11-2023

Continuous Time Recurrent Network for Human Activity Detection

Abdallah Al Zubi
University of Nebraska–Lincoln

Continuous Time Recurrent Network for Human Activity Detection

by

Abdallah Al Zubi

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Architectural Engineering

Under the Supervision of Professor Fadi Alsaleem

Lincoln, Nebraska

November 2023

Continuous Time Recurrent Network for Human Activity Detection

Abdallah Al Zubi, M.S

University of Nebraska, 2023

Adviser: Fadi Alsaleem

Human activity detection is crucial to personalize the control of the building environment. For example, understanding certain human activities (e.g., walking, sitting, etc.) for an occupant in a building helps provide the proper thermal comfort control. However, these applications require large-scale neural networks that are challenging to implement and train.

In this thesis, we implemented a continuous-time recurrent neural network implementation (CTRNN) network that can solve real-time human activity detection with a smaller-size network. The CTRNN uses differential equations with a time constant to describe the neuron equations. It was implemented and trained for the first time using TensorFlow. Specifically, the forward path of the CTRNN was implemented using a new recurrent cell in TensorFlow, and the training was performed while utilizing the auto-differential function to implement the backpropagation through-time algorithm.

The CTRNN showed a high ability to capture a complex pattern in the temporal data of the acceleration measurements of human activities. More importantly, despite the smaller size, we show that the CTRNN outperforms the standard recurrent neural network (RNN) in accuracy, convergence, and stability. The research also investigates the impact of training the time-constant parameters of CTRNN for the first time to improve its performance.

ACKNOWLEDGMENTS

I am grateful to my adviser, Professor Fadi Alsaleem, for his tremendous assistance and mentorship throughout my graduate studies. I particularly appreciate Professor Moe Alahmad and Professor Milad Roohi, who have graciously served on my committee. Furthermore, it is important to express gratitude for the substantial financial support from an IARPA award, which was spearheaded by the Research team at General Electric. Dr. David Lin and his team also deserve recognition for their valuable Input, which significantly enhanced the successful culmination of this project. Finally, I express my deep appreciation for my parents, relatives, acquaintances, and my spouse and girls, whose steadfast backing has served as an essential reservoir of resilience.

Table of Contents

List of Figures

List of Tables

Chapter 1
INTRODUCTION AND BACKGROUND

1.1 Motivation

Human activity detection has many applications in health care, security, and robotics [36-37]. Human activity detection is critical in customizing smart buildings. Metabolic rate is a critical determinant of the comfort zone [36], which may be assessed by human activity analysis. Machine learning algorithms have reached high levels of complexity with many parameters to train that surpass the computational capabilities of average CPUs in personal computers [11-13]. To give a perspective, the Megatron model by Nvidia has 8.3 billion parameters to be trained [11], the BERT (Bidirectional et al. from Transformers) has 110 M trainable parameters [9], and the BART (Bidirectional and Auto-Regressive Transformers) model by Facebook has about 140 M trainable parameters [10]. An alternative is to use specific clusters of servers owned by big companies like Google, Facebook, and IBM to perform such computing. However, those digital-based supercomputers need tremendous power and are not widely spread to be used by average users.

Conversely, analog computing, which mimics how a human's brain works, needs less power to operate [14]. Inspired by the human brain, the Continuous Time Recurrent Network (CTRNN) is a machine learning algorithm that can be solved analogously. Differential equations are used to describe its neuron equation. The response of the biological human cell behavior inspires this differential equation. Our hypothesis is this differential equation brings richness to the network compared to the discrete equations that describe the standard recurrent neural network (RNN) and are solved using digital computers. RNN is a machine learning algorithm that extracts knowledge from sequential data with different input sizes. We expect a CTRNN with fewer neurons and, thus, fewer parameters to perform similarly to an RNN with many neurons. To validate this hypothesis, we explore an innovative approach to implementing and training CTRNN in this thesis and compare its accuracy and network size performance to standard RNN. As a case study, we targeted human activity classification problems. This problem involves the complex task of dealing with the temporal behavior in time series data.

1.2 Literature Review

Human activity detection is crucial in developing smart buildings. For its implementation, it utilizes many technologies like wearable, camera-based, infrared (IR), ultrasonic, and Wi-Fi sensors [39]. Human activity detection enables intelligent, responsive environments that adapt to the resident's needs and preferences. Recognizing human activities will also help save energy consumption, enhance security, improve indoor environmental quality, and provide personalized services. For example, the work [40] employed human activity recognition to manage HVAC systems, lighting, heating, ventilation, and air conditioning by responding to occupancy and activity patterns, decreasing energy usage, and enhancing energy efficiency. Moreover, the work in [41] utilized human activity recognition, which identifies irregularities, like unauthorized access or atypical behavior, thereby bolstering security measures and thwarting potential intrusions.

Complex machine learning algorithms such as the RNN that capture the temporal behavior in the training data are needed for human detection. Capturing such behavior and data patterns by emulating the neuron response of a neural network is not new. Various works attempt to construct mathematical models to simulate and discover these patterns in the data. The Ising model is one of the first formal examples [2]. The Ising model was not originally formulated as a neural network but provided the fundamental basis for machine learning in computer science. It captures dependencies and interactions between variables or features of interest. In the early 80s, the first fundamental form of RNN, Hopfield Networks [1], was published. Hopfield Networks is a recurrent neural network with a straightforward, directed cyclic graph structure. However, it is essential to note that this Hopfield-based RNN type has some constraints regarding its ability to handle large amounts of data and its potential for expansion. Thus, while Hopfield network theory has historically attributed significance to simpler models, the complexity of dealing with more complex applications has driven the need for more complex RNN models and new training algorithms.

In 1990, Elman [3] introduced a trainable RNN. This new recurrent architecture could learn the network parameters and backpropagate throw times. Known as backpropagation, this new training algorithm helps the model tune its parameters by propagating the error from the output layer back into the input layer. Another mathematical model for the RNN is called the Jordan network [12]. However, most of these RNN models have some drawbacks, like exploding and vanishing problems, especially if they have an extended period to remember (long window size). In 1997, the Long Short-Term Memory LSTM [7] model was proposed to tackle those problems. From its name, this cell has different parts, each with a different role. One focuses on the long memory part, the other focuses on the short memory part, and the last works on mixing those memory parts. While LSTM has shown significant improvement compared to the standard RNNs, a major limitation is each neuron has multiple parameters to learn compared to an RNN neuron. Thus, the LSTM has more parameters to train and tune.

Around the same period, Randall Beer officially introduced Continuous Time Recurrent Neural Networks CTRNNs [3]. Beer focused on developing neural models that mimic biological neurons' dynamic behavior. CTRNNs use differential equations to simulate the continuous response of a human brain neuron. CTRNN has many field applications in robotics and signal processing due to its ability to capture the temporal behavior in the data.

Training the CTRNN is currently not well explored. The differential equation of the CTRNN creates many difficulties using standard training algorithms such as backpropagation. For those reasons, scholars have trained the weights of CTRNN in different unconventional ways. These training methods include evolutionary algorithms approaches. For example, in [3], the authors use Differential Evolution (DE) algorithms to find the optimal parameters for CTRNN. The DE algorithm is based on the well-known Genetic algorithms [13]; those search algorithms aim to find a suitable solution in a reasonable time. It keeps developing more elite generations to solve the problem. In [8], CTRNNs were trained using the Genetic algorithm for generating music. This study showed that CTRNNs could capture intricate and continuous temporal structures in musical data, generating more natural and expressive music [8]. Besides trying different training algorithms, there has been a growing use of the CTRNN method for specific applications. For example, the work in [5] uses CTRNN to forecast the glucose level of patients inside the ICU. The study was conducted on 200 patients; they compared the results with different models, such as a linear model and autoregressive gradient boosted trees. In the end, the CTRNN outperformed all other models. Another work used the CTRNN to learn how a bipedal robot stands without falling [4].

Despite the encouraging results of CTRNN, its implementation in real applications is very limited due to the difficulty of training its differential equation parameters. To overcome this challenge, in this thesis, we will develop a novel training algorithm for a CTRNN to train its parameters automatically. These parameters include the CTRNN time constants, which have been mostly assumed to be a constant value in the literature, thus limiting the CTRNN performance. Classifying different types of human activities is used as a case study to validate this new training algorithm.

1.3 Thesis Objectives

This thesis explores the feasibility and efficacy of training a Continuous Time Recurrent Neural Network (CTRNN) to classify human activity detection. It compares the performance of the CTRNN to the standard Recurrent Neural Network (RNN) algorithm. The CTRNN weights, including the time constant parameter, will be trained using the Adaptive Moment Estimation algorithm (ADAM) [19]. The results of both algorithms will be compared in terms of network size and other parameters.

1.4 Problem Statement

The main challenge in this AI is the complexity of the algorithms and the computing resources needed to train the models. Any improvement to the building blocks of those algorithms would significantly impact the big building models. One of the most popular models is the sequence-based model. These models learn how to extract knowledge from sequential data. This thesis explores utilizing sequence-based continuous models that use data and emulate real-time applications. Specifically, we use the CTRNN mathematical model to address human activity classification and compare its performance with the standard RNN model regarding accuracy, stability, and parameter size. The RNN model encounters challenges such as exploding, vanishing, and cataphoric forgetting problems, which hinder its coverage capabilities. We will explore using CTRNN to avoid those problems with smaller network sizes and fewer parameters to be trained. An added benefit of the CTRNN mathematical paradigm is that it will enable physical devices to perform computing. Many physical systems, such as microelectromechanical systems (MEMS), were shown in the literature to have a similar equation to the CTRNN neuron equation [6][21]. Thus, as those tiny mechanical systems vibrate, they will naturally solve the CTRNN equation without needing a digital computer!

Chapter 2
CTRNN MATHEMATICAL MODEL

The CTRNN represents the dynamic behavior of neuroscience within a continuous time domain using a set of differential equations. The differential equations explain the temporal evolution of the neuron's state. The neuron’s states include the response rate of individual neurons as quantified by the time constant and the interplay between the dynamics of neighboring neurons. Additionally, they account for various activation functions that may be employed and how these functions integrate these variables with the external input. This chapter aims to comprehensively describe the continuous-time recurrent neural network (CTRNN) mathematical model. We also explain the proposed training algorithms to learn and optimize its parameters.

2.1 CTRNN Model

The CTRNN mathematical model consists mainly of six preliminary components, as shown in Figure 2.1 and Equation 2.1. The differential part in Equation 2.1 makes them intricately intertwined to mimic the behavior of human brain neuron cells [4].

$$\tau_i \frac{dy_i}{dt} = -y_i + f_i(B_i + \sum_{j=1}^{n} w_{ij}.y_j) + I_i(t) \qquad (2.1)$$

where,

τ_i: Time constant of neuron i,

y_i: Neuron i current state,

f_i: Activation function, the nonlinearity of neuron i.

w_{ij}: Weights vector for neuron i.

B_i: Biase shift for neuron i,

$I_i(t)$: external Input at time t for neuron i.

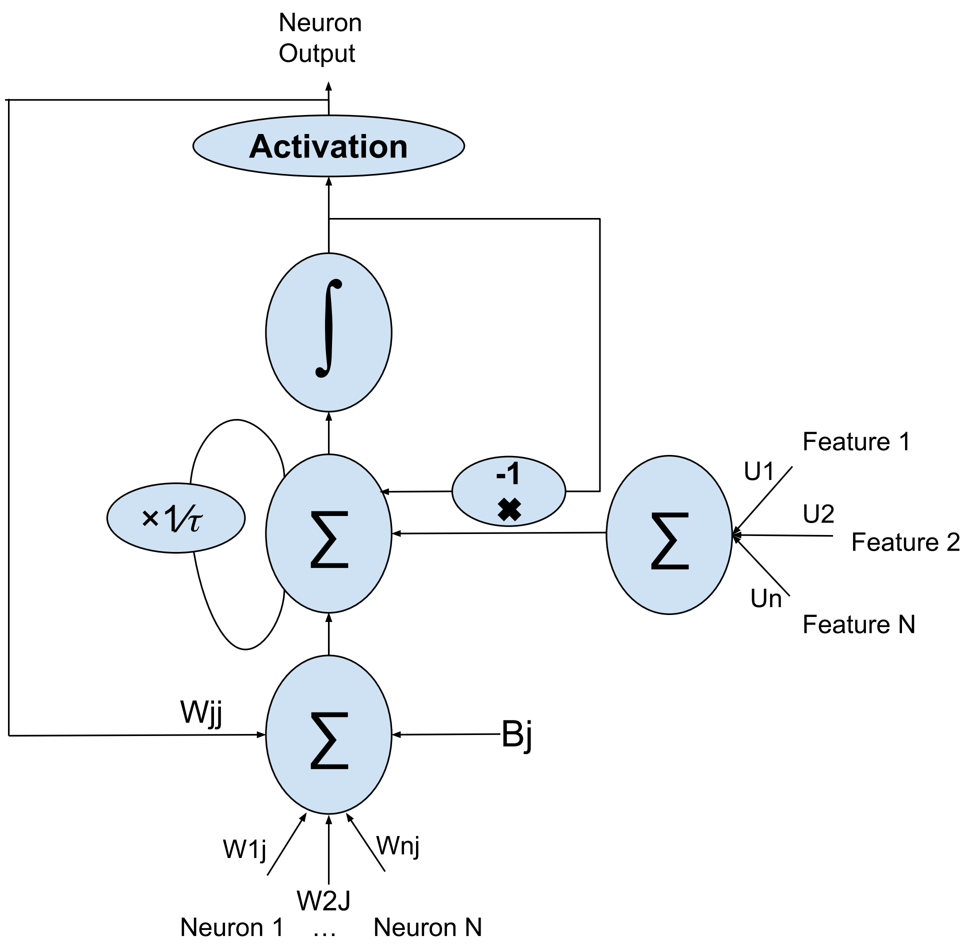


Figure 2.1 Continuous Time Recurrent Neuron (CTRNN) neuron structure.

The first part is the continuous state of the current neuron, represented by y_i. This state changes over time and depends on the previous state and all the other five parts. The second part is the activation or nonlinearity function, which is denoted f_i. Sigmoid, RELU, and hyperbolic tangent function (Tanh) are all examples of widely used functions. The behavior of a biological neuron inspires the activation function to capture the nonlinearity of a complex pattern. As a group of neurons activates together, signal propagation through them in a specific path. Different paths capture different patterns. Thus, another way to think about the activity is as a threshold for the neurons to capture complex patterns. The third part is the synaptic weights vector, represented in w_{ij}. It describes the strength of the connection between neuron j and neuron i. The fourth part is the external input, $I_i(t)$, which represents the continuous input signal from the sensors or other data sources. Usually, the ML model uses a weighted sum rather than the input itself. The fifth part is the time-constant τ_i, which controls how quickly the current cell reacts to any external input; high τ means the neuron needs more time to respond and change its status, while low time constant means this neuron has a rapid response to any change. The last part is the neuron dynamics [16], which is represented by the rate of change of the state (differential part), $\frac{dy_i}{dt}$. CTRNN has a different structure than the RNN, as shown in the simple RNN structure in Figure 2.2.

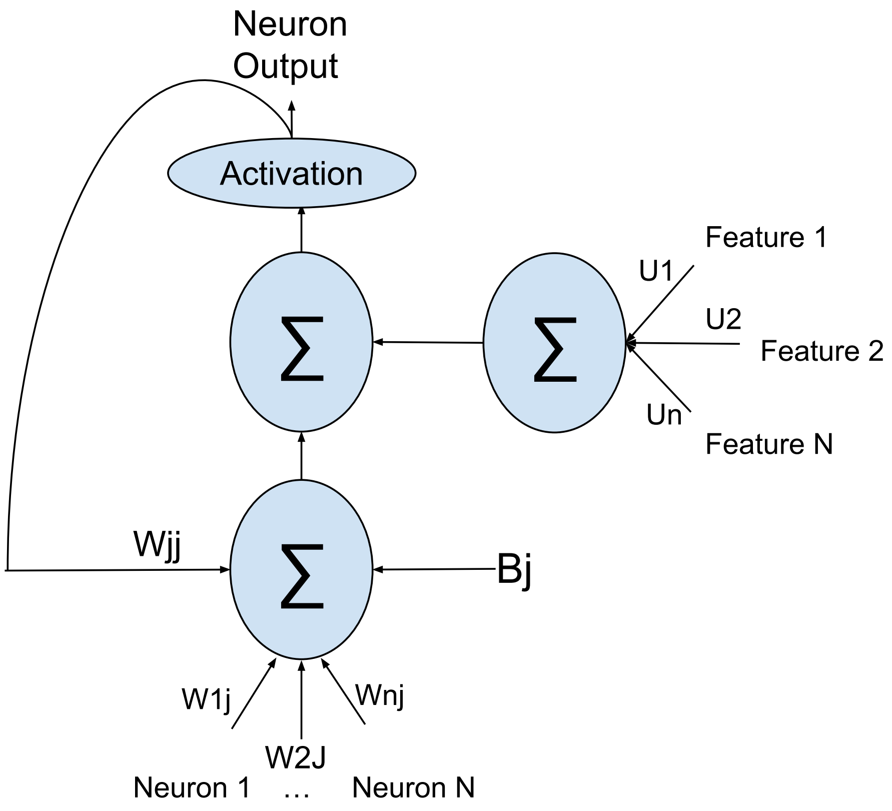


Figure 2.2 RNN Neuron Structure, input from features and others neuron states.

It is evident that while the CTRNN possesses one more parameter, it exhibits a higher level of complexity than the RNN, which is comparatively more straightforward.

2.2 Time Constant (Tau) Effect

To study the effect of the time constant on the CTRNN response, we represent the equation of CTRNN using the Euler method. The Euler method is recognized as one of the straightforward techniques for solving differential equations [22]. For better comprehension, assume that the variable h equals 1.

$$\tau_i \frac{y_{i+1} - y_i}{h} = \frac{1}{\tau_i} * [-y_i + f_i(B_i + \sum_{j=1}^{n} w_{ij}.y_j) + I_i(t)] \quad (2.2)$$

$$y_{i+1} = [y_i + \frac{1}{\tau_i} * [-y_i + f_i(B_i + \sum_{j=1}^{n} w_{ij}.y_j) + I_i(t)] \quad (2.3)$$

The above equations imply that as the value of τ_i approaches a large value, the subsequent state y_{i+1} equals the current state y_i (will not have much change). Conversely, when τ_i approaches zero, the next state will be strongly influenced by input changes or other neurons' status.

It is crucial to select the τ value greater than h [15], as if h>> τ equation 2.2 indicates that the second term of the equation will be multiplied by a value greater than 1, causing system instability. This instability arises from amplifying the other neurons' external input and output, leading to an unstable neuron state. As depicted in Figure 2.3, which shows the output of a single isolated neuron with external input, the selection of h further contributes to the coverage speed in the model. When h=2 and τ=1, the neuron's output oscillates around the external input value. Conversely, when h=2.5, the output diverges. Therefore, to ensure the stability of the neuron and the model, it is recommended to choose h<< τ [16].

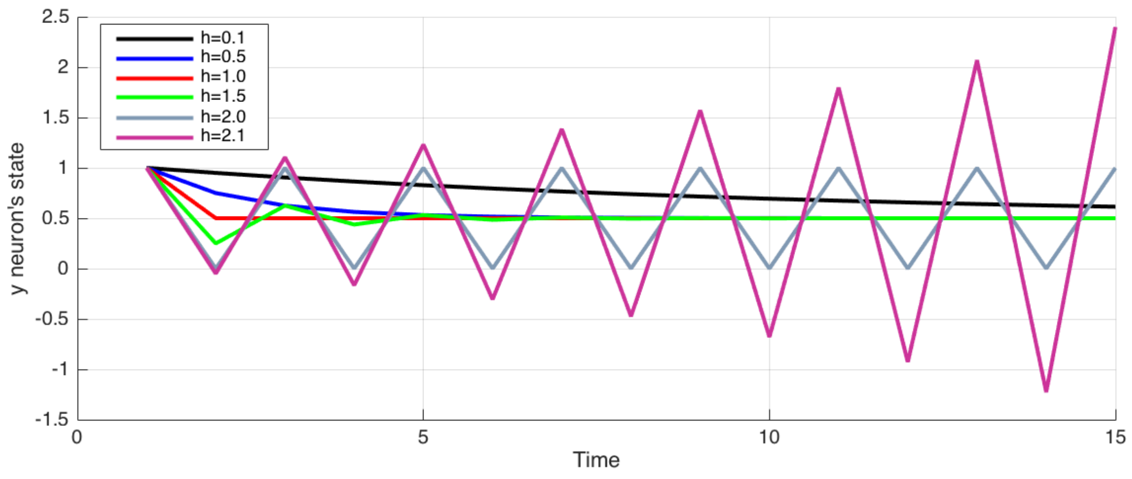


Figure 2.3 Effect of h vs τ on stability of isolated neurons τ =1, external input =0.5 [16].

2.3 Unfolding The Model

CTRNN, like RNN, could be represented as a cyclic graph, where neurons are the nodes graph and edges represent the weights. Each node uses its previous state, external input, and other nodes' output to update its state in this representation. Unfolding refers to the sequential presentation of the cell state across time. In this context, X(t) represents the input at a time, 't', h(t) represents the internal state of the neuron at that moment, and O(t) represents the output at that time. The unfolding of a typical RNN is shown in Figure 2.4. For the CTRNN, it is essential to include the time constant parameter in the equation by creating an extra self-loop, as shown in Figure 2.5.

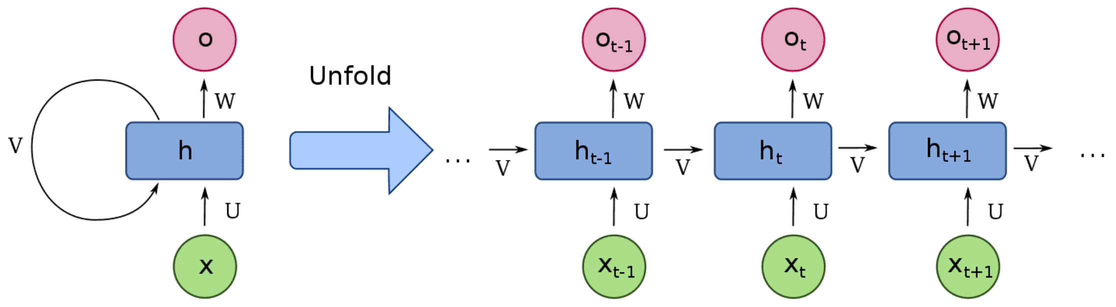


Figure 2.4 RNN cell and RNN unfolding through time [17].

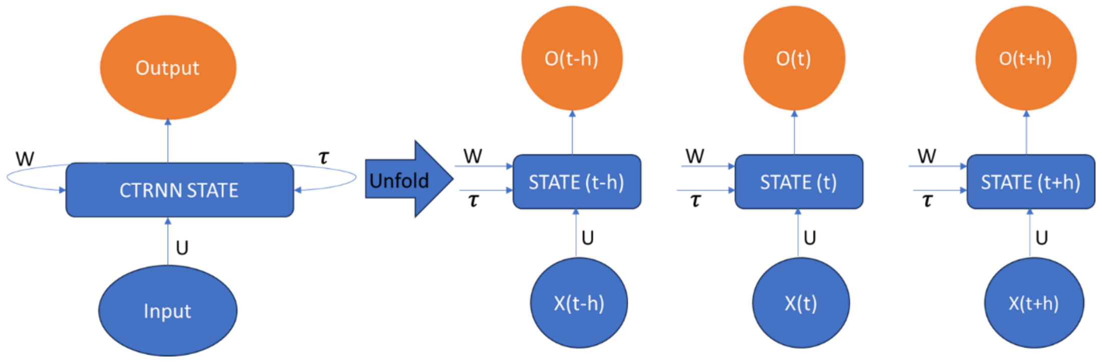


Figure 2.5 CTRNN and CTRNN Unfolding through time.

Figure 2.5 shows that every neuron needs the preceding state, external input, and output from the other neurons to evaluate the next state. Training the input weights, internal weights, and time constant is necessary to optimize the model's performance. In this work, we adapt the backpropagation and backpropagation to perform such training. Further details on those algorithms and optimizing the weights will be discussed in the subsequent sections.

2.4 Backpropagation and Backpropagation Through Time

The Universal approximation theorem states that a neural network with sufficient neurons and appropriate activation functions can approximate any continuous function [18]. The neural network is a set of interconnected layers containing a vector of neurons. The vision of machine learning is to approximate desired functions by learning the best weights that map the input to the output. For example, the function could classify cat vs. non-cat images. In this case, the output will be either Yes (if the image has a cat) or No otherwise. Thinking of the image as a matrix of pixels, the input of the ML function would be a matrix of pixels and an array of weights; the output would be either Yes or No, F(image, Weights)$\rightarrow${Yes, No}.

The main challenge is training those weights to optimize the function. The most popular technique is called backpropagation. In this technique, when data is fed into the network, it flows sequentially from the input layer to the hidden layer and finally to the output layer. The output is then compared to the ground truth (the true data). The loss is a function that mirrors the difference between our model expectation and the true data, is computed. The backpropagation adjusts the weight of the whole network starting from the output layer and propagates back into the input layer to reduce this loss function (reduce the error). As the training goes back through each layer and reaches the recurrent layer with a self-loop, the network should be unfolded, as discussed in section 2.3. Thus, the backpropagation method must go through every time step to train the inner weights. To achieve this, the CTRNN must use the Backpropagation Through Time (BPTT) concept to adjust the internal weights and the time constant.

2.5 Optimizer Algorithm

As discussed in the backpropagation section, to update the weights, the algorithm needs to backpropagate from the loss function to the output layer until it reaches the input layer. To do that, we need to take the partial derivation of the loss function to each weight parameter to know in which direction (opposite direction of the derivative) the weights should be adjusted to optimize our model (reduce the loss function). The learning rate is how we adjust those weights [26]. The equation for an adaptive optimizer is shown in Equation 2.3.

$W_{updted} = W_{old} - learningRate * \frac{dLoss}{dw}$ (2.3)

Figure 2.6 shows a schematic of how the ADAM Optimizer works. Typically, it begins with random weights, symbolized by the red point in the Figure. If the derivative of the loss function to the weight is negative, then this will increase the weight by the learning rate multiplied by the derivative amount. The yellow one will be the new weight. Vice versa, if the solution begins with the green point, the derivative will be positive, and the weights will be decreased.

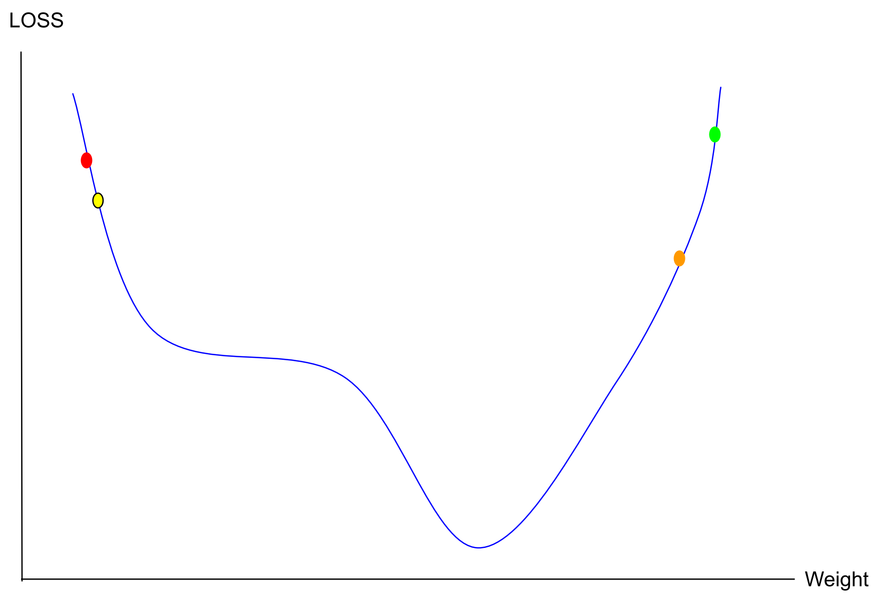


Figure 2.6 ADAM Optimizer algorithms, how it works, and convergence.

The ADAM Optimizer employs adaptive learning rates to accelerate the convergence of the model. During the initial learning phase of the model, it commences with a random initialization. The ADAM optimizer leverages a movement's momentum to accelerate the progression consistently, leading to smooth convergences [19]. Its ability to quickly converge and handle noisy or sparse data makes it a vital training tool for deep neural networks [19].

2.6 SoftMax layer

SoftMax layer is the output layer widely used with multiclass problems [28], like identifying human activity. Usually, this layer has neuron numbers equal to the number of classes. If we have an image containing digits from 0 to 9, we need one neuron for each digit fired when the model classifies the image to this number. In the human activities problem, we have six activities, which means six neurons. It normalizes the score between the neurons to have a sum of one. Thus, Using the SoftMax layer in the model will add the capability to compute the probability of how the model is undoubtedly about its answer. Some real-world applications take critical action and consider the model's confidence above a certain threshold [28].

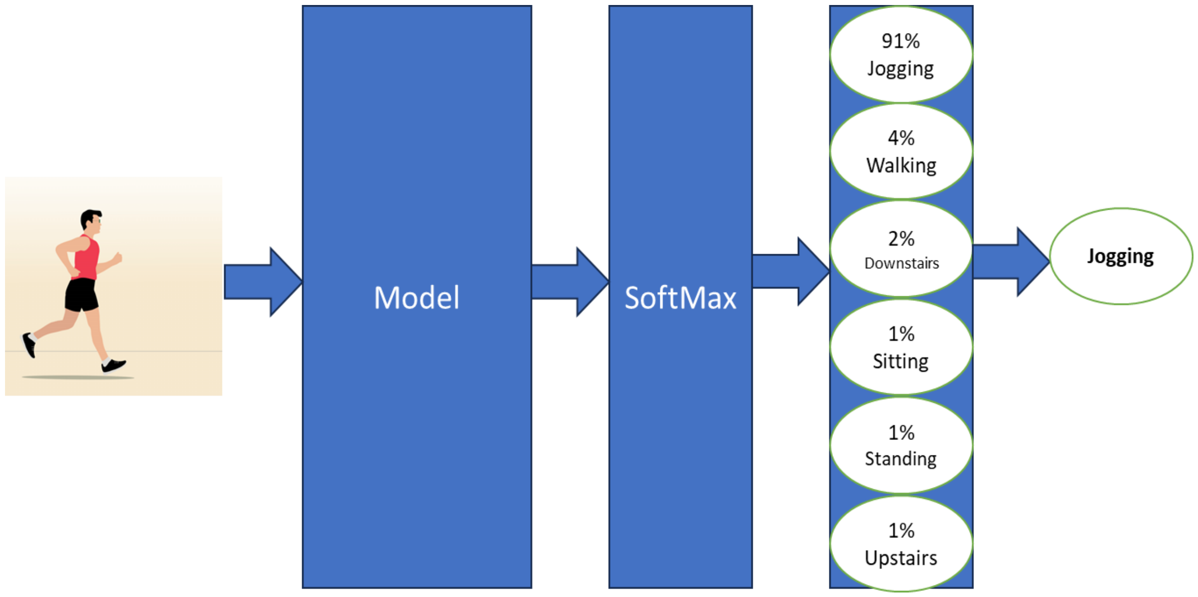


Figure 2.7 Understanding the Function of a SoftMax Layer in Classifying Six Distinct Activities.

2.7 Limitation on Time Constant Parameter (Tau)

The field of machine learning focuses on recognizing and evaluating complex patterns present in data. This pattern is a mathematical function of the input features and the trainable parameters. Machine learning usually focuses on extracting those patterns without considering the parameters' values. Training the time constant in the CTRNN is different as this parameter has a physical meaning and has some restrictions, like it should be a positive number. One could use the regularization technique to add a limitation on training the time constant to account for this constraint. In this method, the parameter of interest, the time constant here, will be added to the loss function to force the training to minimize those parameters [30]. But this method will not always guarantee a parameter not to exceed a specific value.

Alternatively, TensorFlow has non-popular functionality to override the constraints of the parameters of each cell to have values from a specific distribution [35]. In this work, we adapt this method to limit the time constant value during the training.

Chapter 3 METHODOLOGY

This chapter discusses the essential steps associated with implementing CTRNN. These include selecting the training data and the employed libraries, evaluating the performance of the CTRNN models, and performing a comparative analysis compared to RNN models. Furthermore, this chapter explores the operational principles of the CTRNN network using different time constant values.

3.1 Implementing CTRNN Model

To simulate the CTRNNs, it is necessary to solve the associated ordinary differential equations (ODEs). One commonly used numerical method for solving ODEs is the Euler method [22], derived from the Taylor series. This method is discussed in equation (2.3). CTRNN was implemented using the TensorFlow library, widely considered a top-tier machine-learning framework developed by researchers at Google. TensorFlow is esteemed for its trendy nature, reliability, and adaptability [23]. The proposed approach involves constructing a CTRNN cell capable of solving the underlying differential equation and simultaneously learning the weights and time constant. To facilitate the implementation of backpropagation through time (BPTT), the auto-differentiation technique [24] and the built-in ADAM optimizer in TensorFlow are used. As stated before, the built-in ADAM optimizer will be utilized with some modifications to tune the time constant.

3.2 Validating CTRNN MODEL

To validate the CTRNN model, we start testing it on the small and well-known MNST dataset. MNST is a benchmark dataset set comprising handwritten numerical digits ranging from 0 to 9. The dataset has 60 thousand training and 10 thousand test instances, each representing a grayscale digit image with dimensions of 28x28 pixels. The initial findings displayed encouraging results, indicating that the CTRNN exhibited a better learning capability than the RNN. The accuracy of the CTRNN reached 0.714 compared to 0.680 with RNN, as shown in Figure 3.1 for the CTRNN an example. Both networks use 8 recurrent cells in the first layer, another 8 cells in the second layer, and a SoftMax (one neuron for each class) to classify the images.

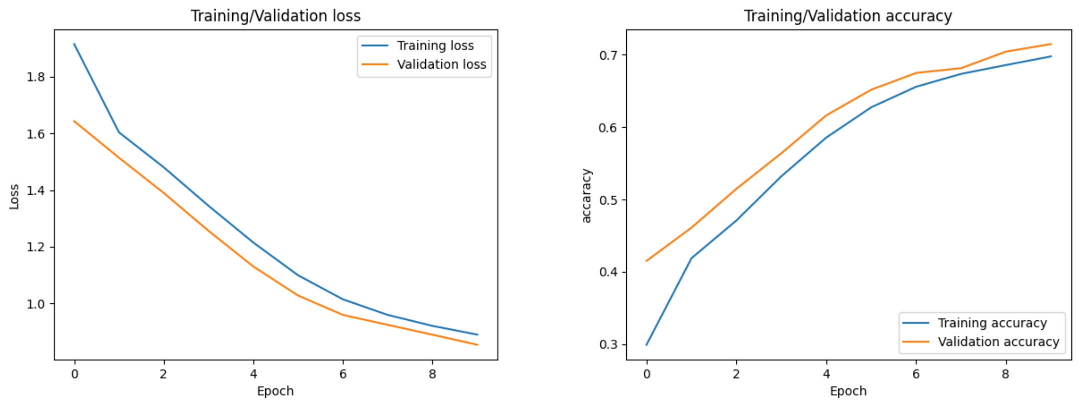


Figure 3.1 Convergence a of CTRNN,8-8-10, running ten epochs accuracy 72%.

After validating our model, the next step is to test it on the human activity dataset and compare it with the naïve RNN model.

3.3 WISDM Dataset

WISDM stands for wireless sensor data mining [20]. The data contains six attributes extracted from smartphones for various groups of people while carrying those devices, and the data has acceleration features. Number of examples: 1,098,207 records; the. It contains six different activities: jogging, walking, upstairs-downstairs, sitting, and standing. Here are snapshots of the dataset: user ID, activity type, timestamp to formulate a sequence, and accelerometer in X, Y, and Z directions.

Table 3.1 Sample Data from the WISDM Dataset, with X, Y, and Z Representing Different Acceleration Directions

ID	**Activity**	**Timestamp (ms)**	**x**	**Y**	**Z**
33	Jogging	491059623	-0.69	1.27E+01	0.50
33	Jogging	491060623	5.01	1.13E+01	0.95
33	Jogging	491061122	4.90	1.09E+01	-0.08
33	Jogging	491062223	-0.61	1.85E+01	3.02

Table 3.2 Exploring Statistical Features and Characteristics of the WISDM Dataset

	X	**Y**	**Z**
count	1085360	1085360	1085360
mean	0.670708	7.34148	0.415928
std	6.889081	6.739406	4.781942
min	-19.61	-19.61	-19.8
25%	-2.96	3.34	-2.26
50%	0.34	8.01	-0.04
75%	4.48	11.65	2.76

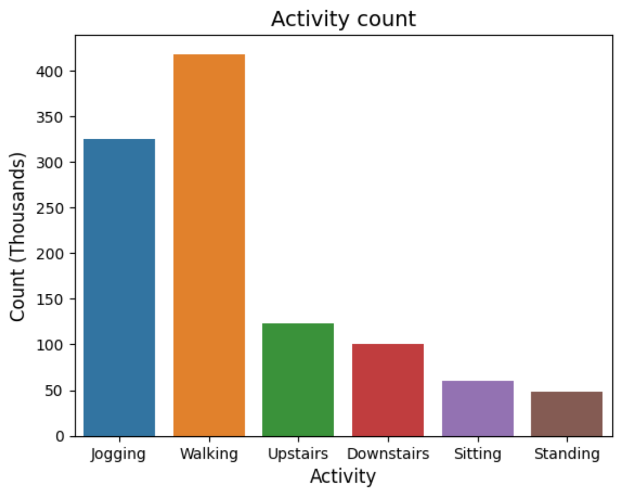


Figure 3.2 WISDM dataset Activities counts walking and jogging represent more than 50% of data.

3.4 Preparing the data for training

The proposed approach systematically traverses the dataset. We used well-known algorithms called sliding windows [33]. The algorithm employs a predetermined window size and constant sliding steps. The purpose is to group subsequent records into one entity. As illustrated in Figure 3.5 we picked the window size to be fifty records, and that should be option based on the literature [34]. To maintain data integrity, we include instances where the dominating activity constitutes at least 70% of the observations. Including this percentage will ensure a higher degree of activity purity.0

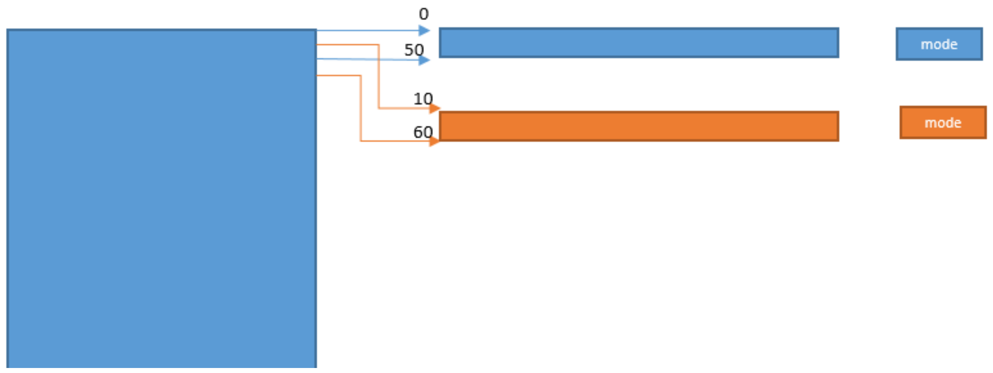


Figure 3.3 Utilizing a sliding windows algorithm to preprocess the WISDM dataset for training.

The data was divided into three distinct portions. Precisely 10% was allocated for testing. while 90% was left for training. From the training data 10% was reserving as a validation dataset to evaluate the model's performance while training. Cross-validation [25] techniques were employed to assess the model's behavior. Cross-validation could help the model to avoid problems like overfitting. Overfitting is a scenario where the model performs well during training but poorly when applied to testing data [26].

3.5 Network Architecture

The network consists of a set of consecutive connected layers. The first layer has 16 recurrent cells. The first layer is fully connected with the second layer, which has 10 cells. The final layer has six cells, one for each human activity.

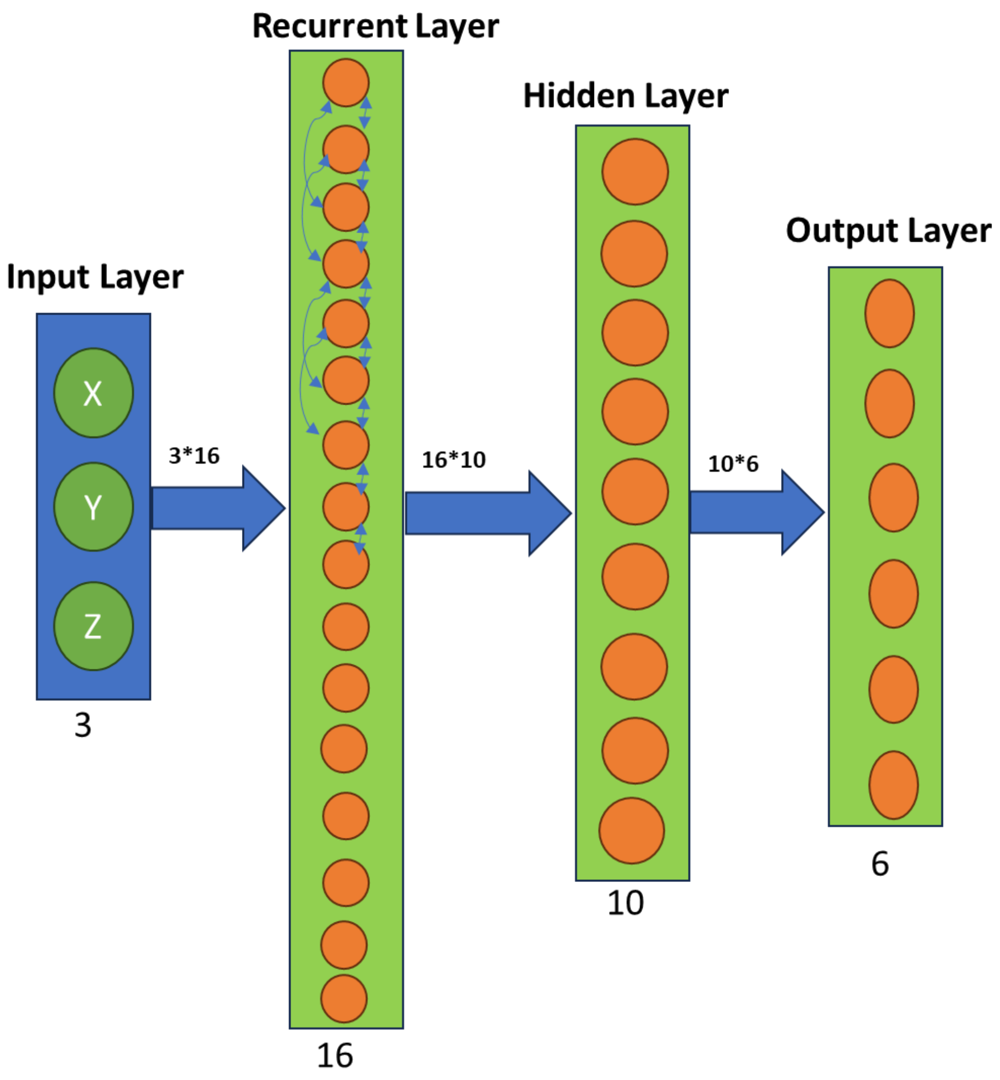


Figure 3.4 WISDM dataset Model CTRNN Network Architecture

Chapter 4 Results

This chapter presents the results of using the CTRNN for human activity classification. We Built the CTRNN and compared it to the standard RNN. The results of this study offer significant insights into CTRNN's capabilities and behavior. Furthermore, we investigated the effect of the time-constant parameter on the training of CTRNN. We conducted a study to examine different scenarios for the time constant. The results reveal the significant influence of this parameter on the training process.

4.1 Training CTRNN on WISDM dataset

To verify that the model converges on the WISDM dataset, we train the model for a limited number of iterations using a small portion of the dataset. In this analysis, the CTRNN model was initially trained for only five epochs, using ten percent of the dataset. The model exhibited promising signs of learning and achieved an accuracy of approximately 65%, as shown in Figure 4.1. In comparison, the RNN took 20 epochs to reach close to that accuracy, as shown in Figure 4.2.

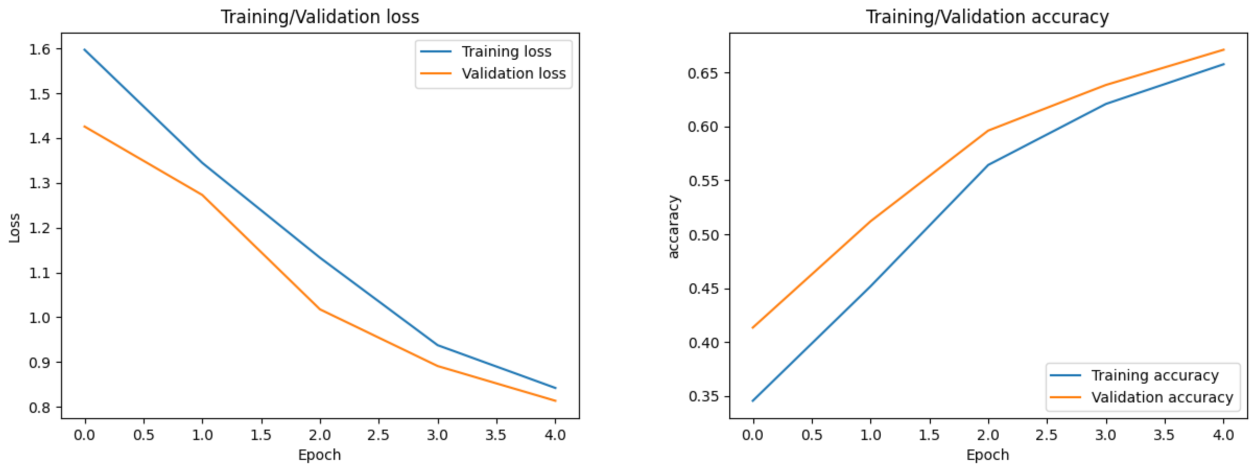


Figure 4.1 Convergence and accuracy curves of CTRNN run for five epochs; the model reaches an accuracy of around 65%.

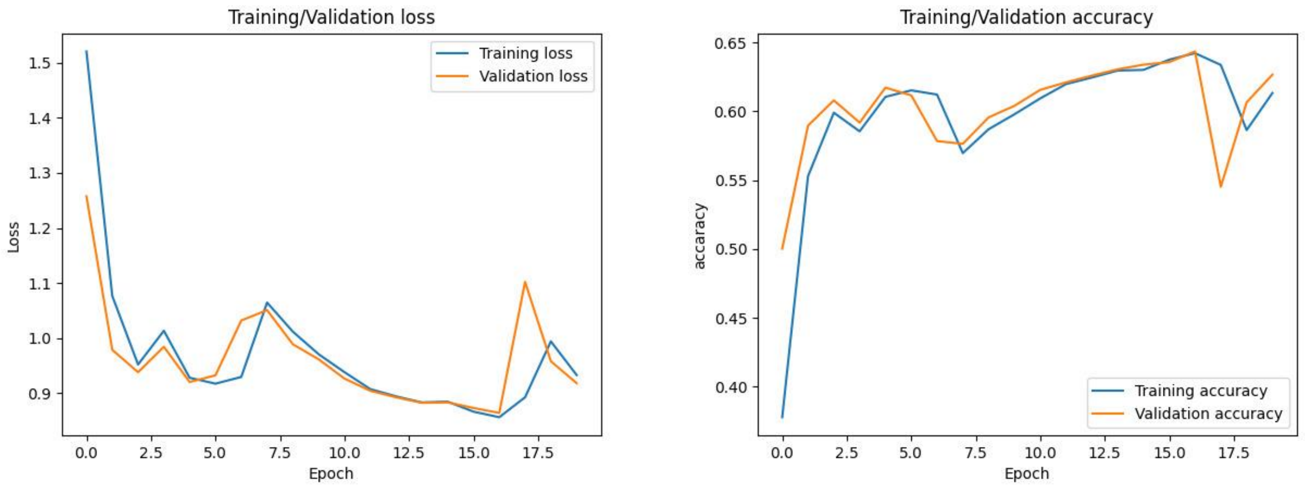


Figure 4.2 Convergence and accuracy curves of RNN Running 20 epochs; the model reaches an accuracy of around 62%.

Subsequently, the model was further trained for 20 epochs on the whole dataset, resulting in an accuracy of 74%, as shown in Figure 4.3. The RNN model accuracy is around 65% in similar conditions, as shown in Figure 4.4.

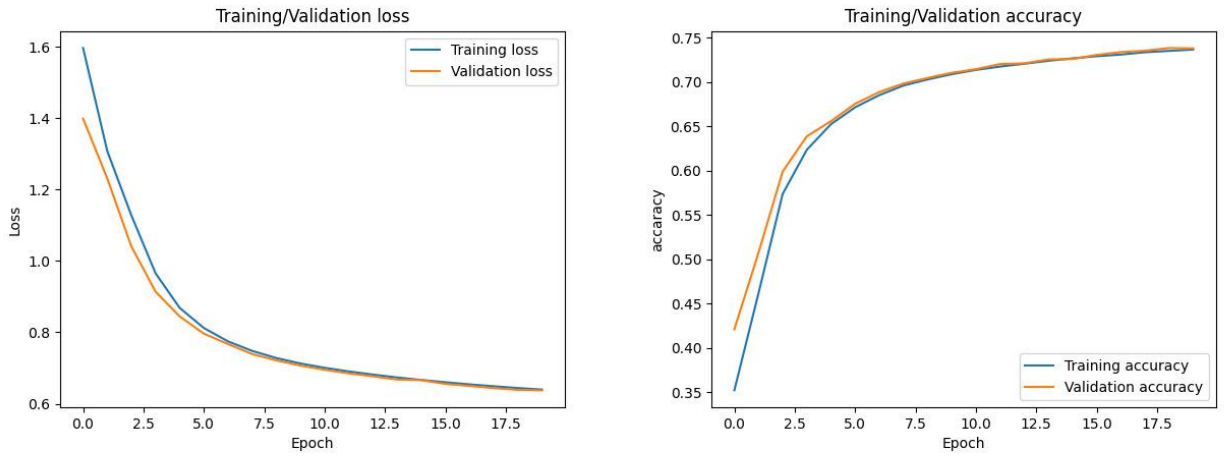


Figure 4.3 Convergence and accuracy curves of CTRNN Running 20 epochs, the model reaches an accuracy of around 74%.

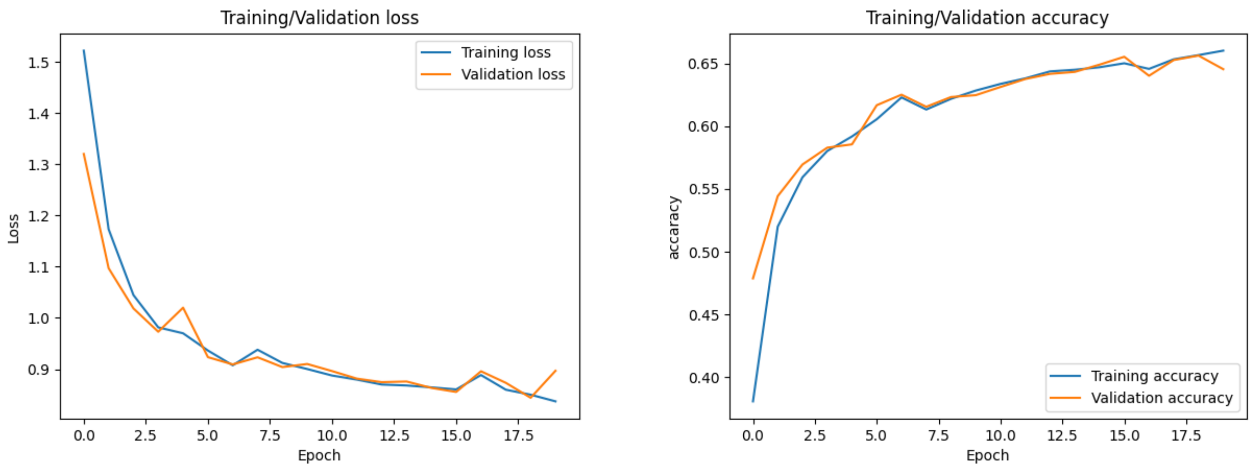


Figure 4.4 Convergence and accuracy curves of RNN Running 20 epochs, the model reaches an accuracy of around 65%.

In both models, it became evident that the model needed further time to converge. Next, we let the model complete an extensive training session of 50 epochs. At this stage, the model attained an approximate accuracy of up to 82% for the CTRNN model, as shown in Figure 4.5.

In this model, we also train the time constant, as explained in the next section.

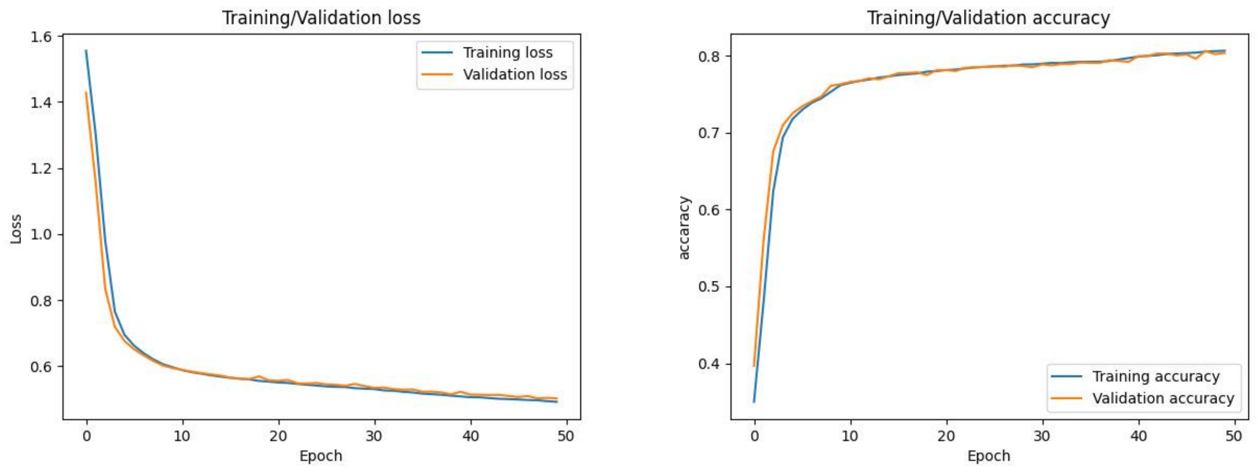


Figure 4.5 Convergence and accuracy curves of CTRNN Running 50 epochs, the model reaches an accuracy of around 82%.

CTRNN demonstrates a smooth convergence process without facing any significant difficulties. Moreover, the validation and training convergence curves come above each other, which is a good sign of smooth learning. In contrast, the convergence behavior of the classic RNN displays significant variability, and the accuracy is less than 73%, as shown in Figure 4.6.

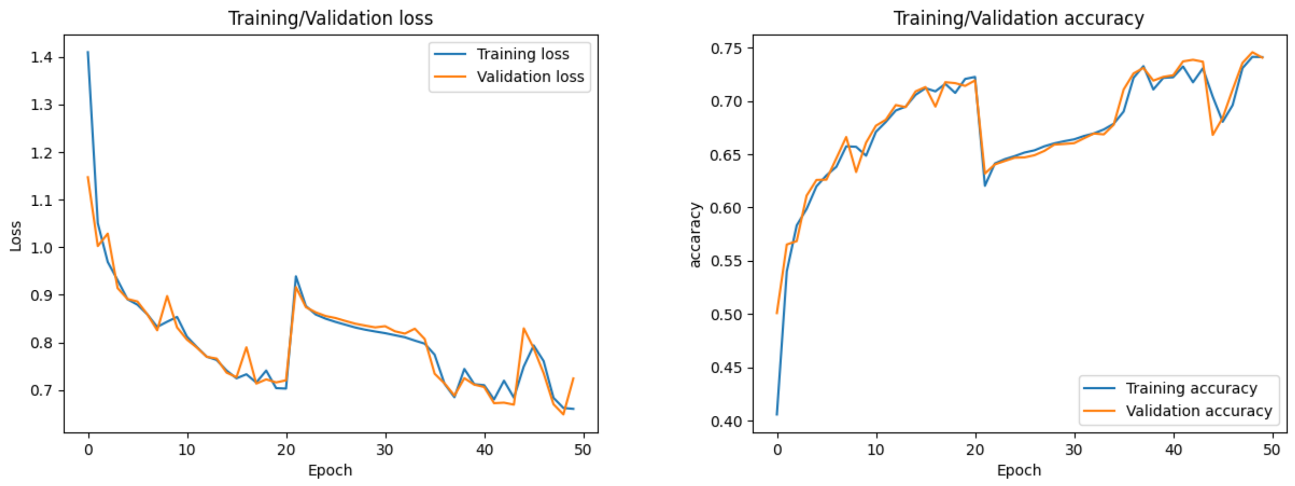


Figure 4.6 Convergence and accuracy curves of RNN Running 50 epochs, the model reaches an accuracy of around 73%.

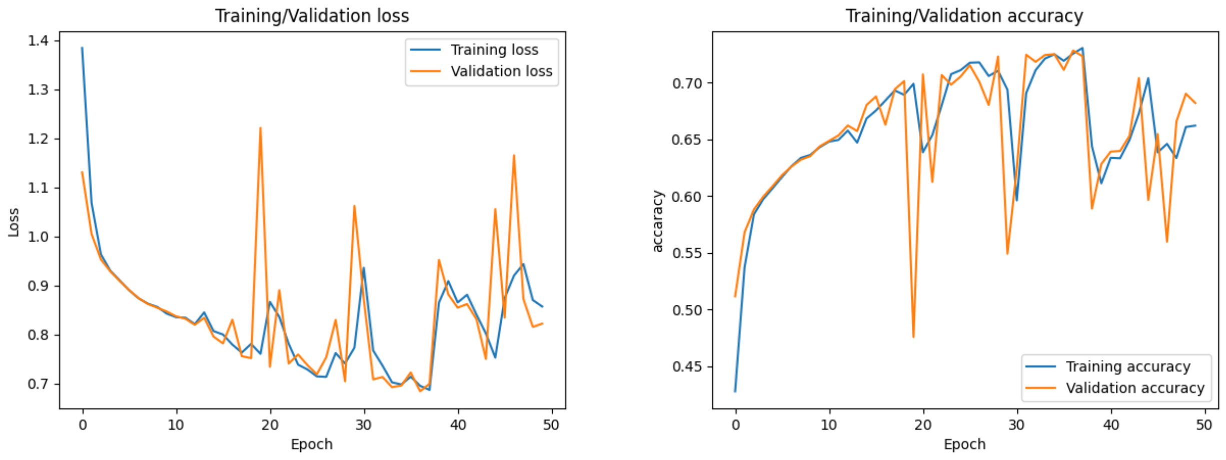


Figure 4.7 Convergence and accuracy curves of RNN Running 50 epochs, the model reaches an accuracy of around 68%.

In another RNN training episode, as shown in Figure 4.7, the convergence of the RNN is rugged, and it suffers from catastrophic forgetting. Catastrophic forgetting happens when the network forgets what it learned while training. In this case, the loss increases as shown around epoch number 20. Thus, the accuracy goes down from 67% to 47%. In summary, CTRNN did a better job than RNN in human activity classification problems in terms of accuracy and avoiding catastrophic forgetting.

4.2 Time constant Training Effects

In machine learning, one of the most crucial goals frequently revolves around attaining substantial accuracy in predictive models. The accuracy of a model is greatly influenced by training and fine-tuning of different parameters. The time constant parameter (tau) in CTRNN is identified as a crucial parameter. In Figure 4.5, we present the results when, for the first time, the time constant of each cell in CTRNN is trained. The accuracy reaches 82%. This section examines the importance of training the time constant parameter and its significant impact on the accuracy of the CTRNN model. Toward this objective, we conducted different experiments. The initial investigation involved training the model by setting a time constant value equal to a constant value of one in all 16 cells of the first layer, which is the standard practice in the literature. The time constant values were next initialized in the subsequent experiment using a uniform random distribution from [0,2]. In all scenarios, the models were trained for 50 epochs.

The findings of the initial experiment, where the time constant is set to one, are presented in Figure 4.8. The accuracy dropped to 68%, and the model successfully converged throughout 50 epochs. In contrast, the model had a better learning opportunity when the time constant was randomly initialized (resulting in varying time constant values for each cell). This is because different time constant values imply different response speeds for each cell, which is needed when dealing with sequence data. This clarifies why the random initialization of time constant parameters outperformed static initialization with ones, allowing the model to attend to distinct temporal segments. The accuracy in this scenario improved to 76%, as depicted in Figure 4.9, which is still less than 82% when the CTRNN time constants are included in the training. Table 1 summarizes our findings with respect to the different time-constant scenarios.

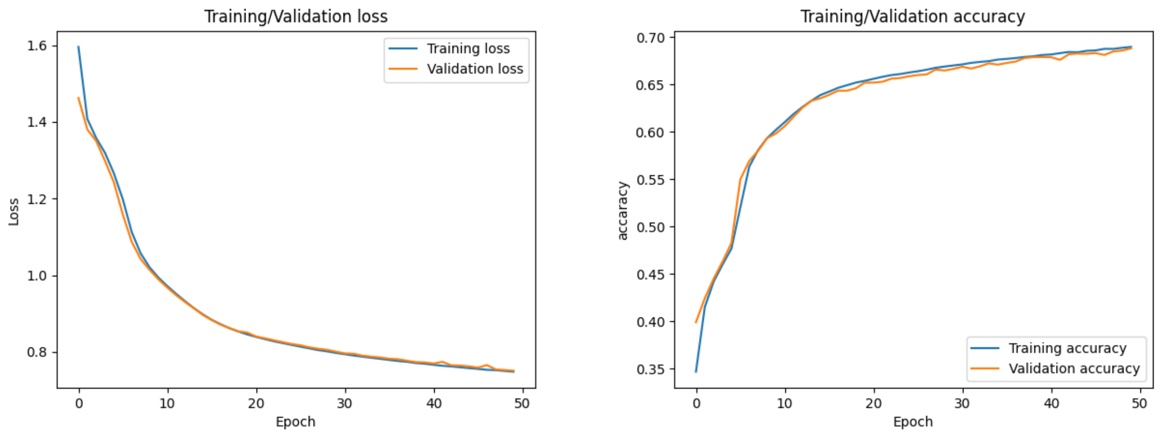


Figure 4.8 Convergence and accuracy curves of CTRNN with a time constant equals to ones, running 50 epochs, the model reaches an accuracy of around 68%.

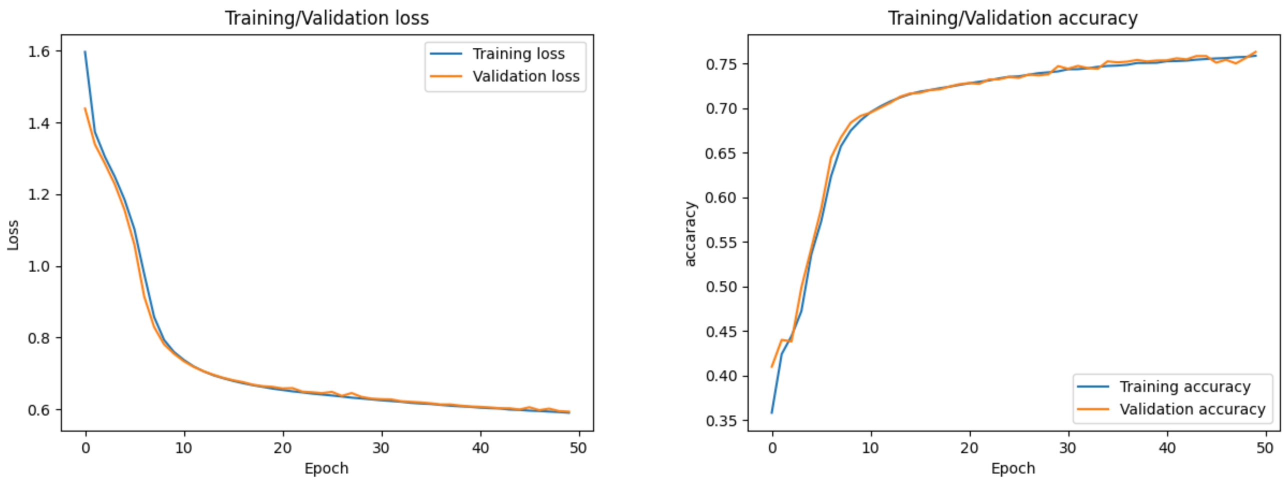


Figure 4.9 Convergence and accuracy curves of CTRNN without constant equals training and random initialization, running 50 epochs, the model reaches an accuracy of around 76%.

Table 4.1 Time constant effect on the CTRNN training

CTRNN Model	**Accuracy (%)**
Static time constant equals one	68
Random Initialization time constant	76
Training the time-constant	82

4.3 Parameter size Comparison between RNN and CTRNN

The model is a sequence of layers. Each layer is a set of cells, and each cell has different parameters that control the behavior of those cells. This section will compare the module regarding parameter count and accuracy. We will answer different questions, starting with the question of how many parameters the RNN needs to reach the same accuracy as the CTRNN.

CTRNN reached 82% with a total number of 572 parameters. The RNN reaches 73%, using almost the same number of parameters. To reach an accuracy close to the RNN accuracy, Figure 4.10 shows a CTRNN with only 4 neurons in the first layer. The number of the CTRNN parameters in this case is only 152 parameters, which is about 0.27% of the RNN parameters (less than One-third of the parameters!).

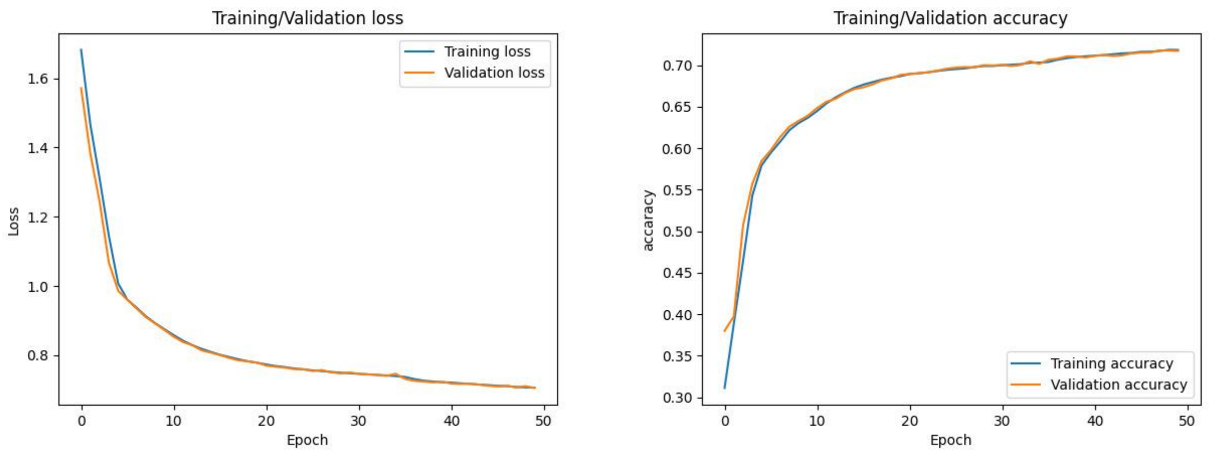


Figure 4.10 Convergence and accuracy curves of CTRNN with 156 parameters, only 4 cells in the first layer, accuracy:71.8%.

The subsequent inquiry in the investigation pertains to determining the requisite number of parameters for the RNN to achieve an accuracy of 82%. In this investigation and as shown in Figure 4.11 an RNN with a parameter count of 1,548, achieved an accuracy of

81.4%, which falls short of the performance exhibited by the CTRNN with only 572 parameters. Therefore, the CTRNN, again while having just 38% of the parameters, showed superior performance compared to the RNN. Table 2 summarizes our findings in this section.

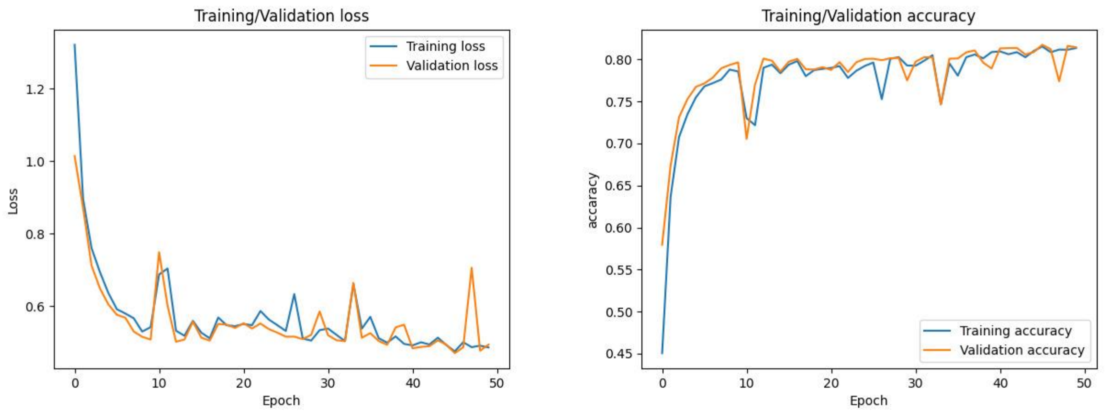


Figure 4.11 Convergence and accuracy curves RNN accuracy 81.4%, 1548 parameters.

Table 4.2 CTRNN vs. RNN parameters size

Model (number of recurrent cells)	**Number of parameters**	**Accuracy%**
CTRNN (16)	572	82
RNN (32)	1,548	81.4
CTRNN (4)	156	71.8
RNN (16)	556	73

4.4 Evaluate the Models

As mentioned in the previous section, The CTRNN models demonstrated a high level of ability by achieving an accuracy above 80% with a small network. This section is directed towards understanding the factors contributing to the remaining 20% inaccuracy. Where does it happen? Moreover, why this could happen? Furthermore, how do we solve it?. For evaluation and illustration purposes, we are using the confusion matrix. Confusion matrix is a tabular form to evaluate the model [29], showing the number of correctly classified items vs incorrectly classified. For those incorrectly classified, the metric also indicates the classes they were erroneously assigned. The confusion matrix has four entries: True Positive (correctly classified as positive), True Negative (correctly classified as positive), False Positive (incorrectly classified as positive), and False Negatives (incorrectly classified as negative). The confusion matrix will help to detect if the model has a problem distinguishing between different activities. Figure 4.12 is the confusion matrix for the CTRNN model after 50 epochs.

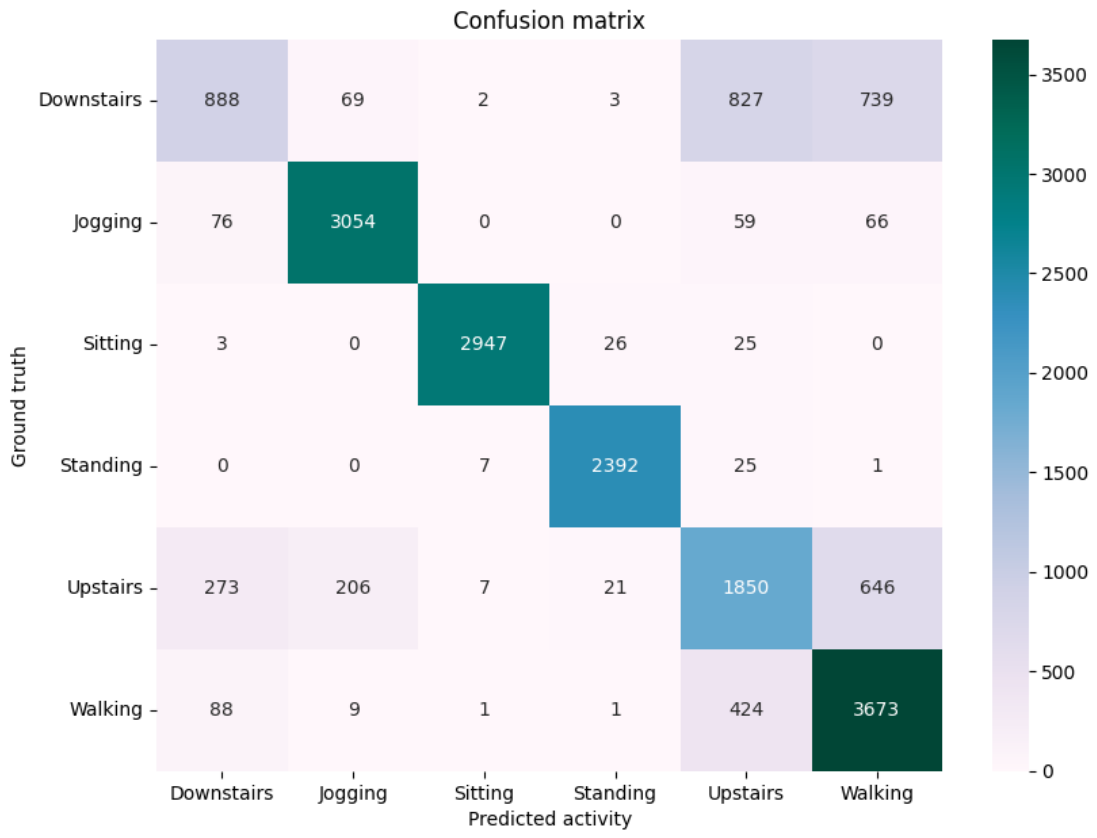


Figure 4. 12 Confusion matrix of CTRNN model run for 50 epochs, network architecture 16-10-6.

Figure 4.12 shows that the model encounters difficulties distinguishing between multiple activities, such as distinguishing "downstairs" from "upstairs" and “walking”, from “upstairs” activities. Next, we investigate those cases very closely.

Distinguishing "downstairs" from "upstairs”

The models confuse "downstairs" and "upstairs," as shown in the confusion matrix Figure 4.12. To better understand this issue, next, we will examine the characteristics of these activities and check the X, Y, and Z accelerations as shown in Figure 4.13. The conflict between the downstairs and upstairs yields valuable insights into the CTRNN model. The Figure shows that downstairs and upstairs movements exhibit similar bodily oscillations. This similarity could pose challenges for the model, as these sequences mirror each other in different directions. Employing a bidirectional CTRNN can be a viable solution to address this issue. Alternatively, we might use an attention model to concentrate on specific channels like the 'Z' direction. Additionally, since we employ a window size of 50, this window might encompass only a tiny portion of an ascent or descent, where their characteristics resemble each other. Hence, incorporating attention layers could further enhance our model's performance.

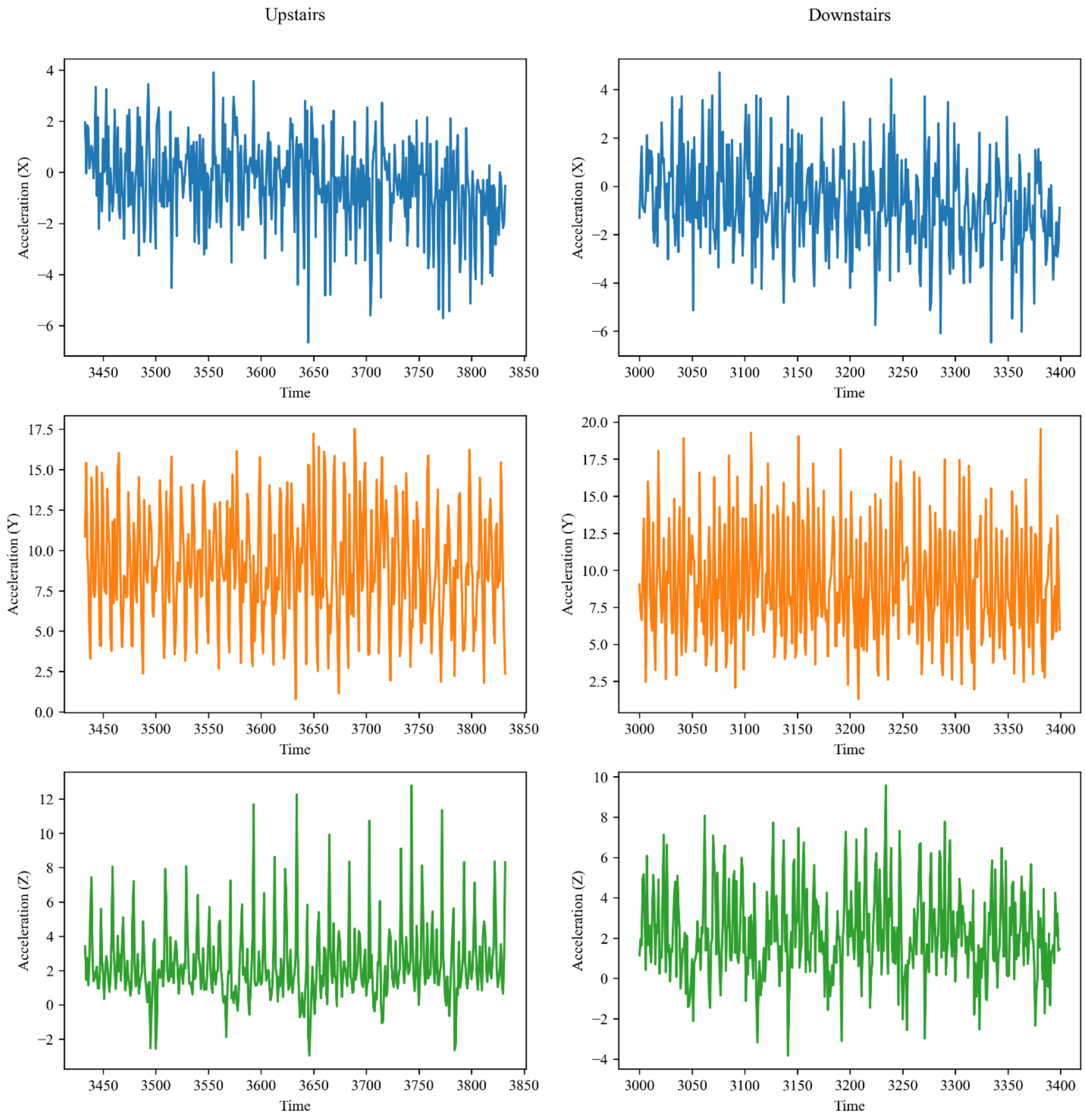


Figure 4.13 Upstairs vs Downstair activities acceleration data in three directions, X, Y and Z.

Distinguishing " walking " from "upstairs”

The second model's challenge is in distinguishing between walking and upstairs activities. This differentiation is based on analyzing the acceleration of these activities, as depicted in Figure 4.14. Both walking and going upstairs exhibit comparable movement patterns characterized by rhythmic and repetitive motions. When an individual walks downstairs, they frequently exhibit step-like motions resembling walking on flat ground, posing difficulty for the model to discern between the two activities merely based on motion patterns. One potential option to address this issue is to increase the model's size. This is demonstrated in Figure 4.15, where the model successfully differentiates between "downstairs" and "walking" activities. Such problems could come from the sensor noise and imbalanced training data in machine learning, which is the case in our data.

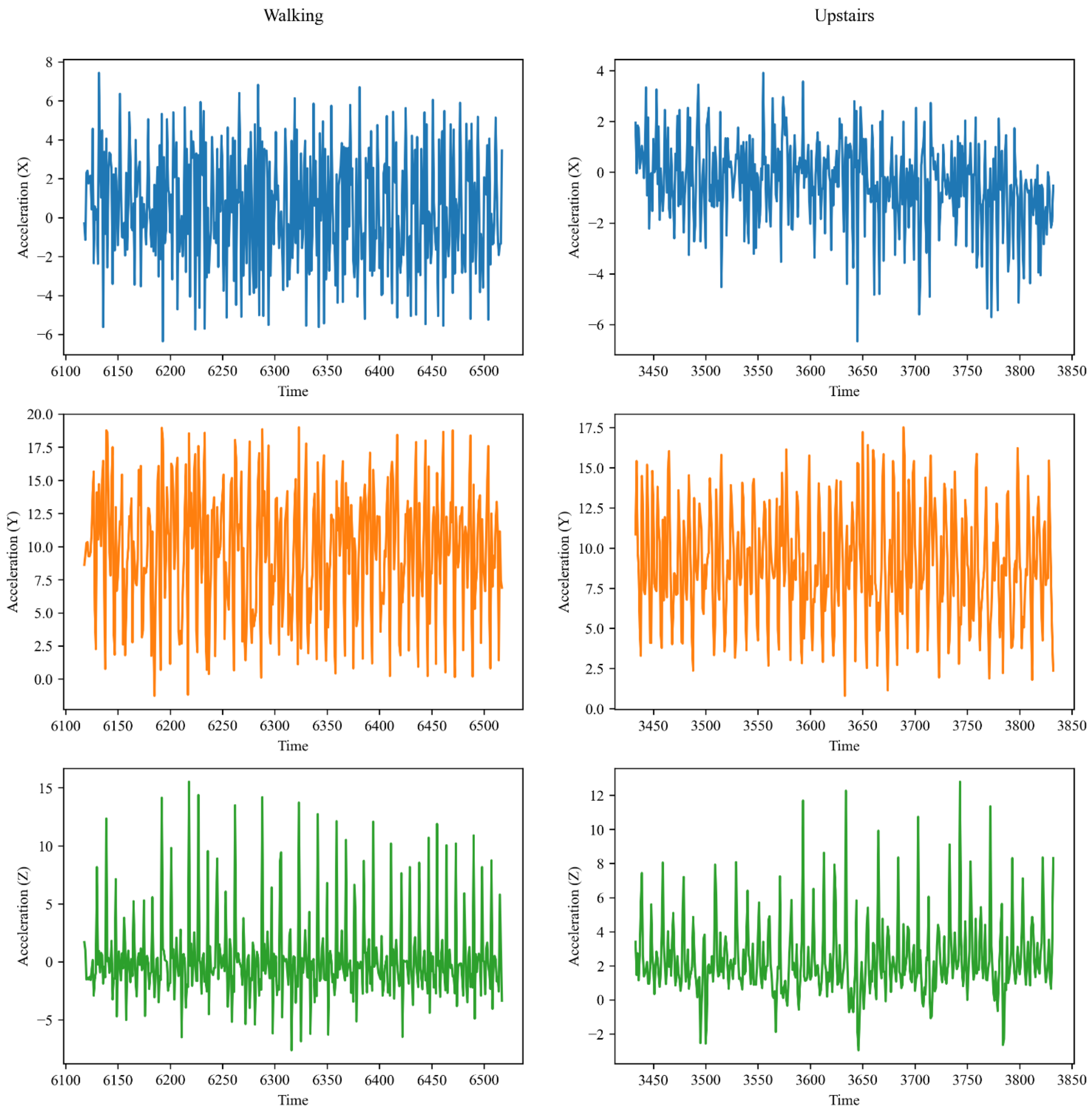


Figure 4. 14 Walking vs Upstairs activities acceleration data in three directions, X, Y and Z.

.

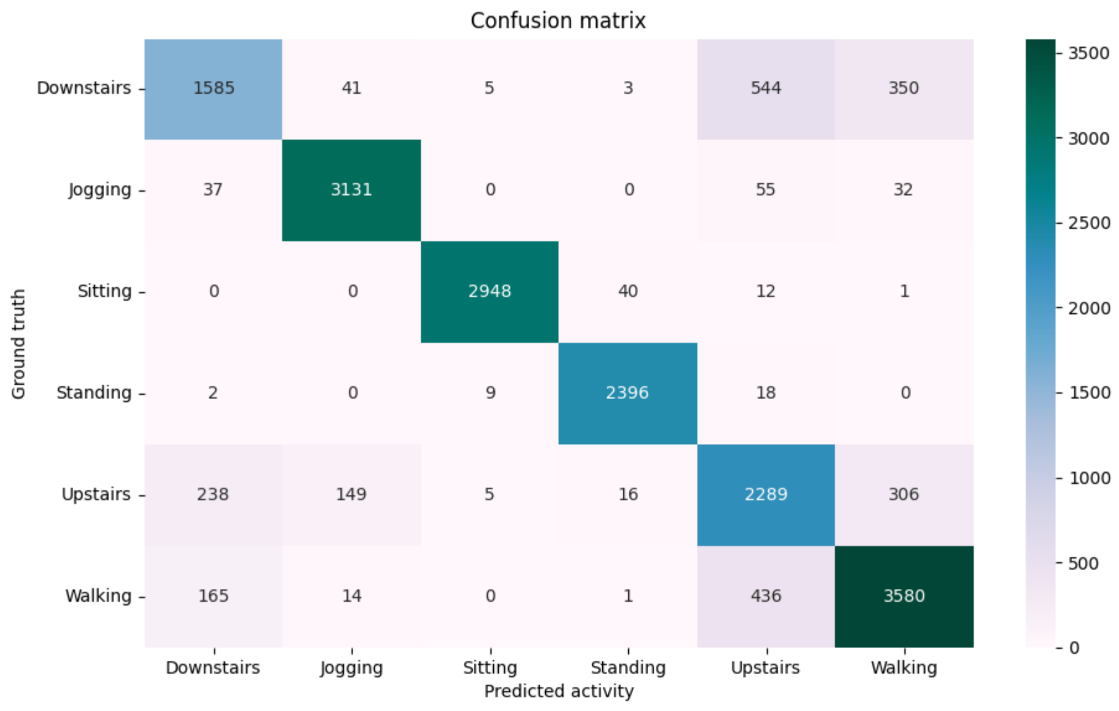


Figure 4. 15 Confusion Matrix of CTRNN model run for 50 epochs, network architecture 32-10-6.

4.5 CTRNN Drawbacks

In this chapter, we showed that CTRNN achieved better accuracy and capability to capture the temporal behavior in data than RNN. Moreover, CTRNN showed smooth converges with fewer parameters to train. However, CTRNN has some drawbacks, making it less popular. The first drawback is the model complexity, which comes from the fact that CTRNN uses a differential equation to mimic the real neuron's continuous time (real-time) behaviors [27]. Those equations are expensive to solve using digital computers. The second drawback is that CTRNN is not widespread, and there is a lack of resources and support. In this next chapter, we discuss some future work to reduce some of these challenges.

Chapter 5
THESIS SUMMARY AND FUTURE WORK

5.1 Thesis Summary and Conclusions

This thesis investigates the potential of implementing the CTRNN network for human activity detection. CTRNN uses differential equations to represent their neuron cell instead of the difference equation for the standard RNN. This novel implementation opens opportunities for analog computing that solves such differential equations. This thesis focused on implementing CTRNN using TensorFlow, a well-known machine learning framework, to optimize the cell's parameters and make it trainable. Then, we tested the network on a small dataset and used it to solve an actual application of human activity detection. The CTRNN shows a high capability to capture temporal behavior in the data with smooth convergences, considering that some parameters have constraints to meet their physical meaning, like the time constant. Our work showed that CTRNN achieved better performance and recognized human activities with less network size (parameters size), with the ability to avoid problems that RNN might face, like disaster forgetting.

Training the CTRNN time constant significantly affected the trained model's performance. We conducted different experiments: one with initializing time constants to ones, a second with random initialization, and a third with training time constant. Of these, training the time constant had the best accuracy, random initialization had the second-best result, and constant initialization was the worst. Evaluating the CTRNN model in depth proves its ability to learn but with difficulty differentiating between some activities with similar behavior. Increasing the model size solves some of those problems. Finally, we discussed some drawbacks of CTRNN, like the complexity that comes from the differential equations, scalability, and low popularity.

5.2 Future Work

To solve the complexity of solving the equations of CTRNN, we plan to implement the MEMS as a learnable CTRNN cell in future work. MEMS are microstructural devices that consist of stationary and moving elements. The dynamic position of the MEMS moving element is governed by a second-order differential equation driven by an input signal [31]. This equation is similar to the CTRNN equation. Thus, we plan to use MEMS to solve the CTRNN equations. Specifically, as the movement of the MEMS is the solution to the differential equation, we do not need to solve the differential equation in the physical hardware, and this will solve the main drawback of the CTRNN. Besides the wide-spreading, MEMS offers a considerable advantage of allowing simultaneous sensing and computational functions at the same level [32], Thereby addressing various issues such as edge computing and providing efficient, low-power solutions for machine learning. Another potential avenue for future research involves utilizing the CTRNN framework to address various challenges, such as the control of robotic systems.

References

1. Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. Proceedings of the national academy of sciences, 79(8), 2554-2558.
2. Ising, E. (1925). Beitrag zur theorie des ferromagnetismus. Zeitschrift für Physik, 31(1), 253-258.
3. De Falco, I., Della Cioppa, A., Donnarumma, F., Maisto, D., Prevete, R., & Tarantino, E. (2008). CTRNN parameter learning using Differential Evolution. In ECAI 2008 (pp. 783-784). IOS Press.
4. Hénaff, P., Scesa, V., Ouezdou, F. B., & Bruneau, O. (2011). Real time implementation of CTRNN and BPTT algorithm to learn on-line biped robot balance: Experiments on the standing posture. Control engineering practice, 19(1), 89-99.
5. Fitzgerald, O., Perez-Concha, O., Gallego-Luxan, B., Metke-Jimenez, A., Rudd, L., & Jorm, L. (2023). Continuous time recurrent neural networks: overview and benchmarking at forecasting blood glucose in the intensive care unit. Journal of Biomedical Informatics, 104498.
6. Alsaleem, F.M., Hasan, M.H. and Tesfay, M.K., 2018. A MEMS nonlinear dynamic approach for neural computing. Journal of Microelectromechanical Systems, 27(5), pp.780-789.
7. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

8. Bown, O., & Lexer, S. (2006, April). Continuous-time recurrent neural networks for generative and interactive musical performance. In Workshops on Applications of Evolutionary Computation (pp. 652-663). Berlin, Heidelberg: Springer Berlin Heidelberg.

9. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

10. Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., ... & Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. arXiv preprint arXiv:1910.13461.

11. Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053..

12. Wysocki, A., & Ławryńczuk, M. (2015, August). Jordan neural network for modelling and predictive control of dynamic systems. In 2015 20th International Conference on Methods and Models in Automation and Robotics (MMAR) (pp. 145-150). IEEE.

13. Forrest, S. (1996). Genetic algorithms. ACM computing surveys (CSUR), 28(1), 77-80.

14. Kumar, A. and Mohanty, P., 2017. Autoassociative memory and pattern recognition in micromechanical oscillator network. Scientific reports, 7(1), p.411.

15. Beer, R. D. (2006). Parameter space structure of continuous-time recurrent neural networks. Neural computation, 18(12), 3009-3051.
16. Garcia, I. (2015, December 17). Analysis of a simple continuous time recurrent neural network (CTRNN) model. analysis-of-a-simple-ctrnn. https://indy9000.blog/posts/analysis-of-a-simple-ctrnn.html.
17. Adaloglou, N. (2020, September 10). Recurrent neural networks: building a custom LSTM cell | AI Summer. AI Summer. https://theaisummer.com/understanding-lstm/ -138.
18. Z Csáji, B. C. (2001). Approximation with artificial neural networks. Faculty of Sciences, Etvs Lornd University, Hungary, 24(48), 7.
19. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980..
20. Kwapisz, J. R., Weiss, G. M., & Moore, S. A. (2011). Activity recognition using cell phone accelerometers. ACM SigKDD Explorations Newsletter, 12(2), 74-82.
21. Emad-Ud-Din, M., Hasan, M.H., Jafari, R., Pourkamali, S. and Alsaleem, F., 2021. Simulation for a Mems-Based CTRNN Ultra-Low Power Implementation of Human Activity Recognition. Frontiers in Digital Health, 3, p.731076.
22. Quarteroni, A., Sacco, R., & Saleri, F. (2010). Numerical mathematics (Vol. 37). Springer Science & Business Media.
23. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). {TensorFlow}: a system for {Large-Scale} machine learning. In 12th USENIX symposium on operating systems design and implementation (OSDI 16) (pp. 265-283).

24. Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. Journal of Marchine Learning Research, 18, 1-43.
25. Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge University Press.
26. Géron, A. (2022). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. " O'Reilly Media, Inc.".
27. Kramer, G. R. (2007). An analysis of neutral drift's effect on the evolution of a CTRNN locomotion controller with noisy fitness evaluation.
28. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.
29. Stehman, S. V. (1997). Selecting and interpreting measures of thematic classification accuracy. Remote sensing of Environment, 62(1), 77-89.
30. Zhu, D., Cai, C., Yang, T., & Zhou, X. (2018). A machine learning approach for air quality prediction: Model regularization and optimization. Big data and cognitive computing, 2(1), 5.
31. Younis, M. I. (2011). MEMS linear and nonlinear statics and dynamics (Vol. 20). Springer Science & Business Media.
32. H Hasan, M., Al-Ramini, A., Abdel-Rahman, E., Jafari, R., & Alsaleem, F. (2020). Colocalized sensing and intelligent computing in micro-sensors. Sensors, 20(21), 6346.
33. Perea, J. A., & Harer, J. (2015). Sliding windows and persistence: An application of topological methods to signal analysis. Foundations of Computational Mathematics, 15, 799-838.

34. Peppas, K., Tsolakis, A. C., Krinidis, S., & Tzovaras, D. (2020). Real-time physical activity recognition on smart mobile devices using convolutional neural networks. Applied Sciences, 10(23), 8482.

35. tf.keras.constraints.MinMaxNorm | TensorFlow v2.14.0. (n.d.). TensorFlow. Retrieved November 6, 2023, fromhttps://www.tensorflow.org/api_docs/python/tf/keras/constraints/MinMaxNorm

36. Hasan, M. H., Alsaleem, F., & Rafaie, M. (2016). Sensitivity study for the PMV thermal comfort model and the use of wearable devices biometric data for metabolic rate estimation. Building and Environment, 110, 173-183.

37. Zhang, S., Li, Y., Zhang, S., Shahabi, F., Xia, S., Deng, Y., & Alshurafa, N. (2022). Deep learning in human activity recognition with wearable sensors: A review on advances. Sensors, 22(4), 1476.

38. Burton, A. (2016). Historical Dictionary of British Spy Fiction. Rowman & Littlefield.

39. Iqbal, A., Ullah, F., Anwar, H., Ur Rehman, A., Shah, K., Baig, A., ... & Kwak, K. S. (2020). Wearable Internet-of-Things platform for human activity recognition and health care. International Journal of Distributed Sensor Networks, 16(6), 1550147720911561.

40. Patel, S. N., Reynolds, M. S., & Abowd, G. D. (2008). Detecting human movement by differential air pressure sensing in HVAC system ductwork: An exploration in infrastructure mediated sensing. In Pervasive Computing: 6th International Conference, Pervasive 2008 Sydney, Australia, May 19-22, 2008 Proceedings 6 (pp. 1-18). Springer Berlin Heidelberg.

41. Yin, J., Yang, Q., & Pan, J. J. (2008). Sensor-based abnormal human-activity detection. IEEE transactions on knowledge and data engineering, 20(8), 1082-1090.

APPENDIX A

Appendix A offers a comprehensive analysis of the diverse case studies presented in the current research, accompanied by supplementary simulation results.

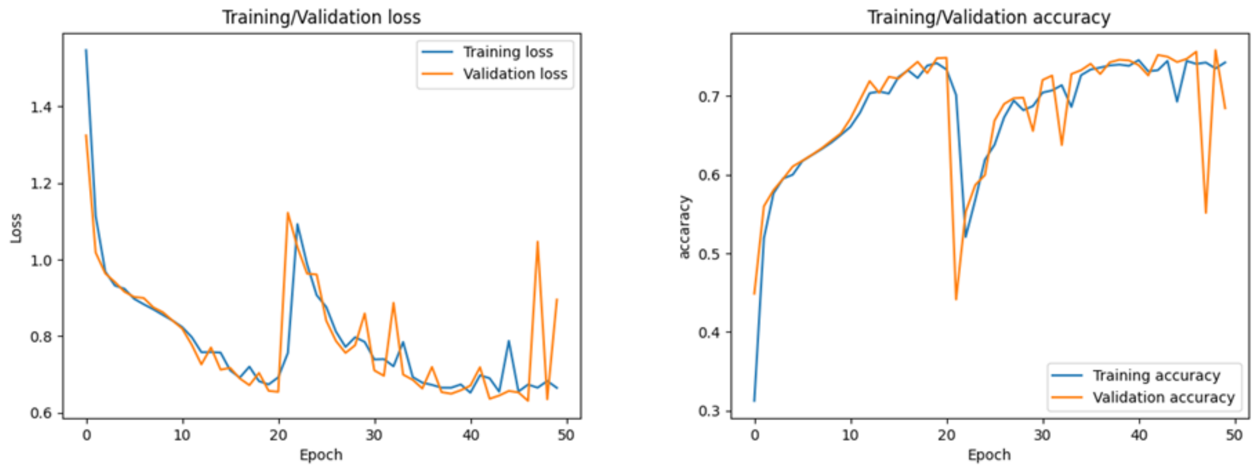


Figure A.1 Convergence and accuracy curves of RNN run for 50 epochs, different runs show catastrophic forgetting.

RNN training does not run smoothly, and it suffers from catastrophic forgetting A.2, the model accuracy goes down from 74% to 43%.

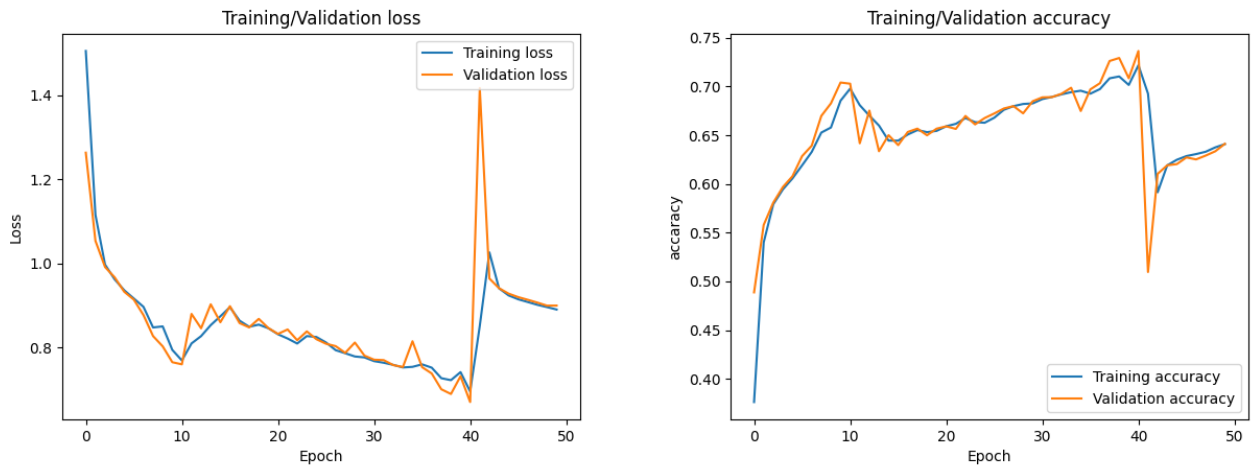


Figure A.2 Convergence and accuracy curves of RNN run for 50 epochs, different runs #2 show catastrophic forgetting.

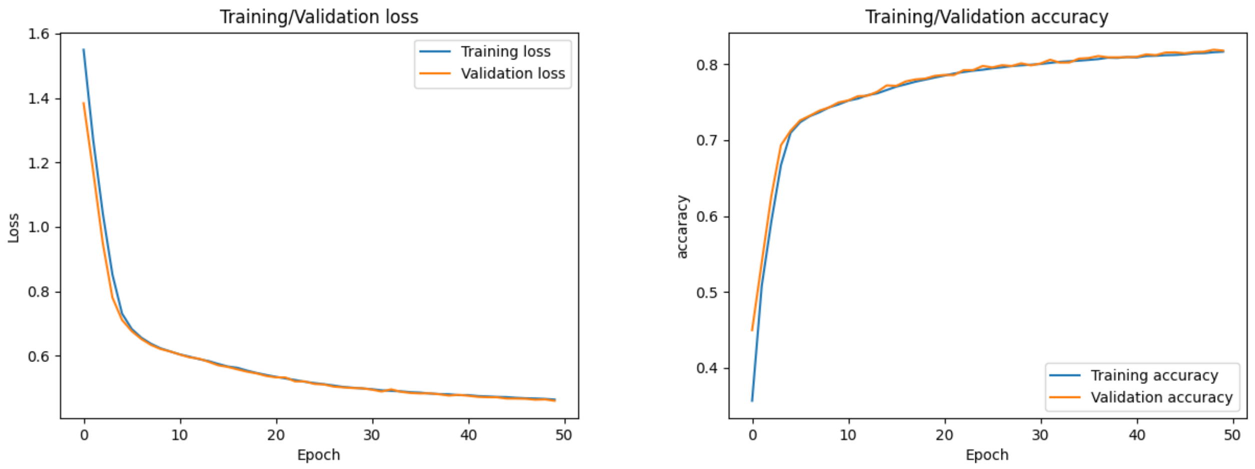
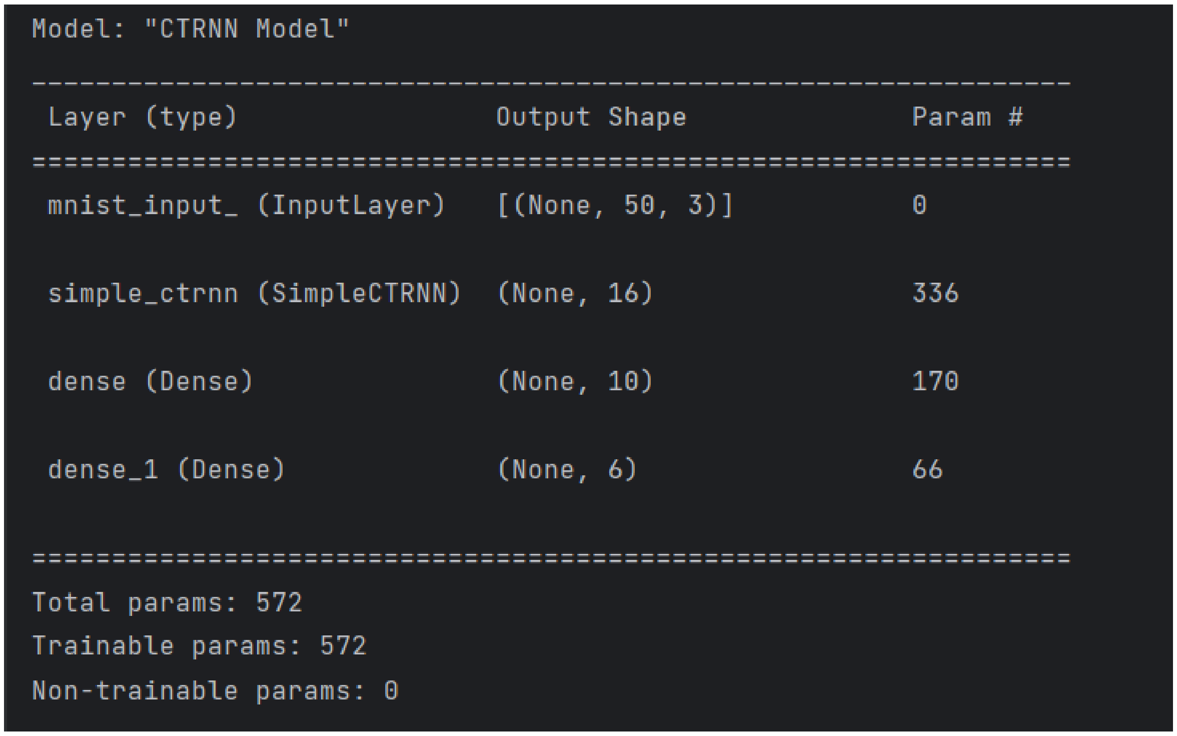


Figure A.3 Convergence and accuracy curves of CTRNN run for 50 epochs, different run, the model reaches an accuracy of around 82%.



```
Model: "CTRNN Model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mnist_input_ (InputLayer)   [(None, 50, 3)]           0

 simple_ctrnn (SimpleCTRNN)  (None, 16)                336

 dense (Dense)               (None, 10)                170

 dense_1 (Dense)             (None, 6)                 66

=================================================================
Total params: 572
Trainable params: 572
Non-trainable params: 0
```

Figure A.4 CTRNN Architecture TensorFlow Information.

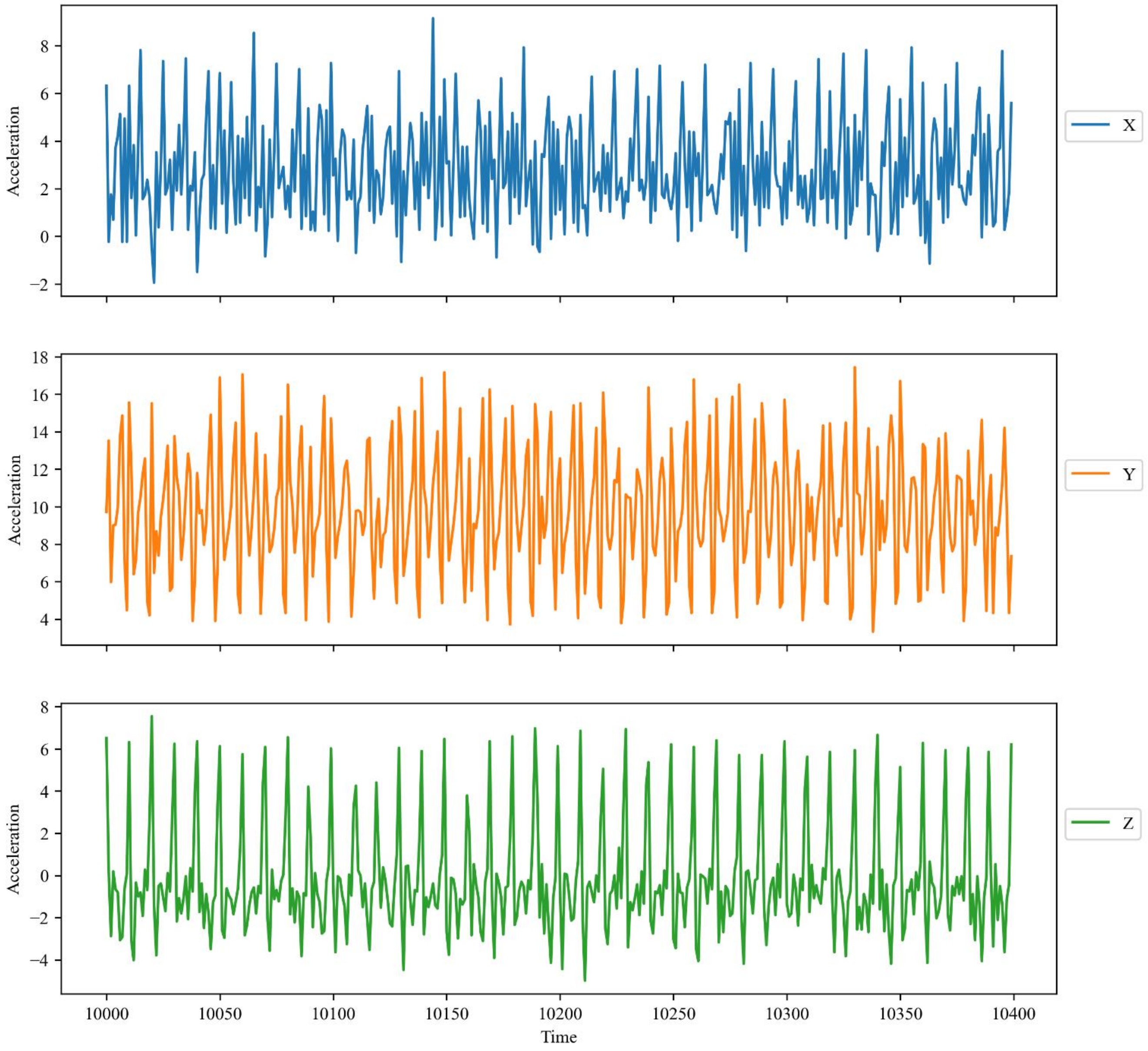


Figure A.5 Walking acceleration data in three directions, X, Y and Z.

.

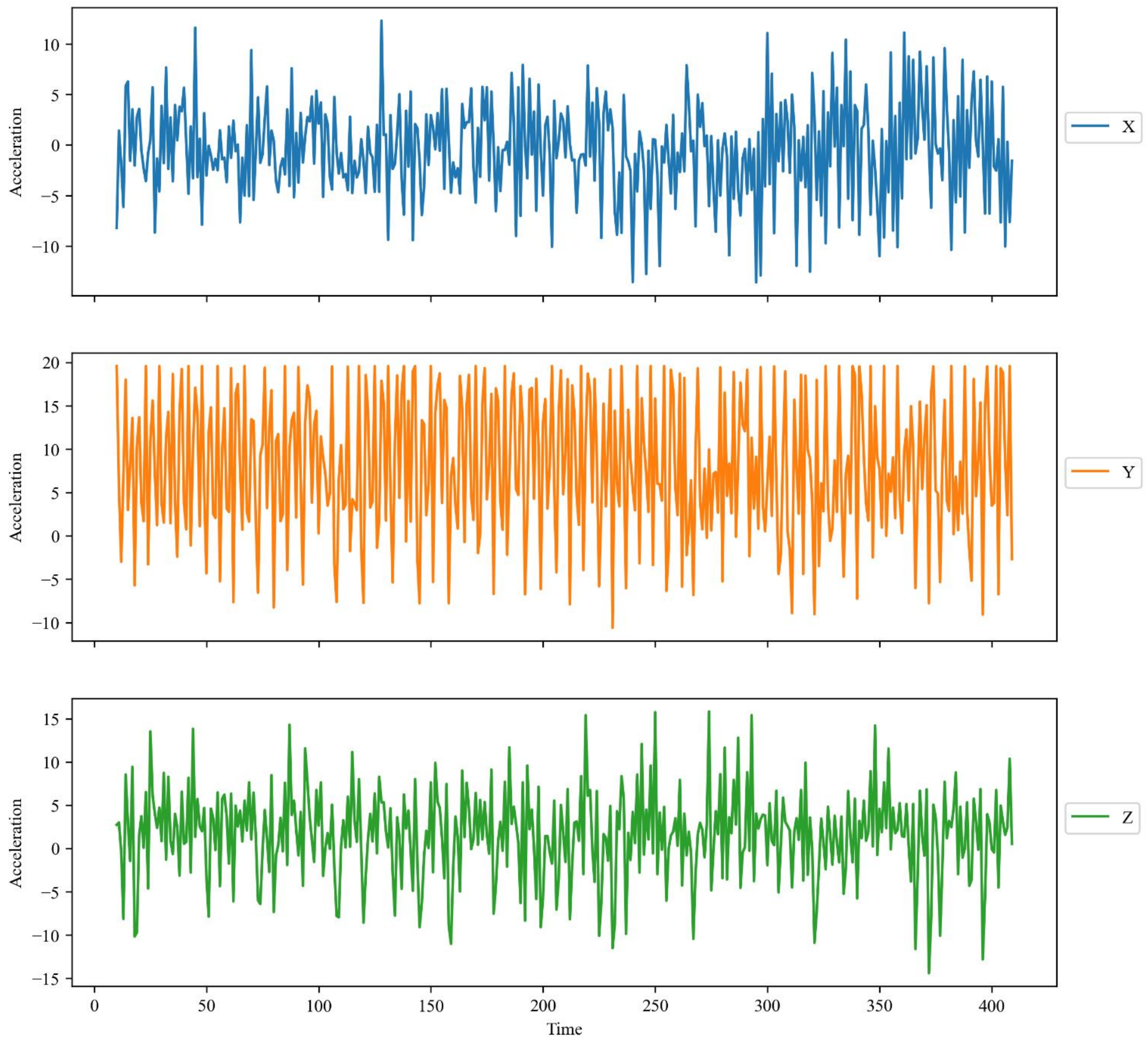


Figure A.6 Jogging acceleration data in three directions, X, Y and Z.

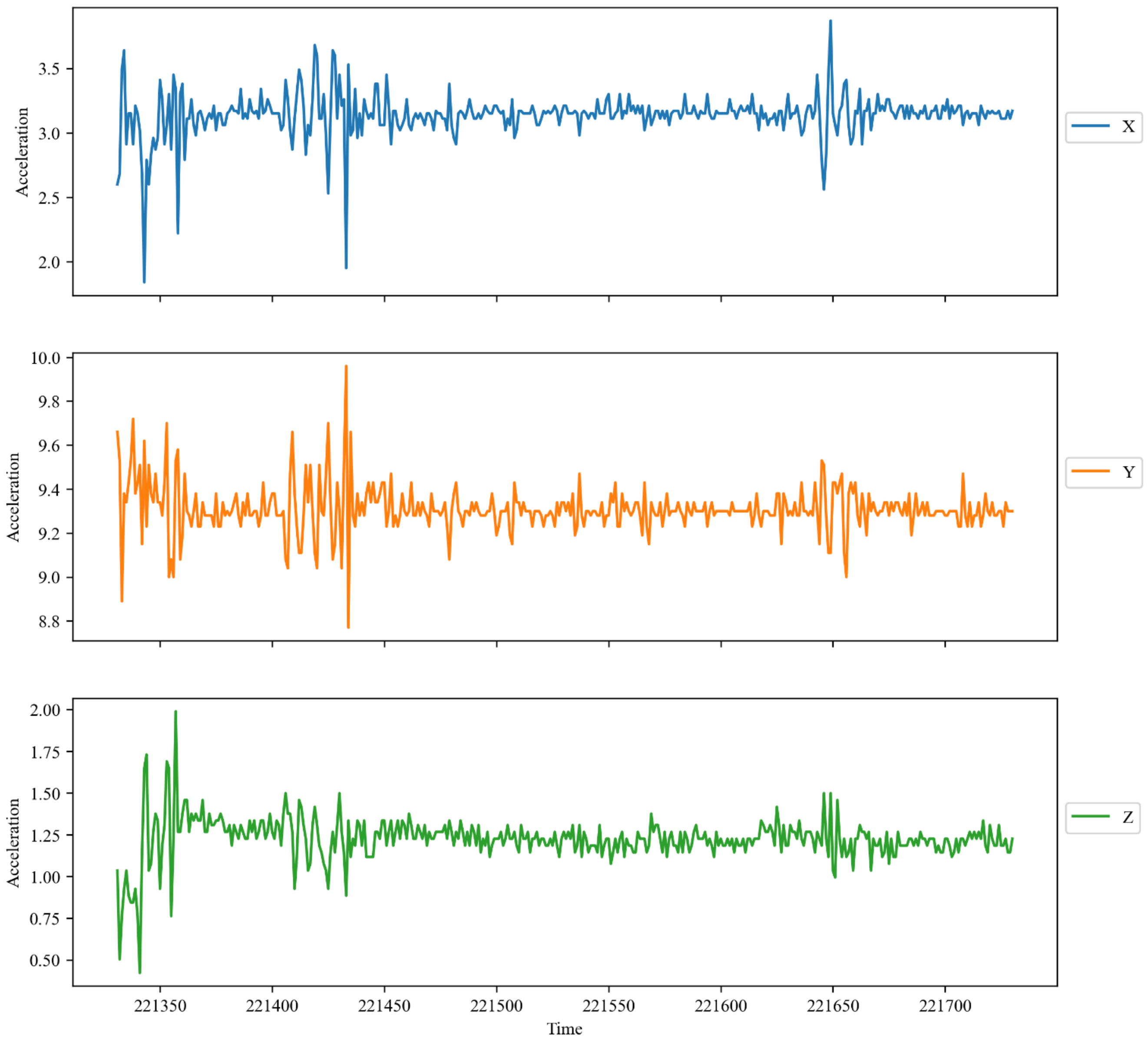


Figure A.7 Sitting acceleration data in three directions, X, Y and Z.

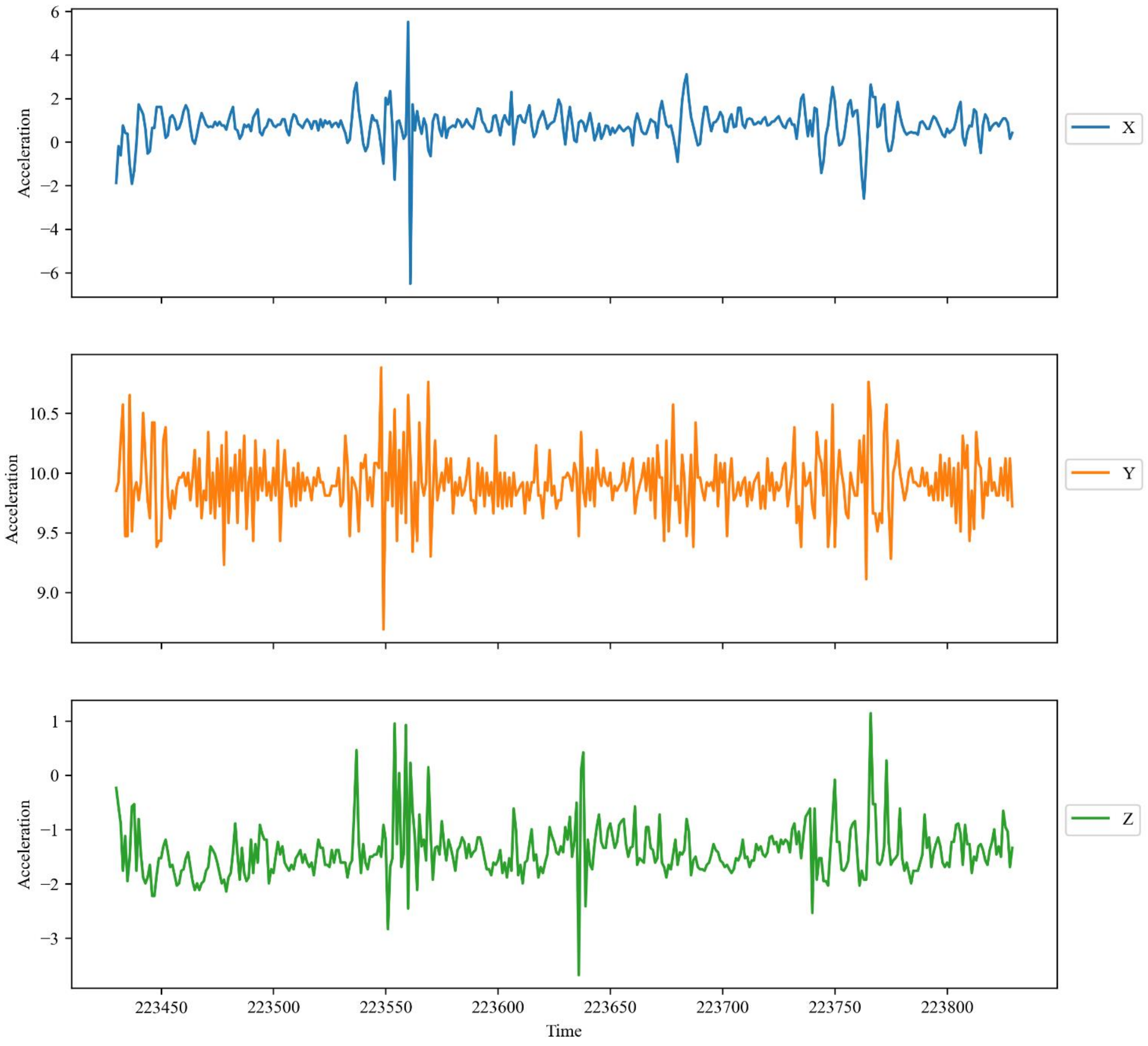


Figure A.8 Standing acceleration data in three directions, X, Y and Z

.

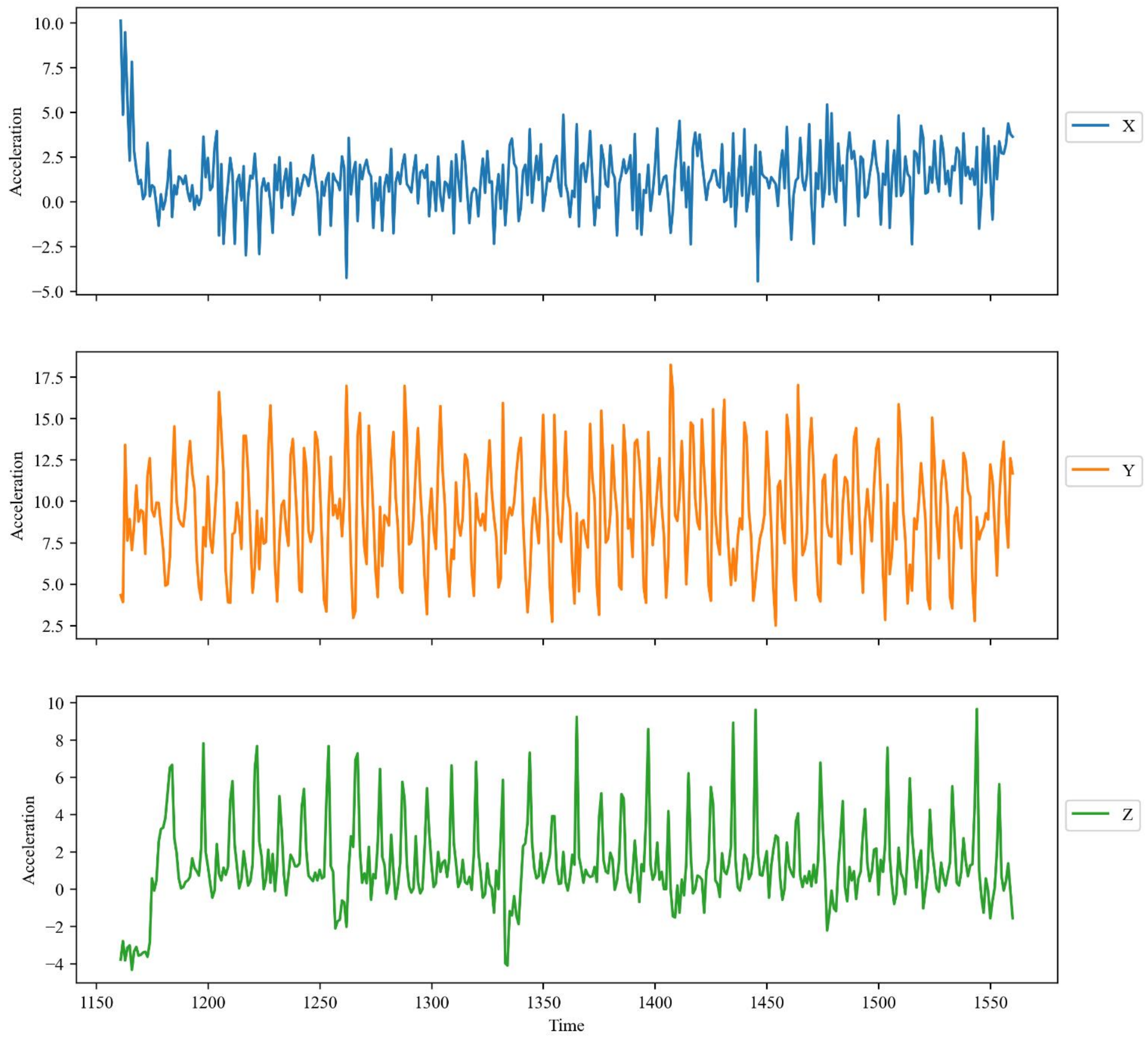


Figure A.9 Upstairs acceleration data in three directions, X, Y and Z

.

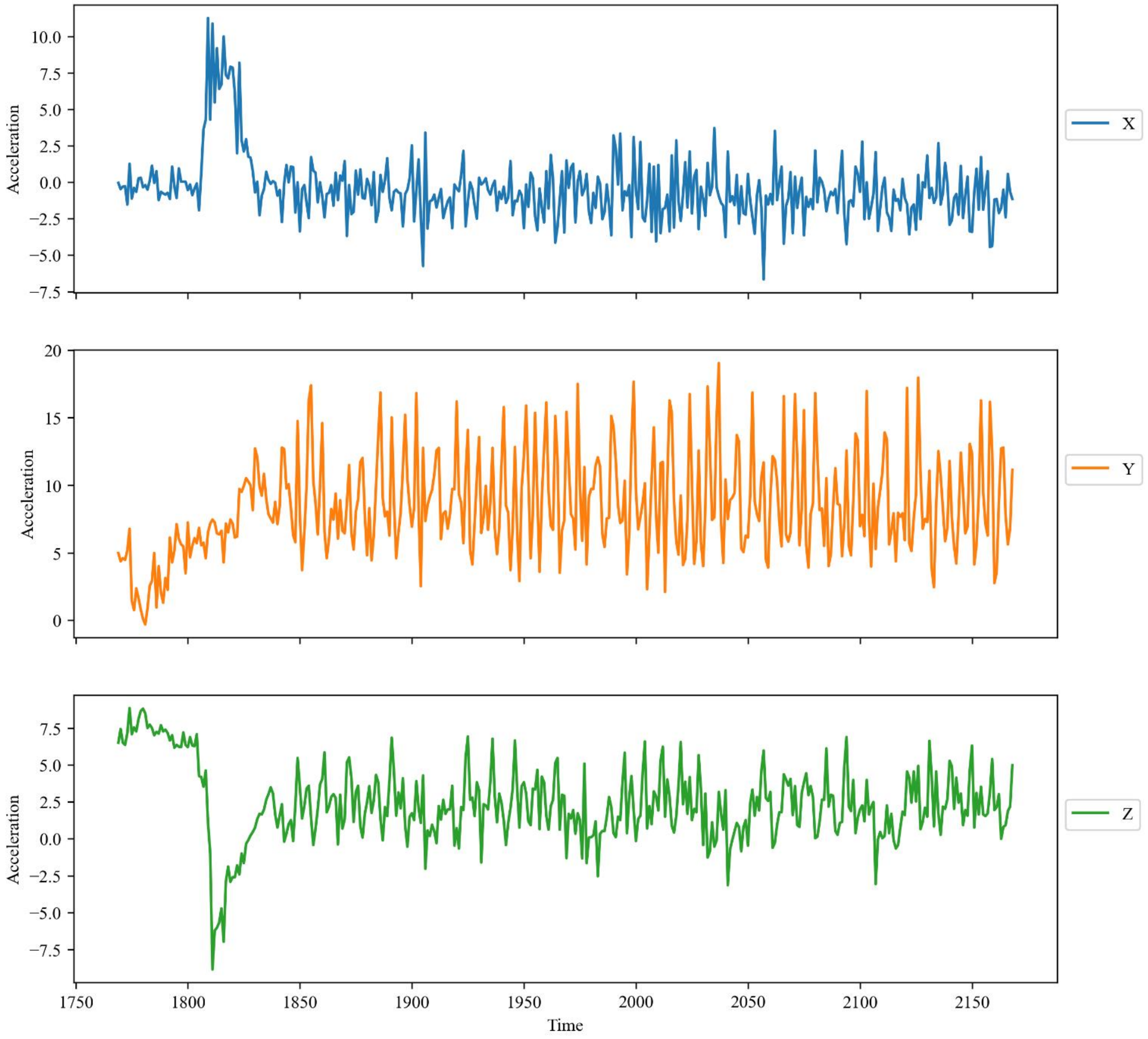


Figure A.10 Downstair acceleration data in three directions, X, Y and Z.

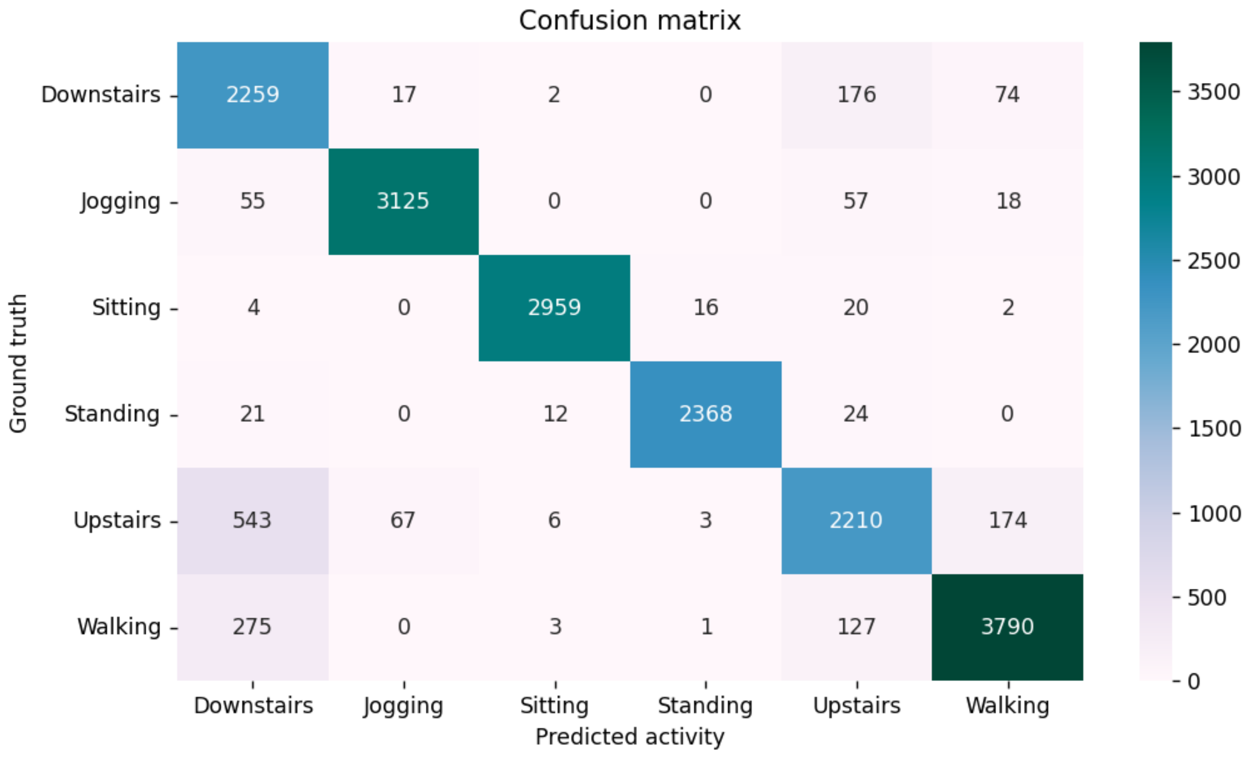


Figure A.11 Confusion matrix of CTRNN model run for 50 epochs, network architecture 16-10-6, accuracy 82%.

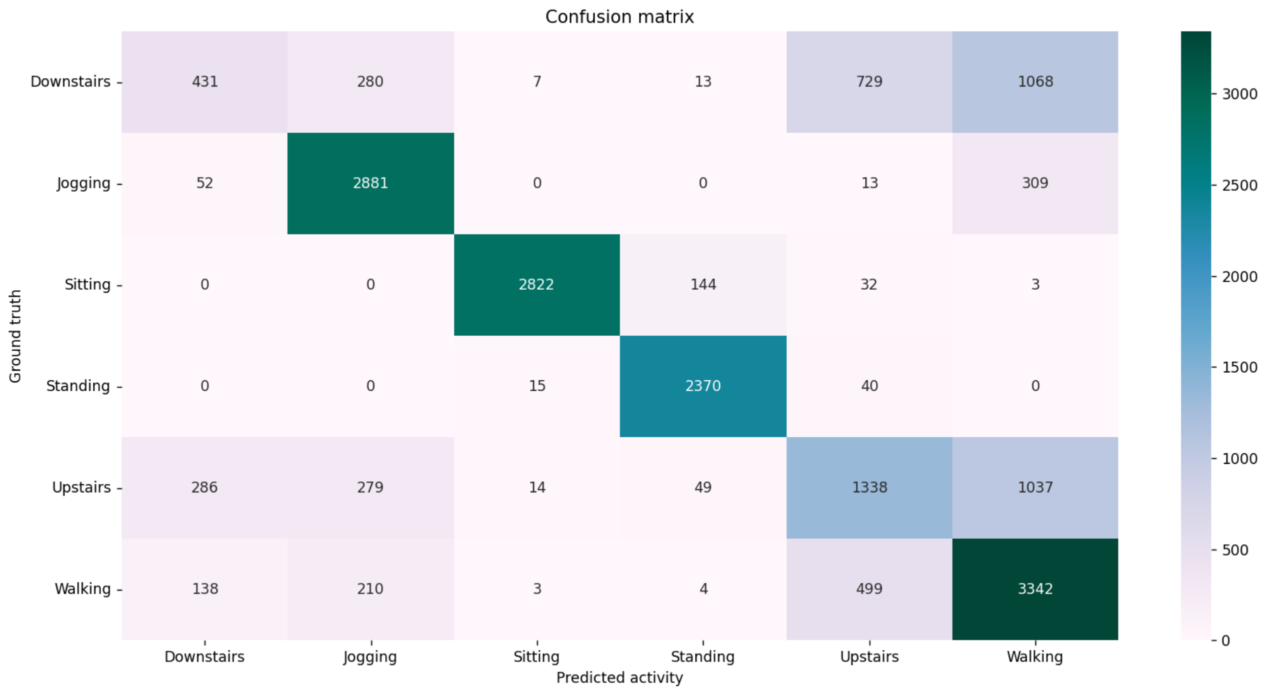
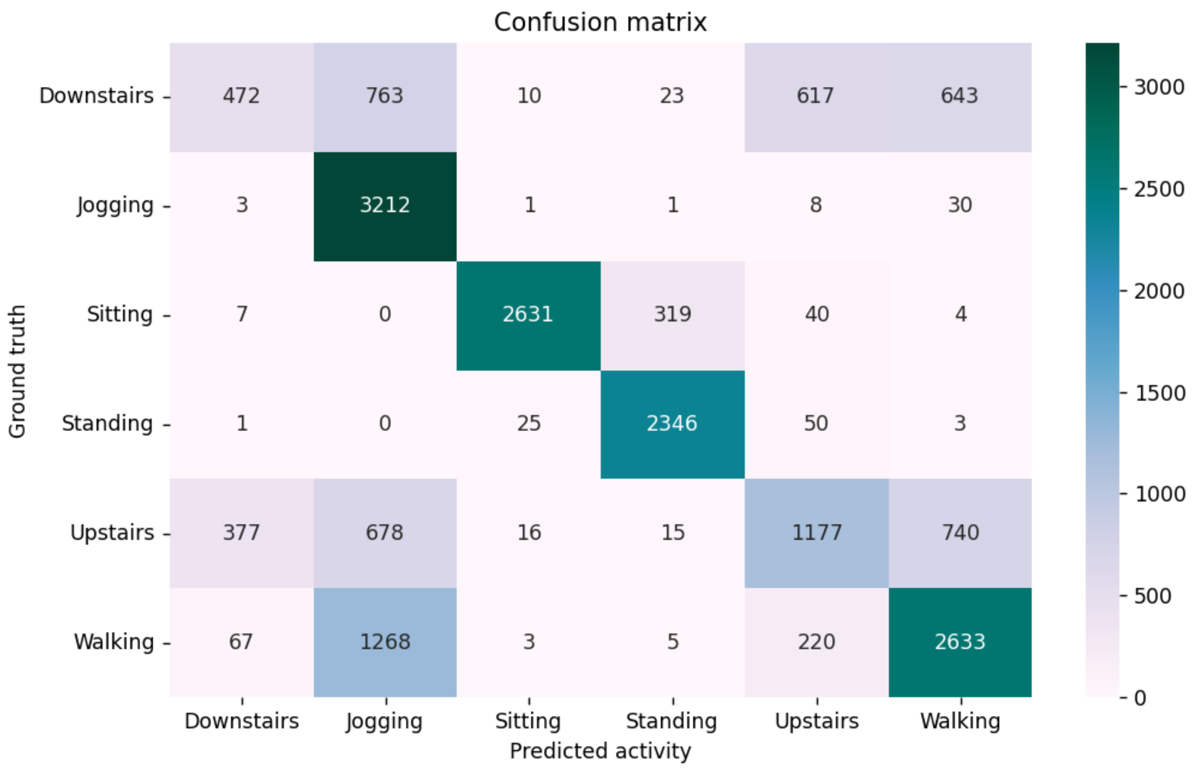
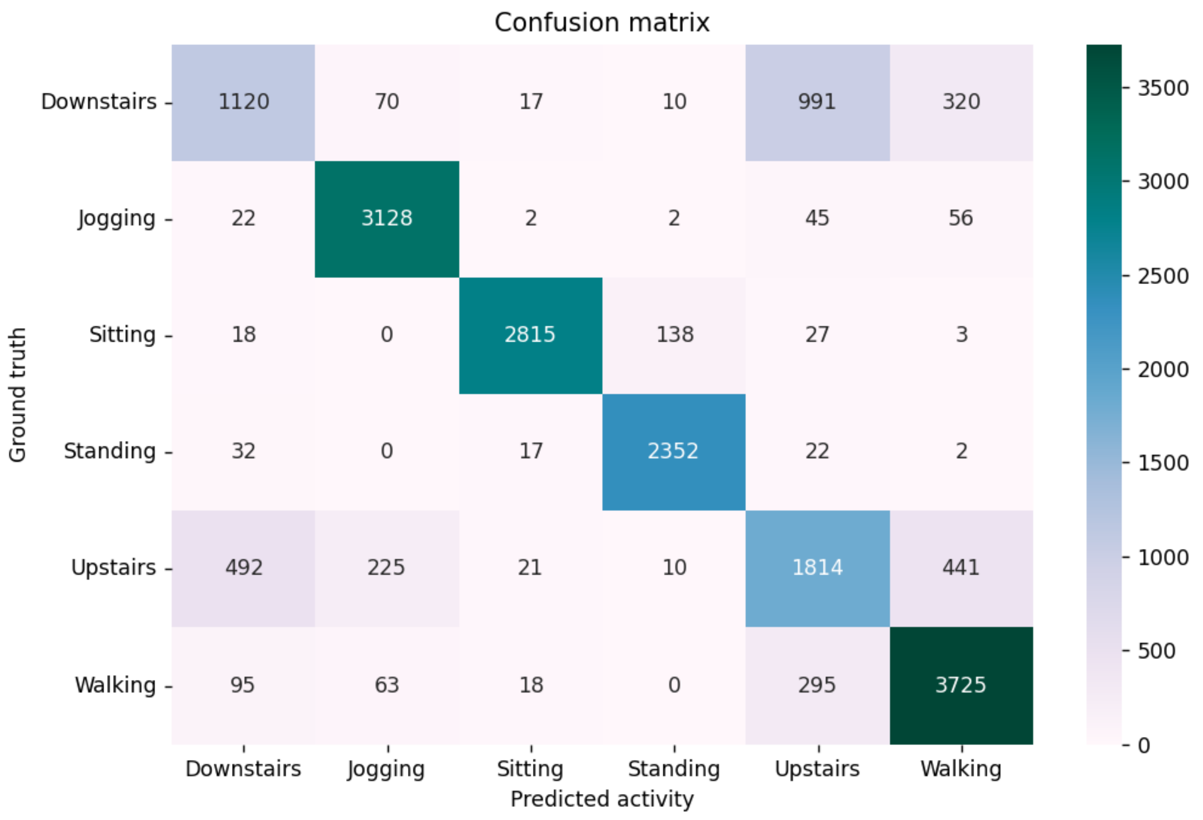


Figure A.12 Confusion matrix of CTRNN model run for 50 epochs, network architecture 4-10-6, 71.6%.



	Downstairs	Jogging	Sitting	Standing	Upstairs	Walking
Downstairs	472	763	10	23	617	643
Jogging	3	3212	1	1	8	30
Sitting	7	0	2631	319	40	4
Standing	1	0	25	2346	50	3
Upstairs	377	678	16	15	1177	740
Walking	67	1268	3	5	220	2633

Figure A.13 Confusion matrix of RNN model run for 50 epochs, network architecture 16-10-6, accuracy 68%.



	Downstairs	Jogging	Sitting	Standing	Upstairs	Walking
Downstairs	1120	70	17	10	991	320
Jogging	22	3128	2	2	45	56
Sitting	18	0	2815	138	27	3
Standing	32	0	17	2352	22	2
Upstairs	492	225	21	10	1814	441
Walking	95	63	18	0	295	3725

Figure A.14 Confusion matrix of RNN model run for 50 epochs, network architecture 32-10-6, accuracy 81.2%.

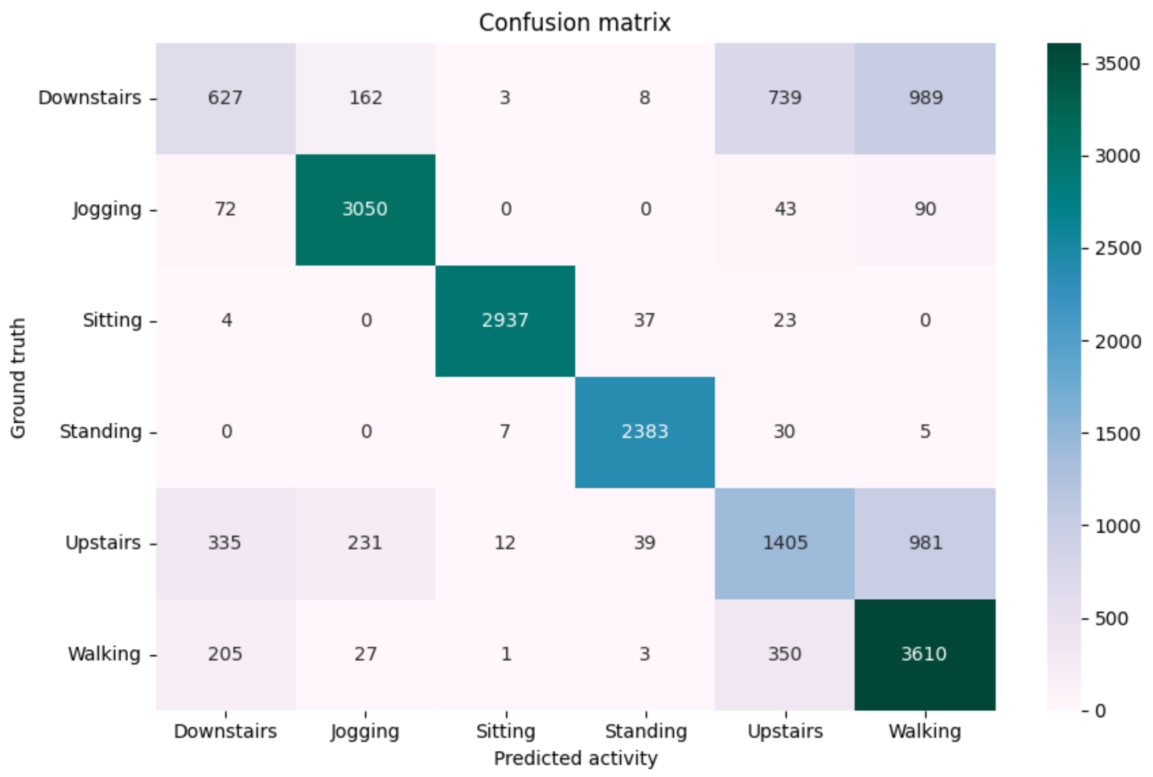


Figure A.15 Confusion matrix of RNN model run for 50 epochs, network architecture 16-10-6, accuracy 68%.

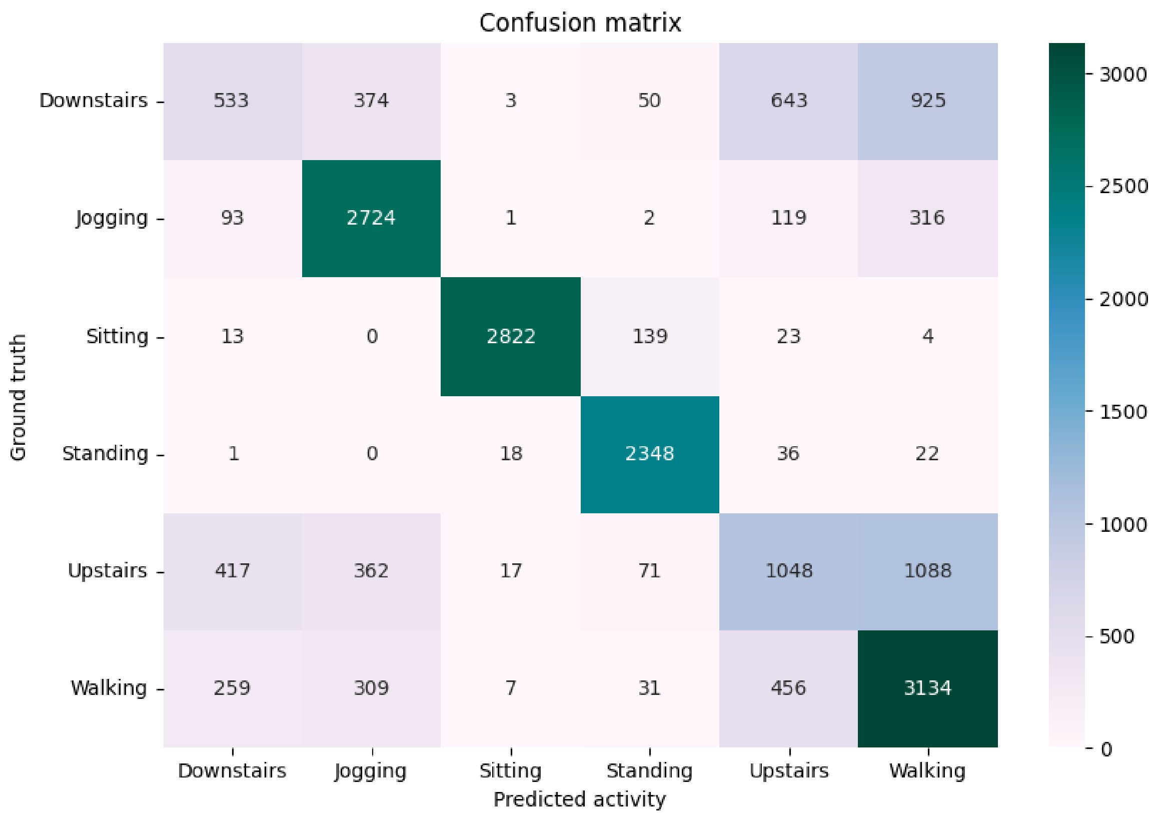


Figure A.16 Confusion matrix of CTRNN model run for 50 epochs, network architecture 16-10-6, accuracy 68.5% no time constant training, tau=Ones.