Convolutional Neural Networks for Human Activity Recognition using Mobile Sensors

Conference Paper · November 2014

DOI: 10.4108/icst.mobicase.2014.257786

Ming Zeng
Carnegie Mellon University
14 PUBLICATIONS **65** CITATIONS
SEE PROFILE

Ole J. Mengshoel
Carnegie Mellon University
94 PUBLICATIONS **1,004** CITATIONS
SEE PROFILE

Pang Wu
Apple Inc.
11 PUBLICATIONS **192** CITATIONS
SEE PROFILE

Convolutional Neural Networks for Human Activity Recognition using Mobile Sensors

Ming Zeng, Le T. Nguyen, Bo Yu, Ole J. Mengshoel, Jiang Zhu, Pang Wu, Joy Zhang

Department of Electrical and Computer Engineering
Carnegie Mellon University
Moffett Field, CA, USA
{ming.zeng,le.nguyen,bo.yu,ole.mengshoel}@sv.cmu.edu
{jiang.zhu,pang.wu,joy.zhang}@sv.cmu.edu

Abstract. A variety of real-life mobile sensing applications are becoming available, especially in the life-logging, fitness tracking and health monitoring domains. These applications use mobile sensors embedded in smart phones to recognize human activities in order to get a better understanding of human behavior. While progress has been made, human activity recognition remains a challenging task. This is partly due to the broad range of human activities as well as the rich variation in how a given activity can be performed. Using features that clearly separate between activities is crucial. In this paper, we propose an approach to automatically extract discriminative features for activity recognition. Specifically, we develop a method based on Convolutional Neural Networks (CNN), which can capture local dependency and scale invariance of a signal as it has been shown in speech recognition and image recognition domains. In addition, a modified weight sharing technique, called partial weight sharing, is proposed and applied to accelerometer signals to get further improvements. The experimental results on three public datasets, Skoda (assembly line activities), Opportunity (activities in kitchen), Actitracker (jogging, walking, etc.), indicate that our novel CNN-based approach is practical and achieves higher accuracy than existing state-of-the-art methods.

Key words: Activity Recognition, Deep Learning, Convolutional Neural Network

1 Introduction

In the recent years, the rapid spread of mobile devices with sensing capabilities has created a huge demand for human activity recognition (AR). Applications that can benefit from AR include daily lifelogging, healthcare, senior care, personal fitness, etc. [7, 32, 31, 9]. As a result, many approaches were proposed for the recognition of a wide range of activities [8, 15, 10, 23].

Feature extraction is one of the key steps in AR, since it can capture relevant information to differentiate among various activities. The accuracy of AR approaches greatly depends on features extracted from raw signals such as accelerometer readings [34]. Many existing AR approaches often rely on statistical features such as mean, variance, entropy or correlation coefficients [3]. Feature extraction is proposed from the frequency domain using FFT [17]. Prior works have shown that some of these heuristically-defined features perform well in recognizing one activity, but badly for others [15]. Therefore, given an application scenario and a set of target activities, one can select a subset of features to optimize the activity recognition performance [34, 15].

Designing hand-crafted features in a specific application requires domain knowledge [23]. This problem is not unique to activity recognition. It has been well-studied in other research areas such as image recognition [22], where different types of features need to be extracted when trying to recognize a handwriting as opposed to recognizing faces. In recent years, due to advances of the processing capabilities, a large amount of Deep Learning (DL) techniques have been developed and successful applied in recognition tasks [2, 28]. These techniques allow an automatic extraction of features without any domain knowledge.

In this work, we propose an approach based on Convolutional Neural Networks (CNN) [2] to recognize activities in various application domains. There are two key advantages when applying CNN to AR:

– **Local Dependency:** CNN captures local dependencies of an activity signals. In image recognition tasks, the nearby pixels typically have strong relationship with each other. Similarly, in AR given an activity the nearby acceleration readings are likely to be correlated.
– **Scale Invariance:** CNN preserves feature scale invariant. In image recognition, the training images might have different scales. In AR, a person may walk with different paces (e.g., with different motion intensity).

We summarize the key contributions of this work as follows:

– We propose an approach based on CNN to extract human activity features without any domain knowledge.
– The proposed approach can capture the local dependencies and scale-invariant features of activity signals. Thus, variations of the same activity can be effectively captured through the extracted features.
– We present the experimental results on three public datasets collected in different domains. The results shown that the proposed approach outperforms the state-of-the-art methods.

The rest of this paper is organized as follows: Section 2 presents related work; Section 3 describes our CNN-based method for activity recognition and improvement; Section 4 presents our experimental results to demonstrate its applications. Finally, we conclude the study in Section 5.

2 Related Work

2.1 Feature Extraction for Activity Recognition

AR can be consider as a classification problem, where the input are time series signals and the output is an activity class label. Fig 1 shows the activity recognition process, which is divided into training phase and classification phase. In the training phase, we extract features from the raw time series data. These features are then used to train a classification model. In the classification phase, we first extract features from unseen raw data and then use the trained prediction model to predict an activity label.

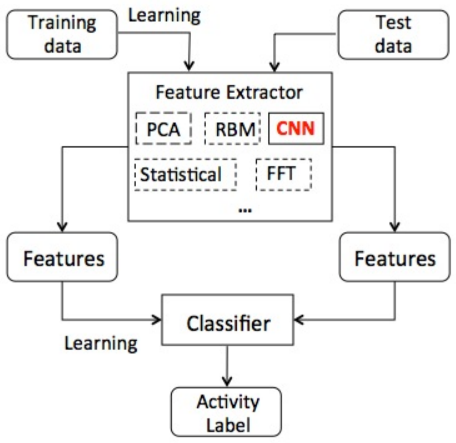

Fig. 1. Feature extraction is one of the key components of activity recognition.

Feature extraction for AR is an important task, which has been studied for many years. Statistical features such as mean, standard deviation, entropy and correlation coefficients, etc. are the most widely used hand-crafted features in AR [8]. Fourier transform and wavelet transform [27] are another two commonly used hand-crafted features, while discrete cosine transform (DCT) have also been applied with promising results [11], as well as auto-regressive model coefficients [12]. Recently, time-delay embeddings [10] have been applied for activity recognition. It adopts nonlinear time series analysis to extract features from time series and shows a significant improvement on periodic activities recognition (cycling involves a periodic, roughly two-dimensional leg movement). However, the features from time-delay embedding are less suitable for non-periodic activities.

In the recent years, some approaches such as principal component analysis (PCA) [11] or restricted Boltzmann machine (RBM) [23] were applied to adapt the feature extraction to the dataset, i.e. the mapping from raw signals to features is not predefined, but rather automatically learned from the training dataset. PCA is a well-established technique to discover compact and meaningful representations of raw data without relying on domain knowledge. The PCA feature extraction is conducted in discrete cosine transform (DCT) domain [11]. After conducting PCA, the most invariant and discriminating information for recognition is maintained. The PCA based on empirical cumulative distribution function (ECDF) is proposed [23] to preserve structural information of the signal.

2.2 Deep Learning for Feature Learning

Although PCA can learn features in an unsupervised manner, its linear combination of raw features does not have sufficient capability to model complex non-linear dependencies [4]. Therefore, deep neural networks (DNN)[1] have been proposed to extract more meaningful features [23]. The one key difference between traditional neural networks and deep neural networks is that DNNs can have many layers in the networks while traditional neural networks contain three layers at most. A key advantage of DNN is its representation of input features. DNN can model diverse activities with much less training data, it can share similar portions of the input space with some hidden units, while keeping other units sensitive to a subset of the input features that are significant to recognition.

DNN in recent made many breakthroughs in many research areas. The deep architectures can represent complex function compactly, which have been shown to outperform state-of-the-art machine learning algorithms in many applications (such as face detection, speech recognition.) [4]. Fig 2 compares a DNN model with existing approaches.

A statistic feature model can be considered as a model of depth 1, where the output nodes represent pre-defined function such as mean, variance, etc. PCA can be also considered as a model with depth 1, where the output nodes represents the k principal components outputted as a linear combination of the input data. DNN is a model with a a depth of n layers, where the complex dependencies of the input data can be captured through hidden layers with non-linear mapping in layers.

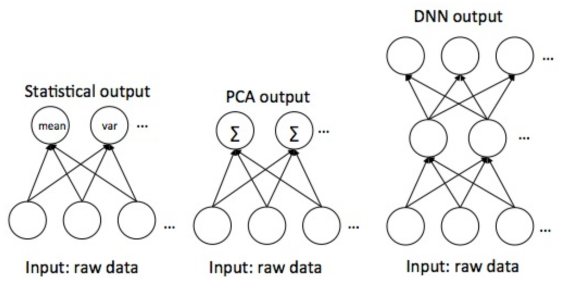

Fig. 2. (a): statistical feature computation, (b): PCA model, (c): DNN model

The Restricted Bozltman Machine (RBM), a particular form of log-linear Markov Random Field (MRF) [13], was proposed as a DNN technique to extract features for AR [23]. It employed Gaussian visible units for the first level and trained the network in a supervised manner. Another DNN model, Shift-Invariant Sparse Coding [29, 5] was used to perform unsupervised learning to train an autoencoder network.

However, RBM [23] and Sparse Coding [29, 5] are fully connected DNN models (as shown in Fig $2(c)$). Therefore, they do not capture local dependencies of the time series signals [1]. Convolution Neural Network (CNN) [20] consists of one or more pairs of

[1] The deep neural networks is built from traditional artificial neural networks (ANN) but conquer its shortcoming. The term “deep” is gained because each layer can be stacked layer by layer. And Geoffrey Hinton and Ruslan Salakhutdinov showed how a many-layered feedforward neural network could be effectively pre-trained one layer at a time [14]. But the traditional ANNs only have three layers at most

convolution and pooling layers[2]. The small local parts of the input were captured by the convolutional layer with a set of local filters. And the pooling layer can preserver the invariant features. Top fully connected layer finally combine inputs from all features to do the classification of the overall inputs. This hierarchical organization generates good results in image processing [18, 16] and speech recognition [1] tasks. In the next section, we will present details of CNN and describe our proposed CNN-based AR approach.

3 CNN-based Activity Recognition

In this section, we discuss our CNN-based feature extraction approach. Fig 3 shows the structure of the proposed approach. Following the settings of [23], given a 3D acceleration time series we use a sliding window with a length of w values and with a certain percentage of overlap to extract input data for the CNN.

Our L-layer CNN-based model has three kinds of layers: 1) An input layer (with units h_i^0) whose values are fixed by the input data; 2) hidden layers (with units h_i^l) whose values are derived from previous layers $l-1$; 3) and output layer (with units h_i^L) whose values are derived from the last hidden layer. The network learns by adjusting a set of weights $w_{i,j}^l$, where $w_{i,j}^l$ is the weight from some input h_i^l's output to some other unit h_j^{l+1}. We use x_i^l to denote the total input to unit u_i^l (ith unit in layer l), and y_i^l denotes the output of unit h_i^l.

3.1 Convolutional Layer

In the following we describe how CNN captures local dependencies and the scale invariant characteristics of the activity signals. In order to capture the local dependencies of the data, one can enforce a local connectivity constraint between units of adjacent layers. For example, in Fig 4 the units (neurons) in the middle layer are only connected to a local subset of units in the input layer. From biology, we know that there are complex arrangement of cells in visual cortex, which are sensitive to small regions of the input, called a receptive field, and are tiled to generate the entire visual field. These filters are local in input space and are thus suited to exploit local correlation hidden in the data, so we also call it local filter. In terms of local filter, the weight of edge connected ith unit with jth, $w_{i,j}$ can be reduced by w_a, and $w_{i,j} = w_{i,j+m} = w_a$, where m is the width of the local filter. In Fig 4, the 1D vector $[w_1, w_2, w_3]$ represents three local filters denoted by different line style, where w_i is weight of edge connecting in two layers. The convolution operation is conducted over the local subset. This topological constraint corresponds to learning a weight matrix with sparsity constraint, which is not only good for extracting local dependencies, but also reduces the computational complexity. The output of such a set of local filters constitute a feature map (Fig 5). At each temporal position, different types of units in different feature maps compute different types of features.

[2] The terms will be defined and discussed in the Section 3

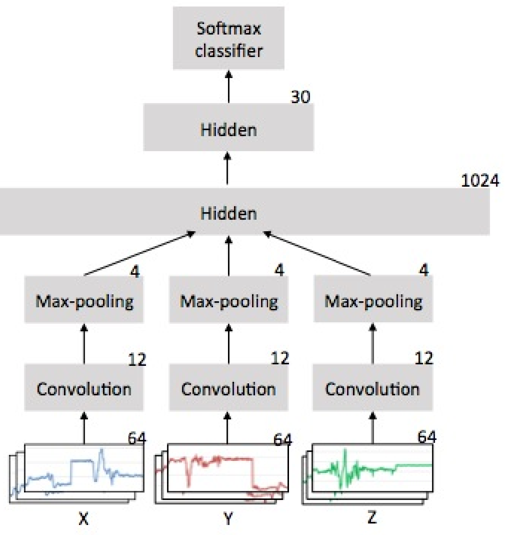

Fig. 3. Structure of CNN for Human Activity Recognition. The dimension of input data is 64, the dimension convolutional output is 12, the dimension max-pooling output is 4. The dimension of two hidden layers is 1024 and 30, respectively. The top layer is a Softmax classifier.

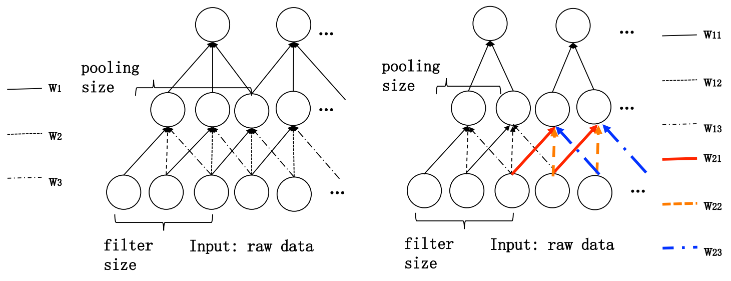

Fig. 4. Left) Traditional weight sharing CNN, Right) Partial weight sharing CNN. Weights denoted by the same line style are shared

Moreover, in order to form a richer representation of the data, the convolutional layers are composed of a set of multiple feature maps (Fig 5), $x^{(\cdot,j)}, j = 1...J$. The following Fig 5 shows two layers of CNN, containing three feature maps $(x^{(0)}, x^{(1)})$ at the left layer and two feature maps at the right layer. Unit's outputs in $x^{(0)}$ and $x^{(1)}$ are computed by convolution operation from units of left layer which fall within their local filter (shown as rectangles in Fig 5). Suppose we have some N unites layer as input which is followed by convolutional layer. If we use m width filter w, the convolutional output will be $(N - m + 1)$ unites. Then the output of convolutional layer l is:

$$x_i^{l,j} = \sigma\left(b_j + \sum_{a=1}^{m} w_a^j x_{i+a-1}^{l-1,j}\right) \tag{1}$$

where $x_i^{l,j}$ is the l convolution layer's output of the jth feature map on the ith unit, and σ is a non-linear mapping, it usually uses hyperbolic tangent function, $tanh(\cdot)$. Take Fig 4 as an example, the first hidden unit of the first local filter is

$$x_{1,1}^{1} = \tanh(w_1^{1,1}x_1^{0,1} + w_2^{1,1}x_2^{0,1} + w_3^{1,1}x_3^{0,1} + b_1)$$

and the second hidden unit of the second local filter is

$$x_{1,1}^{2} = \tanh(w_1^{1,1}x_2^{0,1} + w_2^{1,1}x_3^{0,1} + w_3^{1,1}x_4^{0,1} + b_1)$$

.

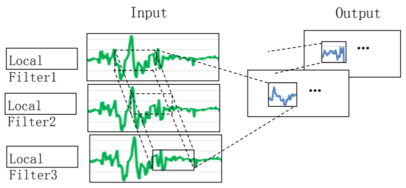

Fig. 5. Feature Map: three feature maps at input layer and two results at output layer

In traditional CNN model [18], each local filter is additionally replicated across the entire input space. That means the weights of local filters are tied and shared by all positions within the whole input space. For example, in Fig 4, weights denoted by the same line style are shared (forced to be identical). The replicated weights allow the features to be detected regardless of the position of the unit, which is also beneficial for scale-invariant preservation.

In image processing task, full weight sharing is suitable because the same image pattern can appear at any position in an image. However, in AR, because different patterns appear in different frame, the signal appearing at different units may behave quite differently. Therefore, it may be better to relax the weight sharing constraint, i.e., weights of the same color and same type are constrained to be identical (means weight sharing, in Fig 4)(Right). This weight sharing strategy is first described in [1], and we called it partial weight sharing in our application. With the partial weight sharing technique, the activation the function in convolution layer is changed as below:

$$x_{i,k}^{l,j} = \sigma\left(b_j + \sum_{a=1}^{m} w_{a,k}^{j} x_{i+(k-1)\times s+a-1}^{l-1,j}\right) \tag{2}$$

where $x_{i,k}^{l,j}$ is one of the ith unit of j feature map of the kth section in the lth layer, and s is the range of section. The difference between Equation 1 and Equation 2 is the range of weight sharing, using the window $(k-1)\times s+i+a$ instead of $i+a$ to conduct the convolution operation.

3.2 Max-Pooling Layer

Once a feature has been detected in the convolutional layer, its exact location becomes less important, as long as its approximate position relative to other features is preserved.

Therefore, an additional layer which performs max-pooling is stacked over convolutional layer to reduce the sensitivity of the output. Scale-invariant feature preservation is another key characteristic of CNN, which is achieved by the max-pooling layer. In the max-pooling layer, features from convolutional layer are split into several partitions, in each partition, we apply max operation to output the values. The activation function in the max-pooling layer in traditional CNN is given by:

$$x_i^{l,j} = max_{k=1}^{r} \left(x_{(i-1)\times s+k}^{l-1,j} \right) \tag{3}$$

Due to the partial weight sharing structure, only local filters that are close to each other share weights and will be aggregated together in the max-pooling layer. The pooling function is slightly different from the traditional pooling function, since the max operation is only carried out in the same shared weight section:

$$x_i^{l,j} = max_{k=1}^{r} \left(x_k^{l-1,j} \right) \tag{4}$$

Thus, filters that work on local time window will provide an efficient way to represent these local structures and their combinations along the whole time axis may be eventually used to recognized different activities.

3.3 Model Structure and Training Process

The CNN can contain one or more pairs of convolution and max-pooling layers, where higher layers use broader filters to process more complex parts of the input. The top layers in CNN are stacked by one or more fully connected normal neural networks. These fully connected neural network are expected to combine different local structures in the lower layers for the final classification purpose.

In this work, we only use one pair convolution and max-pooling layer, and two normal fully connected neural networks. In the training stage, The CNN parameters are estimated by standard forward and backward propagation algorithms to minimize the objective function.

3.4 Training Process

Forward Propagation First, the forward propagation is conducted in convolutional layer with N nodes by equation (2). If we use m filter w, our partial convolutional layer output will be of size $N - m + 1$. The output of the convolutional layer will be feed into the max-pooling layer. The max-pooling is conducted by equation (4). Assume taking some k width window and outputting a single value, which is the maximum in that window. For instance, if the input layer is a N nodes layer, there will then output $\frac{N}{k}$ nodes in the max-pooling layer, as each k window is reduced to just a single value via the max function. Then a fully connected network is followed by the max-pooling layer. The forward propagation is:

$$x_i^l = \sum_j w_{j,i}^{l-1} \sigma(x_i^{l-1}) + b_i^{l-1} \tag{5}$$

The top is softmax classifier:

$$f(x) = argmax_c p(y = c|x) = argmax_c \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T w_k}} \tag{6}$$

where c is a class label, x is a sample feature, y is label variable, and w is weight vector, K is the number of class.

Backward Propagation Once one iteration of forward propagation is done, we will have the error value, with the loss function L (we use negative log-likelihood here), we are able to use gradient descent to update each edge w. For the fully connected layer, the gradient can be calculated in the traditional way as follow:

$$\frac{\partial L}{\partial w_{i,j}^l} = y_i^l \frac{\partial L}{\partial x_j^{l+1}} \tag{7}$$

where non-linear mapping function $y_i^l = \sigma(x_i^l) + b_i^l$, and x_j^{l+1} is node j in $l+1$ layer, $x_i^l = \sum_j w_{j,i}^{l-1} y_j^{l-1}$. the gradient in convolutional layer is computed as follow:

$$\frac{\partial L}{\partial w_{a,b}} = \sum_{i=1}^{N-m-1} y_{(i+a)}^{l-1} \frac{\partial L}{\partial y_i^l} \sigma'(x_i^l) \tag{8}$$

The detailed procedure to obtain (8) is given in the Appendix.

For the gradient in max-pooling layer, as noted earlier, the max-pooling just reduces the size of the convolutional output by introducing sparseness. In forward propagation, k width window is reduced to a single value. Then only this single value contributes an error from backward propagation from the previous layer (convolutional layer). Since it only comes from one place in the k width window, the backpropagated errors from max-pooling layer are rather sparse.

The pseudo-code for the activity recognition CNN model is described in Algorithm 1.

3.5 Regularization for Improvement

In spite of many successes in CNN, there are still many weaknesses, such as local optimum problem, overfitting, etc. In this section, we discuss the regularization terms to train more robust CNN model. In the training stage, we used stochastic gradient descent with batch size of 200 examples and the learning rate of 0.05. The weights of weight decay, momentum and dropout should be set carefully.

Weight Decay The motivation of weight decay is to avoid over-fitting. The learning rate is a parameter that determines how much an updating step influences the current value of the weights, while weight decay is an additional term in the weight update rule that causes the weights to exponentially decay to zero, if no other update is scheduled. Assume that the cost function that we want to minimize is $L(w)$. Gradient descent tells us to modify weights w in the direction of steepest descent in L:

Algorithm 1: Convolutional Neural Network for Activity Recognition

```
Input: Labeled dataset $D_{labeled} = \{((x_i, y_i, z_i), a_i)\}$, an unlabeled dataset
       $D_{unlabeled} = \{(x_i, y_i, z_i)\}$ and positive parameters $\lambda$, $\beta$
Output: Activiy Labels $A$ of the unlabeled data
repeat
    Forward Propagation:
    foreach Accelerometer data from tri-axises, $(x, y, z)$ do
        – Use (2) to conduct convolution operation with the input data
        – Use (4) to conduct max-pooling with the output of convolution
    end
    – Use fully-connected layer to integrate the pooling results of tri-axises $\{((x_i, y_i, z_i), a_i)\}$ data
    – Use soft-max to do classification and update the weight of each edge in the network
    Backward Propagation:
    – Use (7), (8) to conduct backward propagation
until $w_i$ convergences;
– Use the trained network to predict the labels
```

$$w_i = w_i - \eta \frac{\partial L}{\partial w_i} \tag{9}$$

where η is the learning rate, and if it's too large it will have a correspondingly large modification of the weights w_i

In order to effectively limit the number of free parameters in the model and avoid over-fitting, it is possible to regularize the cost function. An easy way is by introducing a zero-mean Gaussian prior over the weights, which is equivalent to changing the cost function to $\hat{L}(w) = L(w) + \frac{\lambda}{2} w^2$. In practice, this penalizes large weights and effectively limits the freedom in the model. The regularization parameter λ determines how one trade off the original cost L with the large weights penalization.

Applying gradient descent to this new cost function we obtain:

$$w_i = w_i - \eta \frac{\partial L}{\partial w_i} - \eta \lambda w_i \tag{10}$$

The new term $-\eta \lambda w_i$ coming from the regularization causes the weight to decay in proportion to its size. More intuitively, the "weight decay" terms models the data well and has "smooth" solutions.

Momentum Momentum is a technique for accelerating gradient descent and attempting to move the global minimum of the function [24] . It accumulates a velocity vector in directions of persistent reduction in the objective across iteration. The intuition behind this strategy is that the current solution w_k has a momentum, which prohibits sudden changes in the velocity. This will effectively filter out high frequent changes in the cost function and allow for larger step in favorable directions. Using appropriate parameters, the rate of convergence is increased while local optima may be overstepped. The cost function $L(w)$ to be minimized, classical momentum is given by

$$\begin{aligned} v_{t+1} &= -\eta(1-\mu)\nabla L(w_t) + \mu v_t \\ w_{t+1} &= w_t + v_{t+1} \end{aligned} \tag{11}$$

where $\eta > 0$ is the learning rate, $\mu \in [0,1]$ is the momentum coefficient, and $\nabla L(w_t)$ is the gradient at w_t. The application of momentum for DNN is proposed in [26]. Note that $\mu = 0$ gives standard descent $w_k = -\eta \nabla L(w_t)$, while $\mu = 1$, we obtain "infinite momentum" $w_k = w_{k-1}$.

Dropout In spite of the appealing representational power of DNNs, the overfitting is still one of the severe weaknesses because of the non-linear hidden layers. Dropout is a simple optimization technique widely used in deep neural network optimization [30]. The term "dropout" refers to removing units (hidden and visible) from a neural network, along with all its incoming and outgoing edges. The choice of which units to drop is random. Moreover, dropping out is done independently for each hidden unit and for each training case. Thus, applying dropout to a neural network also equals to sub-sampling a sub-neural network from it.

Training a network with dropout and using the approximate averaging method at test time leads to significantly lower generalization error on a wide variety of classification problems. The dropout is applied in the top fully-connected layer.

4 Experimental Analysis

4.1 Dataset and Preprocessing

We selected three publically available datasets for our evaluation. All datasets related to human activities in different contexts and have been recorded using tri-axial accelerometers. Sensors were either worn or embedded into objects that subjects manipulated. The sensor data was segmented using a sliding window with a size of 64 continuous samples with 50% overlap. The acceleration values were normalized to have zero mean and unit standard variance for CNN-based approach. All the deep learning based algorithms (CNN-based, and RBM) are performed on a server, equipped with a Tesla K20c GPU and 48G memory. The traditional learning algorithms (PCA, statistics) are run on the same server with an Intel Xeon E5 CPU.

- **Opportunity (Opp)** [25, 6] The dataset contains activities performed in a home environment (kitchen) using multiple worn sensors. The dataset records activities of multiple subjects on different days with 64Hz. The activities contain "open then close the fridge", "open then close the dishwasher", "drink while standing", "clean the table", etc. Our settings on this dataset is the same with [23]: only using one sensor on the right arm, and we consider 11 activities categories, including 10 low-level activities and 1 unknown activity. The dataset contains around 4,200 frames.

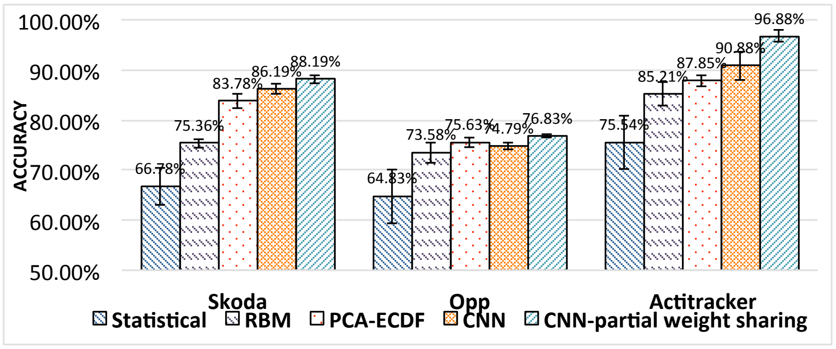

Fig. 6. Accuracy of classification for experimental evaluation of learned features. The Statistical, RBM and PCA-ECDF do not consider local dependency or scale invariant, but CNN-based model take account of local dependency and scale invariant.

– **Skoda** [33] The Skoda Mini Checkpoint dataset describes the activities of assembly-line workers in a car maintenance senario. The dataset records a worker wearing 20 accelerometers in both arms while performing 46 activities in the factory at one of the quality control checkpoint. The activities include “open hood”, “close left hand door” “check steering wheel”, etc. The frequency of sampling is 96Hz resulting around 15,000 frames. The settings of CNN on this data follows that of [23]: use only one accelerometer on the right arm to identify all 10 activities related to right arm and perform 4-fold cross validation.
– **Actitracker** [21] This dataset contains six daily activities collected in an controlled laboratory environment. The activities includes “jogging”, “walking”, “ascending stairs”, and “descending stairs”, etc., which are recorded from 36 users collected using a cell phone in their pocket with 20Hz sampling rate resulting around 29,000 frames. 10-fold cross validation is conducted on this dataset.

4.2 Classification Accuracy

In the first experiment, we evaluate the activity recognition results presented in Fig 6. The CNN is composed of a convolution layer with the partial weight sharing, with the filter size set to 20 and max-pooling size set to 3. The top two fully connected hidden layer have 1024 nodes and 30 nodes respectively. One additional softmax top layer is used to generate state posterior probabilities. All the other compared algorithms used the same settings as [23]: calculating 23 dimension statistical value (mean, standard deviation energy, etc.) as statistical feature; PCA (ECDF prepossessed) with 30 principal component (30 dimension); the structure of RBM is 192-1024-1024-30. KNN is used as the label predictor. To show the general applicability of the methods, the learning parameters and the network layout were tuned on the Skoda dataset via cross-validation and then applied as is for the remaining datasets.

From Fig 6 we can observe that CNN+partial weight sharing could improve the classification accuracy (with 95% confidence) for all the three datasets. This CNN-based model achieves classification accuracy of 88.19%, 76.83% 96.88% on Skoda, Opp, Antitracker respectively, which is 4.41%, 1.2%, 9.02% higher than the best algorithm (PCA-ECDF)[23].

To analyze the results in more detail, we show the confusion matrix for the Actitracker dataset using PCA (Table 1) and CNN (Table 2). The two confusion matrices indicate that many of the prediction error are due to confusion between these three activities: ”walking”, ”walking down”, ”walking up”. This is because these three activities are relatively similar [19]. However, from the results we can observe that the CNN+partial weight sharing model outperforms the PCA-ECDF due to the two characteristics of CNN+partial weight sharing. Note that in the PCA-ECDF confusion matrix, the confusion in (up,walk) and (down,walk) is high. This is because the signal vibration of ”walking up” and ”walking down” activities are like ”walking”. But CNN-based models performs well in these two cases, which indicates CNN could extract better representative features for ”walking down” and ”walking up”.

		Predict Class					
		Jog	**Walk**	**Up**	**Down**	**Sit**	**Stand**
Actual Class	**Jog**	**649**	**13**	**8**	**3**	**0**	**7**
	Walk	**2**	**1146**	**7**	**1**	**2**	**5**
	Up	**5**	**42**	**187**	**30**	**2**	**48**
	Down	**0**	**44**	**65**	**101**	**3**	**42**
	Sit	**0**	**0**	**0**	**0**	**166**	**0**
	Stand	**0**	**0**	**0**	**0**	**0**	**133**

Table 1. Confusion Matrix for PCA-ECDF on ActiTracker Dataset

		Predict Class					
		Jog	**Walk**	**Up**	**Down**	**Sit**	**Stand**
Actual Class	**Jog**	**667**	**5**	**1**	**3**	**0**	**0**
	Walk	**1**	**1145**	**8**	**5**	**0**	**0**
	Up	**5**	**13**	**274**	**17**	**1**	**1**
	Down	**2**	**9**	**13**	**231**	**0**	**0**
	Sit	**0**	**0**	**0**	**0**	**166**	**0**
	Stand	**0**	**0**	**0**	**0**	**0**	**133**

Table 2. Confusion Matrix for CNN on AntiTracker Dataset

4.3 Sensitivity of Parameters

We evaluate the sensitivity of varies pooling window size, the weight decay, momentum and dropout. In the following, we vary the width of pooling window, weight decay, momentum, and dropout respectively while keeping the other parameters as the best settings.

Pooling Size In the following, we evaluate the effect of different pooling sizes of the CNN configuration. Assume CNN is composed of a convolution layer with the partial weight sharing, filter size of 20 units, a max-pooling layer with a sub-sampling factor of 3, and two top fully connected hidden layer having 1024 nodes and 30 node respectively. And one additional the softmax top layer to generate state posterior probabilities. We have tested CNN with pooling size from 1 to 5, where 1 corresponds to the case of no max-pooling.
The recognition results are shown in Fig 7. The results with max-pooling are better than that with no max-pooling because max-pooling is able to preserve scale invariant. The best results are consistently achieved when setting the pooling size to 3. In this case, the recognition accuracy is increased from 85.68% to 88.19% on the Skoda dataset, and from 71.94% to 76.77% on Opp dataset.

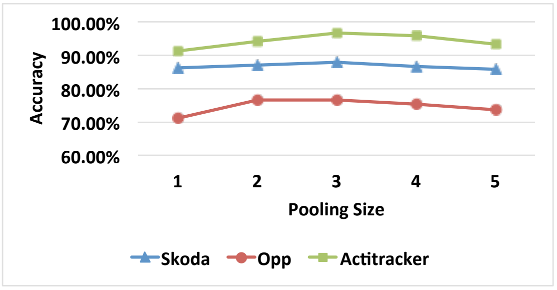

Fig. 7. Influence of pooling size on accuracy.

Weight Decay we evaluate the sensitivity of the weight decay for the weight values $\{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.25, 0.5, 1\}$. The general trend show that, the accuracy of CNN steadily improves from $[0.0001, 0.25]$. Then it decreases even the weight continues increasing. It shows that this small amount of weight decay was important for the model to learn (Fig 8).

Momentum We evaluate the sensitivity of the weights of momentum for the weight values $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$. The general trend shows that, the accuracy of CNN steadily improves from $[0.0, 0.5]$. Then it drops quickly even the weight continues increasing. With increasing weight momentum, the search direction tends to use the initial direction of the last step (Fig 9).

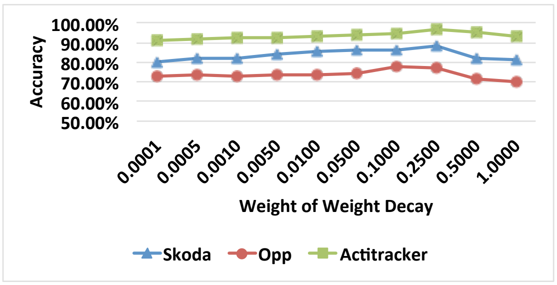

Fig. 8. Influence of weight decay on accuracy.

Dropout Dropout has a tune-able hyperparameter p, which represents the probability of retaining a hidden unit in the network. We explain the effect of varying this hyperparameter. The comparison is done when the number of hidden units is held constant. This means all the nets have the same architecture at test time but they are trained with different amount of dropout. Fig 10 shows the test accuracy obtained as a function of p. It can be observed that the performance is insensitive to the value of p if $0.5 \leq p \leq 0.9$, but drops sharply for small value of p. This is to be expected because for the same number of hidden units, having small p means very few units will turn on during training phase. It is also worth noting that the accuracy on Skoda dataset is lower than the others when p is small, then it increases faster when the p grows high. The reason is the number of Skoda has less samples than the other datasets, but when the number of samples becomes larger, it will perform better.

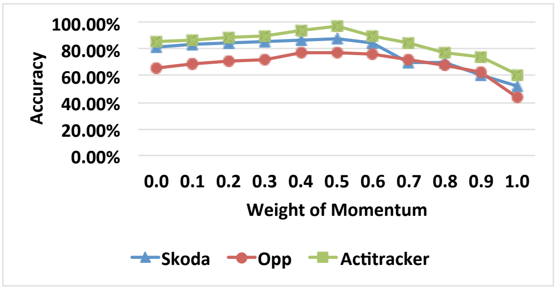

Fig. 9. Influence of momentum on accuracy.

5 Conclusion

In this paper, we have proposed a CNN-based feature extraction approach, which extracts the local dependency and scale invariant characteristics of the acceleration time series. The experimental results have shown that by extracting these characteristics, the CNN-based approach outperforms the state-of-the-art approaches.

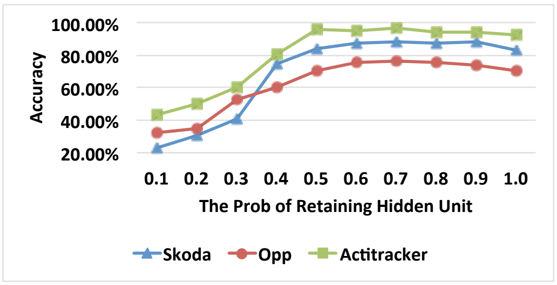

Fig. 10. Influence of dropout on accuracy.

Experiments with larger datasets are needed to further study the robustness of the proposed technique. Further improvements may be achieved by using unsupervised pre-training and repeating pooling operations in multiple layers of the CNN model.

6 Acknowledgement

This work is supported in part by the National Science Foundation through the Smart and Connected Health program under the award IIS1344768.

A
Backward Propagation for Convolutional Layer

Since we know the error from previous layer, we need to compute for the previous layer is the partial of L with respect to each neuron output $\frac{\partial L}{\partial y_i^l}$. According the chain rule, the gradient of w is computed by:

$$\frac{\partial L}{\partial w_{a,b}} = \sum_{i=1}^{N-m-1} \frac{\partial L}{\partial x_{i,j}^l} \frac{\partial x_{i,j}^l}{\partial w_{a,b}} = \sum_{i=1}^{N-m-1} \frac{\partial L}{\partial x_{i,j}^l} y_{(i+a)}^{(l-1)} \tag{12}$$

In order to compute the first term of rightmost of (12), which is straightforward to compute using the chain rule:

$$\frac{\partial L}{\partial x_{i,j}^l} = \frac{\partial L}{\partial y_{i,j}^l} \frac{\partial y_{i,j}^l}{\partial x_{i,j}^l} = \frac{\partial L}{\partial y_{i,j}^l} \frac{\partial}{\partial x_{i,j}^l} \left(\sigma(x_{i,j}^l)\right) = \frac{\partial L}{\partial y_{i,j}^l} \sigma'(x_{i,j}^l) \tag{13}$$

As we can see, since we already know the error at the current layer $\frac{\partial L}{\partial y_i^l}$, we can easily compute $\frac{\partial L}{\partial x^l i}$ at the current layer by just using the derivative of the mapping function, $\sigma'(x)$.

In addition to compute the weights for this convolutional layer, we need to propagate errors back to the previous layer by:

$$\frac{\partial L}{\partial y_{i,j}^{l-1}} = \sum_{a=0}^{m-1} \frac{\partial L}{\partial x_{(i-a)}^l} \frac{\partial x_{(i-a)}^l}{\partial y_{i,j}^{l-1}} = \sum_{a=0}^{m-1} \frac{\partial L}{\partial x_{(i-a)}^l} w_{a,b} \qquad \blacksquare \tag{14}$$

References

1. O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4277–4280. IEEE, 2012.
2. U. Bagci and L. Bai. A comparison of daubechies and gabor wavelets for classification of mr images. In *Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on*, pages 676–679. IEEE, 2007.
3. L. Bao and S. S. Intille. Activity recognition from user-annotated acceleration data. In *Pervasive computing*, pages 1–17. Springer, 2004.
4. Y. Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.
5. S. Bhattacharya, P. Nurmi, N. Hammerla, and T. Plötz. Using unlabeled data in a sparse-coding framework for human activity recognition. *arXiv preprint arXiv:1312.6995*, 2013.
6. R. Chavarriaga, H. Sagha, A. Calatroni, S. T. Digumarti, G. Tröster, J. d. R. Millán, and D. Roggen. The opportunity challenge: A benchmark database for on-body sensor-based activity recognition. *Pattern Recognition Letters*, 34(15):2033–2042, 2013.
7. S. Chennuru, P.-W. Chen, J. Zhu, and J. Y. Zhang. Mobile lifelogger–recording, indexing, and understanding a mobile users life. In *Mobile Computing, Applications, and Services*, pages 263–281. Springer, 2012.
8. D. Figo, P. C. Diniz, D. R. Ferreira, and J. M. Cardoso. Preprocessing techniques for context recognition from accelerometer data. *Personal and Ubiquitous Computing*, 14(7):645–662, 2010.
9. K. Forster, D. Roggen, and G. Troster. Unsupervised classifier self-calibration through repeated context occurences: is there robustness against sensor displacement to gain? In *Wearable Computers, 2009. ISWC'09. International Symposium on*, pages 77–84. IEEE, 2009.
10. J. Frank, S. Mannor, and D. Precup. Activity and gait recognition with time-delay embeddings. In *AAAI*, 2010.
11. Z. He and L. Jin. Activity recognition from acceleration data based on discrete consine transform and svm. In *Systems, Man and Cybernetics. SMC 2009. IEEE International Conference on*, pages 5041–5044. IEEE, 2009.
12. Z.-Y. He and L.-W. Jin. Activity recognition from acceleration data using ar model representation and svm. In *Machine Learning and Cybernetics, 2008 International Conference on*, volume 4, pages 2245–2250. IEEE, 2008.
13. G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.
14. G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
15. T. Huynh and B. Schiele. Analyzing features for activity recognition. In *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, pages 159–163. ACM, 2005.
16. S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(1):221–231, 2013.
17. A. Krause, J. Farringdon, D. P. Siewiorek, and A. Smailagic. Unsupervised, dynamic identification of physiological and activity context in wearable computing. In *2012 16th International Symposium on Wearable Computers*, pages 88–88. IEEE Computer Society, 2003.
18. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, volume 1, page 4, 2012.

19. J. R. Kwapisz, G. M. Weiss, and S. A. Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011.
20. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
21. J. W. Lockhart, G. M. Weiss, J. C. Xue, S. T. Gallagher, A. B. Grosner, and T. T. Pulickal. Design considerations for the wisdm smart phone-based sensor mining architecture. In *Proceedings of the Fifth International Workshop on Knowledge Discovery from Sensor Data*, pages 25–33. ACM, 2011.
22. D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.
23. T. Plötz, N. Y. Hammerla, and P. Olivier. Feature learning for activity recognition in ubiquitous computing. In *Proceedings of the Twenty-Second IJCAI Volume Two*, pages 1729–1734. AAAI Press, 2011.
24. B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.
25. D. Roggen, A. Calatroni, M. Rossi, T. Holleczek, K. Forster, and o. Troster. Collecting complex activity datasets in highly rich networked sensor environments. In *Networked Sensing Systems (INSS), 2010 Seventh International Conference on*, pages 233–240. IEEE, 2010.
26. I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1139–1147, 2013.
27. T. Tamura, M. Sekine, M. Ogawa, T. Togawa, and Y. Fukui. Classification of acceleration waveforms during walking by wavelet transform. *Methods of information in medicine*, 36(4-5):356–359, 1997.
28. Y. Tang, R. Salakhutdinov, and G. Hinton. Robust boltzmann machines for recognition and denoising. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2264–2271. IEEE, 2012.
29. C. Vollmer, H.-M. Gross, and J. P. Eggert. Learning features for activity recognition with shift-invariant sparse coding. In *Artificial Neural Networks and Machine Learning–ICANN 2013*, pages 367–374. Springer, 2013.
30. S. Wang and C. Manning. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 118–126, 2013.
31. P. Wu, H.-K. Peng, J. Zhu, and Y. Zhang. Senscare: Semi-automatic activity summarization system for elderly care. In *Mobile Computing, Applications, and Services*, pages 1–19. Springer, 2012.
32. P. Wu, J. Zhu, and J. Y. Zhang. Mobisens: A versatile mobile sensing platform for real-world applications. *Mobile Networks and Applications*, 18(1):60–80, 2013.
33. P. Zappi, C. Lombriser, T. Stiefmeier, E. Farella, D. Roggen, L. Benini, and G. Tröster. Activity recognition from on-body sensors: accuracy-power trade-off by dynamic sensor selection. In *Wireless Sensor Networks*, pages 17–33. Springer, 2008.
34. M. Zhang and A. A. Sawchuk. A feature selection-based framework for human activity recognition using wearable multimodal sensors. In *Proceedings of the 6th International Conference on Body Area Networks*, pages 92–98. ICST, 2011.