Chapter 5

Real-time Activities of Daily Living Recognition under Long-Tailed Class Distribution

This chapter presents an online system that recognizes ADL while considering the long-tailed class distribution problem. The system first generates hand-crafted and high-level features using conventional learning and deep learning, which cover the advantage of both technologies to recognize ADL. Next, the system uses an ensemble technique to concatenate the generated features. Finally, the system minimizes a loss function, which is a linear combination of focal loss for addressing the long-tailed class distribution problem and center loss for enhancing the discriminative power of the deeply learned features. We conduct several experiments on real-life long-tailed datasets to verify the accuracy of the proposed system.

5.1 Introduction

The world is experiencing growth in the proportion and number of older adults in their population, owing to declining fertility and increasing longevity. Moreover, the

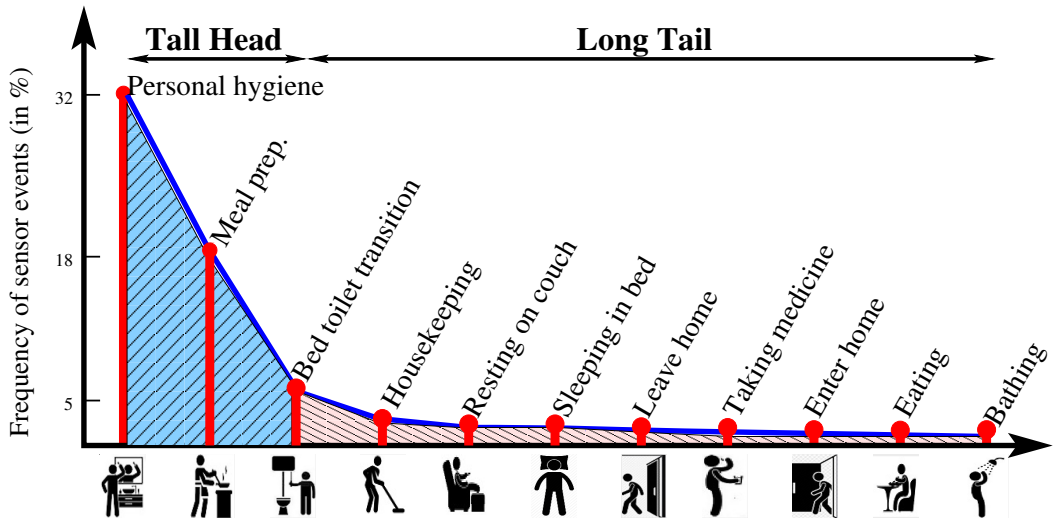


Figure 5.1: Illustration of ADL dataset having long-tailed class distribution.

number of older adults who live alone at home is also increasing. As discussed earlier, despite having different medical and cognitive problems associated with old age, older adults wish to live independently in their own homes for as long as possible rather than move to an aged care facility [13]. Furthermore, according to the World Health Organization, one billion people live with disabilities around the world [114]. Many older and disabled adults have difficulties in performing the ADL. Examples of ADL are shown in Figure 5.1, where x-axis and y-axis illustrate ADL and frequency of sensor events associated with each ADL, respectively. The need to improve the quality of independent living for such people underlines the growing significance of health-assistive technologies.

Smart homes have become widely popular, especially in providing such assistive services. ADLR in smart homes is crucial for developing advanced assistive services and is an active research topic. ADLR identifies the ADL performed by the inhabitant by using the sequence of sensor events. From a machine learning viewpoint, ADLR maps a sequence of sensor events to a matching ADL label.

In recent years, deep learning has become the prevalent approach in ADLR due to its high accuracy. Such approaches can learn features automatically from an ad-

equately large training set without requiring any specific domain knowledge. Even though deep learning based methods now outperform the state-of-the-art in several recognition problems, still many challenges remain. Most apparently, the collection and labeling of enough data for ADLR are often expensive and time-consuming, especially for the elderly and impaired. Another issue is that the collected real-life data follows a *long-tailed class distribution*, *i.e.*, a few classes account for most of the data, while most classes are under-represented [29], as shown in Figure 5.1. The long-tailed class distribution can result in a biased ADLR system optimized to favor the majority classes while failing to identify the discriminant features needed to recognize the minority classes. This behavior is termed the *class-imbalance problem*, and it is intrinsically exhibited in nearly all of the collected ADL databases. Such datasets are termed as *long-tailed datasets*.

Moreover, various ambient intelligence applications require real-time/online ADLR systems, *i.e.*, systems that can recognize the activities as the sensor event sequence arrives. The unpredictability nature of future events and the requirement of dealing with the un-segmented activity streams make online ADLR very challenging [31, 33]. Until now, research on ADLR has mostly centered around offline recognition.

ADLR is a critical module in the design of a smart home. It is valuable for energy-efficient home automation, anomaly detection, and automated context-aware prompt. For example, using ADLR, a smart home can prompt the inhabitant to start essential activities. For illustration, consider "taking medicine" is an essential activity. A smart home recognizes that the "taking meal" activity has occurred. After that, if the smart home does not recognize "taking medicine" activity within a specific timespan, a prompt can be delivered to the inhabitant.

In this chapter, we address the problem: ***how to recognize the activities of daily living online with the long-tailed dataset?*** To address this problem, we propose an ADLR system by using conventional learning and deep learning, which

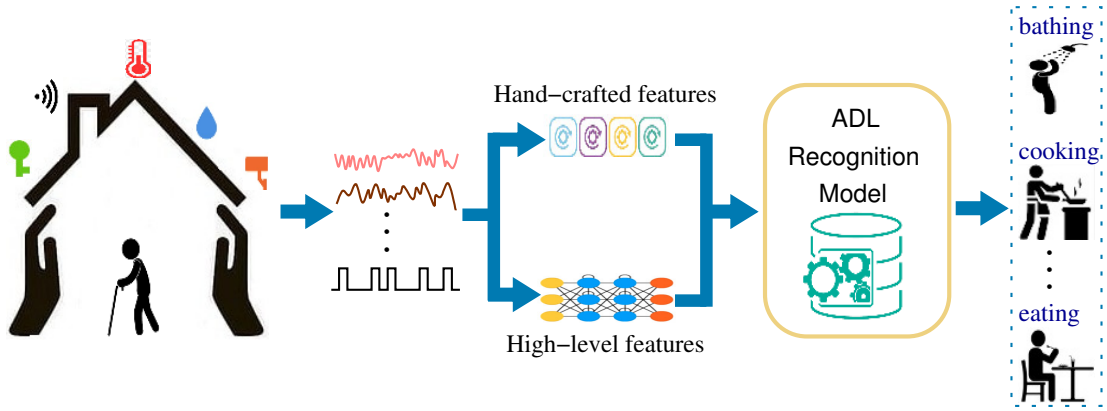


Figure 5.2: Illustration of recognition of ADL in the proposed system.

cover the advantage of both technologies to recognize the ADL better, as shown in Figure 5.2. Furthermore, the system adopts focal loss [103] and center loss [104] jointly as the supervisory signals to boost the ADL recognition performance. Different from the existing work, the proposed ADLR system considers data collected from external/ environmental sensors such as infrared motion, magnetic door, light, or temperature sensors for unobtrusive ADL recognition. The non-intrusive characteristics of such sensors make them practically useful and especially for older and disabled adults.

5.1.1 Motivation

The work in this chapter is motivated by the following limitations noted in the literature. The first limitation of the existing work [19–28] is the use of *video* or *wearable sensors*. However, as discussed earlier, video sensors are not practical due to privacy issues, and wearable sensors may be uncomfortable and inconvenient for inhabitants. In this work, we consider only environmental sensors to overcome this limitation. Next, existing work either uses hand-crafted features or high-level features to recognize the ADL. We effectively combine both sources of information to lift the ADLR performance. Also, most of the approaches for solving long-tailed distribution problems [79–83, 115] work on image dataset. All such approaches are not usually suitable for sensor event-based

ADLR systems because they have limited size datasets. Some existing work [6, 30, 33] uses environmental sensors and high-level features but not handling the long-tailed distribution problem. To this end, we conclude that none of the work recognizes the ADL online with high accuracy addressing the class-imbalance problem.

5.1.2 Major contributions

To the best of our knowledge, we are among the first to address the class-imbalance problem in real-time/online ADL recognition. Apart from this, the major contributions of this chapter are as follows:

- We propose a system that unobtrusively recognizes ADL in real-time from streaming data with very high accuracy.
- To improve ADL recognition performance, we use mutual information of the sensor events and cyclical encoding. The mutual information reduces the weight of irrelevant sensor events, thereby enhances the given sliding window's recognition performance.
- We effectively combine deep learned features with hand-crafted features via an ensemble framework to recognize ADL better.
- We combine the focal loss and center loss functions to minimize the intra-class features variances and maximize the inter-class features differences further for training a more discriminative ADL recognition system.
- Our proposed system achieved new state-of-the-art performance on the datasets collected in CASAS smart home testbeds [41]. We use five CASAS datasets for empirical evaluation.

The rest of the chapter is organized as follows: Section 5.2 states the assumptions and formally defines the notations and overview of the ADLR system. The ADLR system is presented in Section 5.3. Section 5.4 presents the experimental results followed by the conclusions in Section 5.5.

5.2 Preliminary and Overview of the ADLR System

This section describes the terminologies used in this work and then presents an overview of the ADLR system. The block diagram of the ADLR system is shown in Figure 5.2.

5.2.1 Preliminary

The ADLR system considers a scenario where an inhabitant stays alone in a smart apartment. The smart apartment is equipped with motion sensors on the ceiling to detect motion and magnetic sensors on doors and cabinets to detect their openings and closings. The motion sensors are further classified into infrared motion sensors and wide-area infrared motion sensors based on the sensing range of the sensors. The sensors are binary sensors that generate the events only when the inhabitant performs some ADL inside the sensors' sensing range.

Definition 5.1 (Sensor event) *An event is represented by 4-tuple <date (d_i), time (t_i), sensor identification (s_i), sensor value (v_i)> and answers the following questions: what day the sensor event generated? what time of day event is generated?, which sensor generates the event?, and what is the sensor state?, respectively. For binary state motion and magnetic sensors, the value of v_i is ON/OFF, OPEN/CLOSE, respectively.*

Definition 5.2 (Cyclical encoding) *The cyclical encoding of a feature x is defined by the sine and cosine trigonometric transformation of the feature x, i.e., $x_{sin} = \sin\left(\frac{2\pi x}{\max(x)}\right)$ and $x_{cos} = \cos\left(\frac{2\pi x}{\max(x)}\right)$.*

Definition 5.3 (One-Hot encoding) *Let the categorical feature x take m number of different values, i.e., $x_1, x_2, \cdots, x_i, \cdots, x_m$. The one-hot encoding of this feature when having value x_k is defined as $x_k : \{\underbrace{0, \cdots, 0}_{k-1}, 1, \underbrace{0, \cdots, 0}_{m-k}\}$.*

Definition 5.4 (Mutual dependency matrix) *Two sensors are said to be mutually dependent if they generate the events sequentially, i.e., one after another. For a*

given sensor stream $\{e_1, e_2, ...e_N\}$*, the mutual dependency between sensor* $s_i \in \mathcal{S}$ *and* $s_j \in \{\mathcal{S} - s_i\}$ *is given by* $D_{i\leftrightarrow j} = \frac{1}{N}\sum_{x=1}^{N-1}(\mathbb{I}(e_x \text{ and } e_{x+1} \text{ generated by } s_i \text{ and } s_j) + \mathbb{I}(e_x \text{ and } e_{x+1} \text{ generated by } s_j \text{ and } s_i))$*, where identity function* $\mathbb{I}(.)$ *returns true only when* s_i *and* s_j *consecutively generate the events. The mutual dependency matrix of the sensors is given by*

$$\mathbf{D^{k\times k}} = \begin{bmatrix} D_{1\leftrightarrow 1} & \cdots & D_{1\leftrightarrow k} \\ \vdots & \ddots & \vdots \\ D_{k\leftrightarrow 1} & \cdots & D_{k\leftrightarrow k} \end{bmatrix} \tag{5.1}$$

5.2.2 Overview of the ADLR system

Architecture of the ADLR system is illustrated in Figure 5.3. The input of the system is the sensor event stream generated by embedded sensors inside the smart apartment. An example scenario of a smart apartment floor plan and sensor layout is shown in Figure 5.4. The sensor events are generated by sensors when inhabitant performs an activity within the sensing range of these sensors. The system first preprocesses and segments event stream into the fixed-length sliding windows. Next, the system extracts the hand-crafted and high-level features. Such features are concatenated before feeding into Multi-Layer Perceptron (MLP). The system uses the focal loss to address the problem of long-tailed dataset and center loss for discriminative feature learning. Finally, the system generates the output activity label that corresponds to the most recent event in the given sliding window.

5.3 Activities of Daily Living Recognition System

This section presents the **A**ctivities of **D**aily **L**iving **R**ecognition (ADLR) system to recognize ADL while addressing the long-tailed class distribution problem. The architecture of the system is shown in Figure 5.3, which consists of the following steps.

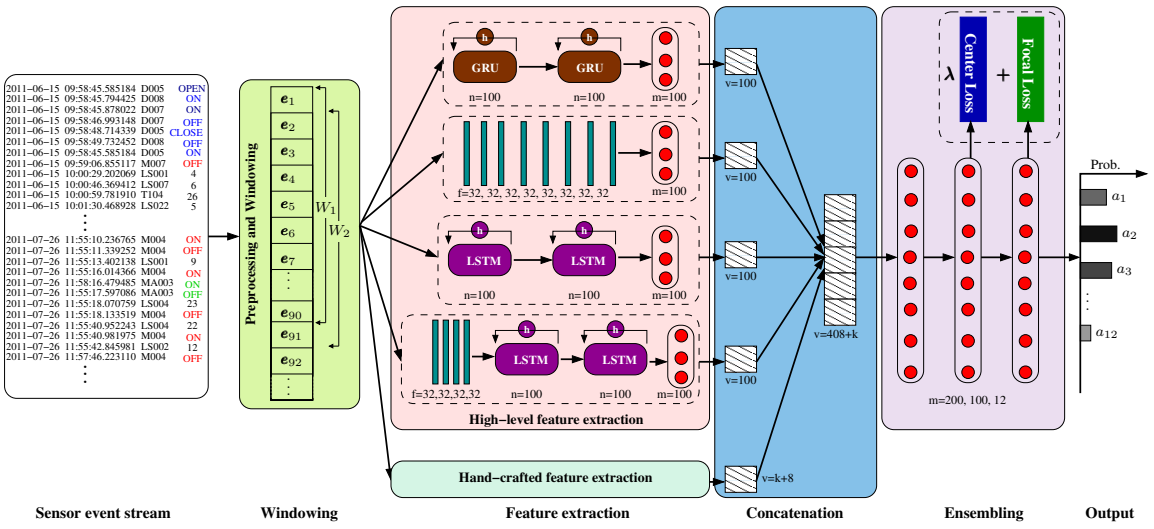


Figure 5.3: Architecture of the ADLR system, where k, f, m, n and v are number of sensors, filters, nodes in a FC layer, nodes in a GRU or LSTM unit, and the dimensions of the feature vector, respectively.

5.3.1 Generating sliding windows from event stream

Let a user perform a set of n activities which are denoted as $\{a_1, a_2, \cdots, a_n\}$. The system generates an event stream whenever a user performs the activities. The system segments such event stream into the fixed-size event count-based window as shown in Figure 5.3. The number of events in a window is known as the length of the window, which is denoted by l. The event stream and window vector are denoted by $\mathbf{e} = \{e_1, e_2, \cdots, e_i, \cdots\}^T$ and $\mathbf{W} = \{W_1, W_2, \cdots, W_j, \cdots\}^T$, respectively. A window W_j consists of the following events $\{e_j, e_{j+1}, \cdots, e_{j+l-1}\}$. The system empirically derives the length l by observing the effect of the different values of l on the system's performance. Procedure 5.3 illustrates the steps of generating the sliding windows. To this end, we address the following challenges:

• *Why does the system use an event count-based window?* The segmentation of the event stream can be done according to the number of events or time interval of the window. The system generates events when the inhabitant performs ADL; therefore, it is quite possible that some time intervals do not have any event. Thus time based

Procedure 5.3: Generating sliding windows

Input: $\mathbf{e} = \{e_1, e_2, \cdots, e_i, \cdots\}^T$, $Accuracy = 0$;
Output: $\mathbf{W} = \{W_1, W_2, \cdots, W_j, \cdots\}^T$, l ;

```
l_min ← min{ΔE_1, ··· , ΔE_i, ··· , ΔE_n};
l_max ← median{ΔE_1, ··· , ΔE_n};
for i ← l_min to l_max do
    for j ← 1 to z − i + 1 do
        Window W_j ← {e_j, e_{j+1}, ··· , e_{j+i−1}};
    W' ← {W_1, W_2, ··· , W_j, ··· }^T;
    The accuracy of system using W' is Acc_i;
    if (Acc_i > Accuracy) then
        Accuracy ← Acc_i, W ← W', and l ← i;
return W and l;
```

segmentation is not suitable.

• *Does the size of the window affect the time complexity of the system?* The time complexity of the system increases with the size of the window. However, a small window gives less accuracy because it provides less context information to classify the window's last event correctly.

• *Does sliding of window per sensor event really require?* Sliding the window per sensor event generates a large number of windows, which helps increase the dataset's size.

5.3.2 Generating hand-crafted features

The system uses Hand-Crafted Features (HCFs) as a complementary source of information along with High-Level Features (HLFs). A fixed dimensional feature vector of HCFs is extracted from each sliding window and concatenated to HLFs, as shown in Figure 5.3. The procedure for extracting of HCFs from each sliding window is as follows:

• **Boundary events of the sliding window:** The system uses the first and last events of each sliding window to estimates the HCFs, which answer the following questions: when is the activity done? And who sensed the activity? Such events consists of *cyclical* time attribute and *categorical data*. To preserve the cyclical nature of time,

Procedure 5.4: Generating hand-crafted features

Input: $\mathbf{W} = \{W_1, W_2, \cdots, W_j, \cdots\}$, l ;
Output: HCFs of $\mathbf{W}$;
1 **for** $j \leftarrow W_1$ *to* $W_{\lfloor \frac{Z}{l} \rfloor}$ **do**
2 /* **HCFs of boundary events of a window** W_j*/
3 Initialize HCF_{W_j} to the empty list;
4 First event of W_j is $< e_j : d_j, t_j, s_k, v_j >$;
5 Last event of W_j is $< e_{j+l-1} : d_{j+l-1}, t_{j+l-1}, s_y, v_{j+l-1} >$;
6 **Add** cyclical encoding of time t_j and t_{j+l-1} to HCF_{W_j}, using Definition 5.2:
$t_j : (t_j)_{\sin}, (t_j)_{\cos}$ $t_{j+l-1} : (t_{j+l-1})_{\sin}, (t_{j+l-1})_{\cos}$;
7 **Add** representation of s_k and s_y to HCF_{W_j}, as per Definition 5.3:
$s_k : \{\underbrace{0, \cdots, 0}_{k-1}, 1, \underbrace{0, \cdots, 0}_{m-k}\}$ $s_y : \{\underbrace{0, \cdots, 0}_{y-1}, 1, \underbrace{0, \cdots, 0}_{m-y}\}$;
8 **Add** $(t_{j+l-1} - t_j)$ to HCF_{W_j};
9 **Add** $Day(d_{j+l-1})$ to HCF_{W_j};
10 /* **HCFs of intermediate events of a window** W_j */
11 Initialize $c_f^j \leftarrow 0$, where $c_f^j \in \mathbf{c^j}$ and $1 \leq f \leq k$;
12 **for** $f \leftarrow 1$ *to* k **do**
13 **for** $g \leftarrow 1$ *to* l **do**
14 **if** $(< e_{j+g-1} : ., s_f, . >)$ **then**
15 $c_f^j \leftarrow c_f^j + 1$;
16 **Add** HCF $(\mathbf{D}[y] \odot \mathbf{c^j})$ to HCF_{W_j};
17 **Add** HCF_{W_j} to All_HCFs;
18 **return** All_HCFs;

the system uses Definition 5.2 and creates HCFs for the time attribute of the first and last events. The system uses One-Hot Encoding as given in Definition 5.3 to convert the sensor identification into a categorical form. The complete procedure is illustrated in steps 5-10 of Procedure 5.4.

• **Intermediate events of the sliding window:** A user in the system continuously performs the activities; therefore, sensor events in a given sliding window may correspond to an activity transition and may not be related to the last sensor event under consideration. For example, the inhabitant finishes the "Personal Hygiene" activity and begins the "Meal preparation" activity. The authors in [33] tackled this problem by defining a weighting scheme using mutual dependency between sensors. The mutual dependence between sensors helps reduce the influence of sensor events from different

functional areas on the HCFs of the sliding window. We count the generated events by each sensor in a given window W_i, *i.e.,* $\mathbf{c^i} = \{c_1^i, \cdots, c_j^i, \cdots, c_k^i\}$, where c_j^i denotes the number of events generated by sensor $s_j \in \mathcal{S}$ in window W_i. Let the last event of W_i is generated by s_y sensor, where $1 \leq y \leq n$. The HCFs of the intermediate events of W_i are computed by $\mathbf{D}[y] \odot \mathbf{c^i}$, where operation $\odot$ denotes the element-wise product and mutual dependency matrix $\mathbf{D}$ is given in Definition 5.1. For k different sensors, the dimension of the HCF is $k + 8$. The HCF of a window W_i, denoted by HCF_{W_i}, is marked with the label of y_i of the last sensor event. Each label y_i depicts an activity class $a_i \in \mathcal{A}$.

5.3.3 Generating high-level features

The ADLR system uses four deep neural networks for extracting high-level features from sliding windows. The system employs CNN to capture the spatial relationship and RNN (LSTM and GRU) to model feature dynamics. A hybrid architecture is also used to combine the goodness of CNN and LSTM. This section briefly discusses CNN, LSTM, GRU, CNN_LSTM architectures.

5.3.3.1 Convolution Neural Network (CNN)

CNN learns spatial features by alternating and stacking convolutional layers, activation layer, and pooling operation. Convolution is the first layer of CNN, which takes a sequence of events, called an input map, and a filter as inputs and generates a feature map. The filter in convolution layers convolves over the input sequence and computes the dot product. Next, the activation layer in CNN increases the non-linearity of the network without affecting the receptive fields of the previous layer. Finally, a pooling layer decreases the dimensionality of the feature map but holds essential information. It joins the outputs of neuron clusters at one layer into a single neuron in the next layer.

• **CNN in ADLR system:** CNN in ADLR system receives sliding windows as input (I), and each sliding window is a sequence data of length l, where $\mathbf{I} = \{e_1, \cdots, e_t, \cdots, e_l\}$, where $e_t \in \mathbb{R}^m$ at each timestamp. The system uses one dimensional (1D) CNN and 32 convolutional filters $\mathbf{L}_f$ of size j, where $0 < j < l$ and $1 \leq f \leq 32$. In this paper, for simplicity, we present the CNN network with one filter. The t^{th} element of feature map is the dot product of events $\{e_t, e_{t+1}, \cdots, e_{t+j-1}\}$ and filter $\mathbf{L}_f$, *i.e.*,

$$C_t^f = \sum_{g=1}^{j} L_f(g,:) I^T(:, t+g-1). \tag{5.2}$$

The activation function and bias of the network are denoted by $f_a(.)$ and b_a, respectively. The system uses a Rectified Linear Unit (ReLU) activation function because it allows the system to learn faster and perform better. The t^{th} element of feature vector after the activation function is given by

$$\begin{aligned} \widetilde{C_t^f} = f_a(C_t^f + b_a) &= \begin{cases} 0 & \text{for}(C_t^f + b_a) \leq 0 \\ C_t^f + b_a & \text{for}(C_t^f + b_a) > 0 \end{cases} \\ &= \mathbb{I}((C_t^f + b_a) > 0)(C_t^f + b_a), \\ &= CNN(e_t). \end{aligned} \tag{5.3}$$

The feature vector of window W_i of l events for $\mathbf{L}_f$ filter is given by

$$\widetilde{\mathbf{C}}_{\mathbf{i}}^{\mathbf{f}} = \{\widetilde{C_1^f}, \widetilde{C_2^f}, \cdots, \widetilde{C}_{l-j+1}^f\}. \tag{5.4}$$

The pooling layer can reduce the length of the feature map, which can further minimize the number of model parameters. We do not use the pooling layer because the sliding window mechanism constrains the input of the system, and this fact limits the possibility of downsampling the data [116].

5.3.3.2 Long Short-Term Memory (LSTM)

The LSTM [113] incorporates memory cells with multiple gates governing information into and out of the cell. The LSTM at each time-step receives a new input and produces an output based on the current input and previous output. The memory cell and gates enable it to memorize the long-period interdependencies in sequential data without suffering from the *vanishing gradients* effect [117]. The temporal dependency learning can be conveniently converted to the spatial domain. The LSTM consists of input, forget, and output gates to answer the following questions: what information is going to store in the memory cell?, what information to throw away from the memory cell?, and what part of the memory cell to output?, respectively.

• **LSTM in ADLR system:** Similar to CNN, input of LSTM is a sequence of sensor events $\mathbf{I} = \{e_1, \cdots, e_t, \cdots, e_l\}$, where $e_t \in \mathbb{R}^m$ at each timestamp. We can utilize the LSTM network to learn the sequence of motion states $\mathbf{h_t} \in \mathbb{R}^m$ to recognize ADL. The LSTM units in the system are updated as

$$\mathbf{i_t} = \sigma(\mathbf{M_{xi}} e_t + \mathbf{M_{hi}} \mathbf{h_{t-1}} + \mathbf{b_i}), \tag{5.5}$$

$$\mathbf{f_t} = \sigma(\mathbf{M_{xf}} e_t + \mathbf{M_{hf}} \mathbf{h_{t-1}} + \mathbf{b_f}), \tag{5.6}$$

$$\mathbf{o_t} = \sigma(\mathbf{M_{xo}} e_t + \mathbf{M_{ho}} \mathbf{h_{t-1}} + \mathbf{b_o}), \tag{5.7}$$

$$\mathbf{g_t} = \tanh(\mathbf{M_{xg}} e_t + \mathbf{M_{hg}} \mathbf{h_{t-1}} + \mathbf{b_g}), \tag{5.8}$$

$$\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \mathbf{g_t}, \tag{5.9}$$

$$\mathbf{h_t} = \mathbf{o_t} \odot \tanh(\mathbf{c_t}), \tag{5.10}$$

where $\mathbf{i_t}$, $\mathbf{f_t}$, $\mathbf{o_t}$, $\mathbf{g_t}$, and $\mathbf{c_t}$ are input gate, forget gate, output gate, input modulation gate, and memory cell, respectively. All gates and memory cells are of the same size as the hidden vector $\mathbf{h_t}$. $\{\mathbf{M_{xi}}, \mathbf{M_{hi}}, \mathbf{M_{xf}}, \mathbf{M_{hf}}, \mathbf{M_{xo}}, \mathbf{M_{ho}}, \mathbf{M_{xg}}, \mathbf{M_{hg}}\} \in \mathbb{R}^{2m}$ are weighted matrices. $\mathbf{b_i}$, $\mathbf{b_f}$, $\mathbf{b_o}$, and $\mathbf{b_g}$ are the biases of LSTM unit. σ is the logistic sigmoid function where $\sigma(x) = 1/(1 + e^{-x})$. The operation $\odot$ denotes the element-wise product

with the gate value. The update of each LSTM unit can be summarized as

$$\mathbf{h_t} = LSTM(\mathbf{h_{t-1}}, e_t, \omega_L), \tag{5.11}$$

where $LSTM(.)$ is a combination of Eqs. 6.1-6.6 and ω_L denotes all the parameters in the LSTM network.

5.3.3.3 Gated Recurrent Unit (GRU)

The GRU is a simplified version of LSTM, which has two gates (reset and update) that modulate the flow of information, however, without having a memory cell. The GRU has a hidden state and the candidate state. The use of fewer gates and parameters makes GRU a bit faster and computational lightweight than LSTM. The reset and update gates address the following questions: how much information to flush from the memory?, and how much information needs to be stored at the current time step for future computations?, respectively. The reset gate in the ADLR system is useful because significant discontinuities occur in the sequence of events generated by human activities.

• **GRU in ADLR system:** The input of GRU is a sequence of sensor events $\mathbf{I} = \{e_1, \cdots, e_t, \cdots, e_l\}$, where $e_t \in \mathbb{R}^m$ at each timestamp. The GRU in the system is updated as

$$\mathbf{r_t} = \sigma(\mathbf{M_{xr}} e_t + \mathbf{M_{hr}} \mathbf{h_{t-1}} + \mathbf{b_r}), \tag{5.12}$$

$$\mathbf{z_t} = \sigma(\mathbf{M_{xz}} e_t + \mathbf{M_{hz}} \mathbf{h_{t-1}} + \mathbf{b_z}), \tag{5.13}$$

$$\tilde{\mathbf{h_t}} = \tanh(\mathbf{M_{xh}} e_t + \mathbf{M_{hh}}(\mathbf{r_t} \odot \mathbf{h_{t-1}}) + \mathbf{b_h}), \tag{5.14}$$

$$\mathbf{h_t} = \mathbf{z_t} \odot \mathbf{h_{t-1}} + (1 - \mathbf{z_t}) \odot \tilde{\mathbf{h_t}}, \tag{5.15}$$

where $\mathbf{r_t}$, $\mathbf{z_t}$, and $\mathbf{h_t}$ are reset gate, update gate, and hidden vector, respectively. All gates and memory cells are the same size as the hidden vector $\mathbf{h_t}$. $\{\mathbf{M_{xr}}, \mathbf{M_{hr}}, \mathbf{M_{xz}}, \mathbf{M_{hz}},$

$\mathbf{M_{xh}}$, and $\mathbf{M_{hh}}\} \in \mathbb{R}^{2m}$ are weighted matrices. $\mathbf{b_r}$, $\mathbf{b_z}$, and $\mathbf{b_h}$ are the biases of the GRU unit. The update of each GRU can be summarized as

$$\mathbf{h_t} = GRU(\mathbf{h_{t-1}}, e_t, \omega_G), \tag{5.16}$$

where $GRU(.)$ is a combination of Eqs. 5.12-5.15 and ω_G denotes all the parameters in the GRU.

5.3.3.4 CNN_LSTM

CNN_LSTM is the hybrid model of convolutional neural networks and long short-term memory. It exploits the benefits of CNN and LSTM. It utilizes the ability of convolution layers to learn the internal representation of time-series data and LSTM layers to identify short-term and long-term dependencies.

• **CNN_LSTM in ADLR system:** In CNN_LSTM, the outcomes of the CNN are fed as the input to the LSTM. The inputs of CNN are the sliding windows and each sliding window is a sequence of l sensor events $\mathbf{I} = \{e_1, \cdots, e_t, \cdots, e_l\}$, where $e_t \in \mathbb{R}^m$ at each timestamp. CNN outputs the intermediate-level representation $\widetilde{\mathbf{C}}^{\mathbf{f}}_{\mathbf{i}}$ of window W_i as shown in Equation 5.4. In CNN_LSTM, the input of LSTM units are $\widetilde{\mathbf{C}}^{\mathbf{f}}_{\mathbf{i}}$ and it generates $\mathbf{h_t}$ as shown in Equation 5.11. The update of CNN_LSTM for a window W_i can be summarized as

$$\begin{aligned} \widetilde{C_t^f} &= CNN(e_t), \\ \mathbf{h_t} &= LSTM(\mathbf{h_{t-1}}, \widetilde{C_t^f}, \omega_{CL}). \end{aligned} \tag{5.17}$$

where $LSTM(.)$ is a combination of Eqs. 6.1-6.6 and ω_{CL} denotes all the parameters in the LSTM network.

5.3.4 Ensemble technique

Ensemble is a technique that combines the outputs of various candidate systems to get better results. It exploits the goodness of all the participating systems. The proposed ensemble model is a Multi-Layer Perceptron (MLP) built on top of learned high-level features of four deep learning models and hand-crafted features. We first separately train and tune all the four deep learning models. We extract intermediate layer activation (high-level features) from these trained models by removing their output layer. These four high-level features are concatenated with hand-crafted features (generated by Procedure 5.4), as shown in Figure 5.3. Procedure 5.5 illustrates the steps of the concatenation of the features. The ensemble model learns the mapping of concatenated features and output labels. It utilizes the richness of different features and learns a joined representation for activity recognition. The dimension of the concatenated features is $k + 408$, *i.e.*, 100 dimension high-level features from each deep learning model plus $k + 8$ dimension hand-crafted features.

Procedure 5.5: Concatenation of the features

Input: $I = \{(W_1, y_1), (W_2, y_2), \cdots, (W_{\lfloor \frac{z}{l} \rfloor}, y_{\lfloor \frac{z}{l} \rfloor})\}$;

Output: Concatenated features

1 Train CNN, GRU, LSTM, and CNN_LSTM base models on input I by using Eqs. 5.4, 5.11, 5.16, and 5.17, respectively;
2 Remove output layer of trained base models;
3 Concatenate the outputs of the base models (high-level features) and hand-crafted features;
4 **return** Concatenated features;

5.3.5 Objective loss formulation

To improve the recognition performance of the proposed ADLR system, we use focal loss and center loss jointly as the supervisory signals. The comparison of these losses with conventional softmax cross-entropy loss will be discussed in the experimental section.

5.3.5.1 Focal loss

The system uses Focal Loss (FL) [103], denoted by $\mathcal{L}_{FL}$, for addressing the long-tailed class distribution problem. The FL adds a modulating factor to the sigmoid cross-entropy loss to reduce the loss for well-classified examples and focus on difficult ones. Let the input sample x with class label $y \in C$ and predicted output from the classifier for all classes are $\mathbf{z} \in \{z_1, z_2, \cdots, z_C\}$. The $\mathcal{L}_{FL}$ of x can be written as $\mathcal{L}_{FL} = -\sum_{i=1}^{C}(1 - \sigma(z_i^x))^{\gamma} \log(\sigma(z_i^x))$. The system uses an α balanced variant of focal loss:

$$\mathcal{L}_{FL} = -\alpha \sum_{i=1}^{C}(1 - \sigma(z_i^x))^{\gamma} \log(\sigma(z_i^x)), \tag{5.18}$$

where γ and σ are the focusing parameter and softmax function, respectively [103].

5.3.5.2 Center loss

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

$$\mathcal{L}_{CL} = \frac{1}{2} \sum_{i=1}^{N} \|\hat{x}_i - c_{y_i}\|_2^2, \tag{5.19}$$

where N, $\hat{x}_i$, y_i, and c_{y_i} are the total training sample in the dataset, i^{th} deep feature, class label of $\hat{x}_i$, and class center of deep features of y_i, respectively.

• **Loss function of ADLR system:** The loss function of the proposed system is defined by combining $\mathcal{L}_{FL}$ and $\mathcal{L}_{CL}$ as:

$$\mathcal{L}_{ADLR} = \mathcal{L}_{FL} + \lambda \mathcal{L}_{CL}, \tag{5.20}$$

where hyperparameter λ balances the two-loss terms and $0 < \lambda \leq 1$. Substituting $\mathcal{L}_{FL}$ and $\mathcal{L}_{CL}$ from Eqs. 5.18 and 5.19, respectively,

$$\mathcal{L}_{ADLR} = -\alpha \sum_{i=1}^{C} (1 - \sigma(z_i^x))^{\gamma} \log(\sigma(z_i^x)) + \frac{\lambda}{2} \sum_{i=1}^{N} \|\hat{x}_i - c_{y_i}\|_2^2 . \tag{5.21}$$

We apply Procedure 5.6 to train the proposed ADLR system. The output of Procedure 5.6 is the trained network parameters, denoted by Θ, where the network consists of the least error. The learning rate of the network at iteration τ is denoted as μ^{τ}. c_k denotes the kth class center defined by the averaging over the features in the kth class. c_k^{τ} denotes the kth estimated class center at the τth iteration and hyperparameter α control the learning rate of the centers.

Procedure 5.6: Training procedure of ADLR system

Input: Training Data $\mathcal{D}$=$\{(x_1, y_1), (x_2, y_2), \cdots\}$, α, λ, μ^{τ};
Output: Trained network parameter Θ ;

while *not converge* **do**
- $\tau = \tau + 1$;
- Compute total loss $\mathcal{L}_{ADLR}$ using Equation 5.21;
- Compute backpropagation error for i as $\frac{\partial \mathcal{L}_{ADLR}}{\partial \hat{x}_i^{\tau}}$;
- Update parameters:
- $c_k^{\tau+1}$=$c_k^{\tau} - \alpha \Delta c_k^{\tau}$;
- $\Theta^{\tau+1}$=$\Theta^{\tau} - \mu^{\tau} \frac{\partial \mathcal{L}_{ADLR}}{\partial \Theta^{\tau}}$= $\Theta^{\tau} - \mu^{\tau} \sum_i \frac{\partial \mathcal{L}_{ADLR}}{\partial \hat{x}_i^{\tau}} \frac{\partial \hat{x}_i^{\tau}}{\partial \Theta^{\tau}}$;

return Θ;

5.4 Experiments and Results

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

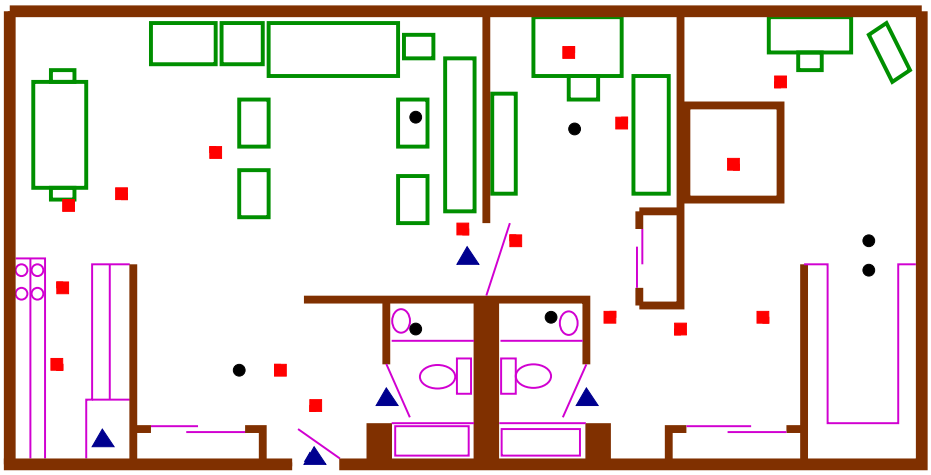


Figure 5.4: CASAS smart home floor plan and sensor layout for $\mathbf{D}_5$. Marks "■", "•", and "▲" are IR motion ($\mathbf{S}_1$), wide-area IR motion ($\mathbf{S}_2$), and magnetic sensors ($\mathbf{S}_3$), respectively.

5.4.1 Datasets

We used publicly available five real datasets collected in CASAS smart home testbeds at Washington State University [31,41,44], referred to as $\mathbf{D}_i$ datasets, where $1 \leq i \leq 5$. Figure 5.4 illustrates the layout of the testbed for $\mathbf{D}_5$. CASAS smart home testbeds use infrared motion sensors, wide-area infrared motion sensors, and magnetic sensors, which are denoted by $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_3$, respectively. The CASAS smart home datasets consist of a set of eleven human activities $\mathcal{A} \in$ {Bathing, Bed toilet transition, Eating, Enter home, Housekeeping, Leave home, Meal preparation, Personal hygiene, Resting on couch, Sleeping in bed, Taking medicine}, which are denoted as $\{a_1, \cdots, a_{11}\}$ [31]. In a real-world setting, there may be sensor events that do not belong to a predefined class, such as events generated due to transitions between different activities or corresponds to some other well-defined activities. We assign them to an "Other" category, denoted by a_{12}. The smart home residents were older adults who were performing normal unscripted activities of daily life. Table 5.1 shows that the percentage of sensor events belongs to activities $\mathcal{A}$ in each dataset. The following points illustrate the motivation of the selection of the datasets. Class imbalance ratios (the ratio between the sample

Table 5.1: The summary of CASAS smart home datasets.

		$\mathbf{D}_1$	$\mathbf{D}_2$	$\mathbf{D}_3$	$\mathbf{D}_4$	$\mathbf{D}_5$
		(s_1,s_2,s_3)	(s_1,s_2,s_3)	(s_1,s_2,s_3)	(s_1,s_2,s_3)	(s_1,s_2,s_3)
Activities $\mathcal{A}$	a_1	(0,2,0), 2.1%	(0,1,0), 0.06%	(0,1,0), 0.12%	(1,1,0), 0.67%	(4,4,0), 1.19%
	a_2	(0,2,0), 2.9%	(0,2,0), 0.25%	(1,1,0), 5.07%	(0,1,0), 0.41%	(3,3,0), 0.63%
	a_3	(1,3,0), 1.63%	(3,8,0), 1.47%	(2,7,0), 0.79%	(5,1,0), 0.44%	(8,3,1), 2.08%
	a_4	(2,2,1), 0.87%	(2,5,1), 0.38%	(2,3,1), 0.89%	(6,3,1), 1.38%	(5,2,1), 0.59%
	a_5	(1,2,0), 5.32%	(4,6,1), 14.0%	(2,4,1), 3.80%	(7,3,0), 14.7%	(16,7,4), 20%
	a_6	(2,1,1), **0.80%**	(2,4,1), 0.42%	(2,3,1), 1.17%	(8,3,1), 1.48%	(2,1,1), 0.60%
	a_7	(1,2,0), 26.8%	(4,5,0), 23.9%	(5,4,2), 18.0%	(6,3,0), 24.5%	(11,4,3), 14.5%
	a_8	(2,5,0), 19.5%	(2,8,1), 11.5%	(2,7,0), 32.1%	(6,4,0), 21%	(15,7,0),25.9%
	a_9	(3,7,3), 2.5%	(4,9,1), 2.5%	(1,5,0), 1.38%	(9,5,0), 3.6%	(16,7,1), 5.6%
	a_{10}	(3,6,0), 5.01%	(2,9,1), 3.9%	(3,6,0),1.33%	(10,4,0), 1.0%	(16,6,0), 1.9%
	a_{11}	(0,2,0), 1.1%	(3,1,0), 2.1%	(0,1,0), 1.02%	(0,1,0), 1.11%	(4,4,1), 0.4%
	a_{12}	(3,7,3), 31.2%	(4,9,3), 39.2%	(5,9,3), 34.2%	(17,7,3), 29.3%	(16,7,4), 26.3%
Regions	R1	√	√	√	√	√
	R2	√	√	√	√	√
	R3	√	√	√	√	√
	R4			√		√
	R5	√	√	√	√	√
	R6	√				√
Sensors	$\mathbf{S}_1$	3	4	5	17	16
	$\mathbf{S}_2$	7	9	9	7	7
	$\mathbf{S}_3$	3	3	3	3	5

size of smallest and largest classes) are $1:39$, $1:655$, $1:275$, $1:71$, and $1:64$ in datasets $\mathbf{D}_1$ to $\mathbf{D}_5$, respectively. Next, the different layout of the sensors deployed in the apartment incorporates the robustness in the experiment results. Finally, we emphasize the number of different types of sensors used in the apartment. Table 5.1 illustrates that $\mathbf{D}_1$ to $\mathbf{D}_5$ datasets used 13, 16, 17, 27, and 28 sensors (sum of $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_3$), respectively. It also shows the number of sensors for each activity. For example, activity a_1 is sensed by (0,2,0) sensors, *i.e.* two $\mathbf{S}_2$ sensors in $\mathbf{D}_1$ dataset.

5.4.2 Implementation details

The proposed models are trained for $\mathbf{D}_i$ datasets, $1 \leq i \leq 5$, using 80% of the data for training and 20% for testing. 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. We use the Adam optimization algorithm with a mini-batch size of 64. The initial learning rate is set to be 0.001. Then we decrease the learning rate by $\times 0.3$ when the accuracy on the training data stops improving [118]. The proposed model converged within about 200 epochs. Each experiment is repeated ten times and took the average. We use Python-based libraries, Keras and Scikit-learn, for implementation. The experiments are performed on a PC with Intel Core *i*7-CPU, 2.83 GHz clock speed, 6 GB RAM, and Ubuntu 18.04 operating system.

5.4.3 Performance metrics

We consider the following performance metrics for estimating the performance of the proposed system.

• **Overall activity classification accuracy:** It is defined as

$$\mathbf{P}_1 = \frac{1}{|\mathcal{A}|}\sum_{i=1}^{|\mathcal{A}|} Acc_i = \frac{1}{|\mathcal{A}|}\sum_{i=1}^{|\mathcal{A}|} \frac{TP_i}{TP_i + FN_i}. \tag{5.22}$$

where $\mathcal{A}$ is the set of all activity classes including the "Other" a_{12} class and TP_i, FP_i, FN_i, and Acc_i are the true positive, false positive, false negative window counts, and classification accuracy of activity $a_i \in \mathcal{A}$, respectively. Moreover, the average overall activity classification accuracy of all the datasets ($\mathbf{D_i}$, $1 \leq i \leq 5$) is given by

$$\mathbf{Q} = \frac{1}{5}\sum_{i=1}^{5} \mathbf{P_1} \text{ of } \mathbf{D_i}. \tag{5.23}$$

• **Predefined activity classification accuracy:** It is computed as

$$\mathbf{P}_2 = \frac{1}{|\mathcal{A}'|}\sum_{i=1}^{|\mathcal{A}'|} Acc_i = \frac{1}{|\mathcal{A}'|}\sum_{i=1}^{|\mathcal{A}'|} \frac{TP_i}{TP_i + FN_i}. \tag{5.24}$$

where $\mathcal{A}' = \mathcal{A} - a_{12}$ is the set of predefined activity classes *i.e.*, set of all activities excluding the "Other" a_{12} class. Note that this activity classification accuracy differs from that employed in [33], where $\mathbf{P}_2 = \sum_{i=1}^{|\mathcal{A}'|} TP_i / \sum_{i=1}^{|\mathcal{A}'|} (TP_i + FN_i)$, which can be

biased to the majority class and therefore not suitable for the class imbalanced datasets.

• **F1-score:** The $F1$ score of the activity set $\mathcal{A}$ is computed as

$$\mathbf{P}_3 = \frac{1}{|\mathcal{A}|}\sum_{i=1}^{|\mathcal{A}|}\frac{2 \times TP_i}{TP_i + FP_i + FN_i}. \tag{5.25}$$

5.4.4 Models for ADLR system

We present the following models for the ablation studies of the proposed work.

5.4.4.1 Baseline models ($\mathbf{M_1}, \mathbf{M_2}$)

We consider two baseline models [31] to compare the performance of our proposed models. The model $\mathbf{M_1}$ uses Baseline Features (BFs) as inputs and SVM as activity classifier. The model $\mathbf{M_2}$ is the extension of the model $\mathbf{M_1}$ where the inputs of the SVM are Baseline features with Mutual Information (BM) as given in Definition 5.1. Such BM features are generated using Procedure 5.4 in Section 5.3.2. In BF computation, a simple count of different sensor events is used, whereas, in BM, it is the sum of every sensor event's contributions weighted by mutual dependency.

5.4.4.2 Deep learning models ($\mathbf{M_3} - \mathbf{M_5}$)

The models $\mathbf{M_3}, \mathbf{M_4}$, and $\mathbf{M_5}$ use CNN, LSTM, and GRU, respectively, to recognize the human activities. The input of these models is a window $\mathbf{W}^{90\times5}$ of the sequence of 90 sensor events. Each event is represented by a tuple $<$*day of the event,* $time_{sin}$, $time_{cos}$, *sensorID, sensor value*$>$, instead of using raw streaming sensor events to gain a better experimental result. Here time_{sin} and time_{cos} are the sine and cosine transform of the time of the event, defined in Definition 5.2. The window $\mathbf{W}$ is generated using Procedure 5.3 in Section 5.3.1.

5.4.4.3 Hybrid model ($\mathbf{M_6}$)

Model $\mathbf{M_6}$ is the combination of CNN and LSTM. It takes window $\mathbf{W}^{90\times5}$ of the sequence of 90 sensor events as input, first pass it through the CNN, and then LSTM.

5.4.4.4 Ensemble models ($\mathbf{M_7} - \mathbf{M_{12}}$)

The ensemble technique creates multiple models and then synthesizes the predictions from these models to produce improved results. Model $\mathbf{M_7}$ uses a separately trained and tuned $\mathbf{M_3}$ model to obtain deep features by removing its output layer. These deep features are concatenated with hand-crafted BM features before feeding into the ensemble model for recognizing human activities. Similarly, models $\mathbf{M_8}$, $\mathbf{M_9}$, and $\mathbf{M_{10}}$ use trained models $\mathbf{M_4}$, $\mathbf{M_5}$, and $\mathbf{M_6}$, respectively, and hand-crafted BM features. Models $\mathbf{M_{11}}$ and $\mathbf{M_{12}}$ use all four deep learning models for deep feature extraction. Model $\mathbf{M_{11}}$ uses separately trained $\mathbf{M_3}$, $\mathbf{M_4}$, $\mathbf{M_5}$, and $\mathbf{M_6}$ models, and the intermediate learned representations are used for the final ensemble. Finally, Model $\mathbf{M_{12}}$ uses all the four base models same as $\mathbf{M_{11}}$, and hand-crafted BM features. The input of all these models is the window $\mathbf{W}^{90\times5}$. The proposed ADLR system uses $\mathbf{M_{12}}$ since it provides the highest accuracy.

5.4.4.5 Ensemble End-to-End model ($\mathbf{M_{13}}$)

$\mathbf{M_{13}}$ is same as $\mathbf{M_{12}}$ except that here all four base models are trained in a unified architecture along with the ensemble model. Model $\mathbf{M_{13}}$ uses the window $\mathbf{W}^{90\times5}$ as input, ensembles CNN, $LSTM$, GRU, CNN_LSTM and BM, and trains the entire model to recognize the human activities.

Table 5.2 summarizes the architecture of $\mathbf{M_3}$ to $\mathbf{M_{13}}$ models. The symbol A is the dimension of the input sliding window, and D is the number of units for the output layer. The $B : [B_1, B_2, \cdots]$ and $C : [C_1, C_2, \cdots]$ indicate the convolution/recurrent layers and dense layers, respectively, where B_i and C_i are the dimension at the ith

Table 5.2: Comparison of the architectures of different models.

Model	**Architectures** $A \xrightarrow{a} \underbrace{[B_1, B_2, \cdots]}_{B} \xrightarrow{b} \underbrace{[C_1, C_2, \cdots]}_{C} \xrightarrow{c} D$
$\mathbf{M}_3$	$90 \times 5 \xrightarrow{a} [32, 32, 32, 32, 32, 32, 32, 32] \xrightarrow{b} 100 \xrightarrow{c} 12$
$\mathbf{M}_4$	$90 \times 5 \xrightarrow{a} [100, 100] \xrightarrow{b} 100 \xrightarrow{c} 12$
$\mathbf{M}_5$	$90 \times 5 \xrightarrow{a} [100, 100] \xrightarrow{b} 100 \xrightarrow{c} 12$
$\mathbf{M}_6$	$90 \times 5 \xrightarrow{a} [32, 32, 32, 32, 100, 100] \xrightarrow{b} 100 \xrightarrow{c} 12$
$\mathbf{M}_7$	$90 \times 5 \xrightarrow{a} \begin{bmatrix} \text{Model X} \\ \mathbf{BM} \end{bmatrix} \xrightarrow{b} \big[200, 100\big] \xrightarrow{c} 12$
$\mathbf{M}_8$	Where Model X= Separately trained $\mathbf{M}_3$, $\mathbf{M}_4$, $\mathbf{M}_5$, and $\mathbf{M}_6$ models for $\mathbf{M}_7$, $\mathbf{M}_8$, $\mathbf{M}_9$, and $\mathbf{M}_{10}$, respectively.
$\mathbf{M}_9$	
$\mathbf{M}_{10}$	
$\mathbf{M}_{11}$	$90 \times 5 \xrightarrow{a} \begin{bmatrix} \mathbf{M}_3 \\ \mathbf{M}_4 \\ \mathbf{M}_5 \\ \mathbf{M}_6 \end{bmatrix} \xrightarrow{b} \big[200, 100\big] \xrightarrow{c} 12$
$\mathbf{M}_{12}$	$90 \times 5 \xrightarrow{a} \begin{bmatrix} \mathbf{M}_3 \\ \mathbf{M}_4 \\ \mathbf{M}_5 \\ \mathbf{M}_6 \\ \mathbf{BM} \end{bmatrix} \xrightarrow{b} \big[200, 100\big] \xrightarrow{c} 12$
$\mathbf{M}_{13}$	Same as $\mathbf{M}_{12}$, except that here all base models are trained in a unified architecture along with the ensemble model.

convolutional/recurrent and dense layer, respectively.

5.4.5 Results and discussion

This section carries out the experimental evaluation to validate the performance of the ADLR system by giving answers to the following questions:

- *What is the influence of the input window size and hyperparameters of the loss functions on the activity recognition accuracy?* The answer to this question gives the optimal window size and hyperparameters for the system's highest accuracy.
- *What is the impact of the loss functions on the activity recognition accuracy?*

The result identifies a loss function that provides the highest activity recognition accuracy by using the optimal hyperparameters.

- *What is the benefit of the inclusion of hand-crafted features in the ensemble architecture on the activity recognition accuracy?* The answer to this question infers that the inclusion of hand-crafted features along with deep features in the ensemble architecture results in an improvement of the activity recognition accuracy.
- *How efficiently can the ADLR system classify human activities compared to baseline models?* The answer to this question illustrates the accuracy of the ADLR system by using optimal hyperparameters and loss function. It shows that $\mathbf{M_{12}}$ gives the highest accuracy.
- *Does the traditional ensemble methods help to improve activity recognition accuracy?* The answer to this question concludes that the proposed ensemble achieves much better performance than any of the traditional ensemble methods.
- *What are the time and space requirements of the ADLR system?* The answer to this question illustrates the required time and space of a given model for recognizing the activities.
- *Does the ADLR system take care of under-represented activity classes?* The answer to this question concludes that the proposed system successfully recognized the given activity classes, including under-represented classes with high accuracy.

5.4.5.1 Impact of the hyperparameter selection

We discuss the selections of two critical hyperparameters, input window size, and λ. Here λ is the ratio between focal loss and center loss as menitoned in Equation 5.20. The hyperparameters α and γ of FL are set 0.25 and 2, respectively, following [106].

The performance of the ADLR varies based on the window size. To determine the optimal window size, we change the size and observe the effects on the accuracy of the system. These tests are performed using $\mathbf{D}_1$ to $\mathbf{D}_5$ datasets. The search space

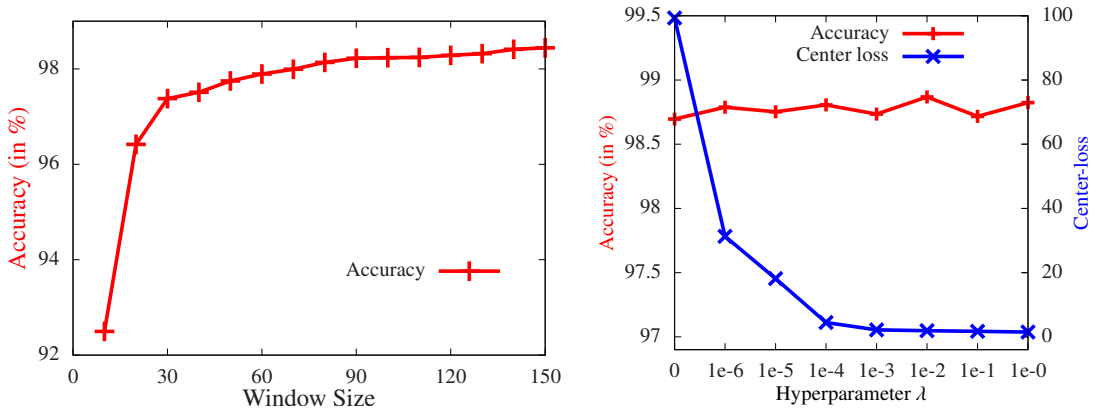
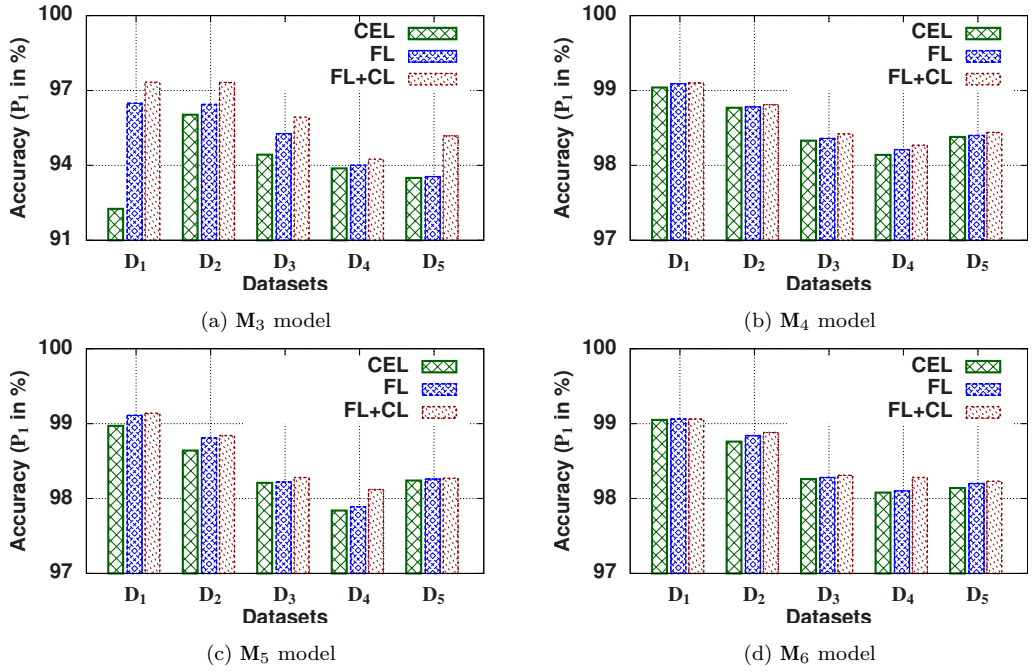


Figure 5.5: Illustration of the impact of window size (part (a)) and λ (part (b)) on the performance of the ADLR system.

of window size is $\{10, 20, \cdots, 150\}$. Part (a) of Figure 5.5 illustrates the impact of the window size on the average performance of the ADLR system (aggregated across all datasets). It shows that the performance increases as window size increases until performance plateaus. After that, the addition of more sensor events does not enhance performance much. According to these findings, we consider a window size of 90 for the experimental results. This window size provides most of the advantages obtained from a broader window size. Increasing the window size further may give a small benefit; however, the complexity rises as the window size increases.

Next, we study the impact of the λ and employ the weight set $\lambda \in \{0, 10^{-6}, 10^{-5}, \cdots, 10^{-1}, 1\}$, which implies the model varying gradually from the model with only FL to the case where the FL and CL ratio is equal to 1. Experiments use the highest imbalanced $\mathbf{D}_2$ dataset. Part (b) of Figure 5.5 demonstrates the impact of the λ on the recognition accuracy and the center loss. We observed that for $\lambda = 0$, *i.e.*, the model with only FL yields a significant value of the center loss. Also, with increasing the value of λ, the center loss decreases, while the recognition accuracy does not change significantly. Considering these observations, we choose $\lambda = 0.01$ in the experimental result.



(a) $\mathbf{M}_3$ model (b) $\mathbf{M}_4$ model

(c) $\mathbf{M}_5$ model (d) $\mathbf{M}_6$ model

Figure 5.6: Performance comparison among three loss functions on $\mathbf{D}_1$-$\mathbf{D}_5$ datasets, where $\mathbf{P}_1$ denotes overall activity classification accuracy.

5.4.5.2 Impact of loss function

Next, we illustrate the impact of the loss function on the performance of $\mathbf{M_3}$, $\mathbf{M_4}$, $\mathbf{M}_5$, and $\mathbf{M}_6$ models in terms of activity recognition accuracy. We present results of these models when supervised by conventional softmax Cross-Entropy Loss (CEL), FL, and a joint loss FL+CL that combines FL and CL. Figure 5.6 illustrates the following observations: (i) The CEL leads to the worst results on all the data sets and models since our datasets are highly imbalanced and cannot be exploited well by deep learning models with CEL. We used the FL to train the proposed deep models. FL employs a modulating factor to the CEL to help focus on hard samples and down-weight the many easy ones. The FL gives better results than the CEL on all the data sets. (ii) Joint loss FL+CL further improves the recognition performance and generalization of the proposed models and provides the highest recognition accuracy on all the datasets. This is because the discrimination ability caused by the CL has a positive impact on recognition

performance. CL adds a cluster-based loss term to the FL, 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. Joint loss FL+CL gives the best recognition accuracy in terms of all the metrics.

Table 5.3: Performance comparison among $\mathbf{M}_1$-$\mathbf{M}_{13}$ models for ADL recognition on $\mathbf{D}_1$-$\mathbf{D}_5$ datasets, where $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$, and $\mathbf{Q}$ denote overall activity classification accuracy, predefined activity classification accuracy, F1-score, and average $\mathbf{P}_1$ of all the datasets, respectively.

Models	Performance															
	$\mathbf{D}_1$ Dataset			$\mathbf{D}_2$ Dataset			$\mathbf{D}_3$ Dataset			$\mathbf{D}_4$ Dataset			$\mathbf{D}_5$ Dataset			$\mathbf{Q}$
	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	
Part (a): Baseline models																
$\mathbf{M}_1$	77.62	76.23	55.02	68.33	57.15	33.22	70.96	71.89	28.68	66.11	63.65	30.61	62.29	63.32	37.60	69.06
$\mathbf{M}_2$	84.82	80.93	60.48	72.42	60.68	38.52	80.12	81.19	34.13	70.33	66.43	35.28	62.51	57.13	43.01	74.04
Part (b): Deep learning models																
$\mathbf{M}_3$	97.33	97.19	94.28	97.32	96.33	82.54	95.93	95.10	79.43	94.25	94.26	85.16	95.18	95.75	89.20	96.00
$\mathbf{M}_4$	99.10	99.16	98.50	98.81	98.85	97.18	98.42	98.26	93.90	98.27	98.60	96.68	98.44	98.84	96.91	98.61
$\mathbf{M}_5$	99.14	99.20	98.53	98.84	98.75	96.15	98.28	98.09	92.80	98.12	98.38	96.19	98.27	98.69	96.49	98.53
$\mathbf{M}_6$	99.06	99.10	98.41	98.88	98.87	96.12	98.31	98.04	93.03	98.28	98.65	96.40	98.23	98.59	96.82	98.55
Part (c): Ensemble models (Single pre-trained deep learning model with BM features)																
$\mathbf{M}_7$	97.36	97.20	94.62	97.56	96.18	85.48	96.17	95.56	81.44	94.29	94.31	84.61	95.18	95.94	89.42	96.11
$\mathbf{M}_8$	99.11	99.02	98.53	98.81	98.73	97.30	98.42	98.30	93.75	98.31	98.64	96.57	98.46	98.79	96.79	98.62
$\mathbf{M}_9$	99.16	99.11	98.59	98.87	98.70	96.26	98.34	98.19	92.10	98.12	98.41	96.15	98.28	98.71	96.84	98.55
$\mathbf{M}_{10}$	99.07	99.14	98.43	98.89	98.75	96.66	98.33	98.16	93.40	98.28	98.34	96.42	98.24	98.64	96.93	98.56
Part (d): Ensemble All together models (All 4 pre-trained deep learning models without and with BM features)																
$\mathbf{M}_{11}$	99.20	99.12	98.65	99.16	99.08	97.32	98.64	98.47	91.57	98.35	98.80	96.54	98.64	98.96	97.29	98.80
$\mathbf{M}_{12}$	99.21	99.23	98.70	99.15	99.15	97.85	98.63	98.43	91.75	98.60	98.88	97.25	98.64	98.98	97.46	98.85
Part (e): Ensemble End-to-End model																
$\mathbf{M}_{13}$	99.20	99.14	98.68	99.08	99.17	97.71	98.58	98.57	93.00	98.28	98.61	96.77	98.47	98.82	97.45	98.72

5.4.5.3 Performance comparison among $\mathbf{M}_1$-$\mathbf{M}_{13}$ models

In this study, we compared the performance of $\mathbf{M}_1$-$\mathbf{M}_{13}$ models. We consider the performance metrics $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_3$ as given in Section 5.4.3, to measure the performance of the models, and results were shown in Table 5.3. The previous result illustrates that the joint loss FL+CL function addresses the class imbalance problem, interclass compactness, and interclass dispersion and provides the highest accuracy. Therefore, we use this loss function in all the models.

• *Observations for* $\mathbf{M}_1$ *and* $\mathbf{M}_2$: Table 5.3 shows that $\mathbf{M}_2$ outperforms $\mathbf{M}_1$ in terms of all performance metrics. For example, the improvement in $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_3$ are 7.2%,

4.7%, and 5.5%, respectively, for $\mathbf{D}_1$ dataset. The reason is that $\mathbf{M}_2$ uses the mutual information, which reduces the weight of sensor events from different functional areas on the features defining the last sensor event within a sliding window. The mutual information accurately detects the correlation among the events in a sliding window.

• *Observations for* $\mathbf{M}_3$-$\mathbf{M}_6$: The performance of CNN, LSTM, GRU, and CNN_LSTM based $\mathbf{M}_3$, $\mathbf{M}_4$, $\mathbf{M}_5$, and $\mathbf{M}_6$ models, respectively, are illustrated in Table 5.3. It shows that these deep learning models provide much better accuracy than the baseline hand-engineered feature-based $\mathbf{M}_1$ and $\mathbf{M}_2$ models, where features are manually extracted using domain knowledge. The reasons are as follows: (i) $\mathbf{M}_3$-$\mathbf{M}_6$, deep learning models, use the non-linear correlations among the events during the recognition of activity and can learn much more high-level and meaningful features automatically through the network. (ii) Through human domain knowledge, only shallow features can be learned [64].

• *Observations for* $\mathbf{M}_7$-$\mathbf{M}_{10}$: Next, we illustrate the benefit of the inclusion of hand-crafted features in the ensemble architecture on the accuracy of the ADLR system. Table 5.3 illustrates the performance of $\mathbf{M}_7$-$\mathbf{M}_{10}$ models which use separately trained $\mathbf{M}_3$-$\mathbf{M}_6$ models, respectively, to obtain deep features by removing their output layer. These deep features are concatenated with hand-crafted BM features before feeding into the ensemble model for recognizing human activities. The results illustrate that the inclusion of hand-crafted features results in an improvement of performance. This is because these ensemble models use hand-crafted BM features, which complement deep features and provide the additional domain knowledge of the dataset, therefore, enhancing the performance.

• *Observations for* $\mathbf{M}_{11}$ -$\mathbf{M}_{12}$: Next, deep learning models $\mathbf{M}_{11}$ and $\mathbf{M}_{12}$ use separately trained $\mathbf{M_3}$, $\mathbf{M_4}$, $\mathbf{M_5}$, and $\mathbf{M_6}$ models together as shown in Table 5.2. Table 5.3 illustrates that $\mathbf{M}_{11}$ model provides better accuracy then all previous models ($\mathbf{M}_1$ to $\mathbf{M}_{10}$). This is because it exploits the richness of different feature representations and the goodness

Table 5.4: Comparison with the traditional ensemble approaches using $\mathbf{M_{12}}$ model, where $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$, and $\mathbf{Q}$ denote overall activity classification accuracy, predefined activity classification accuracy, F1-score, and average $\mathbf{P}_1$ of all the datasets, respectively.

Ensemble	Performance															
	$\mathbf{D}_1$			$\mathbf{D}_2$			$\mathbf{D}_3$			$\mathbf{D}_4$			$\mathbf{D}_5$			$\mathbf{Q}$
	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	
Bagging	94.53	93.49	93.98	93.66	91.72	82.22	94.33	94.83	83.36	94.36	91.84	86.88	92.77	93.30	90.18	93.93
Random Forests	94.35	94.80	92.43	93.48	90.64	80.04	94.08	93.80	86.28	94.60	92.95	88.39	94.90	94.30	94.67	94.28
AdaBoost	56.78	39.03	12.52	51.31	59.25	20.01	53.88	74.50	14.59	49.33	52.87	18.46	20.01	23.05	08.37	46.26
Gradient Boosting	85.58	82.66	64.11	72.06	61.40	37.54	83.17	80.94	48.79	71.20	71.47	44.94	62.98	60.67	48.14	75.00
Extra-trees	94.09	94.69	91.56	93.42	91.65	85.62	94.34	93.21	86.72	92.10	91.85	89.72	92.54	93.00	92.35	93.30
Proposed	99.21	99.23	98.70	99.15	99.15	97.85	98.63	98.43	91.75	98.60	98.88	97.25	98.64	98.98	97.46	98.85

of CNN and RNN. In addition to $\mathbf{M}_{11}$ base models, $\mathbf{M}_{12}$ model additionally uses hand-crafted BM features, which provide more learning the correlation information among the events in a given input window and hence gives slightly more accuracy than $\mathbf{M}_{11}$ model in all the metrics.

• *Observations for* $\mathbf{M}_{13}$: Finally, we study our proposed $\mathbf{M}_{12}$ model in an end-to-end network in which all base models are trained in a unified architecture along with the ensemble model. Table 5.4 illustrates that the accuracy of $\mathbf{M}_{13}$ model is inferior to the $\mathbf{M}_{12}$ model. As per our understanding, since $\mathbf{M}_{13}$ end-to-end ensemble model has a relatively large set of parameters to train, and therefore, requires a large dataset. However, the size of our datasets is limited and not sufficient to optimize the parameters.

5.4.5.4 Comparison with the traditional ensemble methods

Experiments were further conducted to compare our proposed ensemble model with various traditional ensemble methods. The proposed $\mathbf{M}_{12}$ model uses a stacking ensemble technique for constructing an ensemble. The comparative results are shown in Table 5.4, where we consider five traditional ensemble methods Bagging, Random Forests, AdaBoost, Gradient Boosting, and Extra-trees for all the datasets ($\mathbf{D}_1$-$\mathbf{D}_5$). Table 5.4 shows that the proposed ensemble method gives the best performance than any other ensemble method on all performance metrics. This is because the proposed ensemble method exploits the richness of different feature representations (hand-crafted and high-

Table 5.5: Time and space requirements of models on $\mathbf{D_1}$ dataset.

Model	Memory	Train Time	Test Speed	Accuracy $\mathbf{P}_1$
$\mathbf{M}_3$	4.20 MB	1.61 h	0.0732 ms	97.33%
$\mathbf{M}_4$	1.70 MB	8.28 h	0.5272 ms	99.10%
$\mathbf{M}_5$	1.30 MB	8.22 h	0.5170 ms	99.14%
$\mathbf{M}_6$	2.00 MB	9.33 h	0.5641 ms	99.06%
$\mathbf{M}_{12}$	4.30 MB	15.39 h	1.0375 ms	99.21%
$\mathbf{M}_{13}$	10.30 MB	29.61 h	1.0375 ms	99.20%

level features) and learns a combined representation for solving the ADL recognition problem.

5.4.5.5 Time and space requirements of the models

Next, we illustrate the requirement of time and space for the different models. For experiments, we consider dataset $\mathbf{D}_1$ since it gives the highest accuracy. Table 5.5 shows the memory consumption, training time, testing time, and accuracy of different models. Training and testing time indicate the required time to train the model and test on a given sliding window, respectively. Models $\mathbf{M}_{12}$ and $\mathbf{M}_{13}$ uses separately trained base models ($\mathbf{M}_3$-$\mathbf{M}_6$). Therefore, the training time of these models is the sum of the required time to train the ensemble and the maximum required training time of base models ($\max\{\tau_i\}$), where τ_i is training time of $\mathbf{M}_i$ and $3 \leq i \leq 6$. As shown in Table 5.5, CNN ($\mathbf{M}_3$) and end-to-end model ($\mathbf{M}_{13}$) take the least and the most training and testing time, respectively, as expected. LSTM ($\mathbf{M}_4$) takes slightly more training and testing time than GRU ($\mathbf{M}_5$), and both have comparable accuracy. From the perspective of accuracy, $\mathbf{M}_{12}$ outperforms all other models.

5.4.5.6 Illustration of class-wise accuracy of ADLR system

Next, we illustrate the accuracy of the successful recognition of a given class by the ADLR system. Table 5.6 demonstrates the confusion matrix for $\mathbf{D_1}$ dataset. The large

Table 5.6: Confusion matrix for the ADLR system recognizes on $\mathbf{D_1}$ dataset. Each cell consists of $\overset{a}{\underset{(b)}{}}$, where a and b denote the recognized activity in number and percentage, respectively. The • indicates $a = 0$ and $b = 0$.

		Recognized Activities											
		a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
Actual Activities	a_1	487 (99.795)	•	•	•	•	•	•	•	•	•	•	1 (0.205)
	a_2	•	644 (100)	•	•	•	•	•	•	•	•	•	•
	a_3	•	•	362 (98.907)	•	•	•	•	•	•	•	•	4 (1.093)
	a_4	•	•	•	195 (100)	•	•	•	•	•	•	•	•
	a_5	•	•	2 (0.168)	•	1170 (98.319)	•	4 (0.336)	•	•	•	•	14 (1.200)
	a_6	•	•	•	3 (1.667)	•	177 (**98.333**)	•	•	•	•	•	•
	a_7	•	•	•	•	•	•	5992 (99.967)	•	•	•	1 (0.017)	1 (0.017)
	a_8	•	•	•	•	•	•	•	4341 (99.382)	•	•	1 (0.023)	26 (0.595)
	a_9	•	•	•	•	•	•	•	•	511 (90.925)	•	•	51 (9.075)
	a_{10}	•	•	•	•	•	•	•	•	•	1106 (99.640)	•	4 (0.360)
	a_{11}	•	•	•	•	•	•	•	4 (1.594)	•	•	245 (97.610)	2 (0.797)
	a_{12}	•	2 (0.029)	6 (0.086)	•	2 (0.029)	•	1 (0.014)	14 (0.200)	31 (0.444)	2 (0.029)	1 (0.014)	6924 (99.155)

number of zero entries dominating the confusion matrix except for the last column and the diagonal entries suggests that the proposed system results in low confusion amongst the pre-defined activities. Even the system recognizes the under-represented classes with high accuracy. For example, the recognition accuracy of *Leave home* activity (a_6) is 98.333% even if it consists of only 0.807% sensor events (given in Table 5.1). The last column has non-zero entries, indicates that few sliding windows corresponding to pre-defined activities are misclassified as 'Other' activity.

5.4.6 Comparison with other approaches

Table 5.7 lists the state-of-the-art performance achieved by previous works on the CASAS datasets to compare our proposed system with other approaches. For fairness of comparison, we have directly included results reported in the respective papers. The state-of-the-art performance on the CASAS datasets has recently been achieved by [33] with latent knowledge and basic features. Authors in [33] proposed an approach that

Table 5.7: Comparison of the proposed system ($\mathbf{M_{12}}$) with other approaches on the CASAS datasets, where $\mathbf{P}_1$ and $\mathbf{P}_3$ denote overall activity classification accuracy and F1-score, respectively.

Method	Ref.	Performance									
		Tulum		Aruba		Cairo		HHl02		HHl04	
		$\mathbf{P}_1$	$\mathbf{P}_3$	$\mathbf{P}_1$	$\mathbf{P}_3$	$\mathbf{P}_1$	$\mathbf{P}_3$	$\mathbf{P}_1$	$\mathbf{P}_3$	$\mathbf{P}_1$	$\mathbf{P}_3$
TB with RF classifier	[33]	93.28	83.11	99.07	93.66	98.67	89.32	96.15	82.22	96.60	86.88
HM with RF classifier	[33]	92.76	80.36	99.00	93.20	99.72	98.07	92.42	70.97	91.33	71.57
TB with NN classifier	[33]	93.53	84.76	99.22	93.15	98.68	90.71	94.69	82.00	98.38	92.89
HM with NN classifier	[33]	93.76	85.78	99.24	93.75	99.65	94.92	92.17	74.92	95.28	81.15
P-SVM	[35]	-	-	95.00	82.00	-	-	-	-	-	-
DCNN	[6]	-	-	98.54	79.00	-	-	-	-	-	-
DCNN	[30]	-	-	-	-	95.20	90.57	-	-	-	-
ADLR	**ours**	98.61	97.85	99.53	97.71	99.91	99.05	98.64	97.46	99.29	98.85

learns the latent knowledge from explicit-activity windows and determines the given sliding window's prediction. Their approach then feeds the prediction with other sliding window features into a classifier to produce the final class prediction for each sliding window. The authors considered ten ADL for recognition and conducted experiments on five CASAS datasets. As revealed in Table 5.7, our proposed system outperforms previous approaches with significant margins from the aspects of both overall activity classification accuracy and F1-score.

5.5 Conclusion

This chapter presented the design, implementation, and evaluation of an online ADL recognition system. The proposed system uses focal loss to handle the long-tailed class distribution and center loss to enhance the discriminative power of the deeply learned features. We effectively combine deep learned features with hand-crafted features via an ensemble technique. The proposed system achieved new state-of-the-art performance on the CASAS datasets.