Chapter 3

Sensor Signals based Early Dementia Detection System using Travel Pattern Classification

This chapter presents an early dementia detection system using inhabitant travel pattern classification. The system segments the movements into travel episodes and classifies them using a recurrent neural network. The system handles the unbalanced classes of travel patterns by using the focal loss and enhances the discriminative power of the deeply learned features by the center loss function. We conduct several experiments on real-life datasets to verify the accuracy of the system.

3.1 Introduction

Many older adults suffer from some degree of cognitive impairment, such as dementia. Dementia is an overall term for diseases and conditions characterized by a decline in memory, language, problem-solving, and other thinking skills. There is no one test to determine if someone has dementia [98, 99]. Inefficient travel (or wandering) patterns of inhabitant are one of the first and primus indicators of progressive dementia. Many attempts have been made to understand the travel patterns of People with Dementia (PwD); one prevailing strategy has been the classification of travel patterns based on the geographical region [100]. The authors categorized the travel patterns into four basic types: direct travel, random travel, lapping, and pacing. Except direct travel pattern, other patterns are considered inefficient wandering patterns.

Different sensor modalities used to gather movement (travel pattern) information include wearable sensors, environmental sensors, and video sensors [30]. As discussed earlier, environmental sensors are considered acceptable solutions for sensing smart homes due to their non-intrusive characteristics, making them suitable for environments where user acceptance and privacy are needed. The literature on the classification of traveling patterns used conventional machine learning [101].

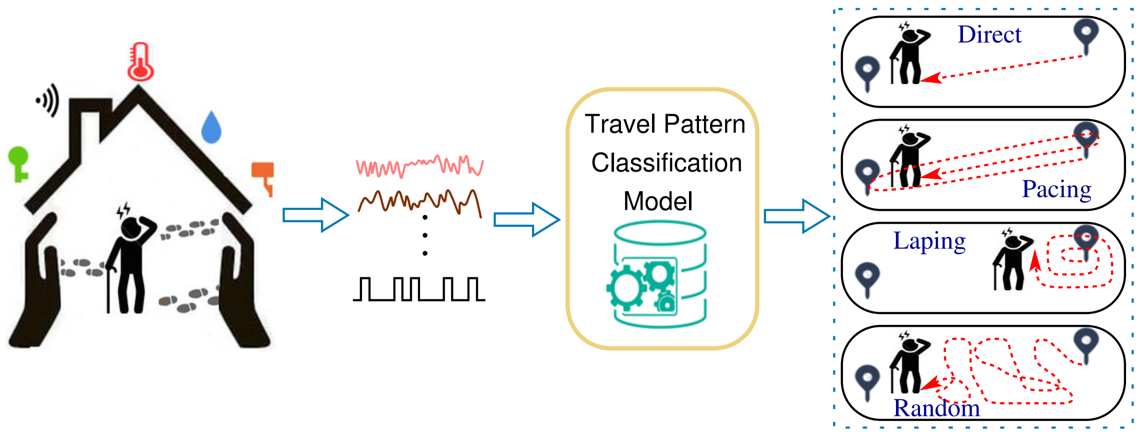


Figure 3.1: Illustration of classification of travel patterns in the EDD system.

In this chapter, we address the problem: ***how to detect dementia of an inhabitant at an early stage by using travel patterns of the inhabitant?*** To address this problem, we propose an Early Dementia Detection (EDD) system by using Recurrent Neural Network (RNN), which automatically extracts the high-level features, as shown in Figure 3.1. We consider the environmental passive infrared motion sensors for sensing the movements of the inhabitant. The system segments the movements into the travel episodes and classifies the episodes. The proposed system directly deals with raw movement sensory data and is fully end-to-end. To do this, we use RNN architecture, especially Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), for extracting the high-level features. The system also handles the unbalanced classes by using a special loss function.

3.1.1 Motivation

The work in this chapter is motivated by the following limitations noted in the literature. The first limitation of the existing work [36, 59, 61, 62] is the use of *wearable sensors* or *video sensors*. As discussed earlier, wearable sensors are inconvenient for inhabitants, and video sensors are not practical due to privacy issues. Furthermore, multimodality based solutions where using two or more types of sensors, (*e.g.*, wearable and video sensors) consist of the limitations of both types of sensors [102]. In this work, we consider only environmental sensors to overcome this limitation. Next, existing work either based on feature engineering [36, 59, 101], which is usually limited by the domain knowledge of humans or transforming the travel patterns into image-like representations [2].

3.1.2 Major contributions

The major contributions of this chapter are as follows:

- This work considers non-intrusive environmental sensors for acquiring the inhabitant movement information.
- The proposed system is fully end-to-end and directly deals with raw movement sensory data without requiring any other domain-specific knowledge, unlike previous methods where some hand-crafted feature engineering or image-like representations are required.
- Focal loss is applied to address the class imbalance problem. Center loss enhances the discriminative power of the deeply learned features.

- We evaluate the accuracy of the EDD system by using five real-life datasets collected in the CASAS smart home testbeds [44].

The rest of the chapter is organized as follows: Section 3.2 states the assumptions and overview of the EDD system. Section 3.3 presents EDD system in detail. Section 3.4 reports the experimental results followed by the conclusions in Section 3.5.

3.2 Preliminary and Overview of the EDD System

This section describes the terminologies used in this work and presents an overview of the EDD system.

3.2.1 Smart home

The EDD system considers a scenario where an inhabitant stays alone in a smart home denoted by region Ψ. We assume that region Ψ equipped with motion sensors mounted on the ceiling to detect inhabitant movements. Furthermore, the sensors are passive infrared binary sensors that generate events only when movement is present under their sensing region. The region Ψ consists of sub-regions (*e.g.*, bathroom, bedroom, living room, office, kitchen, *etc*) and consists of the obstacles (*e.g.*, table, chair, bed, door, television, *etc*). We assume that the region Ψ consists of total N sub-regions and obstacles, which are denoted by ψ_i, where $0 < i \leq N$. The *location* of ψ_i, denoted by L_i, be the coordinates of the ψ_i.

Definition 3.1 (Movement) *A movement, denoted by m_{ij}, is defined as an action taken to move from ψ_i to ψ_j, i.e., L_i to L_j, where $i \neq j$, $L_i \neq L_j$, and $1 \leq \{i, j\} \leq N$.*

Definition 3.2 (Episode) *An episode consists of one or more sequential movements, and each episode has a start and a stop location. An episode with n sub-regions $\psi_1, \psi_2, \cdots, \psi_n$ is denoted by the sequence in chronological order of the locations of*

ψ_i, *i.e.,*

$$E = \{L_1, L_2, ..., L_n\}, \tag{3.1}$$

where $L_i \neq L_{i+1}$ *and* $1 \leq i \leq n-1$*. The length of episode* E *is* n*, i.e., total number of locations it covers. Using episode* E*, the sequence of movements is given by*

$$\begin{aligned} M =&\{m_{12}, m_{23}, \cdots, m_{n-1n}\} \\ =&\{(L_1, L_2), (L_2, L_3), ..., (L_{n-1}, L_n)\}. \end{aligned} \tag{3.2}$$

3.2.2 Travel pattern model for PwD

We classify the travel patterns of PwD into the following two categories, as modeled in [100].

3.2.2.1 Efficient travel

Efficient travel is a straightforward path from one location to another without rerouting and diversion. Part (a) of Figure 3.2 illustrates an example scenario of direct pattern. An episode consists of an efficient travel, called as direct pattern, if $M = \{(L_i, L_j), (L_j, .), \cdots (., L_k)\}$, where $L_i \neq L_j \neq L_k$, and $1 \leq \{i, j, k\} \leq n$.

3.2.2.2 Inefficient travel

Inefficient travel have been used to define the wandering behavior of PwD [66,67]. The term *wandering* covers different types of behavior, including aimless movement without a discernible purpose. It consists of pacing, lapping, and random patterns as shown in parts (b)-(d) of Figure 3.2.

Pacing pattern is defined as a repeated path back and forth between two locations; it contains at least three consecutive to-and-fro movements. Next, lapping pattern has repeated circular path involving at least three locations. It contains at least two repeated circular routes involving at least three different locations; it can be the same or opposite direction. Finally, random pattern is a path, which has multiple locations with no particular order.

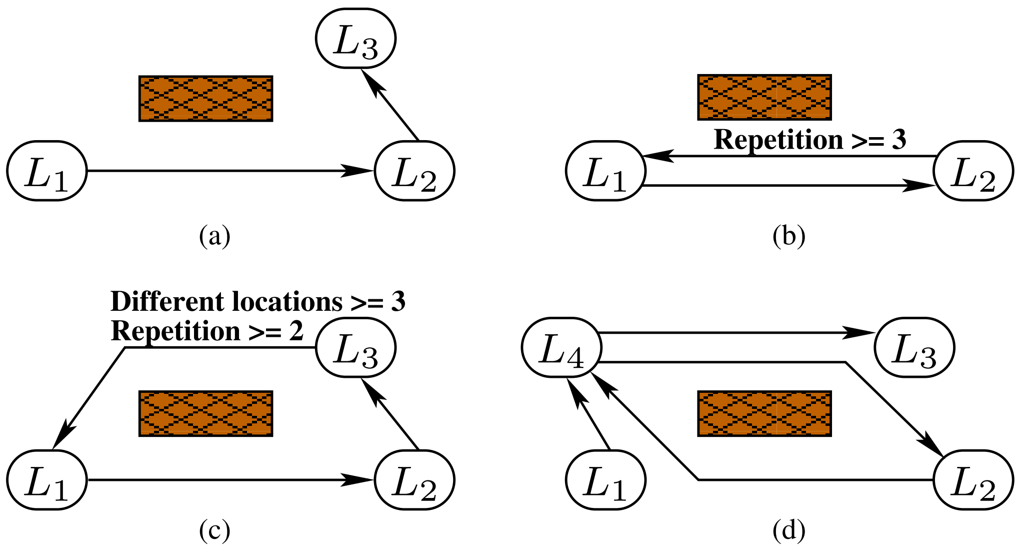


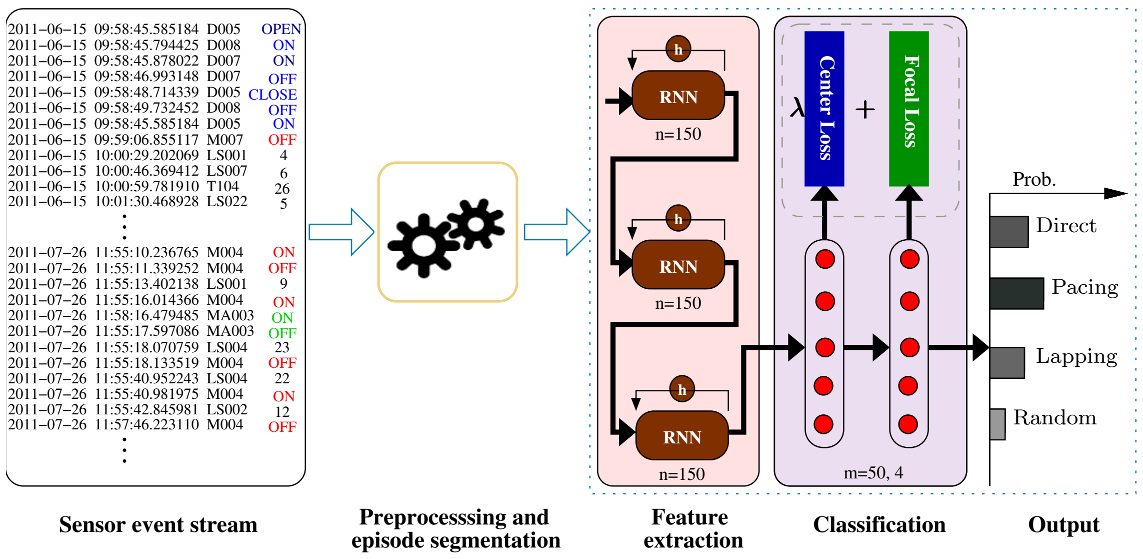
Figure 3.2: Travel patterns: (a) Direct (b) Pacing (c) Lapping (d) Random.

3.2.3 Overview of the EDD system

This work presents an Early Dementia Detection (EDD) system through the classification of indoor travel patterns as shown in Figure 3.3. The input of the EDD system is the sensor event stream generated by motion sensors embedded inside the smart home. The system first preprocesses the sensor event stream and generates the episodes (travel patterns). Next, the system uses recurrent neural network to extract the high-level features from episodes. Finally, the system minimizes a loss function, which is a linear combination of focal loss and center loss.

3.3 Early Dementia Detection System

This section presents a sensors based Early Dementia Detection (EDD) system through the classification of inhabitant travel patterns. Algorithm 3.1 illustrates the EED system which consists the following phases.



```
2011–06–15  09:58:45.585184  D005     OPEN
2011–06–15  09:58:45.794425  D008     ON
2011–06–15  09:58:45.878022  D007     ON
2011–06–15  09:58:46.993148  D007     OFF
2011–06–15  09:58:48.714339  D005     CLOSE
2011–06–15  09:58:49.732452  D008     OFF
2011–06–15  09:58:45.585184  D005     ON
2011–06–15  09:59:06.855117  M007     OFF
2011–06–15  10:00:29.202069  LS001    4
2011–06–15  10:00:46.369412  LS007    6
2011–06–15  10:00:59.781910  T104     26
2011–06–15  10:01:30.468928  LS022    5
                      ⋮
2011–07–26  11:55:10.236765  M004     ON
2011–07–26  11:55:11.339252  M004     OFF
2011–07–26  11:55:13.402138  LS001    9
2011–07–26  11:55:16.014366  M004     ON
2011–07–26  11:58:16.479485  MA003    ON
2011–07–26  11:55:17.597086  MA003    OFF
2011–07–26  11:55:18.070759  LS004    23
2011–07–26  11:55:18.133519  M004     OFF
2011–07–26  11:55:40.952243  LS004    22
2011–07–26  11:55:40.981975  M004     ON
2011–07–26  11:55:42.845981  LS002    12
2011–07–26  11:57:46.223110  M004     OFF
                      ⋮
```

Figure 3.3: EDD system to classify travel patterns, where m and n are number of nodes in a FC layer and nodes in a RNN unit, respectively.

3.3.1 Episode segmentation from sensor events

Let an inhabitant performs moving inside the region Ψ. The system generates an event stream whenever inhabitant comes inside the sensing range of a sensor, as shown in Figure 3.3. Segmentation of an episode is a process of separating the long consecutive movements into groups of movements that have a start and stop locations. Procedure 3.1 illustrates the steps of episode segmentation for a given z sensor events stream. The procedure iterates over all the sensor events (Step 1). The starting location of the first episode is the first 'ON' value of the binary sensor (Steps 4-7). The start location of any subsequent episode would be the next location after the stop location of the previous episode. The stop location of an episode is defined as one where the inhabitant spent more than a threshold time interval denoted by T_{th} (Steps 10-14). The time interval of two consecutive movements in an episode is not more than T_{th} (Step 15-17). Due to some visitor(s), there may be some multiperson episodes. We excluded such episodes by using a visitor detection algorithm [2] to acquire correct episodes that belong to inhabitant only.

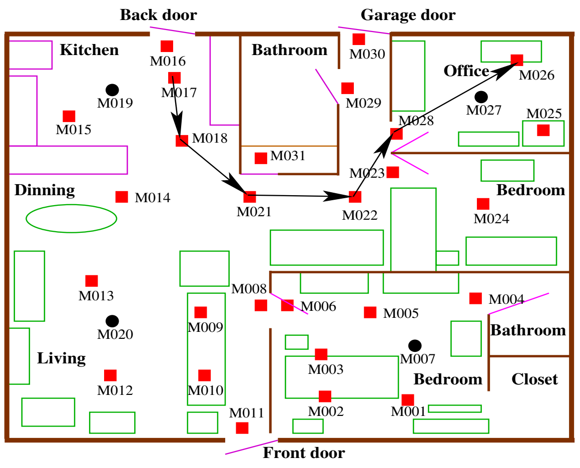
Procedure 3.1: Episode segmentation

```
    Input: Sensor events $\mathbf{e} = \{e_1, e_2, ..., e_z\}$ ;
    Output: Episodes $\mathbf{E} = (E_1, E_2, ..., E_n)$ ;
    Initialization: $e_i = 1$ # episode index;
 1  for $i \leftarrow 1$ to $z$ do
 2      /* For each event $< e_i >$ of stream $\mathbf{e}$*/
 3      if ($s_i \in \mathcal{M}'$ and $v_i ==$'ON') then
 4          if ($s_i$ is the first event) then
 5              Add $s_i$ to $E_1$;
 6              $time_{prev} = t_i$ and $location_{prev} = s_i$;
 7          else
 8              $interval = t_i - time_{prev}$;
 9              if $interval > T_{th}$ then
10                  $e_i++$ and Add $s_i$ to $E_{ei}$;
11                  $time_{prev} = t_i$ and $location_{prev} = s_i$;
12              else if $interval \leq T_{th}$ and $location_{prev} \neq s_i$ then
13                  Add $s_i$ to $E_{ei}$;
14                  $time_{prev} = t_i$ and $location_{prev} = s_i$;
15  Remove all $E_i$ from $\mathbf{E}$, where length of $E_i == 2$;
16  return $\mathbf{E}$;
```

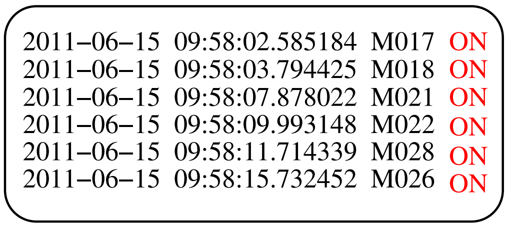
Example of Procedure 3.1: Parts (a) and (b) of Figure 3.4 show an example of the movement from the kitchen (M017) to the office (M026) and the corresponding motion sensor events generated due to those movements, respectively. This episode can be represented as M017→M018→M021→M022→M028→M026. If we assume that the inhabitant stayed more than T_{th} at location M021, then the movements would be divided into two separate episodes *i.e.* M017→M018→M021 and M021→M022→M028→M026.

3.3.2 High-level features generation

Next, the EDD system uses recurrent neural network for extracting high-level features. Recurrent neural networks are a generalization of the standard feedforward neural networks that contain cyclic connections, to allow it to model sequential data. Given a general input sequence $[\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_T}]$ where $\mathbf{x_i} \in \mathbb{R}^d$ (different samples may have different sequence length $\mathbf{T}$), at each time-step of RNN modeling, a hidden state is produced, resulting in a hidden sequence of $[\mathbf{h_1}, \mathbf{h_2}, ..., \mathbf{h_T}]$. The activation of the hidden state at



(a)



```
2011–06–15  09:58:02.585184  M017  ON
2011–06–15  09:58:03.794425  M018  ON
2011–06–15  09:58:07.878022  M021  ON
2011–06–15  09:58:09.993148  M022  ON
2011–06–15  09:58:11.714339  M028  ON
2011–06–15  09:58:15.732452  M026  ON
```

(b)

Figure 3.4: Illustration of episode segmentation: part (a) depicts a direct episode and part (b) shows corresponding generated sensor events. Marks "🟥" and "●" are IR motion, and wide-area IR motion sensors, respectively.

time-step **t** is computed as a function f of the current input $\mathbf{x_t}$ and previous hidden state $\mathbf{h_{t-1}}$ as

$$\mathbf{h_t} = H(\mathbf{W_{xh}x_t} + \mathbf{W_{hh}h_{t-1}} + \mathbf{b_h}) \tag{3.3}$$

At each time step, an optional output $\mathbf{o_t}$ can be produced as

$$\mathbf{y_t} = O(\mathbf{W_{ho}h_t} + \mathbf{b_o}) \tag{3.4}$$

where the **W** terms denote weight matrices and the **b** terms denote bias vectors. $H(\cdot)$ and $O(\cdot)$ are the activation functions in the hidden layer and the output layer, respectively. This kind of connection is not very useful to keep the information stored for long time durations; also, it does not allow forgetting nonessential information. Back-Propagation Through Time (BPTT) algorithm is used to train an RNN. The popular cells of the RNN are LSTM and GRU. We select the parameters of RNN based on the accuracy of the system, as shown in Section 3.4.

3.3.3 Objective loss formulation

The EDD system uses focal loss and center loss jointly as the supervisory signals to improve the travel pattern classification performance.

3.3.3.1 Focal loss

We applied Focal Loss (FL) [103] to address the class imbalance problem. Focal loss is a variant of standard cross-entropy loss, and it adds a modulating factor to the cross-entropy loss to reduce the loss for well-classified examples and focus on difficult ones. We denote focal loss by $\mathcal{L}_{FL}$. Let the input sample x with class label $y \in C$ and predicted output from the classifier for all classes are $\mathbf{z} \in \{z_1, z_2, ..., z_C\}$. The focal loss of the sample x can be written as [103]

$$\mathcal{L}_{FL} = -\sum_{i=1}^{C}(1 - \sigma(z_i^x))^\gamma \log(\sigma(z_i^x)) \tag{3.5}$$

We use an α balanced variant of focal loss:

$$\mathcal{L}_{FL} = -\alpha\sum_{i=1}^{C}(1 - \sigma(z_i^x))^\gamma \log(\sigma(z_i^x)) \tag{3.6}$$

where γ is the focusing parameter and σ is the Softmax function.

3.3.3.2 Center loss

The Center Loss (CL) [104], denoted by $\mathcal{L}_{CL}$, enhances the discriminative power of the deeply learned features. The center loss simultaneously learns a center for deep features of each class and penalizes the distances between the deep features and their corresponding class centers. The center loss function can be written as [104]

$$\mathcal{L}_{CL} = \frac{1}{2}\sum_{i=1}^{N} \|\hat{x}_i - c_{y_i}\|_2^2 \tag{3.7}$$

where N, $\hat{x}_i$, y_i, and c_{y_i} are the size of mini-batch, i^{th} deep feature, class label of $\hat{x}_i$, and class center of deep features of y_i, respectively.

• **Joint Supervision of Focal Loss and Center Loss:** The loss function of the proposed system is defined by combining the focal loss $\mathcal{L}_{FL}$ and center loss $\mathcal{L}_{CL}$ as:

$$\mathcal{L} = \mathcal{L}_{FL} + \lambda \mathcal{L}_{CL} \tag{3.8}$$

where hyperparameter λ balances the two-loss terms and $0 < \lambda \leq 1$. Substituting $\mathcal{L}_{FL}$ and $\mathcal{L}_{CL}$ from Eqs. 6.10 and 6.11, respectively,

$$\mathcal{L} = -\alpha \sum_{i=1}^{C} (1 - \sigma(z_i^x))^{\gamma} \log(\sigma(z_i^x)) + \frac{\lambda}{2} \sum_{i=1}^{N} \|\hat{x}_i - c_{y_i}\|_2^2 \tag{3.9}$$

Algorithm 3.1: Early Dementia Detection System

```
   Input: Event stream e, test instance $\mathbf{e}' \notin \mathbf{e}$, $\eta$, $\alpha$, $\gamma$ & $\lambda$;
   Output: Predicted class label for $\mathbf{e}'$;
1  Obtain $E = \{E_1, E_2, \cdots, E_n\}$ from e using Proc. 3.1;
2  Obtain label's of $E$ using [59], now labeled dataset
   $= \{(E_1, y_1), (E_2, y_2), \cdots, (E_n, y_n)\}$;
3  Initialize network parameter $\Theta$;
4  while not converge do
5      $t = t + 1$;
6      Forward propagation and compute loss;
7      Compute gradients ($g^t$);
8      Update weight parameters $\Theta$ by $\Theta^{t+1} = \Theta^t - \eta g^t$;
9  Obtain $E' = \{E'_1, E'_2, \cdots, E'_x\}$ from $\mathbf{e}'$ using Proc. 3.1;
10 for $i \leftarrow 1$ to $x$ do
11     Use the model $\Theta$ to classify $E'_i$;
```

3.4 Experiments and Results

This section presents experimental results for evaluating the performance of the EDD system and compares it with other state-of-the-art approaches.

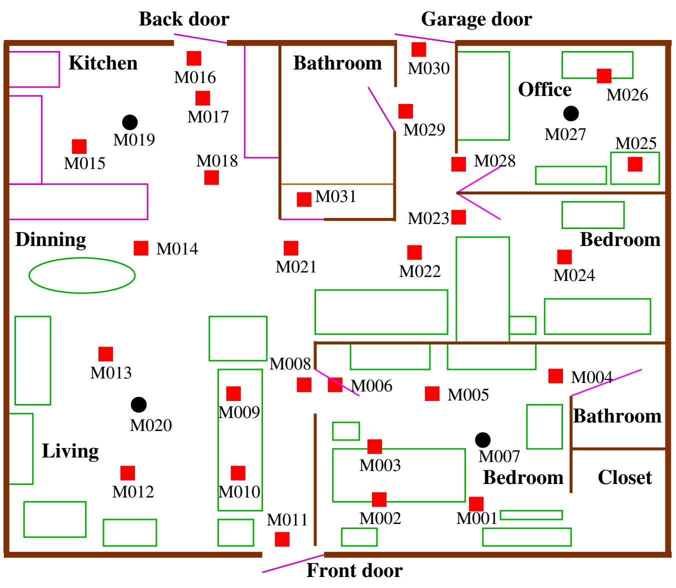


Figure 3.5: Aruba smart home floor plan and sensor layout. Marks "🟥" and "●" are IR motion, and wide-area IR motion sensors, respectively.

3.4.1 Datasets

We used publicly available five datasets collected in the CASAS [41] as shown in Table 3.1. The CASAS smart home testbeds assume that the infrared motion, magnetic door sensors, and temperature sensors are placed inside the smart home for observing physical surroundings of the inhabitants. Figure 3.5 illustrates a layout of Aruba testbed for collecting $\mathbf{D}_1$ dataset. Typical samples from the raw dataset are represented as *sensor event stream* in Figure 3.3. Motion sensors in a smart home are relevant for this study to capture inhabitant movements; therefore, we depicted only motion sensors in Figure 3.5. Motion sensors generate an ON event only when motion is present under their coverage area.

3.4.1.1 Training and testing sets

For training and testing sets preparation, first, we segmented episodes from the raw sensor dataset using the episode segmentation procedure illustrated in Procedure 3.1. For example, Procedure 3.1 classified dataset $\mathbf{D}_1$ in 46235, 20366, 12943, and 50162

Table 3.1: Summary of the datasets used in the EDD system.

ID	No. of events	No. of sensors	Time span (days)	Dataset updated
$\mathbf{D}_1$	5228654	27	625	2010
$\mathbf{D}_2$	669330	14	107	2011
$\mathbf{D}_3$	569330	14	60	2011
$\mathbf{D}_4$	216255	17	60	2017
$\mathbf{D}_5$	484931	22	60	2017

episodes as direct, pacing, lapping, and random, respectively. It shows that the dataset $\mathbf{D}_1$ was imbalanced. We randomly chosen 90% of the episodes from each pattern as the training set and the remaining 10% as the testing set. We used Vuong's MS pattern algorithm [59] for creating the labeled dataset. The Vuong's MS pattern algorithm classifies the episodes into one of the patterns: direct, pacing, lapping, or random. The algorithm determines whether an episode can be classified into the first three patterns *i.e.* direct, pacing, and lapping. If not, then the episode is classified as a random pattern. For the multi-pattern episode scenario, the episode is considered as a concatenation of single-pattern sub episodes. To classify multi-pattern episodes, first, the numbers of occurrence of each type of inefficient patterns are counted in the entire multi-pattern episode. Then the episode is classified as the pattern which has the highest number of count. When there are multiple patterns with the same number of the highest count, then the episode is classified based on the severity of the inefficient patterns, which is random, followed by lapping then pacing.

3.4.2 Implementation details

The weight matrices in the neural network layers are initialized by the *Xavier* Initialization and *Uniform He* initialization [105] when the layers use softmax and ReLU activation functions, respectively. The optimization algorithm is Adam, with a mini-batch size of 64. The initial learning rate is set to be 0.001. After each epoch, the models shuffle the training data to make different mini-batches. The proposed system converged within about 100 epochs. The hyperparameters α and γ of focal loss are set 0.25 and 2, respectively, following [106]. Empirically, we choose $\lambda = 0.01$, *i.e.*, the ratio between focal loss and center loss in Equation 3.8. To handle variable-length input sequences (episodes) in RNNs, we use padding and masking. We padded shorter episodes with zeros to the length of the longest episode *i.e.* 128 for dataset $\mathbf{D}_1$ for T_{th} of 15 seconds as given in [59]. We consider padding to make all the episodes to the equal length. We run the experiments under the environment of Intel Core i7-CPU @2.80-GHz, 8 GB RAM, GeForce GTX 1050 graphics card, and Ubuntu 18.04 operating system. We use Python based libraries, Keras, and Scikit-learn for implementation.

3.4.3 Performance metrics

We consider the following five widely used performance metrics for multi-class classification to estimate the performance of the proposed framework.

• **Precision:** measures the proportion of correct episodes of a class to the entire episodes classified as that class, calculated using the formula

$$Precision = \frac{TP}{TP + FP} \tag{3.10}$$

• **Recall:** (or Sensitivity) represents the proportion of episodes correctly classified as a given class to the actual total episodes in that class, calculated as

$$Recall = \frac{TP}{TP + FN} \tag{3.11}$$

• **Specificity:** measures the goodness of classifier at avoiding false alarms, calculated as

$$Recall = \frac{TN}{TN + FP} \tag{3.12}$$

• **F$_1$ score:** represents the harmonic mean of precision and recall. To evaluate the performance over the highly imbalanced dataset we assessed F_1 score, which is calculated as

$$F_1 = 2.\frac{Precision.Recall}{Precision + Recall} \tag{3.13}$$

• **Accuracy:** represents the proportion of correctly classified episodes over all classifications, calculated as

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}.100 \tag{3.14}$$

where TP, TN, FP, and FN are the true positives, true negatives, false positives, and false negatives, respectively. Apart from this, we also measure the test time (computation cost), which indicates the required time to classify an episode.

3.4.4 Baseline comparison

We consider six traditional machine learning classifiers *i.e.* Naive Bayes, KNN, SVC, Decision Tree, Random Forest, and Gradient Boost for the baseline comparison. We compute the following eight features for each travel episode which are used by the machine learning classifiers: entropy (F1), repeated locations (F2), repeated movements (F3), number of pairs of opposite movements (F4), number of movements (F5), time duration (F6), approximate distance (F7), and approximate average speed (F8) [2, 59]. Let, a given episode $E = (L_1, L_2, ..., L_n)$ and its corresponding movements $M = ((L_1 L_2), (L_2, L_3), ..., (L_{n-1}, L_n))$. The set of distinct elements in E is given by:

$$S_E = \{L_i, 1 \ll i \ll n | L_i \in L\} \tag{3.15}$$

The set of distinct elements in M is given by:

$$S_M = \{(L_i, L_{i+1}), 1 \ll i \ll n - 1 | (L_i, L_{i+1}) \subseteq M\} \tag{3.16}$$

The frequency of occurrence of each element in S_E is given as:

$$f_i = (\text{number of occurrences of } L_i \text{ in } E)/n, 1 \leq i \leq n \tag{3.17}$$

Then, F1-F8 features can be formulated as follows:

$$F1 = -\sum_{i=1}^{n} f_i log f_i \tag{3.18}$$

$$F2 = n - ||S_E|| \tag{3.19}$$

$$F3 = n - 1 - ||S_M||, \tag{3.20}$$

$$F4 = ||L_{j+1} \wedge L_{i+1}|| \tag{3.21}$$

$$F5 = n - 1 \tag{3.22}$$

$$F6 = time_{stop} - time_{start} \tag{3.23}$$

$$F7 = \sum_{i=1}^{n-1} \sqrt{(x_{i,2} - x_{i,1})^2 + (y_{i,2} - y_{i,1})^2} \tag{3.24}$$

$$F8 = \frac{F7}{F6} \tag{3.25}$$

where $x_{i,2}$, $x_{i,1}$ are x coordinates of two locations in i-th movement; similarly, $y_{i,2}$, $y_{i,1}$ are y coordinates of two locations in i-th movement.

3.4.5 Results and discussion

This section carries out the experimental evaluation to validate the performance of the proposed EDD system.

3.4.5.1 Performance comparison of different network architectures

Table 3.2 shows the comparison of different network architectures which can be represented by a general form as

$$A \xrightarrow{a} \underbrace{[B_1, ..., B_n]}_{B} \xrightarrow{b} C \xrightarrow{c} D \tag{3.26}$$

Symbol A is the dimension for each input sequence. The symbol B indicates n stacked recurrent layers, as shown in Figure 3.3, and B_i is the number of units at the i-th recurrent layer. Moreover, the symbol C is the size of the dense (fully connected) layer, and D is the number of units for the output layer (also the number of travel pattern classes).

The third column in Table 3.2 shows the recurrent unit type (LSTM or GRU) for each model. Different networks are compared from five aspects, including Memory consumption in the fourth column, total training time (in hours) in the fifth column, latency or testing speed (in millisecond) for one episode in the sixth column, training accuracy in the seventh column, and test accuracy in the last column. Latency (testing speed) is the time spends during the classification of an episode. Other configurations are the same as described in Section 3.4.2 to give a fair comparison of different architectures. It is shown that the best performance (test accuracy) is achieved by ARCH6.

Table 3.2: Comparison of different architectures for travel pattern classification using $\mathbf{D}_1$ dataset.

Name	Architecture	Recurrent Type	Memory	Train Time	Latency	Train Acc.	Test Acc.
	Part (a): Comparison of different depths						
ARCH1	$128 \xrightarrow{a} [150] \xrightarrow{b} 50 \xrightarrow{c} 4$	LSTM	1.20 MB	3.28 h	0.0042 ms	97.37%	96.80%
ARCH2	$128 \xrightarrow{a} [150] \xrightarrow{b} 50 \xrightarrow{c} 4$	GRU	957 KB	2.89 h	0.0025 ms	97.12%	96.88%
ARCH3	$128 \xrightarrow{a} [150,150] \xrightarrow{b} 50 \xrightarrow{c} 4$	LSTM	3.40 MB	6.38 h	0.0085 ms	97.96%	97.59%
ARCH4	$128 \xrightarrow{a} [150,150] \xrightarrow{b} 50 \xrightarrow{c} 4$	GRU	2.60 MB	5.02 h	0.0057 ms	98.47%	98.17%
ARCH5	$128 \xrightarrow{a} [150,150,150] \xrightarrow{b} 50 \xrightarrow{c} 4$	LSTM	5.60 MB	11.22 h	0.0181 ms	99.21%	98.13%
ARCH6	$128 \xrightarrow{a} [150,150,150] \xrightarrow{b} 50 \xrightarrow{c} 4$	GRU	4.20 MB	6.10 h	0.0130 ms	98.83%	**98.63%**
	Part (b): Comparison of different number of recurrent units						
ARCH7	$128 \xrightarrow{a} [100,100,100] \xrightarrow{b} 50 \xrightarrow{c} 4$	LSTM	3.23 MB	5.14 h	0.0165 ms	98.16%	98.12%
ARCH8	$128 \xrightarrow{a} [100,100,100] \xrightarrow{b} 50 \xrightarrow{c} 4$	GRU	1.81 MB	4.05 h	0.0116 ms	98.38%	97.96%
ARCH9	$128 \xrightarrow{a} [100,150,200] \xrightarrow{b} 50 \xrightarrow{c} 4$	LSTM	5.80 MB	11.37 h	0.0245 ms	99.18%	98.25%
ARCH10	$128 \xrightarrow{a} [100,150,200] \xrightarrow{b} 50 \xrightarrow{c} 4$	GRU	4.40 MB	6.30 h	0.0196 ms	98.46%	98.06%
ARCH11	$128 \xrightarrow{a} [200,200,200] \xrightarrow{b} 50 \xrightarrow{c} 4$	LSTM	9.12 MB	14.7 h	0.0269 ms	98.89%	97.86%
ARCH12	$128 \xrightarrow{a} [200,200,200] \xrightarrow{b} 50 \xrightarrow{c} 4$	GRU	7.05 MB	8.20 h	0.0212 ms	98.66%	97.90%

3.4.5.2 Comparison of LSTM and GRU

In this section, multiple RNN models with either LSTM or GRU recurrent units were trained and compared under the same configurations. As shown in part (a) of Table 3.2, from the perspective of test accuracy, ARCH2 outperforms ARCH1, ARCH4 beats ARCH3, and ARCH6 is better than ARCH5. Similarly, part (b) of Table 3.2

shows, ARCH8, ARCH10, and ARCH12 outperforms ARCH7, ARCH9, and ARCH11, respectively, in test accuracy. However, the differences are not significant. Therefore, the only conclusion we can draw is that LSTM and GRU have comparable classification accuracies, and GRU is performing better on our classification task. Furthermore, as revealed in Table 3.2, from the perspectives of memory consumption, training time, and especially latency, we can conclude that GRU is much better than LSTM. The GRU can be viewed as a light-weight version of LSTM, and still shares similar functionalities with LSTM, which makes GRU favored by practical applications with particular requirements on memory or speed.

3.4.5.3 Comparison of different depths

Different depths for the networks are compared in Table 3.2(a). Compared with only one recurrent layer (ARCH1 and ARCH2), stacking two layers (ARCH3 and ARCH4) and three layers (ARCH5 and ARCH6) can indeed improve both the training and test accuracies. Moreover, as shown in part (a) of Table 3.2, with more stacked recurrent layers, memory size, training time, and testing time are increasing. Beyond the depth of three layers, increasing network capacity further did not yield any performance improvement. From the perspectives of accuracy, ARCH6 is preferred among the other network architectures.

3.4.5.4 Comparison of different number of recurrent units

Different number of units in recurrent layers are also compared in part (b) of Table 3.2. The test accuracy does not vary significantly as we vary the number of recurrent units. As shown in Table 3.2, ARCH6 is the best performing network architecture, which has 150 units in each recurrent layer.

3.4.5.5 Comparison of different RNN variants

In this section, we tested the following RNN variants: LSTM, GRU, Bi-directional LSTM, Bi-directional GRU [107], LSTM with peephole connections [108], MUT1, MUT2, and MUT3 [109]. MUT1, MUT2, and MUT3 recurrent neural architectures evolved from a pool of several thousand candidate architectures [109]. Table 3.3 shows the comparison results. We found that the GRU (ARCH6) outperformed all other variants and the performance of MUT2 was comparable to GRU. This may be because most of the variants of RNN are developed based on the image processing where the dimension of the data is very huge. Due to the limited size of sensor dataset, such co-relations and modifications are not very effective.

Table 3.3: Comparison of different RNN variants for travel pattern classification using $\mathbf{D}_1$ dataset.

RNN variant	Test Accuracy
LSTM	98.13%
GRU (ARCH6)	**98.63%**
Bi-directional LSTM	97.23%
Bi-directional GRU	97.47%
LSTM with peephole connections	97.89%
MUT1	98.43%
MUT2	98.60%
MUT3	98.17%

3.4.5.6 Effectiveness of loss function

To check the effectiveness of loss function, we used dataset $\mathbf{D}_1$ and retrained ARCH6 with conventional softmax cross-entropy loss, and we witnessed accuracy was decreased to 97.31 percent. Since dataset $\mathbf{D}_1$ was highly imbalanced, therefore could not be exploited well by deep learning models with cross-entropy loss. Moreover, we also retrained ARCH6 with focal loss, and the accuracy was 98 percent, which was better than the earlier case. This suggests that imbalanced datasets can be exploited well by deep learning models with focal loss. Focal loss employs a modulating factor to the cross-entropy loss to help focus on hard samples and down-weight the many easy ones. When we retrained ARCH6 with a joint loss combining focal loss and center loss, the accuracy was further improved to 98.63 percent. This is because the discrimination ability caused by the center loss has a positive impact on classification performance. Center loss adds a cluster-based loss term to the focal loss, ensuring that the learned representations have both large interclass margins and small intraclass variations. That is, the learned representations are not only separable but also discriminative.

3.4.6 Comparison with other approaches

Table 3.4 lists the performance achieved by different machine learning classifiers on five datasets $\mathbf{D}_1$-$\mathbf{D}_5$ to compare the EDD system with other approaches. It is shown that the deep learning based proposed system outperforms the traditional machine learning methods with significant margins. Recently, state-of-the-art performance on the Aruba dataset has been achieved by [2] with a DCNN classifier for indoor travel patterns. The authors considered interval time (T_{th}) of 10 seconds for episode segmentation and reported 0.979, 0.978, 0.993, 0.978, and 97.84% for precision, recall, specificity, F_1 score, and accuracy, respectively. The work in [2] has a particular requirement of transforming the sensor data into image representations, which increases the time complexity and adds new constraints that the episodes can be of a maximum length of 32 only. The EDD system directly deals with the raw movement data and therefore has the potential to exploit additional information that is discarded in the spatial representations. Moreover, the EDD system does not have any episode length constraint and classifies travel episodes of all given length. The EDD system is end-to-end, depending only on generic priors about sequential data processing, and not requiring any other domain-specific knowledge. As shown in Table 3.4, the EDD system (ARCH6) outperforms all baseline approaches and DCNN [2] in all performance metrics.

Table 3.4: Comparison with other approaches, where **C**, **D**, **P**, **R**, **S**, F_1, and **A**, denote classifier, dataset, precision, recall, specificity, F_1 score, and accuracy, respectively.

C	**D**	**P**	**R**	**S**	F_1	**A**	Latency
Naive Bayes	$\mathbf{D}_1$	0.7033	0.5962	0.9141	0.5985	76.90%	0.0004 ms
	$\mathbf{D}_2$	0.7833	0.6982	0.9342	0.6984	86.80%	0.0013 ms
	$\mathbf{D}_3$	0.8503	0.8746	0.9877	0.8591	95.59%	0.0013 ms
	$\mathbf{D}_4$	0.7668	0.6865	0.9265	0.6589	81.84%	0.0009 ms
	$\mathbf{D}_5$	0.7256	0.6568	0.9178	0.6658	82.65%	0.0011 ms
KNN	$\mathbf{D}_1$	0.7959	0.7867	0.9489	0.7903	85.06%	0.0293 ms
	$\mathbf{D}_2$	0.7251	0.6549	0.9003	0.6798	78.95%	0.0235 ms
	$\mathbf{D}_3$	0.8222	0.7297	0.9246	0.7668	84.50%	0.0249 ms
	$\mathbf{D}_4$	0.7565	0.7326	0.9125	0.7388	83.68%	0.0264 ms
	$\mathbf{D}_5$	0.7863	0.7482	0.9321	0.7484	84.80%	0.0274 ms
SVC	$\mathbf{D}_1$	0.8373	0.8135	0.9568	0.8232	87.63%	2.5926 ms
	$\mathbf{D}_2$	0.8720	0.7332	0.9301	0.7865	86.60%	0.0739 ms
	$\mathbf{D}_3$	0.1434	0.2500	0.7500	0.1823	57.39%	0.0740 ms
	$\mathbf{D}_4$	0.7123	0.5582	0.9051	0.5855	79.98%	0.0854 ms
	$\mathbf{D}_5$	0.7633	0.7282	0.9042	0.7984	80.81%	0.0963 ms
Decision Tree	$\mathbf{D}_1$	0.8582	0.8613	0.9679	0.8597	90.37%	0.0001 ms
	$\mathbf{D}_2$	0.9334	0.9246	0.9678	0.9284	95.69%	0.0009 ms
	$\mathbf{D}_3$	0.9394	0.9648	0.9944	0.9506	98.06%	0.0009 ms
	$\mathbf{D}_4$	0.9033	0.7962	0.9541	0.8985	93.90%	0.0010 ms
	$\mathbf{D}_5$	0.8833	0.7956	0.9342	0.8984	93.80%	0.0013 ms
Rand. Forest	$\mathbf{D}_1$	0.8680	0.8779	0.9715	0.8725	91.20%	0.0095 ms
	$\mathbf{D}_2$	0.9288	0.9011	0.9893	0.9087	96.34%	0.0173 ms
	$\mathbf{D}_3$	0.9536	0.9709	0.9959	0.9610	98.59%	0.0179 ms
	$\mathbf{D}_4$	0.8083	0.8962	0.9154	0.7985	89.90%	0.0204 ms
	$\mathbf{D}_5$	0.9033	0.9082	0.9542	0.8912	92.65%	0.0225 ms
Grad. Boost	$\mathbf{D}_1$	0.8682	0.8925	0.9731	0.8783	91.39%	0.0034 ms
	$\mathbf{D}_2$	0.9419	0.9492	0.9831	0.9442	96.38%	0.0041 ms
	$\mathbf{D}_3$	0.9536	0.9709	0.9959	0.9610	98.59%	0.0042 ms
	$\mathbf{D}_4$	0.9063	0.9062	0.9741	0.8985	92.90%	0.0044 ms
	$\mathbf{D}_5$	0.9133	0.8982	0.9652	0.9254	92.80%	0.0052 ms
DCNN [2]	$\mathbf{D}_1$	0.9717	0.9716	0.9821	0.9716	96.41%	0.4351 ms
	$\mathbf{D}_2$	0.9635	0.9530	0.9829	0.9550	96.53%	0.4347 ms
	$\mathbf{D}_3$	0.9715	0.9718	0.9907	0.9716	97.66%	0.4340 ms
	$\mathbf{D}_4$	0.9533	0.9662	0.9741	0.9685	96.90%	0.4386 ms
	$\mathbf{D}_5$	0.9643	0.9712	0.9742	0.9584	96.80%	0.4405 ms
Prop. EED	$\mathbf{D}_1$	0.9863	0.9862	0.9950	0.9862	98.63%	2.0980 ms
	$\mathbf{D}_2$	0.9866	0.9865	0.9956	0.9865	98.65%	0.4536 ms
	$\mathbf{D}_3$	0.9898	0.9898	0.9966	0.9898	98.98%	0.4305 ms
	$\mathbf{D}_4$	0.9733	0.9862	0.9941	0.9865	98.35%	0.4402 ms
	$\mathbf{D}_5$	0.9813	0.9882	0.9942	0.9884	98.91%	0.4652 ms

Table 3.5 demonstrates the confusion matrix for the proposed system recognitions on datasets $\mathbf{D}_1$-$\mathbf{D}_5$. In this experiment, we take the average value of the correct classifications of patterns of five datasets. The large number in diagonal entries of the matrix shows that the proposed EDD system successfully recognized the given episode. Finally, Table 3.6 illustrates the difference of the correctly classification of pattern, between proposed EDD and existing [2], *i.e.*, probability of correctly classifying a given pattern by EDD-probability of correctly classifying a given pattern by [2]. The result shows that the proposed EDD gives higher accuracy for classifying the patterns.

Table 3.5: Confusion matrix for EDD system datasets.

		Predicted label			
		Direct	**Lapping**	**Pacing**	**Random**
True label	**Direct**	[98.71%]	[0]	[0]	[1.29%]
	Lapping	[0]	[99.40%]	[0.34%]	[0.26%]
	Pacing	[0]	[0.79%]	[98.28%]	[0.93%]
	Random	[0.82%]	[0.05%]	[0.78%]	[98.35%]

Table 3.6: Confusion matrix for difference between EDD and existing work [2].

		Predicted label			
		Direct	**Lapping**	**Pacing**	**Random**
True label	**Direct**	[1.45%]	[-0.70%]	[0]	[-0.75%]
	Lapping	[-1.0%]	[1.52%]	[-0.02%]	[-0.5%]
	Pacing	[0]	[-1.0%]	[1.04%]	[-0.04%]
	Random	[-0.70%]	[-0.05%]	[-0.75%]	[1.5%]

3.5 Conclusion

This chapter proposed a sensor based EDD system by using RNN, which automatically extracts the high-level features. We considered the environmental passive sensors for sensing the movements of the inhabitant. The system segments the movements into the episodes and classifies the travel patterns. We used focal loss for handling the unbalanced classes of travel patterns, and center loss to enhance the discriminative power of the deeply learned features. Experimental results support our hypothesis that using high-level features, focal loss, and center loss together can converge faster to optimal accuracy, even using passive sensors and imbalanced classes. We believe that the proposed system can be very useful for automated dementia detection at an early stage.