DeepSense: a Unified Deep Learning Framework for Time-Series Mobile Sensing Data Processing

Shuochao Yao[†]
syao9@illinois.edu

Shaohan Hu[‡]
shaohan.hu@ibm.com

Yiran Zhao[†]
zhao97@illinois.edu

Aston Zhang[†]
lzhang74@illinois.edu

Tarek Abdelzaher[†]
zaher@illinois.edu

[†]University of Illinois at Urbana-Champaign, Urbana, IL USA [‡]IBM Research, Yorktown Heights, NY USA

ABSTRACT

Mobile sensing and computing applications usually require time-series inputs from sensors, such as accelerometers, gyroscopes, and magnetometers. Some applications, such as tracking, can use sensed acceleration and rate of rotation to calculate displacement based on physical system models. Other applications, such as activity recognition, extract manually designed features from sensor inputs for classification. Such applications face two challenges. On one hand, on-device sensor measurements are noisy. For many mobile applications, it is hard to find a distribution that exactly describes the noise in practice. Unfortunately, calculating target quantities based on physical system and noise models is only as accurate as the noise assumptions. Similarly, in classification applications, although manually designed features have proven to be effective, it is not always straightforward to find the most robust features to accommodate diverse sensor noise patterns and heterogeneous user behaviors. To this end, we propose DeepSense, a deep learning framework that directly addresses the aforementioned noise and feature customization challenges in a unified manner. DeepSense integrates convolutional and recurrent neural networks to exploit local interactions among similar mobile sensors, merge local interactions of different sensory modalities into global interactions, and extract temporal relationships to model signal dynamics. DeepSense thus provides a general signal estimation and classification framework that accommodates a wide range of applications. We demonstrate the effectiveness of DeepSense using three representative and challenging tasks: car tracking with motion sensors, heterogeneous human activity recognition, and user identification with biometric motion analysis. DeepSense significantly outperforms the state-of-the-art methods for all three tasks. In addition, we show that DeepSense is feasible to implement on smartphones and embedded devices thanks to its moderate energy consumption and low latency.

Keywords

Deep Learning; Mobile Computing; Mobile Sensing; Internet of Things; Tracking; Activity Recognition; User Identification

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052577

1. INTRODUCTION

A wide range of mobile sensing and computing applications require time-series measurements from such sensors as accelerometers, gyroscopes, and magnetometers to generate inputs for various signal estimation and classification applications [23]. Using these sensors, mobile devices are able to infer user activities and states [35, 39] and recognize surrounding context [31, 42]. These capabilities serve diverse application areas including health and wellbeing [20, 33, 24], tracking and imaging [25, 46], mobile security [29, 41], and vehicular road sensing [17, 19, 45].

Although mobile sensing is becoming increasingly ubiquitous, key challenges remain in improving the accuracy of sensor exploitation. In this paper, we consider the general problem of estimating signals from noisy measurements in mobile sensing applications. This problem can be categorized into two subtypes: regression and classification, depending on whether prediction results are continuous or categorical, respectively.

For regression-oriented problems, such as tracking and localization, sensor inputs are usually processed based on physical models of the phenomena involved. Sensors on mobile devices generate time-series measurements of physical quantities such as acceleration and angular velocity. From these measurements, other physical quantities can be computed, such as displacement through double integration of acceleration over time. However, measurements of commodity sensors are noisy. The noise in measurements is nonlinear [3] and correlated over time [32], which makes it hard to model. This makes it challenging to separate signal from noise, leading to estimation errors and bias.

For classification-oriented problems, such as activity and context recognition, a typical approach is to compute appropriate features derived from raw sensor data. These hand-crafted features are then fed into a classifier for training. This general workflow for classification face the challenge that designing good hand-crafted features can be time consuming; it requires extensive experiments to generalize well to diverse settings such as different sensor noise patterns and heterogeneous user behaviors [39].

In this work, we propose DeepSense, a unified deep learning framework that directly addresses the aforementioned customization challenges that arise in mobile sensing applications. The core of DeepSense is the integration of convolutional neural networks (CNN) and recurrent neural networks (RNN). Input sensor measurements are split into a series of data intervals along time. The frequency representation of each data intervals is fed into a CNN to learn intra-interval local interactions within each sensing modality and intra-interval global interactions among different sensor inputs, hierarchically. The intra-interval representations along time are then fed into an RNN to learn the inter-interval relationships.

The whole framework can be easily customized to fit specific mobile computing (regression or classification) tasks by three simple steps, as will be described later.

For the regression-oriented mobile sensing problem, DeepSense learns the composition of physical system and noise model to yield outputs from noisy sensor data directly. The neural network acts as an approximate transfer function. The CNN part approximates the computation of sensing quantities within the time interval, and the RNN part approximates the computation of sensing quantities across time intervals. Instead of using a model-based noise analysis method that assumes a noise model with experience or observations, DeepSense can be regarded as a model-free noise analysis that learns the non-linear and correlated-over-time noises among sensor measurements.

For the classification-oriented mobile sensing problem, the neural network acts as an automatic feature extractor encoding local, global, and temporal information. The CNN part extracts local features within each sensor modality and merges the local features of different sensory modalities into global features hierarchically. The RNN part extracts temporal dependencies.

We demonstrate the effectiveness of our DeepSense framework using three representative and challenging mobile sensing problems, which illustrate the potential of solving different tasks with a single unified modeling methodology:

- *Car tracking with motion sensors:* In this task, we use dead reckoning to infer position from acceleration measurements. One of the major contributions of DeepSense is its ability to withstand nonlinear and time-dependent noise and bias. We chose the car tracking task because it involves double-integration and thus is particularly sensitive to error accumulation, as acceleration errors can lead to significant deviations in position estimate over time. This task thus constitutes a worst-case of sorts in terms of emphasizing the effects of noise on modelling error. Traditionally, external means are needed to reset the error when possible [7, 17, 27]. We intentionally forgo such means to demostrate the capability of DeepSense for learning accurate models of target quantities in the presence of realistic noise.

- *Heterogeneous human activity recognition:* Although human activity recognition with motion sensors is a mature problem, Stisen *et al.* [39] illustrated that state-of-the-art algorithms do not generalize well across users when a new user is tested who has not appeared in the training set. This classification-oriented problem therefore illustrates the capability of DeepSense to extract features that generalize better across users in mobile sensing tasks.

- *User identification with biometric motion analysis:* Biometric gait analysis can be used to identify users when they are walking [13, 35]. We extend walking to other activities, such as biking and climbing stairs, for user identification. This classification-oriented problem illustrates the capability of DeepSense to extract distinct features for different users or classes.

We evaluate these three tasks with collected data or existing datasets. We compare DeepSense to state-of-the-art algorithms that solve the respective tasks, as well as to three DeepSense variants, each presenting a simplification of the algorithm as described in Section 5.3. For the regression-oriented problem: car tracking with motion sensors, DeepSense provides an estimator with far smaller tracking error. This makes tracking with solely noisy on-device motion sensors practical and illustrates the capability of DeepSense to perform accurate estimation of physical quantities from noisy sensor data. For the other two classification-oriented problems, DeepSense outperforms state-of-the-art algorithms by a large margin, illustrating its capability to automatically learn robust and distinct features. DeepSense outperforms all its simpler variants in all three tasks, which shows the effectiveness of its design components. Despite a general shift towards remote cloud processing for a range of mobile applications, we argue that it is intrinsically desirable that heavy sensing tasks be carried out locally on-device, due to the usually tight latency requirements, and the prohibitively large data transmission requirement as dictated by the high sensor sampling frequency (e.g. accelerometer, gyroscope). Therefore, we also demonstrate the feasibility of implementing and deploying DeepSense on mobile devices by showing its moderate energy consumption and low overhead for all three tasks on two different types of smart devices.

In summary, the main contribution of this paper is that *we develop a deep learning framework, DeepSense, that solves both regression-oriented and classification-oriented mobile computing tasks in a unified manner. By exploiting local interactions within each sensing modality, merging local interactions of different sensing modalities into global interactions, and extracting temporal relationships, DeepSense learns the composition of physical laws and noise model in regression-oriented problems, and automatically extracts robust and distinct features that contain local, global, and temporal relationships in classification-oriented problems. Importantly, it outperforms the state of the art, while remaining implementable on mobile devices.*

The rest of this paper is organized as follows. Section 2 introduces related work on deep learning in the context of mobile sensing and computing. We describe the technical details of DeepSense in Section 3 and the way to customize DeepSense to mobile computing problems in Section 4. The evaluation is presented in Section 5. Finally, we discuss the results in Section 6 and conclude in Section 7.

2. RELATED WORK

Recently, deep learning [5] has become one of the most popular methodologies in AI-related tasks, such as computer vision [16], speech recognition [10], and natural language processing [4]. Lots of deep learning architectures have been proposed to exploit the relationships embedded in different types of inputs. For example, Residual nets [16] introduce shortcut connections into CNNs, which greatly reduces the difficulty of training super-deep models. However, since residual nets mainly focus on visual inputs, they lose the capability to model temporal relationships, which are of great importance in time-series sensor inputs. LRCNs [11] apply CNNs to extract features for each video frame and combine video frame sequences with LSTM [14], which exploits spatio-temporal relationships in video inputs. However, it does not consider modeling multimodal inputs. This capability is important to mobile sensing and computing tasks, because most tasks require collaboration among multiple sensors. Multimodal DBMs [38] merge multimodal inputs, such as images and text, with Deep Boltzmann Machines (DBMs). However, the work does not model temporal relationships and does not apply tailored structures, such as CNNs, to effectively and efficiently exploit local interactions within input data. To the best of our knowledge, DeepSense is the first architecture that possesses the capability for both (i) modelling temporal relationships and (ii) fusing multimodal sensor inputs. It also contains specifically designed structures to exploit local interactions in sensor inputs.

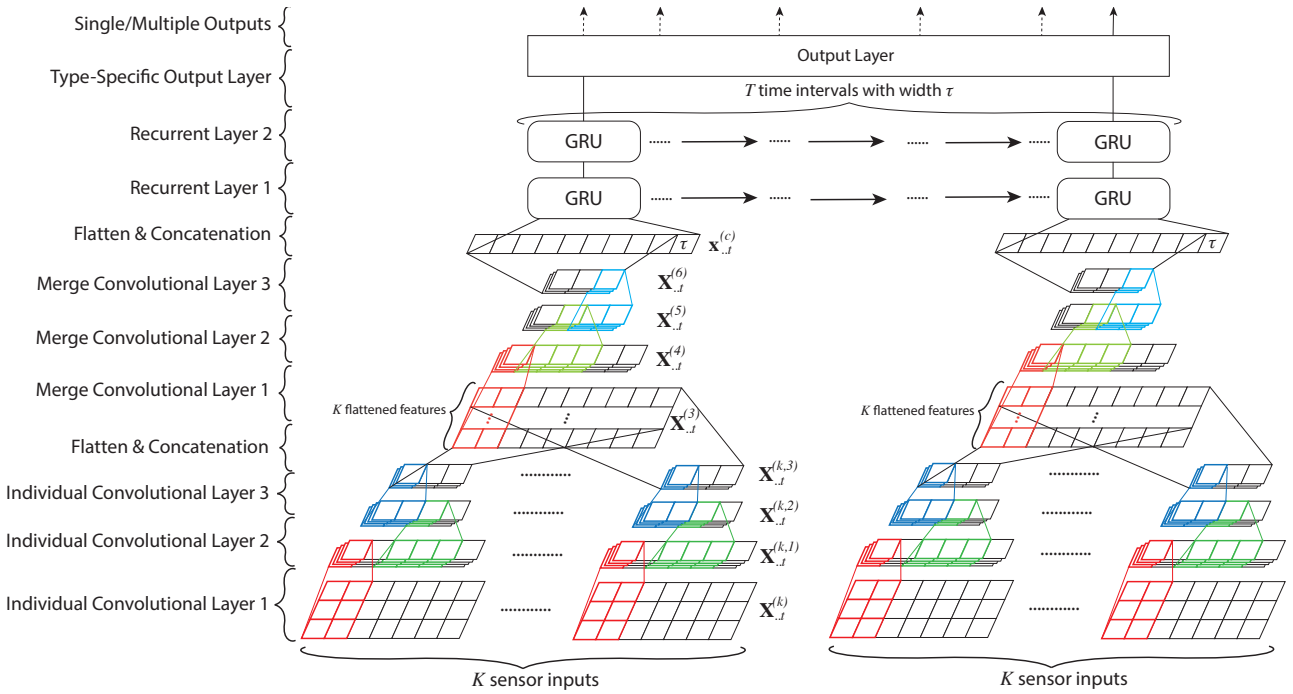


Figure 1: Main architecture of the DeepSense framework.

There are several illuminating studies, applying deep neural network models to different mobile sensing applications. DeepEar [22] uses Deep Boltzmann Machines to improve the performance of audio sensing tasks in an environment with background noise. RBM [6] and MultiRBM [34] use Deep Boltzmann Machines and Multimodal DBMs to improve the performance of heterogeneous human activity recognition. IDNet [13] applies CNNs to the biometric gait analysis task. DeepX [21], RedEye [26], and ConvTransfer [30] reduce the energy consumption or training time of deep neural networks, based on software and hardware, respectively. However, these studies do not capture the temporal relationships in time-series sensor inputs, and, with the only exception of MultiRBM, lack the capability of fusing multimodal sensor inputs. In addition, these techniques focus on classification-oriented tasks only. To the best of our knowledge, DeepSense is the first framework that directly solves both regression-based and classification-based problems in a unified manner.

3. DEEPSENSE FRAMEWORK

We introduce DeepSense, a unified framework for mobile applications with sensor data inputs, in this section. We separate our description into three parts. The first two parts, convolutional layers and recurrent layers, are the main building blocks for DeepSense, which are the same for all applications. The third part, the output layer, is the specific layer for two different types of applications; regression-oriented and classification-oriented.

For the rest of this paper, all vectors are denoted by bold lower-case letters (e.g., $\mathbf{x}$ and $\mathbf{y}$), while matrices and tensors are represented by bold upper-case letters (e.g., $\mathbf{X}$ and $\mathbf{Y}$). For a vector $\mathbf{x}$, the j^{th} element is denoted by $\mathbf{x}_{[j]}$. For a tensor $\mathbf{X}$, the t^{th} matrix along the third axis is denoted by $\mathbf{X}_{..t}$, and other slicing denotations are defined similarly. We use calligraphic letters to denote sets (e.g., $\mathcal{X}$ and $\mathcal{Y}$). For any set $\mathcal{X}$, $|\mathcal{X}|$ denotes the cardinality of $\mathcal{X}$.

For a particular application, we assume that there are K different types of input sensors $\mathcal{S} = \{S_k\}$, $k \in \{1, \cdots, K\}$. Take a sensor S_k as an example. It generates a series of measurements over time. The measurements can be represented by a $d^{(k)} \times n^{(k)}$ matrix $\mathbf{V}$ for measured values and $n^{(k)}$-dimensional vector $\mathbf{u}$ for time stamps, where $d^{(k)}$ is the dimension for each measurement (e.g., measurements along x, y, and z axes for motion sensors) and $n^{(k)}$ is the number of measurements. We split the input measurements $\mathbf{V}$ and $\mathbf{u}$ along time (i.e., columns for $\mathbf{V}$) to generate a series of non-overlapping time intervals with width τ, $\mathcal{W} = \{(\mathbf{V}_t^{(k)}, \mathbf{u}_t^{(k)})\}$, where $|\mathcal{W}| = T$. Note that, τ can be different for different intervals, but here we assume a fixed time interval width for succinctness. We then apply Fourier transform to each element in $\mathcal{W}$, because the frequency domain contains better local frequency patterns that are independent of how time-series data is organized in the time domain [36]. We stack these outputs into a $d^{(k)} \times 2f \times T$ tensor $\mathbf{X}^{(k)}$, where f is the dimension of frequency domain containing f magnitude and phase pairs. The set of resulting tensors for each sensor, $\mathcal{X} = \{\mathbf{X}^{(k)}\}$, is the input of DeepSense.

As shown in Fig. 1, DeepSense has three major components; the convolutional layers, the recurrent layers, and the output layer, stacked from bottom to top. In the following subsections, we detail these components, respectively.

3.1 Convolutional Layers

The convolutional layers can be further separated into two parts: an individual convolutional subnet for each input sensor tensor $\mathbf{X}^{(k)}$, and a single merge convolutional subnet for the output of K individual convolutional subnets' outputs.

Since the structures of individual convolutional subnet for different sensors are the same, we focus on one individual convolutional subnet with input tensor $\mathbf{X}^{(k)}$. Recall that $\mathbf{X}^{(k)} \in \mathbb{R}^{d^{(k)} \times 2f \times T}$, where $d^{(k)}$ is the sensor measurement dimension, f is the dimension of frequency domain, and T is the number of time intervals. For each time interval t, the matrix $\mathbf{X}_{..t}^{(k)}$ will be fed into a CNN

architecture (with three layers in this paper). There are two kinds of features/relationships embedded in $\mathbf{X}_{\cdot\cdot t}^{(k)}$ we want to extract. The relationships within the frequency domain and across sensor measurement dimension. The frequency domain usually contains lots of local patterns in some neighbouring frequencies. And the interaction among sensor measurement usually including all dimensions. Therefore, we first apply 2d filters with shape $(d^{(k)}, cov1)$ to $\mathbf{X}_{\cdot\cdot t}^{(k)}$ to learn interaction among sensor measurement dimensions and local patterns in frequency domain, with the output $\mathbf{X}_{\cdot\cdot t}^{(k,1)}$. Then we apply 1d filters with shape $(1, cov2)$ and $(1, cov3)$ hierarchically to learn high-level relationships, $\mathbf{X}_{\cdot\cdot t}^{(k,2)}$ and $\mathbf{X}_{\cdot\cdot t}^{(k,3)}$.

Then we flatten matrix $\mathbf{X}_{\cdot\cdot t}^{(k,3)}$ into vector $\mathbf{x}_{\cdot\cdot t}^{(k,3)}$ and concat all K vectors $\{\mathbf{x}_{\cdot\cdot t}^{(k,3)}\}$ into a K-row matrix $\mathbf{X}_{\cdot\cdot t}^{(3)}$, which is the input of the merge convolutional subnet. The architecture of the merge convolutional subnet is similar as the individual convolutional subnet. We first apply 2d filters with shape $(K, cov4)$ to learn the interactions among all K sensors, with output $\mathbf{X}_{\cdot\cdot t}^{(4)}$, and then apply 1d filters with shape $(1, cov5)$ and $(1, cov6)$ hierarchically to learn high-level relationships, $\mathbf{X}_{\cdot\cdot t}^{(5)}$ and $\mathbf{X}_{\cdot\cdot t}^{(6)}$.

For each convolutional layer, DeepSense learns 64 filters, and uses ReLU as the activation function. In addition, batch normalization [18] is applied at each layer to reduce internal covariate shift. We do not use residual net structures [16], because we want to simplify the network architecture for mobile applications. Then we flatten the final output $\mathbf{X}_{\cdot\cdot t}^{(6)}$ into vector $\mathbf{x}_{\cdot\cdot t}^{(f)}$; concatenate $\mathbf{x}_{\cdot\cdot t}^{(f)}$ and time interval width, $[\tau]$, together into $\mathbf{x}_t^{(c)}$ as inputs of recurrent layers.

3.2 Recurrent Layers

Recurrent neural networks are powerful architectures that can approximate function and learn meaningful features for sequences. Original RNNs fall short of learning long-term dependencies. Two extended models are Long Short-Term Memory (LSTM) [14] and Gated Recurrent Unit (GRU) [8]. In this paper, we choose GRU, because GRUs show similar performance as LSTMs on various tasks [8], while having a more concise expression, which reduces network complexity for mobile applications.

DeepSense chooses a stacked GRU structure (with two layers in this paper). Compared with standard (single-layer) GRUs, stacked GRUs are a more efficient way to increase model capacity [5]. Compared to bidirectional GRUs [37], which contain two time flows from start to end and from end to start, stacked GRUs can run incrementally, when there is a new time interval, resulting in faster processing of stream data. In contrast, we cannot run bidirectional GRUs until data from all time intervals are ready, which is infeasible for applications such as tracking. We apply dropout to the connections between GRU layers [43] for regularization and apply recurrent batch normalization [9] to reduce internal covariate shift among time steps. Inputs $\{\mathbf{x}_t^{(c)}\}$ for $t = 1, \cdots, T$ from previous convolutional layers are fed into stacked GRU and generate outputs $\{\mathbf{x}_t^{(r)}\}$ for $t = 1, \cdots, T$ as inputs of the final output layer.

3.3 Output Layer

The output of recurrent layer is a series of vectors $\{\mathbf{x}_t^{(r)}\}$ for $t = 1, \cdots, T$. For the regression-oriented task, since the value of each element in vector $\mathbf{x}_t^{(r)}$ is within ± 1, $\mathbf{x}_t^{(r)}$ encodes the output physical quantities at the end of time interval t. In the output layer, we want to learn a dictionary $\mathbf{W}_{out}$ with a bias term $\mathbf{b}_{out}$ to decode $\mathbf{x}_t^{(r)}$ into $\hat{\mathbf{y}}_t$, such that $\hat{\mathbf{y}}_t = \mathbf{W}_{out} \cdot \mathbf{x}_t^{(r)} + \mathbf{b}_{out}$. Therefore, the output layer is a fully connected layer on the top of each interval with sharing parameter $\mathbf{W}_{out}$ and $\mathbf{b}_{out}$.

For the classification task, $\mathbf{x}_t^{(r)}$ is the feature vector at time interval t. The output layer first needs to compose $\{\mathbf{x}_t^{(r)}\}$ into a fixed-length feature vector for further processing. Averaging features over time is one choice. More sophisticated methods can also be applied to generate the final feature, such as the attention model [4], which has illustrated its effectiveness in various learning tasks recently. The attention model can be viewed as weighted averaging of features over time, but the weights are learnt by neural networks through context. In this paper, we still use averaging features over time to generate the final feature, $\mathbf{x}^{(r)} = (\sum_{t=1}^{T} \mathbf{x}_t^{(r)})/T$. Then we feed $\mathbf{x}^{(r)}$ into a softmax layer to generate the predicted category probability $\hat{\mathbf{y}}$.

4. TASK-SPECIFIC CUSTOMIZATION

In this section, we first describe how to trivially customize the DeepSense framework to different mobile sensing and computing tasks. Next, we instantiate the solution with three specific tasks used in our evaluation.

4.1 General Customization Process

In general, we need to customize a few parameters of the main architecture of DeepSense, shown in Section 3, for specific mobile sensing and computing tasks. Our general DeepSense customization process is as follows:

1. Identify the number of sensor inputs, K. Pre-process the sensor inputs into a set of tensors $\mathcal{X} = \{\mathbf{X}^{(k)}\}$ as input.

2. Identify the type of the task. Whether the application is regression or classification-oriented. Select one of the two types of output layer according to the type of task.

3. Design a customized cost function or choose the default cost function (namely, mean square error for regression-oriented tasks and cross-entropy error for classification-oriented tasks).

Therefore, if opt for the default DeepSense configuration, we need only to set the number of inputs, K, preprocess the input sensor measurements, and identify the type of task (i.e., regression-oriented versus classification-oriented).

The pre-processing is simple, as stated at the beginning of Section 3. We just need to align and chunk the sensor measurements, and apply Fourier transform to each sensor chunk. For each sensor, we stack these frequency domain outputs into $d^{(k)} \times 2f \times T$ tensor $\mathbf{X}^{(k)}$, where $d^{(k)}$ is the sensor measurement dimension, f is the frequency domain dimension, and T is the number of time intervals.

To identify the number of sensor inputs K, we usually set K to be the number of different sensing modalities available. If there exist two or more sensors of the same modality (e.g., two accelerometers or three microphones), we just treat them as one multi-dimensional sensor and set its measurement dimension accordingly.

For the cost function, we can design our own cost function other than the default one. We denote our DeepSense model as function $\mathcal{F}(\cdot)$, and a single training sample pair as $(\mathcal{X}, \mathbf{y})$. We can express the cost function as:

$$\mathcal{L} = \ell(\mathcal{F}(\mathcal{X}), \mathbf{y}) + \sum_j \lambda_j P_j \tag{1}$$

where $\ell(\cdot)$ is the loss function, P_j is the penalty or regularization function, and λ_j controls the importance of the penalty or regularization term.

4.2 Customize Mobile Sensing Tasks

In this section, we provide three instances of customizing DeepSense for specific mobile computing applications used in our evaluation.

Car tracking with motion sensors (CarTrack): In this task, we apply accelerator, gyroscope, and magnetometer to track the trajectory of a car without initial speed. Therefore, according to our general customization process, carTrack is a regression-oriented problem with $K = 3$ (i.e. accelerometer, gyroscope, and magnetometer). Instead of applying default mean square error loss function, we design our own cost function according to Equation (1).

During the training step, the ground-truth 2D displacement of car in each time interval, $\mathbf{y}$, is obtained by GPS signal, where $\mathbf{y}_{[t]}$ denotes the 2D displacement in time interval t. Yet a problem is that GPS signal also contains noise. Training the DeepSense model to recover the displacement obtained from by GPS signal will generate sub-optimal results. We apply Kalman filter to covert displacement $\mathbf{y}_{[t]}$ into a 2D Gaussian distribution $\mathbf{Y}_{[t]}(\cdot)$ with mean value $\mathbf{y}^{(t)}$ in time interval t. Therefore, we use negative log likelihood as loss function $\ell(\cdot)$ with additional penalty terms:

$$\begin{aligned}\mathcal{L}=&-\log\left(\mathbf{Y}_{[t]}\left(\mathcal{F}(\mathcal{X})_{[t]}\right)\right)\\&+\sum_{t=1}^{T}\lambda\cdot\max\left(0,\cos(\theta)-S_c\left(\mathcal{F}(\mathcal{X})_{[t]},\mathbf{y}^{(t)}\right)\right)\end{aligned}$$

where $S_c(\cdot,\cdot)$ denotes the cosine similarity, the first term is the negative log likelihood loss function, and the second term is a penalty term controlled by parameter λ. If the angle between our predicted displacement $\mathcal{F}(\mathcal{X})_{[t]}$ and $\mathbf{y}^{(t)}$ is larger than a pre-defined margin $\theta \in [0, \pi)$, the cost function will get a penalty. We introduce the penalty, because we find that predicting a correct direction is more important during the experiment, as described in Section 5.4.1.

Heterogeneous Human activity recognition (HHAR): In this task, we perform leave-one-user-out cross-validation on human activity recognition task with accelerometer and gyroscope measurements. Therefore, according to our general customization process, HHAR is a classification-oriented problem with $K = 2$ (accelerometer and gyroscope). We use the default cross-entropy cost function as the training objective.

$$\mathcal{L} = H(\mathbf{y}, \mathcal{F}(\mathcal{X}))$$

where $H(\cdot,\cdot)$ is the cross entropy for two distributions.

User Identification with motion analysis (UserID): In this task, we perform user identification with biometric motion analysis. We classify users' identity according to accelerometer and gyroscope measurements. Similarly, according to our general customization process, UserID is a classification-oriented problem with $K = 2$ (accelerometer and gyroscope). Similarly as above, we use the default cross-entropy cost function as the training objective.

For further adapting the DeepSense architecture for a specific mobile sensing task, please refer to Section 6 for the discussion about architecture modification.

5. EVALUATION

In this section, we evaluate DeepSense on three mobile computing tasks. We first introduce the experimental setup for each, including datasets and baseline algorithms. We then evaluate the three tasks based on accuracy, energy, and latency. We use the abbreviations, CarTrack, HHAR, and UserID, as introduced in Section 4.2, to refer to the aforementioned tasks.

5.1 Data Collection and Datasets

For the CarTrack task, we collect 17,500 phone-miles worth of driving data. Namely, we collect around 500 driving hours in total using three cars fitted with 20 mobile phones in the Urbana-Champaign area. Mobile devices include Nexus 5, Nexus 4, Galaxy Nexus, and Nexus S. Each mobile device collects measures of accelerometer, gyroscope, magnetometer, and GPS. GPS measurements are collected roughly every second. Collection rates of other sensors are set to their highest frequency. After obtaining the raw sensor measurements, we first segment them into data samples. Each data sample is a zero-speed to zero-speed journey, where the start and termination are detected when there are at least three consecutive zero GPS speed readings. Each data sample is then separated into time intervals according to the GPS measurements. Hence, every GPS measurement is an indicator of the end of a time interval. In addition, each data sample contains one additional time interval with zero speed at the beginning. Furthermore, for each time interval, GPS latitude and longitude are converted into map coordinates, where the origin of coordinates is the position at the first time interval. Fourier transform is applied to each sensor measurement in each time interval to obtain the frequency response of the three sensing axes. The frequency responses of the accelerator, gyroscope, and magnetometer at each time interval are then composed into the tensors as DeepSense inputs. At last, for evaluation purposes, we apply a Kalman filter to coordinates obtained by the GPS signal, and generate the displacement distribution of each time interval. The results serve as ground truth for training.

For both the HHAR and UserID tasks, we use the dataset collected by Allan et al. [39]. This dataset contains readings from two motion sensors (accelerometer and gyroscope). Readings were recorded when users executed activities scripted in no specific order, while carrying smartwatches and smartphones. The dataset contains 9 users, 6 activities (biking, sitting, standing, walking, climbStair-up, and climbStair-down), and 6 types of mobile devices. For both tasks, accelerometer and gyroscope measurements are model inputs. However, for HHAR, activities are used as labels, and for UserID, users' unique IDs are used as labels. We segment raw measurements into 5-second samples. For DeepSense, each sample is further divided into time intervals of length τ, as shown in Figure 1. We take $\tau = 0.25$ s. Then we calculate the frequency response of sensors for each time interval, and compose results from different time intervals into tensors as inputs.

5.2 Evaluation Platforms

Our evaluation experiments are conducted on two platforms: Nexus 5 with Qualcomm Snapdragon 800 SoC [2] and Intel Edison Compute Module [1]. We train DeepSense on Desktop with GPU. And trained DeepSense models are run solely on mobile with CPU: quad core 2.3 GHz Krait 400 CPU on Nexus 5 and dual-core 500 MHz Atom processor on Intel Edison. In this paper, we do not exploit the additional computation power of mobile GPU and DSP units [21].

5.3 Algorithms in Comparison

We evaluate our DeepSense model and compare it with other competitive algorithms in three tasks. There are three global baselines, which are the variants of DeepSense model by removing one design component in the architecture. The other baselines are specifically designed for each single task.

DS-singleGRU: This model replaces the 2-layer stacked GRU with a single-layer GRU with larger dimension, while keeping the number of parameters. This baseline algorithm is used to verify the efficiency of increasing model capacity by staked recurrent layer.

DS-noIndvConv: In this mode, there are no individual convolutional subnets for each sensor input. Instead, we concatenate the

input tensors along the first axis (i.e., the input measurement dimension). Then, for each time interval, we have a single matrix as the input to the merge convolutional subnet directly.

DS-noMergeConv: In this variant, there are no merge convolutional subnets at each time interval. Instead, we flatten the output of each individual convolutional subnet and concatenate them into a single vector as the input of the recurrent layers.

CarTrack Baseline:

• **GPS:** This is a baseline measurement that is specific to the CarTrack problem. It can be viewed as the ground truth for the task, as we do not have other means of more accurately acquiring cars' locations. In the following experiments, we use the GPS module in Qualcomm Snapdragon 800 SoC.

• **Sensor-fusion:** This is a sensor fusion based algorithm. It combines gyroscope and accelerometer measurements to obtain the pure acceleration without gravity. It uses accelerometer, gyroscope, and magnetometer to obtain absolute rotation calibration. Android phones have proprietary solutions for these two functions [28]. The algorithm then applies double integration on pure acceleration with absolute rotation calibration to obtain the displacement.

• **eNav (w/o GPS):** eNav is a map-aided car tracking algorithm [17]. This algorithm constrains the car movement path according to a digital map, and computes moving distance along the path using double integration of acceleration derived using principal component analysis that removes gravity. The original eNav uses GPS when it believes that dead-reckoning error is high. For fairness, we modified eNav to disable GPS.

HHAR Baselines:

• **HAR-RF:** This algorithm [39] selects all popular time-domain and frequency domain features from [12] and ECDF features from [15], and uses random forest as classifier.

• **HAR-SVM:** Feature selection of this model is same as the HAR-RF model. But this model uses support vector machine as classifier [39].

• **HRA-RBM:** This model is based on stacked restricted Boltzmann machines with frequency domain representations as inputs [6].

• **HRA-MultiRBM:** For each sensor input, the model processes it with a single stacked restricted Boltzmann machine. Then it uses another stacked restricted Boltzmann machine to merge the results for activity recognition [34].

UserID Baselines:

• **GaitID:** This model extracts the gait template and identifies user through template matching with support vector machine [40].

• **IDNet:** This model first extracts the gait template, and extracts template features with convolutional neural networks. Then this model identifies user through support vector machine and integrates multiple verifications with Wald's probability ratio test [13].

5.4 Effectiveness

In this section, we will discuss the accuracy and other related performance metrics of the DeepSense model, compared with other baseline algorithms.

5.4.1 CarTrack

We use 253 zero-speed to zero-speed car driving examples to evaluate the CarTrack task. The histogram of evaluation data driving distance is illustrated in Fig. 2.

During the whole evaluation, we regard filtered GPS signal as ground truth. CarTrack is a regression problem. Therefore, we first evaluate all algorithms with mean absolute error (MAE) between predicted and true final displacements with 95% confidence interval except for the eNav (w/o GPS) algorithm, which is a map-aided algorithm without tracking real trajectories. The results about mean absolute errors are illustrated in the second column of Table 1.

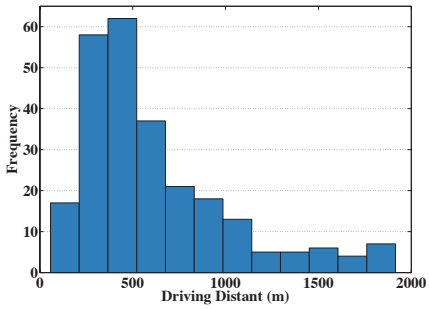


Figure 2: Histogram of Driving Distance.

Table 1: CarTrack Task Accuracy

	MAE (meter)	Map-Aided Accuracy
DeepSense	$\mathbf{40.43 \pm 5.24}$	$\mathbf{93.8}\%$
DS-SingleGRU	44.97 ± 5.80	90.2%
DS-noIndvConv	52.15 ± 6.24	88.3%
DS-noMergeConv	53.06 ± 6.59	87.5%
Sensor-fusion	606.59 ± 56.57	
eNav (w/o GPS)		6.7%

Compared with senior-fusion algorithm, DeepSense reduces the tracking error by an order of magnitude, which is mainly attributed to its capability to learn the composition of noise model and physical laws. Then, we compare our DeepSense model with three variants as mentioned before. The results show the effectiveness of each designing component of our DeepSense model. The individual and merge convolutional subnets learn the interaction within and among sensor measurements respectively. The stacked recurrent structure increases the capacity of model more efficiently. Removing any component will cause performance degradation.

DeepSense model achieves $40.43 \pm 5.24m$ mean absolute error. This is almost equivalent to half of traditional city blocks ($80m \times 80m$), which means that, with the aid of map and the assumption that car is driving on roads, DeepSense model has a high probability to provide accurate trajectory tracking. Therefore, we propose a naive map-aided track method here. For each segment of original tracking trajectory, we assign them to the most probable road segment on map (i.e., the nearest road segment on map). We then compare the resulted trajectory with ground truth. If all the trajectory segments are the same as the ground truth, we regard it as a successful tracking trajectory. Finally, we compute the percentage of successful tracking trajectories as accuracy. eNav (w/o GPS) is a map-aided algorithm, so we directly compare the trajectory segments. Sensor-fusion algorithm generates tracking errors that are comparable to driving distances, so we exclude it from the comparison. We show the accuracy of map-aided versions of algorithms in the third column of Table 1. DeepSense outperforms eNav (w/o GPS) with a large margin, because eNav (w/o GPS) intrinsically depends on occasional on-demand GPS samples to correct tracking error.

We next examine how tracking performance is affected by driving distances. We first sort all evaluation samples according to driving distance. Then we separate them into 10 groups with 200m step size. Finally, we compute mean absolute error and accuracy of map-aided track for DeepSense algorithm separately for each group. We illustrate the results in Fig. 3. For the mean absolute er-

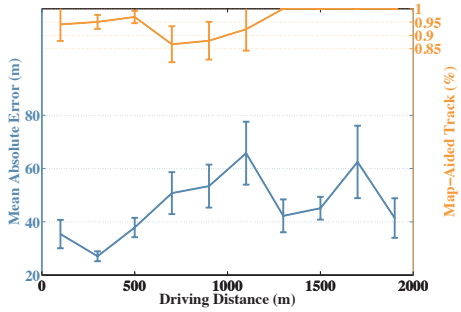
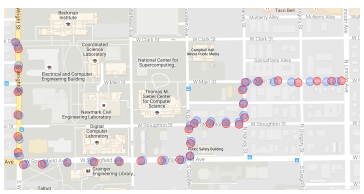
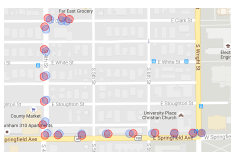


Figure 3: Performance over driving distance.

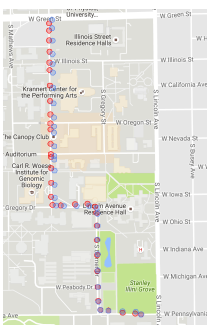
ror metric, driving longer distance generally results in large error, *but the error does not accumulate linearly over distance*. There are mainly two reasons for this phenomenon. On one hand, we observe that the error of our predicted trajectory usually occurs during the beginning of the driving, where uncertainty in predicting driving direction is the major cause. This is also the motivation that we add the penalty term for cost function in Section 4.2. On the other hand, longer-driving cases in our testing samples are more stable, because we extract the trajectory from zero-speed to zero-speed. For the map-aided track, longer driving distances even yields slightly better accuracy. This is because long-distance trajectory usually contains long trajectory segments, which can help to find the ground truth on the map.



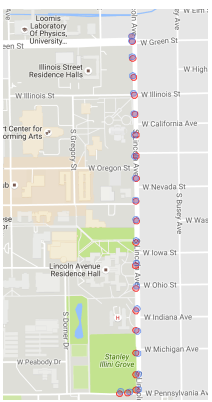
(a) Trajectory a



(b) Trajectory b



(c) Trajectory c



(d) Trajectory d

Figure 4: Examples of tracking trajectory without the help of map: Blue trajectory (DeepSense) and Red trajectory (GPS)

Finally, some our DeepSense tracking results (without the help of map and with downsampling) are illustrated in Fig. 4.

5.4.2 HHAR

For HHAR task, we perform leave-one-user-out evaluation (i.e., leaving the whole data from one user as testing data) on datasets consisting of 9 users, which are labelled from a to i. We illustrate the result of evaluations according to three metrics: accuracy, macro F_1 score, and micro F_1 score with 95% confidence interval in Fig. 5.

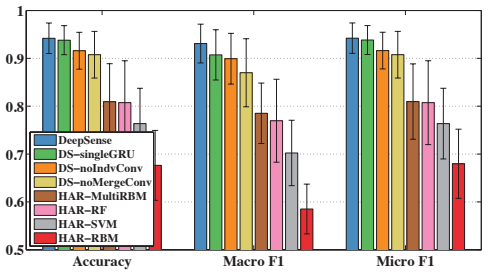


Figure 5: Performance metrics of HHAR task.

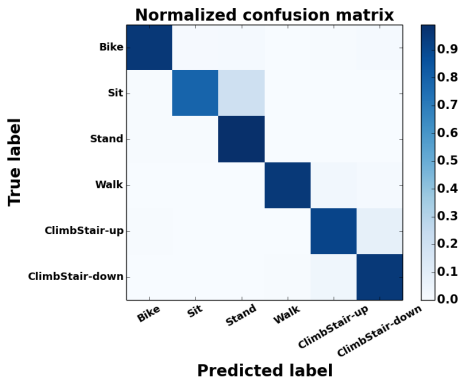


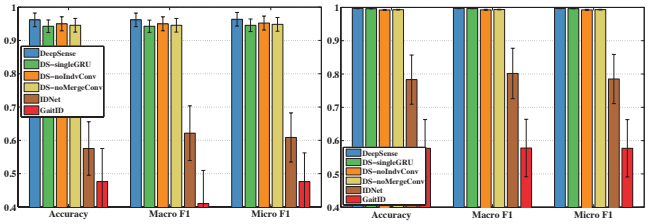
Figure 6: Confusion matrix of HHAR task.

The DeepSense based algorithms (including DeepSense and three variants) outperform other baseline algorithms with a large margin (i.e., at least 10%). Compared with two hand-crafted feature based algorithms HAR-RF and HAR-SVM, DeepSense model can automatically extract more robust features, which generalize better to the user who does not appear in the training set. Compared with a deep model, such as HAR-RBM and HAR-MultiRBM, DeepSense model exploit local structures within sensor measurements, dependency along time, and relationships among multiple sensors to generate better and more robust features from data. Compared with three variants, DeepSense still achieves the best performance (accuracy: 0.942 ± 0.032, macro F_1: 0.931 ± 0.041, and micro F_1: 0.942 ± 0.032). This reinforces the effectiveness of our design components in DeepSense model.

Then we illustrate the confusion matrix of best-performing DeepSense model in Fig. 6. Predicting Sit as $Stand$ is the largest error. It is hard to classify these two, because two activities should have similar motion sensor measurements by nature, especially when we have no prior information about testing users. In addition, the algorithm has a minor error about misclassification between $ClimbStair\text{-}up$ and $ClimbStair\text{-}down$.

5.4.3 UserID

This task focuses on user identification with biometric motion analysis. We evaluate all algorithms with 10-fold cross validation. We illustrate the result of evaluations according to three metrics: accuracy, macro F_1 score, and micro F_1 score with 95% confidence interval in Fig. 7. Specifically, Fig. 7a shows the results when algorithms observe 1.25 seconds of evaluation data, Fig. 7b shows the results when algorithms observe 5 seconds of evaluation data.



(a) 5 time intervals: 1.25s (b) 20 time intervals: 5s

Figure 7: Performance metrics of UserID task.

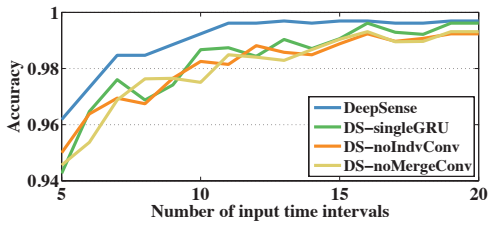


Figure 8: Accuracy over input measurement length of UserID task.

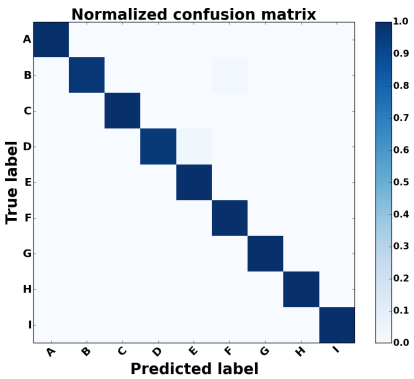


Figure 9: Confusion matrix of UserID task.

Figure 10: Confusion matrix of HHAR and UserID tasks.

DeepSense and three variants outperform other baseline algorithms with a large margin again (i.e. at least 20%). Compared with the template extraction and matching method, GaitID, DeepSense model can automatically extract distinct features from data, which fit well to not only walking but also all other kinds of activities. Compared with method that first extracts templates and then apply neural network to learn features, IDNet, DeepSense solves the whole task in the end-to-end fashion. We eliminate the manually processing part and exploit local, global, and temporal relationships through our architecture, which results better performance. In this task, although the performance of different variants is similar when observing data with 5 seconds, DeepSense still achieves the best performance (accuracy: 0.997 ± 0.001, macro F_1: 0.997 ± 0.001, and micro F_1: 0.997 ± 0.001).

We further compare DeepSense with three variants by changing the number of evaluation time intervals from 5 to 20, which corresponds to around 1 to 5 seconds. We compute the accuracy for each case. The results illustrated in Fig. 8 suggest that DeepSense performs better than all the other variants with a relatively large margin when algorithms observe sensing data with shorter time. This indicates the effectiveness of design components in DeepSense.

Then we illustrate the confusion matrix of best-performing DeepSense model when observing sensing data with 5 seconds in Fig. 9. It shows that the algorithm gives a pretty good result. On average, only about two misclassifications appear during each testing.

5.5 Latency and Energy

Final, we examine the computation latency and energy consumption of DeepSense—stereotypical deep learning models are traditionally power hungry and time consuming—we illustrate, through our careful measurements in all three example application scenarios, the feasibility of directly implementing and deploying DeepSense on mobile devices without any additional optimization.

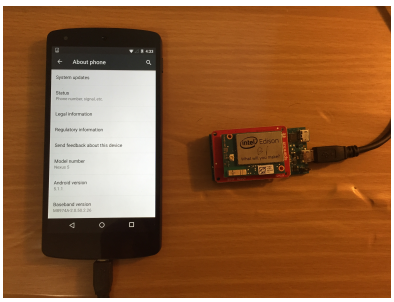
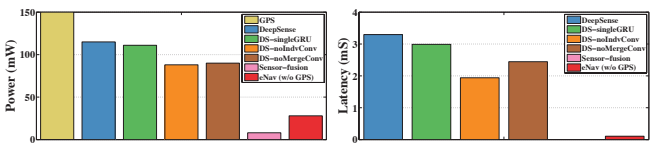
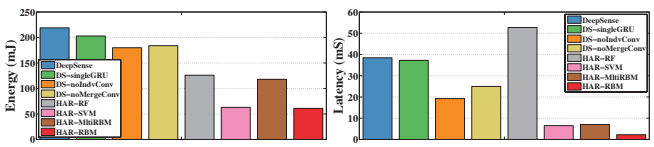


Figure 11: Test Platforms: Nexus5 and Intel Edison.



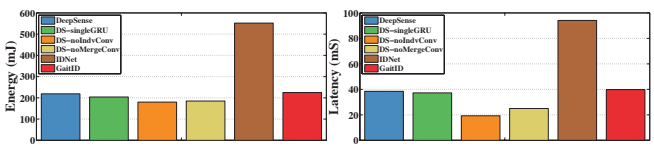
(a) Power (b) Latency

Figure 12: Power and Latency of carTrack solutions on Nexus 5



(a) Energy (b) Latency

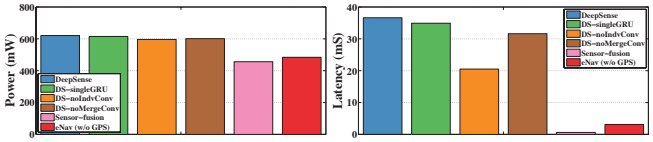
Figure 13: Energy and Latency of HHAR solutions on Nexus 5



(a) Energy (b) Latency

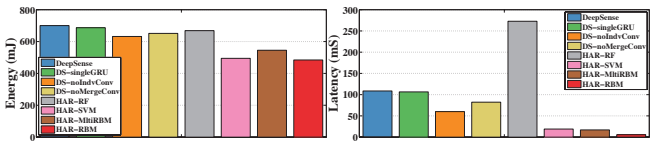
Figure 14: Energy and Latency of UserID solutions on Nexus 5

Experiments measure the whole process on smart devices including reading the raw sensor inputs and are conducted on two kinds of devices: Nexus 5 and Intel Edison, as shown in Fig. 11. The energy consumption of applications on Nexus 5 is measured by PowerTutor [44], while the energy consumption of Intel Edison is measured by an external power monitor. The evaluations of energy and latency on Nexus 5 are shown in Fig. 12, 13, and 14, and



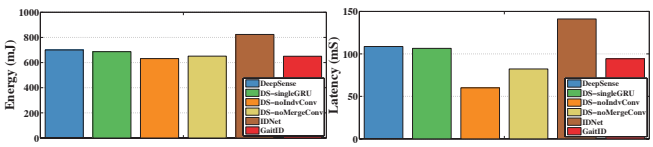
(a) Power (b) Latency

Figure 15: Power and Latency of carTrack solutions on Edison



(a) Energy (b) Latency

Figure 16: Energy and Latency of HHAR solutions on Edison



(a) Energy (b) Latency

Figure 17: Energy and Latency of UserID solutions on Edison

Intel Edison Fig. 15, 16, and 17. Since algorithms for carTrack are designed to report position every second, we show the power consumption in Fig. 12a and 15a. Other two tasks are not periodical tasks by nature. Therefore, we show the per-inference energy consumption in Fig. 13a, 16a, 14a, and 17a. For experiments on Intel Edison, notice that we measured total energy consumption, containing 419mW idle-mode power consumption.

For the carTrack task, all DeepSense based models consume a bit less energy compared with 1-Hz GPS samplings on Nexus 5. The running times are measured in the order of microsecond on both platforms, which meets the requirement of per-second measurement.

For the HHAR task, all DeepSense based models take moderate energy and low latency to obtain one classification prediction on two platforms. An interesting observation is that HHAR-RF, a random forest model, has a relatively longer latency. This is due to the fact that random forest is an ensemble method, which involves combining a bag of individual decision tree classifiers.

For the UserID task, except for the IDNet baseline, all other algorithms show similar running time and energy consumption on two platforms. IDNet contains both a multi-stage pre-processing process and a relative large CNN, which takes longer time and more energy to compute in total.

6. DISCUSSION

This paper focuses on solving different mobile sensing and computing tasks in a unified framework. DeepSense is our solution. It is a framework that requires only a few steps to be customized into particular tasks. During the customization steps, we do not tailor the architecture for different tasks in order to lessen the requirement of human efforts while using the framework. However, particular changes to the architecture can bring additional performance gains to specific tasks.

One possible change is separating noise model and physical laws for regression-oriented tasks. The original DeepSense directly learns the composition of noise model and physical laws, providing the capability of automatically understanding underlying physical process from data. However, if we know exactly the physical process, we can use DeepSense as a powerful denoising component, and apply physical laws to the outputs of DeepSense.

The other possible change is removing some design components to trade accuracy for energy. In our evaluations, we show that some variants take acceptable degradation on accuracy with less energy consumption. The basic principle of removing design components is based on their functionalities. Individual convolutional subnets explore relationship within each sensor; merge convolutional subnet explores relationship among different sensors; and stacked RNN increases the model capacity for exploring relationship over time. We can choose to omit some components according to the demands of particular tasks.

In addition, although our three evaluation tasks focus mainly on motion sensors, which are the most widely deployed sensors, we can directly apply DeepSense to almost all other sensors, such as microphone, Wi-Fi signals, Barometer, and light sensor. We need further study on applying DeepSense to explore new applications on smart devices.

At last, for a particular sensing task, if there is drastic change in the physical environment, DeepSense might need to be re-trained with new data. However, on one hand, the traditional solution with pre-defined noise model and physical laws (or hand-crafted features) would also need redesigns anyways. On the other hand, an existing trained DeepSense framework can serve as a good initialization stage for the new training process that aids in optimization and reduce generalization error [10].

7. CONCLUSION

In this paper we introduced our unified DeepSense framework for mobile sensing and computing tasks. DeepSense integrates convolutional and recurrent neural networks to exploit different types of relationships in sensor inputs, thanks to which, it is able to learn the composition of physical laws and noise model for regression-oriented problems, and automatically extract robust and distinct features on local, global, and temporal domains to effectively carry out classification tasks—the two major focuses in mobile sensing literature. We evaluated DeepSense via three representative mobile sensing tasks, where DeepSense outperformed state of the art baselines by significant margins while still claiming its mobile-feasibility through moderate energy consumption and low latency on both mobile and embedded platforms. Our experience with the multiple DeepSense variants also provided us with valuable insights and promising guidelines in the opportunities of further framework adaptation and customization for a wide range of applications.

8. ACKNOWLEDGMENTS

We sincerely thank the anonymous reviewers for their invaluable comments. Research reported in this paper was sponsored in part by NSF under grants CNS 16-18627 and CNS 13-20209 and in part by the Army Research Laboratory under Cooperative Agreement W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory, NSF, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

9. REFERENCES

[1] Intel edison compute module. http://www.intel.com/content/dam/support/us/en/documents/edison/sb/edison-module_HG_331189.pdf.

[2] Qualcomm snapdragon 800 processor. https://www.qualcomm.com/products/snapdragon/processors/800.

[3] W. T. Ang, P. K. Khosla, and C. N. Riviere. Nonlinear regression model of a low-g mems accelerometer. *IEEE Sensors Journal*, 2007.

[4] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2014.

[5] I. G. Y. Bengio and A. Courville. Deep learning. Book in preparation for MIT Press, 2016.

[6] S. Bhattacharya and N. D. Lane. From smart to deep: Robust activity recognition on smartwatches using deep learning. In *PerCom Workshops*, 2016.

[7] G. Chandrasekaran, T. Vu, A. Varshavsky, M. Gruteser, R. P. Martin, J. Yang, and Y. Chen. Tracking vehicular speed variations by warping mobile phone signal strengths. In *PerCom*, 2011.

[8] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv:1412.3555*, 2014.

[9] T. Cooijmans, N. Ballas, C. Laurent, and A. Courville. Recurrent batch normalization. *arXiv:1603.09025*, 2016.

[10] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE TASLP*, 2012.

[11] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015.

[12] D. Figo, P. C. Diniz, D. R. Ferreira, and J. M. Cardoso. Preprocessing techniques for context recognition from accelerometer data. *Pers. Ubiquit. Comput.*, 2010.

[13] M. Gadaleta and M. Rossi. Idnet: Smartphone-based gait recognition with convolutional neural networks. *arXiv:1606.03238*, 2016.

[14] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *arXiv:1503.04069*, 2015.

[15] N. Y. Hammerla, R. Kirkham, P. Andras, and T. Ploetz. On preserving statistical characteristics of accelerometry data using their empirical cumulative distribution. In *ISWC*, 2013.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

[17] S. Hu, L. Su, S. Li, S. Wang, C. Pan, S. Gu, M. T. Al Amin, H. Liu, S. Nath, et al. Experiences with enav: a low-power vehicular navigation system. In *UbiComp*, 2015.

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

[19] L. Kang, B. Qi, D. Janecek, and S. Banerjee. Ecodrive: A mobile sensing and control system for fuel efficient driving. In *MobiCom*, 2015.

[20] J. Ko, C. Lu, M. B. Srivastava, J. A. Stankovic, A. Terzis, and M. Welsh. Wireless sensor networks for healthcare. *Proc. IEEE*, 2010.

[21] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *IPSN*, 2016.

[22] N. D. Lane, P. Georgiev, and L. Qendro. Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *UbiComp*, 2015.

[23] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell. A survey of mobile phone sensing. *IEEE Commun. Mag.*, 2010.

[24] C.-Y. Li, C.-H. Yen, K.-C. Wang, C.-W. You, S.-Y. Lau, C. C.-H. Chen, P. Huang, and H.-H. Chu. Bioscope: an extensible bandage system for facilitating data collection in nursing assessments. In *UbiComp*, 2014.

[25] T. Li, C. An, Z. Tian, A. T. Campbell, and X. Zhou. Human sensing using visible light communication. In *MobiCom*, 2015.

[26] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong. Redeye: analog convnet image sensor architecture for continuous mobile vision. In *ISCA*, pages 255–266, 2016.

[27] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao. Energy-accuracy aware localization for mobile devices. In *MobiSys*, 2010.

[28] G. Milette and A. Stroud. *Professional Android sensor programming*. John Wiley & Sons, 2012.

[29] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury. Tapprints: your finger taps have fingerprints. In *MobiSys*, 2012.

[30] F. J. O. Morales and D. Roggen. Deep convolutional feature transfer across mobile activity recognition domains, sensor modalities and locations. In *ISWC*, 2016.

[31] S. Nath. Ace: exploiting correlation for energy-efficient and continuous context sensing. In *MobiSys*, 2012.

[32] M. Park. *Error analysis and stochastic modeling of MEMS-based inertial sensors for land vehicle navigation applications*. Library and Archives Canada= Bibliothèque et Archives Canada, 2005.

[33] M. Rabbi, M. H. Aung, M. Zhang, and T. Choudhury. Personal sensing: Understanding mental health using ubiquitous sensors and machine learning. In *UbiComp*, 2015.

[34] V. Radu, N. D. Lane, S. Bhattacharya, C. Mascolo, M. K. Marina, and F. Kawsar. Towards multimodal deep learning for activity recognition on mobile devices. In *UbiComp: Adjunct*, 2016.

[35] Y. Ren, Y. Chen, M. C. Chuah, and J. Yang. Smartphone based user verification leveraging gait recognition for mobile healthcare systems. In *SECON*, 2013.

[36] O. Rippel, J. Snoek, and R. P. Adams. Spectral representations for convolutional neural networks. In *NIPS*, 2015.

[37] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Trans Sig. Process.*, 1997.

[38] N. Srivastava and R. R. Salakhutdinov. Multimodal learning with deep boltzmann machines. In *NIPS*, 2012.

[39] A. Stisen, H. Blunck, S. Bhattacharya, T. S. Prentow, M. B. Kjærgaard, A. Dey, T. Sonne, and M. M. Jensen. Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition. In *Sensys*, 2015.

[40] H. M. Thang, V. Q. Viet, N. D. Thuc, and D. Choi. Gait identification using accelerometer on mobile phone. In *ICCAIS*, 2012.

[41] C. Wang, X. Guo, Y. Wang, Y. Chen, and B. Liu. Friend or foe?: Your wearable devices reveal your personal pin. In *AsiaCCS*, 2016.

[42] C. Xu, S. Li, G. Liu, Y. Zhang, E. Miluzzo, Y.-F. Chen, J. Li, and B. Firner. Crowd++: unsupervised speaker count with smartphones. In *UbiComp*, 2013.

[43] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv:1409.2329*, 2014.

[44] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *CODES+ISSS*, 2010.

[45] Y. Zhao, S. Li, S. Hu, L. Su, S. Yao, H. Shao, and T. Abdelzaher. Greendrive: A smartphone-based intelligent speed adaptation system with real-time traffic signal prediction. In *ICCPS*, 2017.

[46] Y. Zhu, Y. Zhu, B. Y. Zhao, and H. Zheng. Reusing 60ghz radios for mobile radar imaging. In *MobiCom*, 2015.