Combining Neural Density Estimation and Computational State Space Models for Human Activity Recognition

Master Thesis

presented by
Benedikt Jakob Lasotta
Matriculation Number 1610104

submitted to the
Institute for Enterprise Systems
Dr. Stefan Lüdtke
University of Mannheim

January 2023

Contents

1 Introduction **1**

2 Theoretical Background **3**
2.1 Introduction to Computational State Space Models 3
2.2 Distribution Estimation with Masked Autoregressive Flow 6

3 Related Work **11**

4 Methods **14**
4.1 Data pre-processing 15
4.2 Density Estimation 17
4.3 Recognition Model 18

5 Experimental Evaluation **21**
5.1 Datasets 21
5.2 Experiments 22
5.3 Results 23

6 Conclusion **27**
6.1 Summary 27
6.2 Future Work 27

A Program Code / Resources **32**

B Further Experimental Results **33**

List of Figures

2.1 Illustration of MADE 8

4.1 Hybrid architecture components 14

List of Tables

5.1 Dataset statistics 21
5.2 Evaluation Results Carrot 24
5.3 Evaluation Results Motion Sense 26

Chapter 1

Introduction

The task of recognizing human activities from sensor data is relevant for many real world applications. Its uses range from optimizing logistics in warehouses to providing healthcare and assistance systems for elderly and disabled people [15, 5]. Consequently, human activity recognition (HAR) has been an active topic of research in recent years. Historically, methods to perform sensor-based HAR, broadly fall into either of two categories, each with their respective strengths and weaknesses: Symbolic methods as proposed by Krüger et al. or Ramirez and Geffner [14, 22] can employ prior domain knowledge and thus may need less training data, but often lack the capacity to learn from complex, time-series sensor data. One such approach are computational state space models (CSSMs) [14], which employ probabilistic inference on top of symbolic, rule based representations of possible actions within a domain to reason about observed activities. Contrary to this, modern, data-driven methods are capable of handling complex, multi-channel sensor data and have shown promising results in the past. An approach using a convolutional neural network (CNN) is described by Moya Rueda et al. [18]. In recent literature Transformers have been applied to the task of recognizing human activities. Such an approach is described by Shavit and Klein [24], which uses an encoder-only Transformer architecture to aggregate sensor data from the entire time series via the attention mechanism. Their results indicate that such a system improves prediction accuracy over models based on CNNs. Next to this, many other deep learning methods for sensor-based HAR exist. A comprehensive survey over current deep learning HAR methods and challenges is given by Chen et al. [4]. Among the drawbacks of purely data-driven methods is the fact that they are oblivious to the causal structure of activities in the domain and thus may require more training data which is often hard to obtain. Thus recently, attempts have been made to integrate symbolic and data driven models. In the line of this work

such approaches are referred to as hybrid models. The goal of these is to combine both approaches in order to obtain state of the art performance while being able to incorporate prior knowledge to reduce the need for training data. This work is not the first to propose such a hybrid model, three other notable systems are listed below: DeepProbLog [17], extends probabilistic-logic modeling with neural networks. The system proposed by Rueda et al. [23], employs a CNN to learn from sensor data and uses the output of this deep model to infer the most probable activity class via Bayesian filtering. DUSTIN [2] is a hybrid system which concatenates features obtained by a knowledge based reasoner to features extracted via a neural network. In this way information about the causal structure of activities can be captured. A more thorough overview of related work is given in chapter 3.

To benefit from the advantages of symbolic and data-driven methods, the main goal of this thesis is to develop a hybrid system for sensor-based HAR by combining existing components in a novel way. The aim of this hybrid system is to achieve a greater sample efficiency than current state of the art systems. That is, with a fixed amount of limited training data the hybrid system should outperform both data-driven and purely symbolic approaches. To build this hybrid model an implementation of computational state space models (CSSM) [14] referred to as computational causal behavior models (CCBM) will be combined with an observation model that is obtained from sensor data via neural density estimation methods. Initial experiments will be conducted with a type of normalizing flows called masked autoregressive flow (MAF) [20] since it has shown promising performance on similar density estimation tasks [11]. The system will be evaluated on three datasets containing sensor data from inertial measurement units (IMUs) of people performing activities of daily living: The Carrot dataset [13], MotionSense [16], and the UCI HAR dataset [1] will be used.

Additionally, an important research goal of this thesis is to examine which component contributes in what part to the final performance. To this end, an ablation study will be conducted. For this, the performance of the symbolic approach with a simple observation model, the density estimation component with a simple prediction, the full hybrid model, and a baseline of quadratic discriminant analysis (QDA) will be compared on the data set introduced above. The results of this ablation study can be found in chapter (insert ref).

Chapter 2

Theoretical Background

2.1 Introduction to Computational State Space Models

The following paragraphs will introduce some preliminary knowledge and terminology that is required to follow the remainder of this thesis. Following convention, vector-valued variables will be written bold. Random variables are indicated by capital letters, while assignments to those random variables are indicated by non-capital letters.

The goal of human activity recognition is to recognize activities in a sequence of potential actions that a human could perform. As such the HAR context is always a dynamic one, i.e. entities within this system are subject to changing states. Starting from some initial state and following a transition model the system state changes as time progresses. Due to combinatorial explosion the space of potential states is very large. Thus, a system which allows tractable inference in dynamic systems is required. In a feasibility study by Krüger et al. [14], Computational State Space models (CSSMs) have been shown to fulfill this property, even in complex, real-world applications. The Authors found that the domain model used in their evaluation of the Carrot data set had circa 10^8 states. Yet, the system was able to outperform Hidden Markov Models (HMMs) if the correct inference procedures were chosen. This section introduces the necessary background and notation to understand such models.

In CSSMs the transition model of the dynamic system is described by a computable function, which differentiates it from systems that require the explicit enumeration of states or paths, like HMMs. The behavior of the dynamic system is characterized as a labeled transition system (LTS). It consists of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, and labeled transition relations $\rightarrow \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ that describe transitions between states. Thus, an LTS is a triple of the form $(\mathcal{S}, \mathcal{A}, \rightarrow)$. It is

possible that $\mathcal{S}$ and $\rightarrow$ become infinite while $\mathcal{A}$ is finite. As such it is necessary to find a suitable computational description to avoid explicit enumeration of all (potentially infinitely many) states. This description is then called the *computational action language* and is used to define the transition model. It is especially applicable when the modeled process (for example the act of cooking) can be interpreted as performing sequential computations to arrive at some goal state starting from an initial state. That is, when the observed behavior is somehow goal directed.

In general the model should allow tractable inference in the sense that the hidden state of the LTS can be inferred at time t given a sequence of past observations $Y_{1:t}$ from a space $\mathcal{Y}$ and a sequence of states $X_{1:t}$ from space $\mathcal{X}$. If the joint distribution $p(x_{1:t}, y_{1:t})$ factorizes over time into a *transition model* $p(X_t|X_{t-1})$ and an *observation model* $p(Y_t|X_t)$ such that

$$p(x_{1:t}, y_{1:t}) = p(y_1|x_1)p(x_1)\prod_{i=2}^{t}(p(y_i|x_i)p(x_i|x_{i-1})), \tag{2.1}$$

the joint distribution can be described by a state space model. The underlying idea of this thesis is to obtain the observation model $p(y_t|x_t)$ via a neural density estimator like masked autoregressive flow (MAF) as outlined in section 2.2, and then apply an implementation of CSSM known as CCBM to this observation data.

CCBM follows a symbolic approach that uses precondition-effect rules to define the causal model of the domain it operates on. The computational action language used in CCBM is an extension of the planning domain definition language (PDDL). This makes it possible to specify the causal structure of a domain with its actions, preconditions and effects. The observation model describes which expected observation y_t is caused by any given state x_t at time t, while the transition model describes how states change over time. In their feasibility study on CSSMs [14], Krüger et al. show that the choice of observation model has the largest impact on model accuracy. As a limitation of their work they note, that the sensor model used by them is fairly primitive: Each activity class is represented by a multivariate normal distribution with unconstrained covariance. Improving upon this basic model is thus a logical step for further research and is exactly the approach that this thesis takes. With MAF a powerful model is trained on the sensor data which aims to fit the distribution of the observation data more accurately. Thus, it enables the computation of $p(y_t|x_t)$. The data used to train this model is continuous raw sensor data. A thorough introduction to these datasets can be found in section 5.1. Due to the limited number of actions and components in the State space model, the state space of the examined domain is discrete categorical.

The rules that govern an action, i.e., what are the parameters of an action, which preconditions have to be fulfilled, what is the effect and duration of the action, are

defined for every possible action in a so called action template which is written in the action language. Given all action templates, an initial state distribution and the goal(s), a directed graph from the initial to the goal state(s) can be generated. In particular, given some state x the probability of selecting action a in CCBM is defined by a Log-Linear model with features f_1, f_2, f_3 in the following way [23]:

$$p(a|x) \propto \exp(\sum_{k=1}^{3} \lambda_k f_k(a, x)) \tag{2.2}$$

$$f_1(a, x) = \log \gamma(a(x)) \tag{2.3}$$

$$f_2(a, x) = \log s(a) \tag{2.4}$$

$$f_3(a, x) = \delta(a(x)) \tag{2.5}$$

Each of the features can be weighted by a scalar λ_k. The first feature $\gamma(a(x))$ is the revisiting factor, which is 0 if the state resulting from taking action a in state x has already been visited. It controls if already visited states can be visited again. The parameter $s(a)$ is the saliency of action a which is a weight that can be assigned to any action. A higher saliency indicates the action is more likely to be selected with respect to actions with lower saliency. Lastly, $\delta(a(x))$ is the goal distance of the state x' that results from performing action a in state x. The underlying assumption is, that actions which lead to states that are closer to the goal are selected with a higher probability.

The CCBM system is used to compile a filter from which it is possible to perform inference about the most probable action given a system sate. Due to the large state space exact inference is computationally infeasible. Hence, particle and marginal filters are used to perform approximate inference. In categorical domains the marginal filter has been shown to outperform the particle filter [14], which is the reason why it is used in this work. Standard Bayesian filtering methods are then employed to reason about state and action sequences from sensor data. To do this, first the prediction of the current state x_t is calculated based on the previous state x_{t-1} and transition probabilities, which are obtained by plugging in equation 2.2 into equation 2.6 below.

$$p(x_t|x_{t-1}) = \sum_{x_t=a(x_{t-1})} p(a|x_t) \tag{2.6}$$

$$p(x_t|y_{1:t-1}) = \sum_{x_{t-1}} p(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1}) \tag{2.7}$$

$$p(x_t|y_{1:t}) = \frac{p(y_t|x_t)p(x_t|y_{1:t-1})}{p(y_t|y_{1:t-1})} \tag{2.8}$$

The transition probabilities $p(x_t|x_{t-1})$ are then used for the actual *prediction* step of Bayesian filtering in equation 2.7, producing an estimated state. Using the observation model $p(y_t|x_t)$, the *correction* step then corrects this estimated state by taking into account the most recent observation as shown in equation 2.8.

2.2 Distribution Estimation with Masked Autoregressive Flow

The task of estimating the joint probability distribution $p(\boldsymbol{X})$ from a set of examples $\{\boldsymbol{X}_n\}_{n=1}^N$ is a central aspect in many machine learning applications. In the line of this work it is of great interest to obtain density estimates for the sensor data as this can be used for the observation model. This approach allows greater flexibility than using a simple multivariate normal distribution for all actions of a given class, which was used in the original feasibility study on CSSMs [14].

Let $\boldsymbol{x}$ be an example consisting of D random variables $X_1, X_2, ..., X_D$. The chain rule of probability states that any joint distribution of variables can be decomposed into a product of joint conditional probabilities, where variable X_d only depends on the prior variables $X_{1:d-1}$. According to this rule, the joint distribution $p(\boldsymbol{x})$ can be expressed as

$$p(\boldsymbol{x}) = \prod_{d=1}^{D} p(x_d|x_{1:d-1}). \tag{2.9}$$

This property can be exploited in order to estimate the joint distribution. To perform this task, many methods exist. In the line of this work the focus will be on methods based feed-forward neural networks which fulfill certain properties that allow correct estimation of probability densities. In particular this section will briefly introduce masked autoregressive flow (MAF) [20] as a model of performing neural density estimation from a set of examples. MAF can be thought of as stacking multiple layers of autoregressive models on top of each other to obtain

greater modeling flexibility. In the case of this work each individual MAF layer is a masked autoencoder for distribution estimation (MADE) [9] proposed by Germain et al. in 2015. A MADE is an autoencoder, that is a feed forward neural network which takes as input some D-dimensional vector $\boldsymbol{x}$ and learns a compressed hidden representation $h(\boldsymbol{x})$ from which a reconstruction $\hat{\boldsymbol{x}}$ of input $\boldsymbol{x}$ is obtained. It is possible to use an autoencoder for density estimation if each partial output $\hat{x}_d$ only depends on the previous variables x_1 to x_{d-1} and computes the probability of x_d given these previous variables. Note, that these conditionals are the factors of the joint distribution in formula 2.9 and thus their product gives the joint distribution $p(\boldsymbol{x})$. If each output unit only depends on the previous variables the model fulfills the so called *autoregressive property*. To ensure that a MADE has this property, binary masking matrices are multiplied element wise to the encoder and decoder weight matrices of the autoencoder. These masking matrices have the value 1 where connections should be retained and 0 where they should be dropped. This guarantees that variables X_1 to X_{d-1} are connected to the output $\hat{X}_d$, while the remaining variables X_d to X_D are dropped. The MADE paper [9] proposes a straightforward but effective procedure to perform this masking which is explained below.

Distribution Estimation with MADE To illustrate the principle of a masked autoencoder which fulfills the autoregressive property we look at a simple example which starts from a fully connected autoencoder with one hidden layer consisting of K hidden units. This standard autoencoder computes the hidden representation $h(\boldsymbol{x})$ of an input $\boldsymbol{x}$ and from the hidden representation computes the reconstruction $\hat{\boldsymbol{x}}$ of. Let $W \in \mathbb{R}^{K \times D}$ be the input weight matrix for the single hidden layer and $V \in \mathbb{R}^{D \times K}$ be the output weight matrix of that layer. These two matrices parameterize the autoencoder. As elaborated above, a masked autoregressive autoencoder can be constructed from this fully connected network by calculating the binary masking matrices $\mathbf{M}^W$, $\mathbf{M}^V$ and multiplying them element wise to the weight matrices. In the network this corresponds to dropping connections which would introduce information from subsequent variables. Calculating the masking matrices is straightforward and only requires that each of the K hidden units is assigned a value $m(k)$ in the range from 1 to $D - 1$. The value $m(k)$ can be viewed as the number of inputs the k^{th} hidden unit is connected to. Hence, $m(k) = 0$ and $m(k) = D$ are disallowed because they would create hidden units which are connected to none, respectively all input units. Both of these would not be useful for modeling conditionals. To assign these $m(k)$ values Germain et al. [9] suggest that for each hidden unit the value is sampled from a uniform discrete distribution defined on the integers from 1 to $D - 1$. In expectation this means that each pos-

sible value of $m(k)$ will be assigned to an approximately equal number of hidden units. Following this, the masking matrices $\mathbf{M}^W$, $\mathbf{M}^V$ can be computed through the equations 2.10 and 2.11 respectively. For $d \in \{1, ..., D\}$ and $k \in \{1, ..., K\}$:

$$\mathbf{M}^W_{k,d} = \begin{cases} 1 & \text{if } m(k) \geq d \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

$$\mathbf{M}^V_{d,k} = \begin{cases} 1 & \text{if } d > m(k) \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

Figure 2.1 illustrates this masking for a simple autoencoder with $D = 3$ and one hidden layer with $K = 5$ hidden units. However, the process can be extended to autoencoders with multiple hidden layers. Furthermore it is not necessary that the order of inputs is kept. In fact, Germain et al. suggest that order-agnostic training, in which multiple MADEs are trained on different random orderings, may be beneficial [9].

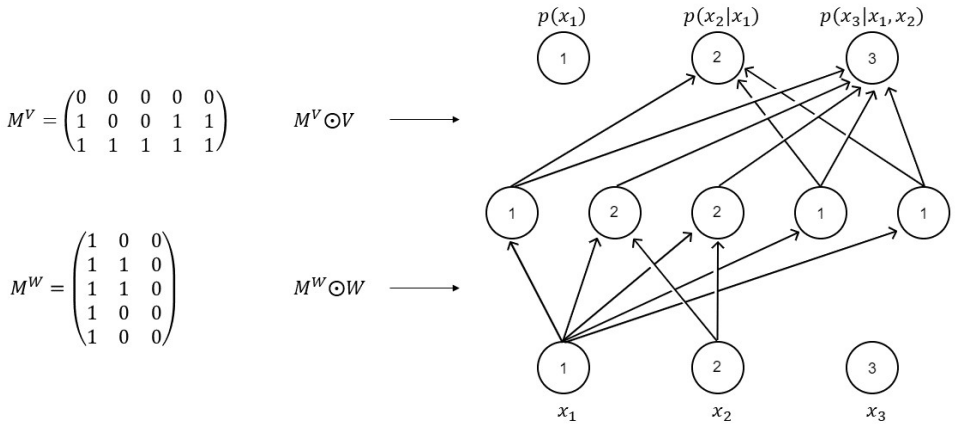

Figure 2.1: Illustration of the effect of masking matrices on the graph structure of an autoencoder. The digits inside the hidden layer nodes are the $m(k)$ values that were used to calculate $\mathbf{M}^W$ and $\mathbf{M}^V$, the digits inside input / output nodes indicate the ordering. Connections where the masking matrix is 0 are dropped while others are retained (own figure).

Advantages of MAF A main disadvantage of MADE is that the order of the conditionals has a large impact on the quality of the learned density. With the correct order it may be possible to learn the density perfectly while for other orders this

is not the case. To remedy this, MAF stacks L MADEs and models the random numbers which are used by the MADE in layer l with the model in layer $l+1$ and so on, until the random numbers of the model in the last layer are modeled by a standard Gaussian. Although each individual MADE unit has unimodal conditionals the authors of MAF argue that such a model can learn multimodal conditionals [20]. According to them, this added flexibility improves model fit.

The central idea behind MAF is, that the distribution to be learned is viewed as a transformation of a base density $\pi_u(\boldsymbol{u})$ to the target distribution via a differentiable and invertible transformation f. The base density should be simple in the sense that $\pi_u(\boldsymbol{u})$ can be easily evaluated for all inputs $\boldsymbol{u}$. For example the standard multivariate Gaussian where $\boldsymbol{u} \sim \mathcal{N}(0, I)$ is a common choice. This means that f takes an input $\boldsymbol{u}$, which follows the base density distribution $\boldsymbol{u} \sim \pi_u(\boldsymbol{u})$ and transforms it to data space via the invertible function $\boldsymbol{x} = f(\boldsymbol{u})$. As f is invertible it holds that $\boldsymbol{u} = f^{-1}(\boldsymbol{x})$, and the density $p(\boldsymbol{x})$ can be calculated as

$$p(\boldsymbol{x}) = \pi_u\big(f^{-1}(\boldsymbol{x})\big)\left|\det\left(\frac{\partial f^{-1}}{\partial \boldsymbol{x}}\right)\right|. \tag{2.12}$$

According to Papamakarios et al. [20] the computation of this expression is tractable by design. The first part is easy to compute because f is by design invertible and the base density can be evaluated for any input. The absolute determinant of the Jacobian is tractable, because the Jacobian matrix is triangular. This is due to the autoregressive property where the d-th variable x_d only depends on the prior $d-1$ variables $x_{1:d-1}$. Consequently, the Jacobian of f^{-1} is zero for all variables that this variable does not depend on. In each layer the mean and the log standard deviation of the d-th conditional given only the previous previous variables are computed via functions which are implemented as MADE units. Stacking multiple such units on top of each other is possible because if f_1 and f_2 are differentiable and invertible functions their composition $f_1 \circ f_2$ is also invertible and differentiable. This means the necessary properties to keep expression 2.12 tractable are maintained, even when stacking multiple layers. The use of masking makes it possible to compute the density $p(\boldsymbol{x})$ of data $\boldsymbol{x}$ in a single forward pass.

Conditional MAF At test time this allows to take a set of samples and produce for each of those samples the likelihood of the observed data. For reasons of numerical stability during training and testing, the log likelihood (natural logarithm) is used in this work. It is important to note here, that obtaining a single density from a sample is not yet what we want to achieve. The final goal of MAF in this use case is to obtain the observation model from it. For this the likelihood of observed sensor data given the system state should be computed.

That is, from the set of labeled training pairs $p(\boldsymbol{x}|y)$ can be computed instead of computing $p(\boldsymbol{x})$. According to Papamakarios et al. [20] this extension to handle class information y comes naturally in MAF. It is obtained by conditioning each term in the chain rule decomposition of the joint probability on the available class information y. This means that $p(\boldsymbol{x}|y)$ decomposes into $p(\boldsymbol{x}|y) = \prod_i p(x_i|x_{1:d-1}, y)$. The additional class information is added to the input as a one hot encoded class vector. This information is then used by each layer as an additional input to the raw data, as the class information is always given at every time step no connections need to be masked from the class inputs to the remainder of the network, as this does not violate the autoregressive property. For this work conditional MAF was employed exclusively, since all of the three datasets used for evaluation contain labeled data, and conditional MAF was shown to significantly outperform unconditional MAF if labeled data is available [20].

Chapter 3

Related Work

While it has been an active research topic for many years, the field of neurosymbolic AI has recently gained renewed traction after calls for more explainable and semantically sound AI systems have gotten louder [8]. This section gives a concise overview of recent developments in this field and introduces successful models related to the presented work. Intellectually closest to this thesis is the approach proposed by Rueda et al. [23]. They present the idea of using deep neural architectures as the observation model required by CSSMs. In particular the authors use a CNN to learn from time-series sensor data provided by inertial measurement units (IMUs), and predict the observed action class. In that sense the purpose of the system is very similar to the model presented here. The authors evaluated their model on the same Carrot data set that will be used for evaluation in this thesis and found that the model performance was comparable to that of state of the art deep models at the time. In the line of this work, a similar approach will be taken. With the difference being, that the observation model is obtained by a neural density estimator like MAF. The benefit of this approach over using a CNN is that MAF actually learns a distribution and can compute the density of sensor data in a single forward pass. While a CNN computes as surrogate the predicted action from sensor data, MAF in fact allows computation of the observation model $p(y_t|x_t)$. Subsequently, this observation model is combined with CCBM to perform probabilistic inference. As the same dataset is used for evaluation in this thesis and the work introduced above, it will be interesting to compare the results with this approach.

Next to this, many other approaches to combine deep neural architectures with symbolic reasoning have been proposed in recent literature. DeepProbLog [17] is introduced as a framework which combines neural networks and probabilistic-logical modeling via the existing language ProbLog [21]. This is done by extending ProbLog with neural predicates. These special predicates essentially com-

pute the probability of atomic expressions in a probabilistic logic via a neural network. The evaluation of DeepProbLog shows that this framework is capable of performing symbolic reasoning as well as deep learning from examples in an end-to-end fashion. The benefit of this system over previous work is, that it integrates neural networks into a probabilistic-logical framework instead of approximating reasoning with neural nets. Consequently, the resulting model is able to perform probabilistic-logical inference, deep learning and combinations thereof. The downside of this approach is that its inference algorithms and language are ill-suited for dynamic systems, such as HAR.

Another hybrid architecture for HAR in an order-picking scenario is proposed by Lüdtke et al. [15]. This model first predicts higher level movement descriptors from sensor data via a temporal convolutional neural network. These attributes are then used by a shallow classifier to estimate the most likely activity class. Additionally, the current process step can be taken into account as context information by making it accessible to the shallow classifier. One main benefit of this system is that it allows the integration of prior context information without re-training the entire network. Consequently, the deep model can be interpreted as a tool for feature extraction from sensor data. This means that it is largely domain independent and only the shallow prediction head has to be adjusted when switching domains. This model is reported to achieve state-of-the-art HAR performance even in the absence of context information, but performance increases further if relevant information from the process step is included, even if it is not always correct.

A very recent approach that shows promising results for HAR is DUSTIN [2] which employs knowledge infusion on top of features extracted via a neural network. That is, the model uses a CNN to extract features from time series sensor data as well as the high-level context and concatenates to this representation a set of features which are obtained by a knowledge based reasoner before its classification layer. Just as for CSSMs, this has the added benefit that common-sense knowledge can be introduced to the model, which means that estimated action sequences are consistent with the user-context. However, unlike most models DUSTIN allows infusion of additional knowledge into the deep model itself. That is instead of applying such information before or after the learning process, additional knowledge from a symbolic reasoner is added before the classification layer of the deep model. This is achieved by aggregating raw context data (e.g. the GPS data of the user) to high level context data (e.g. the users semantic location in the world). From this, the symbolic reasoner produces only activities which are consistent with the observed context (e.g. if the user is at a library, they are not doing sports). The evaluation of DUSTIN shows that it is able to outperform other state-of-the-art neuro-symbolic approaches while achieving a high sample efficiency. In comparison with other hybrid approaches that employ knowledge infusion [3, 25], DUSTIN is the

first HAR model to apply the knowledge infusion *within* the deep model instead of applying it before or after the learning process.

Chapter 4

Methods

At its core, the hybrid model proposed in this paper consists of three main components which can be exchanged with other methods and combined in a modular way. Figure 4.1 illustrates a schematic of these components and their interactions. Ultimately, the goal of the hybrid model is to achieve sate of the art HAR performance with increased sample efficiency compared to approaches which solely rely on deep learning methods. To this end three components are defined which can be altered to investigate the effects of each component.

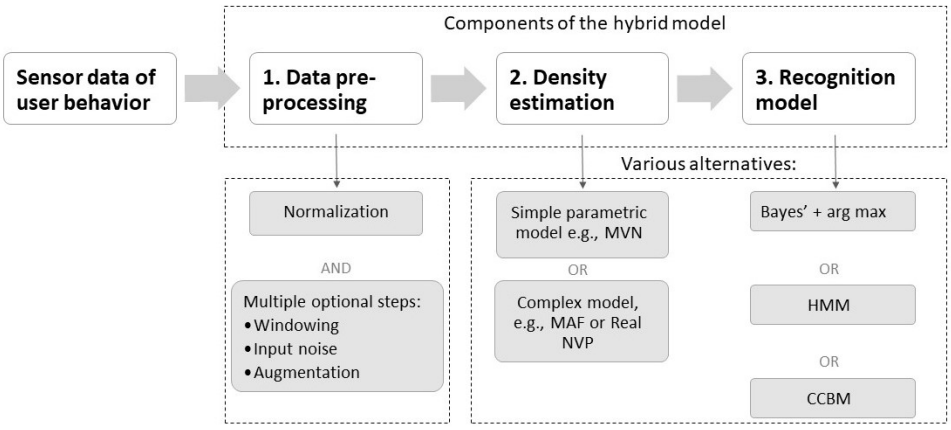

Figure 4.1: Components of the hybrid architecture and some of their possible variations (own image).

First is the data pre-processing stage in which numerical raw data is read and processed. Various steps can be undertaken here to make the training of the following module more robust. Secondly, a method for density estimation is employed to

obtain sample densities. The third and last step in this process is the recognition model which may use the obtained densities directly for activity recognition or as an observation model for more complex models. From this the concrete prediction of an activity class or state is obtained. The remainder of this chapter introduces each component and the implementation thereof in greater detail. Furthermore, some other models choices for these components are discussed briefly.

4.1 Data pre-processing

The data of all three datasets used in this work stems from inertial measurement units (IMUs) which measure acceleration and angular rates. MotionSense [16] and the UCI HAR [1] dataset use the accelerometer and gyroscope sensors of smartphones, while the Carrot dataset [13] uses standalone IMUs which are located in 5 different locations on the body. Furthermore, the UCI HAR dataset has precomputed features while the remaining two contain raw sensor data. As UCI HAR is already pre-processed it was chosen not to employ any methods that alter this default pre-processing of the UCI HAR dataset. Thus, he processing steps below are applicable to the Carrot and MotionSense datasets. In terms of pre-processing several steps were undertaken and their effects are evaluated in section 5.3 of this thesis. Normalization is always done for both datasets, as without it the training of the MAF is unstable. In addition to this mandatory step, three optional preprocessing steps may be employed for both datasets. These steps are windowing, augmentation, and noise addition. It is possible to select a subset of these operations, including none or all of them together.

Normalization To begin with, each channel in the data is normalized to the interval $[0, 1]$. This is achieved by fitting a *MinMaxScaler* to the training data, the same transformation with this scaler is then applied to the test and validation data to normalize it in the same way. Also tested was z-score normalization per channel, but it quickly became apparent that training was more stable with the MinMaxScaler.

Windowing The process of windowing segments the raw data into windows of a specified window size *T*, with a step size of *s*, indicating by how many samples to step the subsequent window. Note, that window size and step size can be used to create overlapping windows. If *T* is the same as *s* no overlap occurs, if for example $s = \frac{T}{2}$, subsequent windows have an overlap of 50 %. Such overlap creates more samples than windowing without it. The class of a window is the majority class of its components. It is apparent that with a window size of $T = 1$ the notion of a sample (i.e., an example at a single time step) and a window agree. For simplicity,

a data window of arbitrary size will be referred to as a *sample* in the remainder of this thesis.

The idea behind windowing the data is that actions in reality have a duration that is typically much greater than the sampling rate of the sensors. Thus, many individual time steps can be segmented together in order to obtain a sample which corresponds to a real-world activity. Thus such a segmentation is typically done for HAR applications. The UCI HAR dataset is segmented by default into windows of size $T = 128$.

Data Augmentation Next to windowing, two simple data augmentation methods can be used to augment the training data. These are adapted from the paper on data augmentation methods for sensor data by Um et al. [27]. Namely scaling and jittering of data are used subsequently. If used, a proportion of the training data, i.e., from 0 to 100 % is added to the original data in a slightly altered fashion to increase the number of training samples. The employed augmentation methods are jittering, that is adding Gaussian noise with $\mu = 0, \sigma = 0.05$, and scaling, that is scaling each channel by multiplying with a factor drawn from a normal distribution with $\mu = 1, \sigma = 0.1$. This step was added as an option because the amount of training data is often very limited.

Input Noise Injection To combat overfitting of the subsequent MAF module another option was introduced: Adding noise directly to the raw data. If this option is set, Gaussian noise with $\mu = 0, \sigma = 0.025$ is added to training samples. The idea here is, that this does not change the samples enough to where they would be considered a different class, but enough that some of the interpersonal variability between how different subjects perform certain activities is captured. Adding noise differs from the data augmentation option where the original data is left unchanged and only augmented samples are added to the training data. It is also possible to use both mechanisms at the same time.

Pre-Processing Alternatives One of the main difficulties of HAR lies in the fact that different subjects often produce greatly differing sensor readings for the same activity, that is a high intra-class variability. This is a consequence of various physiological differences between subjects. Factors such as variances in age, height or weight can impact how actions are performed, and thus sensor readings, strongly. Consequently, it is preferable for HAR tasks to have a dataset that captures most of this variability in its training set by containing data of many different subjects. However, labeled motion data is often hard to obtain. To counteract this, the pre-processing component of the hybrid model could be refined further. For example,

more advanced augmentation methods, such as the ones proposed by Um et al. [27], are available. These aim to combat a lack of labeled data. Furthermore, representation learning from unlabeled time series in a self-supervised manner as proposed by Eldele et al. [7] may be an interesting research direction. In this way helpful representations may be learned which are useful for the downstream task of recognizing human activities.

4.2 Density Estimation

The second component of the proposed hybrid system is the density estimation component. This module computes the probability density of samples which can be used later on in the recognition model either to build the observation model or to predict actions from conditional densities directly. Just like the other two modules, many concrete implementations are plausible for this component.

Multivariate Normal Distribution Potentially the simplest method of density estimation is a simple parametric model for which the probability densities can be computed easily. An example of this is the multivariate normal (MVN) distribution. For this thesis one MVN per activity class y was fitted for benchmarking purposes. Thus a set of MVNs with unconstrained covariance matrices allow conditional density estimation. Each MVN can be used to obtain the density $p(\boldsymbol{x}|y)$ of a sample, conditioned on activity class y.

Advanced Density Estimation Methods However, there are more sophisticated methods of obtaining probability densities from samples. One such method is density estimation based on neural networks such as the previously discussed MADE and MAF models. The goal of the neural architecture of the hybrid system is to obtain a likelihood from given sensor data which is useful for the downstream task of recognizing human activities. As MAF has shown promising performance in density estimation tasks [11] it was elected for the implementation of this component. At its heart, MAF is trained on data to fit the assumed true data distribution from samples. As discussed in section 2.2, MAF calculates densities of samples within a single forward pass, which makes obtaining densities with a trained MAF computationally inexpensive. It was chosen to use a conditional MAF as the density estimation component of the hybrid model. To build this model a base implementation of MAF in PyTorch [12] was used and heavily adapted to fit the requirements of this thesis. Each Layer of the MAF consists of a MADE followed by a Batch normalization layer. The MADE uses the ReLu activation function. The number of MAF modules that are stacked on top of each other is a tunable hyperparameter.

However, five blocks of the above architecture provided the most promising results in many of the experiments.

MAF is trained by minimizing the negative log likelihood (NLL) loss, over a number of max epochs which can be set as a hyperparameter. The best model is selected by the lowest loss on the validation data. Since, the downstream task is the goal of recognizing activities it was also tested to select the best model according to recognition accuracy. However, in most cases these two methods of selecting the best model agree or differ only marginally. Therefore, minimizing the NLL loss was chosen as a suitable mechanism for model selection. Furthermore, it is important that training samples are shuffled to ensure that they are independent and identically distributed. If this is not done, most batches only contain samples from one class and MAF tends to learn unimodal or other very simple distributions.

In theory other generative models can be employed in a very similar fashion to obtain (conditional) densities of samples. Another notable example that can be used for density estimation and is also based on invertible flows is Real NVP [6] or extensions thereof like Glow [10].

4.3 Recognition Model

The last component of the proposed hybrid model is the recognition model. The goal of this model is to accurately recognize human activities given the output of the previous step. To achieve this, many concrete options are applicable. The following paragraph will introduce three of the potential mechanisms.

Bayes Classifier The simplest recognition model is to compute $p(y_c|\boldsymbol{x})$ for each class $c \in C$ via Bayes' theorem and then predicting the most probable class via the $\arg\max$ operation. This mechanism is known as a Bayes classifier and is implemented for this work as follows: At test time the set of test samples is passed through the MAF and the $\log p(\boldsymbol{x}|y)$ of each sample is computed. To obtain a prediction of activities from these densities the log density of sample $\boldsymbol{x}$ conditioned on each individual class $c \in C$ is calculated using the MAF. As discussed in section 2.2, the $\log p(\boldsymbol{x}|y_c)$ for class c is obtained with a single forward pass of the sample through the MAF, which makes estimation of sample densities fairly cheap. By doing this for all classes, a matrix containing the logarithmic density of each test sample for each of the classes is computed. From there Bayes' theorem is used to calculate $p(y_c|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y_c) \cdot p(y_c)}{p(\boldsymbol{x})}$. As $p(\boldsymbol{x})$ is the same within each sample $\boldsymbol{x}$, the denominator is a constant factor per test sample. Hence, it can be left out to achieve a proportional result which suffices for predicting the most probable class. The prior $p(y_c)$ for all classes c is easily computed as the class distribution of training

samples. As operations occur in log space, the numerator of Bayes' theorem is obtained by adding the values $\log p(\boldsymbol{x}|y_c) + \log p(y_c)$. For the prediction itself, the most probable class can be predicted by getting the $\arg\max$ over all classes y for each sample.

Hidden Markov Model The previously described method however does not take into account the any other prior information that might be available such as the temporal structure of actions. To include such knowledge other methods of estimating states from observations are better suited. For example the simple Bayesian recognition model could be exchanged for a hidden Markov model (HMM). A HMM is fully specified by its transition model, emission probabilities and a prior distribution of states. In this recognition mode the emission probabilities of an HMM are obtained by the distribution over activity classes given the observation at time t. This is obtained as a result of the density estimation component. The transition model can be computed empirically from the observation data by counting how often each state follows each other state, resulting in an $M \times M$ matrix where M is the number of distinct states of the HMM. Likewise, the prior is obtained by measuring state frequencies from data. To perform the filtering task at time t, i.e., estimating the probability of a state given all observations up to this time, the prediction (4.1) and correction (4.2) steps introduced in section 2.1 are employed. By doing this sequentially a trajectory of the most probable states can be obtained which is the recognition of the individual activities as required by HAR.

$$p(x_t|y_{1:t-1}) = \sum_{x_{t-1}} p(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1}) \tag{4.1}$$

$$p(x_t|y_{1:t}) = \frac{p(y_t|x_t)p(x_t|y_{1:t-1})}{p(y_t|y_{1:t-1})} \tag{4.2}$$

Computational Causal Behavior Models Conceptually, activity recognition using CCBM works the same way as for an HMM. The difference lies in how the transition model of states is obtained. In CCBM the transition probabilities between actions are computed following the model presented in equation 2.2 of this work. The use of CCBM as the recognition model is especially sensible if the sequence of actions to be recognized follows some goal directed behavior. This is for example the case in the Carrot data set. In contrast to the exact inference in the paragraph explained above, CCBM typically uses approximate filtering algorithms as state spaces get large quickly. In such spaces the marginal filter has been shown by Nyolt et al. [19] to be an effective method of performing the filtering task. To perform activity recognition using CCBM, first the observation model is obtained

from the density estimation component. Following this, the observation model is then compiled with the action templates, initial and goal states into a marginal filter. Using this filter, a trajectory of the most probable activity states can be computed.

Chapter 5

Experimental Evaluation

5.1 Datasets

The three datasets used in the evaluation of this work are the Carrot [13], MotionSense [16] and UCI HAR [1] datasets. All three sets contain measurements from inertial measurement units. The individual datasets are introduced in greater detail in the paragraphs below. An overview of their features can be found in table 5.1.

Carrot The Carrot dataset uses standalone IMUs which are attached to the subject its body in five locations: upper torso, left forearm, right forearm, left calf, right calf. Seven subjects were tasked with a typical activity of daily living to produce this data. The task was to prepare, cook, serve and eat a carrot soup. Afterwards cleaning up and washing the dishes was required. During this task, acceleration and angular rates were measured with the IMUs and annotated based on a motion capturing system. It contains 16 action classes and additionally more fine grained annotations of objects and places. However, this work only uses the 16 action classes as labels. The data is recorded at a sampling rate of 120 Hz. Each sensor at each of the five locations records the angular rate and acceleration in x,y and z direction. Thus, the dataset consists of $D = 5 \cdot 6 = 30$ channels.

Dataset	Classes	Channels	Features	Train	Valid	Test	Subjects
Carrot	16	30	30	146570	36643	30036	7
MotionSense	6	12	12	812035	315156	285674	24
UCI HAR	6	6	561	5881	1471	2947	30

Table 5.1: Dataset statistics

MotionSense The MotionSense dataset includes time-series data generated by accelerometer and gyroscope sensors. Attitude, gravity, acceleration, and rotation rate are measured on the x,y and z axis. Consequently this dataset has $D = 4 \cdot 3 = 12$ channels. The data was collected with the sensors of an iPhone 6s. Only a single location of the sensor was taken into account, which is the front pocket of the subject. Data was collected at a sampling rate of 50 Hz. In total, the data stems from 24 subjects with differing physiological properties. They performed six activities in 15 trials: downstairs, upstairs, walking, jogging, sitting, and standing.

UCI HAR The UCI HAR dataset is slightly different as it is already processed and contains pre-computed features. Acceleration and angular rates were measured along the x,y and z axis at a sampling rate of 50 Hz, resulting in $D = 2 \cdot 3 = 6$ channels. All data was recorded with smartphone sensors from a Samsung Galaxy S II positioned at the waist. Data was labeled manually via video recordings. The pre processing of this dataset segments data into windows of 2.56 s (i.e., 128 samples) with 50 % overlap. The acceleration signal was then separated into a body acceleration and gravitational component using a Butterworth low-pass filter. Afterwards 561 distinct features from the time and frequency domain of the raw signals were calculated from each segment. In total, 30 subjects ranging in age from 19 - 48 years performed six activities: walking, upstairs, downstairs, sitting, standing and laying.

5.2 Experiments

To evaluate the usefulness of neural density estimation methods for HAR applications several experiments were conducted on the previously introduced datasets. From the set of optional pre-processing steps, five tunable parameters are obtained. Three boolean flags can be set for windowing (W), augmentation (A), and input noise injection (N). If windowing is true, some window size $T \geq 2$ needs to be specified. In this case, also a step size s with $1 \leq s \leq T$ should be given for stepping the segments in the training data. In the experiments presented here, the step size is fixed to $s = \frac{T}{2}$ such that the training data segments have an overlap of 50 %. Validation and testing sets use segmentation without overlap by default. If windowing is false, this indicates that no explicit windowing is performed, or in other words window size $T = 1$. The two remaining boolean flags A and N control whether the augmentation of training data, respectively the input noise injection as described in section 4.1 is performed. The hyperparameters of these methods which control the amount of noise added and scaling done are the default settings used in the implementation of data augmentation for wearable sensor

data [26]. Various combinations of these parameters for each dataset have been selected and evaluated. In particular, the usefulness of MAF as the density estimation component is evaluated and compared to the baseline method of estimating densities with a MVN per activity class as described in section 4.2. For this evaluation, the average log likelihood obtained over all test samples is reported. As the downstream task is the recognition of activities, two different components for the density estimation component are compared using the simple recognition model. More precisely, the recognition accuracy as well as macro weighted F1 score of both MVN and MAF densities are compared. To predict activities from sample densities, the simple Bayes classifier is used. The results of all experiments are reported in section 5.3.

Train-Test Split To facilitate an evaluation that is as close to a real world scenario as possible, a leave-one-subject-out strategy was employed for splitting the data into test and training set. Realistically, the model would be trained on some data and each new subject would not have its data in the training set. This fact is emulated by leaving one or more subjects completely out of the training set and using this subject for testing. This strategy was employed for the Carrot and MotionSense datasets, for the UCI HAR dataset the predefined train-test split was used which also leaves a set of subjects completely out of the training set. For the Carrot dataset it was decided arbitrarily that subject 7 would be the test subject and the remaining data would be used with a 70 % - 30 % split for training and validation. The MotionSense data was split in such a way that subjects 1 to 14 are used for training, 15 to 19 for validation and 20 to 24 as test subjects. UCI HAR splits the data such that 56 % of subjects are used for training, 14 % for validation and 30 % for testing.

Settings All experiments are conducted with a five layer MAF, the number of hidden neurons in each MADE Layer is $H = 512$. The parameters are updated by minimizing the negative log likelihood over the training data. The Adam optimizer is used with a base learning rate of $\lambda = 5 \times 10^{-5}$ and a weight decay of $d = 1 \times 10^{-6}$. The batch size is set to $bs = 128$. The model was trained using CUDA on an NVIDIA RTX 2070 SUPER.

5.3 Results

Table 5.2 shows the results of experiments where MAF was only trained for 2 epochs. The individual settings regarding the optional pre-processing steps can be read off the table.

Dataset	W	T	s	A	N	MVN LL	MAF LL	MVN ACC	MAF ACC
CARROT	F	1	1	F	F	63.1	**68.2**	**0.6006**	0.5384
CARROT	F	1	1	F	T	56.8	**59.7**	0.5932	**0.6037**
CARROT	F	1	1	T	F	52.1	**68.2**	**0.5683**	0.5624
CARROT	F	1	1	T	T	45.7	**59.2**	0.5103	**0.6166**
CARROT	T	26	13	F	F	**2986.3**	1555.2	**0.3299**	0.3290
CARROT	T	26	13	F	T	**2016.8**	1416.1	**0.4398**	0.3861
CARROT	T	26	13	T	F	**1881.1**	1720.2	**0.4312**	0.4121
CARROT	T	26	13	T	T	**1566.9**	1513.4	0.4909	**0.5351**
CARROT	T	8	4	T	F	530.5	**695.9**	0.3759	**0.5341**
CARROT	T	64	32	T	F	1376.0	**3319.6**	0.1215	**0.4030**
UCIHAR	T	128	64	F	F	**717.9**	351.8	**0.9444**	0.6291

Table 5.2: Log likelihoods and prediction accuracies of MAF and a baseline multivariate normal distribution trained per class with unconstrained covariance matrix. The Settings of the experiments are reported in the first 6 columns, the results in the remaining four. MAF was trained for a maximum of 2 epochs with the settings described above.

Dataset	**W**	**T**	**s**	**A**	**N**	**LL**		**Accuracy**			
						MVN	MAF	MVN + Bay.	MAF + Bay.	MVN + HMM	MAF + HMM
CARROT	F	1	1	F	F	63.1	**68.2**	**0.6006**	0.5384		
CARROT	F	1	1	F	T	56.8	**59.7**	0.5932	**0.6037**		
CARROT	F	1	1	T	F	52.1	**68.2**	**0.5683**	0.5624		
CARROT	F	1	1	T	T	45.7	**59.2**	0.5103	**0.6166**		
CARROT	T	26	13	F	F	**2986.3**	1555.2	**0.3299**	0.3290		
CARROT	T	26	13	F	T	**2016.8**	1416.1	**0.4398**	0.3861		
CARROT	T	26	13	T	F	**1881.1**	1720.2	**0.4312**	0.4121		
CARROT	T	26	13	T	T	**1566.9**	1513.4	0.4909	**0.5351**		
CARROT	T	8	4	T	F	530.5	**695.9**	0.3759	**0.5341**		
CARROT	T	64	32	T	F	1376.0	**3319.6**	0.1215	**0.4030**		
UCIHAR	T	128	64	F	F	**717.9**	351.8	**0.9444**	0.6291		

The first result that becomes apparent is that MAF achieves higher likelihoods if the data is not windowed or window sizes are relatively small. This result is expected, as MAF is a much more flexible model than a set of MVN conditioned on the class. MAF should in theory be able to fit an arbitrary distribution while MVN is expected to only be a good fit if individual features are (approximately) normally distributed. Given this discrepancy in log likelihood (LL) it is surprising that MVN significantly outperforms MAF in the base case where no windowing, augmentation or noise injection is done. In that case MVN achieves a LL of 63.1 while that of the MAF is 68.2. Nevertheless, MVN achieves the higher recognition accuracy at 60.06 %, an increase of 6.22 percentage points (pp) from the result of MAF at 53.84 %. This outcome warrants the investigation into the Carrot dataset to check if features may indeed be normally distributed which could explain the good performance of the MVN. To examine this, a 30×30 pair plot was drawn, which plots a histogram per feature from 1600 samples of the Carrot data and compares each feature to all others. An excerpt of this plot can be found in appendix B, the script to generate it is contained in the attached code. The plot shows that circa half of the 30 features from the Carrot dataset are somewhat well represented by a normal distribution. Given this result, the fact that a MVN performs well is less surprising. Still, the overall recognition accuracy is relatively low at around 60 %. This can be explained by the fact that the Carrot dataset is very challenging to learn from. The first challenge is, that it is relatively small. Data stems from only 7 subjects and the number of total (non-windowed) training samples is also low at 146570. As discussed in section 4.1 a low number of training subjects might lead to difficulties during the learning process due to the high intra-class variability in HAR tasks. In that sense it is plausible that a less flexible model can achieve better results simply because it cannot overfit to the training data as much and thus generalize better to sensor readings from unseen subjects which perform activities slightly differently from those subjects in the training data. The second difficulty of the Carrot data is that some activities occur much less frequently at a shorter duration or have many different concrete expressions. To illustrate this let us look at two activities that are hard to recognize in greater detail (insert picture of Confusion matrix to actually show this.). The activity *stand up* for example has the lowest relative frequency in the dataset at 0,0036. Also the duration of that activity typically very short. At prediction time it is often confused with other activities that are more frequent. The activity *put* is third most frequent (0,1151) in the training data, but still recognizing it accurately seems difficult. The confusion matrix shows that it has a large spread of other activities that is is confused with at test time. A plausible explanation for this observation is that *put* can have various expressions, which differ greatly in the sensor readings they produce: Putting a glass in a cupboard above your head produces different measurements from putting

Dataset	W	T	s	A	N	MVN LL	MAF LL	MVN ACC	MAF ACC
MOSENSE	True	128	64	False	False	-242.2431	1264.8010	0.4966	0.7487
MOSENSE	True	128	64	False	True	435.8816	1200.5637	0.6580	0.8254
MOSENSE	True	128	64	True	False	587.4257	1582.0021	0.6731	0.8030
MOSENSE	True	128	64	True	True	210.0161	1284.2458	0.6544	0.7916
MOSENSE	True	64	32	True	True	579.2101	868.5525	0.8166	0.8046
MOSENSE	True	32	16	True	True	289.0209	516.7429	0.7376	0.7864
UCIHAR	True	0	0	False	False	846.4878	1256.1664	0.9474	0.8792
UCIHAR	True	0	0	False	True	846.4878	868.8217	0.9474	0.9433

Table 5.3: Log likelihoods and prediction accuracies of MAF and a baseline multivariate normal distribution trained per class with unconstrained covariance matrix. The Settings of the experiments are reported in the first 6 columns, the results in the remaining four. MAF was trained for a maximum of 50 epochs with the settings described above.

a dish in the sink at waist level. It is also interesting to see that this activity is often confused with the activity *eat*. Interpreting eating as putting food in ones mouth may explain this. These results are supported by other researchers as well. For example Rueda et al. [23] report an overall recognition accuracy on subject 7 of the Carrots data of 65.2 % using a CNN. However, the recognition accuracy of the activity *put* is only 35.1 % and that of *stand up* is 11.7 %.

Chapter 6

Conclusion

6.1 Summary

6.2 Future Work

Bibliography

[1] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L. Reyes-Ortiz. A Public Domain Dataset for Human Activity Recognition Using Smartphones. In *21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, Bruges, Belgium, April 2013.

[2] Luca Arrotta, Gabriele Civitarese, and Claudio Bettini. Knowledge Infusion for Context-Aware Sensor-Based Human Activity Recognition. In *2022 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–8, June 2022. ISSN: 2693-8340.

[3] Claudio Bettini, Gabriele Civitarese, and Riccardo Presotto. CAVIAR: Context-driven Active and Incremental Activity Recognition. *Knowledge-Based Systems*, 196:105816, May 2020.

[4] Kaixuan Chen, Dalin Zhang, Lina Yao, Bin Guo, Zhiwen Yu, and Yunhao Liu. Deep Learning for Sensor-based Human Activity Recognition: Overview, Challenges, and Opportunities. *ACM Computing Surveys*, 54(4):1–40, May 2022.

[5] Liming Chen, Jesse Hoey, Chris D. Nugent, Diane J. Cook, and Zhiwen Yu. Sensor-Based Activity Recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):790–808, November 2012. Conference Name: IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews).

[6] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP, February 2017. arXiv:1605.08803 [cs, stat].

[7] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee Keong Kwoh, Xiaoli Li, and Cuntai Guan. Time-Series Representation Learning via Temporal and Contextual Contrasting. In *Proceedings of*

the Thirtieth International Joint Conference on Artificial Intelligence, pages 2352–2359, Montreal, Canada, August 2021. International Joint Conferences on Artificial Intelligence Organization.

[8] Artur d'Avila Garcez and Luis C. Lamb. Neurosymbolic AI: The 3rd Wave, December 2020. arXiv:2012.05876 [cs].

[9] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked Autoencoder for Distribution Estimation. page 9, 2015.

[10] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions, July 2018. arXiv:1807.03039 [cs, stat].

[11] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, November 2021. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[12] Ilya Kostrikov. pytorch-flows. `https://github.com/ikostrikov/pytorch-flows`, September 2018. Online; accessed 2022-10-14.

[13] Frank Krüger, Albert Hein, Kristina Yordanova, and Thomas Kirste. Recognising user actions during cooking task (Cooking task dataset): IMU Data. `https://rosdok.uni-rostock.de/resolve/id/rosdok_document_0000010639`, 2011. Publisher: University of Rostock.

[14] Frank Krüger, Martin Nyolt, Kristina Yordanova, Albert Hein, and Thomas Kirste. Computational State Space Models for Activity and Intention Recognition. A Feasibility Study. *PLOS ONE*, 9(11):e109381, November 2014. Publisher: Public Library of Science.

[15] Stefan Lüdtke, Fernando Moya Rueda, Waqas Ahmed, Gernot A. Fink, and Thomas Kirste. Human Activity Recognition using Attribute-Based Neural Networks and Context Information, October 2021. arXiv:2111.04564 [cs, eess].

[16] Mohammad Malekzadeh, Richard G. Clegg, Andrea Cavallaro, and Hamed Haddadi. Mobile Sensor Data Anonymization. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, pages 49–58, Montreal, Quebec, Canada, 2019. ACM.

[17] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural Probabilistic Logic Programming. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[18] Fernando Moya Rueda, René Grzeszick, Gernot A. Fink, Sascha Feldhorst, and Michael Ten Hompel. Convolutional Neural Networks for Human Activity Recognition Using Body-Worn Sensors. *Informatics*, 5(2):26, June 2018. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[19] Martin Nyolt, Frank Krüger, Kristina Yordanova, Albert Hein, and Thomas Kirste. Marginal filtering in large state spaces. *International Journal of Approximate Reasoning*, 61:16–32, June 2015.

[20] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked Autoregressive Flow for Density Estimation. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[21] Luc De Raedt. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. page 6.

[22] Miquel Ramirez and Hector Geffner. Goal Recognition over POMDPs: Inferring the Intention of a POMDP Agent. In *Twenty-second international joint conference on artificial intelligence*, page 6, 2011.

[23] Fernando Moya Rueda, Stefan Lüdtke, Max Schröder, Kristina Yordanova, Thomas Kirste, and Gernot A. Fink. Combining Symbolic Reasoning and Deep Learning for Human Activity Recognition. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 22–27, March 2019.

[24] Yoli Shavit and Itzik Klein. Boosting Inertial-Based Human Activity Recognition With Transformers. *IEEE Access*, 9:53540–53547, 2021.

[25] Abdul Syafiq Abdull Sukor, Ammar Zakaria, Norasmadi Abdul Rahim, Latifah Munirah Kamarudin, Rossi Setchi, and Hiromitsu Nishizaki. A hybrid approach of knowledge-driven and data-driven reasoning for activity recognition in smart homes. *Journal of Intelligent & Fuzzy Systems*, 36(5):4177–4188, May 2019.

[26] Terry Taewoong Um. Python implementation of data augmentation for wearable sensor data. `https://github.com/terryum/Data-Augmentation-For-Wearable-Sensor-Data`, August 2017. Online; accessed 2022-12-01.

[27] Terry Taewoong Um, Franz Michael Josef Pfister, Daniel Pichler, Satoshi Endo, Muriel Lang, Sandra Hirche, Urban Fietzek, and Dana Kulić. Data Augmentation of Wearable Sensor Data for Parkinson's Disease Monitoring using Convolutional Neural Networks. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, pages 216–220, November 2017. arXiv:1706.00527 [cs].

Appendix A

Program Code / Resources

The source code to run all experiments in this thesis is uploaded to a GitHub Repository. To run this code please follow the instructions given in the readme file of that repository. Steps are: 1. Downloading the required data; 2. Change the *BASEPATH* variable in the file pytorch-flows/dataloader/loaders.py, respectively the file pytorch-flows/dataloader/motionSense.py to the absolute path of the dataloader directory. For example in my case C:/Users/bened/PythonWork/Test_MA/pytorch-flows/dataloader

Appendix B

Further Experimental Results

In the following further experimental results are ...

Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Master-/Bachelorarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Er- klärung rechtliche Folgen haben wird.

Mannheim, den 31.09.2022 Unterschrift