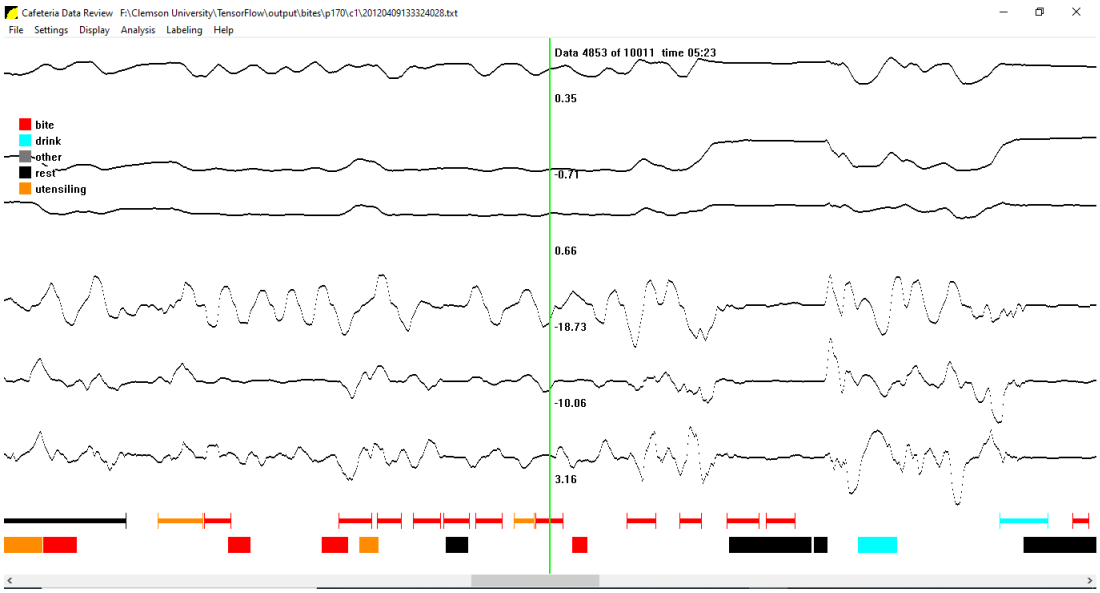


Figure 3.8: Example 1: An outlier meal from the data set.

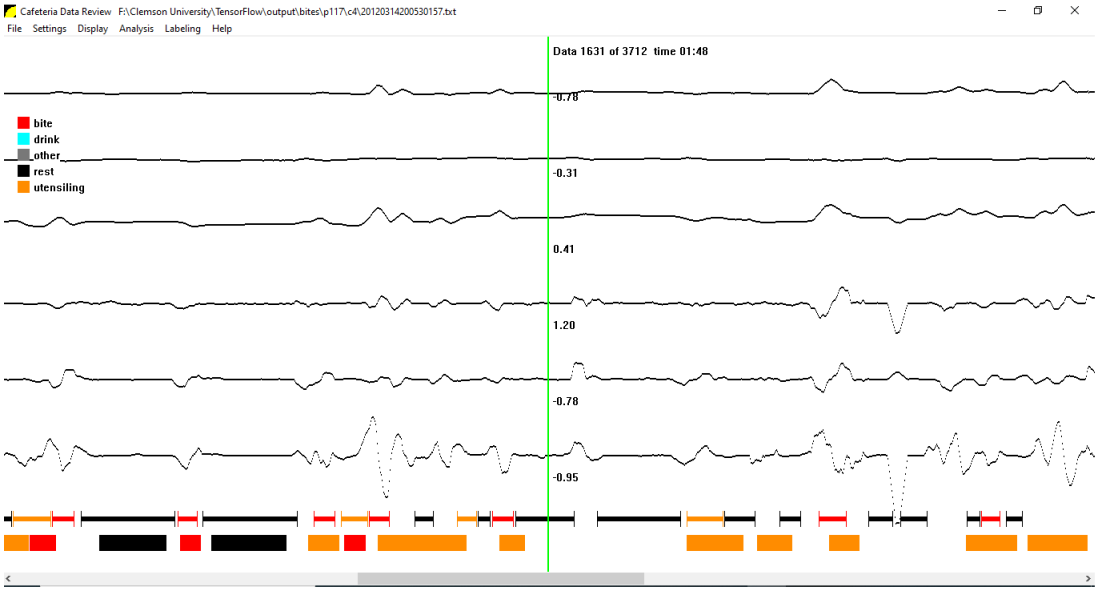


Figure 3.9: Example 2: An outlier meal from the data set.

Chapter 4

Conclusions

This research considers the application of deep learning neural networks for automatically identifying and segmenting eating related gestures from data recorded by hand-worn IMU sensors inside a watch-like device. It builds on the work in [27], and takes its inspiration from [25] and [30], in which a deep learning classifier was used for the task of image segmentation.

Our group has been investigating using wrist-worn sensors for characterizing different eating related activities for tracking periods of intake [8] and also for characterizing the movement from hand-to-mouth for subjects [7], [24]. As explained in [28] these sensors can be encapsulated in a watch-like device, making its use inconspicuous and promoting long-term daily usage. This fits well with our end goal of accurately tracking eating in subjects, and providing the end user with an accurate estimate of the number of calories consumed in a day, and to promote healthier eating in subjects.

Most recently in [27] the researchers identified a potential use for a HMM-based classifier for accurately detecting eating gestures from the same data set as that used for this research. However they considered pre-segmented data of fixed duration for training their model. They also identified the need for a model that could automatically segment gestures from the same data set, without the need for pre-segmented data. This research is a step in that direction, and proposes using a deep learning neural network for segmenting and classifying segments of data from a fixed set of user defined gestures.

Chapter 1 provides a brief introduction to our research, stating that it is of practical use in today's world, owing to the increase in the percentage of people both in the United States of America

and around the world that are deemed to be either overweight or obese. These lead to a variety of diseases, and can be prevented by monitoring one's energy intake, while simultaneously increasing energy expenditure as well. While the majority of focus is on monitoring energy expenditure, there is a lack of clear understanding in terms of monitoring energy intake, which is the main reason why the researchers from our group feel motivated to contribute to this area. This chapter also provides an introduction to IMU sensors, and deep learning neural networks particularly for image segmentation, for readers that may be unaware of these technologies.

Chapter 2 describes our data set and the gesture labeling methodology used in [27], which was used to create the data set used in this research. Based on the original data, windowed segments of measurement data and their corresponding ground truth labels were separated for each meal. This large array of data was created from all meals belonging to the original data, and used to train the neural network. This chapter also provides detailed explanation of our deep learning classifier, its components and design strategies for reliably training the classifier on time-series data such as the one considered in this research. It concludes by explaining how a trained classifier should be evaluated when segmenting and identifying gestures from our data set.

Chapter 3 provides an evaluation for our trained classifier, and compares the average percentage of correctly classified gestures against categories such as missed, mislabeled, mangled and false-positives which are undesirable when segmenting data. Chapter 3 also includes results related to the performance of the model at identifying intake related gestures correctly. It also provides an explanation, when the classifier is unable to correctly identify gestures from some meals. On average our classifier was able to identify 77.77% of all gestures correctly per meal, while the average percentage of correctly recognized 'bite' and 'drink' gestures was 79.6% and 80.7% per meal respectively. In addition our classifer identified 78.71% of gestures corresponding to the class 'utensiling' correct on average per meal, and 80.97% of gestures corresponding to 'rest' were correctly identified on average per meal. However the classifier was not able to identify gestures belonging to the category 'other'. This is most likely due to the imbalance in the training data set corresponding to this gesture type. This gesture occurred most infrequently in our data set, while having a lot of variability in the input signals recorded corresponding to it. Hence our classifier was not able to correctly identify these gestures from any meal within our data set.

It is observed that some meals have ground truth gestures marked by the human rater for activity that occurred with the uninstrumented hand of the subject. Hence the neural network could

not correctly identify gestures from such meals as well. It is known that motions conducted by one wrist tend to cause related motions in the other wrist, and hence we hope that the classifier built will be able to recognize the motions in the instrumented hand that are related to eating gestures by the other hand. This is discussed in section 4.1.

4.1 Future Work

In the future, we recommend training the neural network classifier on data collected from windows of different sizes. It is however unclear how the size of the window would affect the performance of the classifier. A smaller window size would lead to an improvement in the training time of the neural network, but may lead it to miss out on important temporal relationships due to the shorter sequence length of the input. In contrast a bigger window size would mean a longer time to train the neural network, but important temporal relationships would be modeled effectively. Hence the trade-off between different window sizes needs to be carefully considered by researchers in the future.

In addition, the ground truth labels could be expanded to include the instrumented hand with which the gesture occurred. Such an expansion of classes would most likely improve the classification accuracy of the neural network as well, by characterizing the sequence of input data better.

Finally increasing the number of convolutional and deconvolutional blocks in the classifier could potentially lead to an improvement in detecting motions in the instrumented hand that occur due to eating activity in the uninstrumented hand. A deeper neural network would definitely lead to better feature mappings of wrist micromovements that occur during such activities. The concept of wrist micromovements has been previously discussed in [18], where the authors used a deep learning classifier for detecting periods of food intake in subjects. Wrist micromovements are thought to characterize activity of the wrist related to both eating as well as non-eating related gestures. Hence building a deeper neural network to capture more representative wrist micromovements could lead to an improvement in the classifier accuracy for detecting gestures in the instrumented and uninstrumented hands of the subjects.

Bibliography

[1] Oliver Amft, Mathias Stäger, Paul Lukowicz, and Gerhard Tröster. Analysis of chewing sounds for dietary monitoring. In *International Conference on Ubiquitous Computing*, pages 56–72. Springer, 2005.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

[3] Centers for Disease Control and Prevention. Adult Obesity Prevalence Maps. https://www.cdc.gov/obesity/data/prevalence-maps.html, 2019.

[4] Centers for Disease Control and Prevention. Adult Obesity Facts. https://www.cdc.gov/obesity/data/adult.html, 2020.

[5] Dan Ciresan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in Neural Information Processing Systems*, pages 2843–2851, 2012.

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

[7] Yujie Dong, Adam Hoover, Jenna Scisco, and Eric Muth. A new method for measuring meal intake in humans via automated wrist motion tracking. *Applied Psychophysiology and Biofeedback*, 37(3):205–215, 2012.

[8] Yujie Dong, Jenna Scisco, Mike Wilson, Eric Muth, and Adam Hoover. Detecting periods of eating during free-living by tracking wrist motion. *IEEE Journal of Biomedical and Health Informatics*, 18(4):1253–1260, 2013.

[9] Yang Gao, Ning Zhang, Honghao Wang, Xiang Ding, Xu Ye, Guanling Chen, and Yu 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.

[10] Luke Gemming, Aiden Doherty, Jennifer Utter, Emma Shields, and Cliona Ni Mhurchu. The use of a wearable camera to capture and categorise the environmental and social context of self-identified eating episodes. *Appetite*, 92:118–125, 2015.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[12] J Hins, F Series, N Almeras, and A Tremblay. Relationship between severity of nocturnal desaturation and adaptive thermogenesis: preliminary data of apneic patients tested in a whole-body indirect calorimetry chamber. *International Journal of Obesity*, 30(3):574–577, 2006.

[13] Steve Hodges, Lyndsay Williams, Emma Berry, Shahram Izadi, James Srinivasan, Alex Butler, Gavin Smyth, Narinder Kapur, and Ken Wood. Sensecam: A retrospective memory aid. In *International Conference on Ubiquitous Computing*, pages 177–193. Springer, 2006.

[14] A. Hoover. Clemson Cafeteria Dataset. http://cecas.clemson.edu/ ahoover/cafeteria/, 2020.

[15] Qianyi Huang, Wei Wang, and Qian Zhang. Your glasses know your diet: Dietary monitoring using electromyography sensors. *IEEE Internet of Things Journal*, 4(3):705–712, 2017.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[17] G. Koley. ECE 6320, Instrumentation: Class Notes, 2020.

[18] Konstantinos Kyritsis, Christos Diou, and Anastasios Delopoulos. Modeling wrist micromovements to measure in-meal eating behavior from inertial sensor data. *IEEE Journal of Biomedical and Health Informatics*, 23(6):2325–2334, 2019.

[19] R. O'Reilly, A. Khenkin, and K. Harney. Sonic Nirvana: Using MEMS Accelerometers as Acoustic Pickups in Musical Instruments. https://www.analog.com/en/analog-dialogue/articles/mems-accelerometers-as-acoustic-pickups.html, 2020.

[20] Sebastian Päßler and Wolf-Joachim Fischer. Food intake monitoring: Automated chew event detection in chewing sounds. *IEEE Journal of Biomedical and Health Informatics*, 18(1):278–289, 2013.

[21] Sebastian Päßler, Wolf-Joachim Fischer, and Ivan Kraljevski. Adaptation of models for food intake sound recognition using maximum a posteriori estimation algorithm. In *2012 Ninth International Conference on Wearable and Implantable Body Sensor Networks*, pages 148–153. IEEE, 2012.

[22] Temiloluwa Prioleau, Elliot Moore II, and Maysam Ghovanloo. Unobtrusive and wearable systems for automatic dietary monitoring. *IEEE Transactions on Biomedical Engineering*, 64(9):2075–2089, 2017.

[23] P. Louis Pröve. An Introduction to different Types of Convolutions in Deep Learning. https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d, 2017.

[24] Raul I Ramos-Garcia, Eric R Muth, John N Gowdy, and Adam W Hoover. Improving the recognition of eating gestures using intergesture sequential dependencies. *IEEE Journal of Biomedical and Health Informatics*, 19(3):825–831, 2014.

[25] Olaf Ronneberger, Philipp Fischer, and Thomas 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.

[26] Dale A Schoeller. Limitations in the assessment of dietary energy intake by self-report. *Metabolism*, 44:18–22, 1995.

[27] Y. Shen. *Using Contextual Information to Improve Hidden Markov Model Recognition of Wrist Motions During Eating Activities*. PhD thesis, Clemson University, December 2018.

[28] Yiru Shen, James Salley, Eric Muth, and Adam Hoover. Assessing the accuracy of a wrist motion tracking method for counting bites across demographic and food variables. *IEEE Journal of Biomedical and Health Informatics*, 21(3):599–606, 2016.

[29] Mingui Sun, Lora E Burke, Zhi-Hong Mao, Yiran Chen, Hsin-Chen Chen, Yicheng Bai, Yuecheng Li, Chengliu Li, and Wenyan Jia. ebutton: a wearable computer for health monitoring and personal assistance. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6, 2014.

[30] TensorFlow. Image Segmentation. https://www.tensorflow.org/tutorials/images/segmentation, 2020.

[31] Guotai Wang, Wenqi Li, Maria A Zuluaga, Rosalind Pratt, Premal A Patel, Michael Aertsen, Tom Doel, Anna L David, Jan Deprest, Sébastien Ourselin, et al. Interactive medical image segmentation using deep learning with image-specific fine tuning. *IEEE Transactions on Medical Imaging*, 37(7):1562–1573, 2018.

[32] J. Watson. MEMS Gyroscope Provides Precision Inertial Sensing in Harsh, High Temperature Environments. https://www.analog.com/en/technical-articles/mems-gyroscope-provides-precision-inertial-sensing.html, 2020.

[33] World Health Organization. Obesity. https://www.who.int/topics/obesity/en, 2020.

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

[35] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535. IEEE, 2010.

[36] Rui Zhang and Oliver Amft. Bite glasses: measuring chewing using emg and bone vibration in smart eyeglasses. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers*, pages 50–52, 2016.

[37] Rui Zhang and Oliver Amft. Monitoring chewing and eating in free-living using smart eyeglasses. *IEEE Journal of Biomedical and Health Informatics*, 22(1):23–32, 2017.