Communications in Mathematics and Applications
Vol. 10, No. 3, pp. 541–560, 2019
ISSN 0975-8607 (online); 0976-5905 (print)
Published by RGN Publications
DOI: 10.26713/cma.v10i3.1269

http://www.rgnpublications.com

Research Article

A Deep One-Pass Learning based on Pre-Training Weights for Smartphone-Based Recognition of Human Activities and Postural Transitions

Setthanun Thongsuwan [1,*], Praveen Agarwal[2] and Saichon Jaiyen[3,*]

[1,3] *Advanced Artificial Intelligence (AAI) Research Laboratory, Department of Computer Science, King Mongkuts Institute of Technology Ladkrabang, Bangkok 10520, Thailand*
[2] *Department of Mathematics, Anand International College of Engineering, Jaipur 303012, India*
***Corresponding authors:** [1]tsetthanun@gmail.com, [3]saichon.ja@kmitl.ac.th

Abstract. We describe a new deep learning model – *Deep One-Pass Learning* (DOPL) for Smartphone-Based Recognition of Human Activities and Postural Transitions based on the Pre-Trained Weights, DOPL consists of several stacked convolutional layers to learn the features of the input and is able to learn features automatically, followed by the Extreme gradient boosting (XGBoost) as the last layer for predicting the class labels. DOPL is much faster in the training phase, because the input weights are optimal weights from the Pre-Trained weights module and it does not have to re-adjust weights repeatedly. Further, we replaced the final fully connected layer with XGBoost to increase predictive efficiency. In the worst case, our model with demonstrated an accuracy of 99.2% for the smartphone sensors database data, which was significantly better than CNN or XGBoost alone as well as several other models assessed.

Keywords. Human activity recognition; Machine learning; Deep learning; Convolutional neural network; Feature learning; Classification; Extreme gradient boosting; XGBoost; Pre-trained weights

MSC. 68T10

Received: July 5, 2019 **Accepted:** July 18, 2019

1. Introduction

An attempt to recognize patterns of human behavior is a starting point for many research areas: it can lead to societal benefits and contribute to the quality of human life in areas such as healthcare [31, 48] and various industrial areas [30]. Human Activity Recognition

(HAR) is a key research topic for human behavior classification problems [22, 36, 49], and creates more challenges in the research field of *machine learning* (ML). Several techniques have been proposed to solve these problems. Recently, *deep learning* (DL) have been shown to be successful and attracted significant attention. In general, DL has been widely used with image data, to solve problems in computer vision and many research areas, *e.g.*biomedicine [43], transportation [33], manufacturing systems [45], consumer devices and services [26], and including previous research on recognising human activities in smart homes [28] etc.

Convolutional Neural Network (CNN) [24] is one of the DL techniques. In addition, to good ability to learn data, it can handle complex HAR tasks well. Wang *et al.* [44] have surveyed recent advances of deep learning in activity recognition. CNN has been used for sensor-based activity recognition [21,38–40], it has expanded the scope in and state-of-the-art and for many research challenges. Nweke *et al.* [34] reviewed combinations of CNN and other DL techniques, that affect the efficiency of the model in predicting class labels for mobile sensor activity recognition. However, the performance of the CNN model has been improved by combining its capabilities with other models [17, 25, 27]. We note that, in combining those abilities, each model has its own capabilities. If we combine them properly, it will increase the efficiency of the model.

We describe a new deep learning model – *Deep One-Pass Learning* (DOPL) algorithm as a modified CNN model for HAR problem: smartphone-based recognition of human activities and postural transitions [37]. Our model combines the performance of a CNN and Extreme Gradient Boosting (XGBoost) [6], the motivations are: We have already observed that a single model is not sufficient for complex data in many fields and research areas. We consider the advantage of the CNN model for handling complex data including a feature learning process. Furthermore, we choose to use only the capabilities of the convolutional layer and weights from a pre-trained CNN. Due to the pre-training, we do not need to re-adjust the weights in each the convolutional layer. Therefore, the backpropagation algorithm in the loop is not necessary, including the *Fully Connected* (FC) layer. We looked for models that have good performance in predicting class labels on their own, and which will handle the data passing the feature learning step as the first step: XGBoost is a scalable machine learning system for tree boosting, commonly used by data scientists and successful in many machine learning competitions (*e.g.* Kaggle). Finally both models – CNN and XGBoost are state-of-the-art and many researchers show good performance [6, 26, 33, 43, 45].

DOPL consists of two main sections: first, it has a convolutional layer stacked in several layers for learning information features. The second stage is responsible for processing the training data passed from the feature learning step to predicts class labels. However, DOPL adds improvements for weights received from the pre-trained model. As we show, this increases accuracy and reduces training time, because it does not start with randomly chosen weights, which may be far from the optimal weight. In addition, we added the pooling layer to reduce the size of the training set.

The main contributions of this paper are:

- A a new deep learning model called 'DOPL' for HAR problem based on Pre-Trained weights.

- DOPL takes less time to train data than CNN, see Table 6.
- Our model is effective for automatic feature learning and there is time complexity as $\mathcal{O}(Ld^2mnpq)+\mathcal{O}(r(Kt+\log B))$ see in Section 3.4.
- DOPL provides higher accuracy than the two individual models, CNN and XGBoost, which are the current prototypes for modeling, and other extant models, *i.e.*, Logistic Regression (LR) [7,10,47], Extra Trees Classifier (ETC) [13], Gradient Boosting Classifier (GBC) [11, 12, 16], Random Forest Classifier (RFC) [3], Gaussian Naive Bayes (GNB) [4], Decision Tree Classification (DTC) [2], Multilayer Perceptron (MLP) [19] and Support Vector Classification (SVC) [5]. In addition, we evaluated the performance of other research CNNs [21, 38–40], previously reported for HRA tasks, listed in Wang *et al.*'s survey [45].

DOPL differs from previous work of Jabri *et al.* [20] and Lioutas *et al.* [29], in that Pre-training weights are commonly used for Visual Question Answering (VQA) tasks, using weights acquired from a library or public framework [46] (*e.g.* Theano [42], Lasagne [8], ImageNet [15], word2vec [32], AlexNet [23], GoogLeNet [41], and ResNet [18] *etc.*). However, the weights for our model were directly generated from the HAR data set for training with our model and saved as the pre-trained weights in the prediction stage.

The remainder of this paper covers: in Section 2, we review related theories and research. In Section 3, we set out details of DOPL, including the architecture, pre-training weights, data preprocessing and learning algorithm. Section 4 describes design of experiments for evaluating performance. Then, we evaluate the performance from the results and, finally, conclude.

2. Material and Methods

2.1 Human Activity Recognition (HAR) Data Set

The smartphone-based recognition of human activities and postural transitions is public data set, it used to evaluate the performance of our model, was built by Jorge-L *et al.* [37] from 30 people, for human activities and postural transitions definition. All volunteers wore a sensor equipped smartphone (Samsung Galaxy S II) on the waist. Data the sensors and video images were recorded and then manually labeled. Activities were classified as six basic activities: three static postures (standing, sitting, lying) and three dynamic activities (walking, walking downstairs and walking upstairs). By adding transitions between the basic activities, we end up with 12 activity labels: listed in Table 1: six basic activities: composed of three dynamic activities - WALKING, WALKING_UPSTAIRS and WALKING_DOWNSTAIRS - and three static ones - SITTING, STANDING and LYING DOWN - and possible six transitions between them: STAND_TO_SIT, SIT_TO_STAND, SIT_TO_LIE, LIE_TO_SIT, STAND_TO_LIE, and LIE_TO_STAND. We assumed our volunteers were not acrobats or contortionists. Details of the sensors are shown in Table 2. Data used for activity recognition was obtained from a smartphone sensor. This data set was sourced from the University of California at Irvine (UCI) Repository of machine learning data sets [9] – UCI smartphone: It is commonly used in research for high-level activity understanding [44]: the data set is described in Definition 2.1.

Each element of the vector consists of a classes label, including 10,929 instances, each with 561 attributes. There were no missing values. There were 12 classes in this data set (see Example 2.2). The data set was divided with three-fold cross-validation (see Section 4.2) for evaluating models, with 7,286 assigned to training and 3,643 assigned to test activities.

Definition 2.1. Input: Let $\mathbb{I} = \{x_i, y_i | 1 \le i \le M\}$, be the training set of M vectors, each vector x_i consists of N features, $x_i = \{p_j | 1 \le j \le N\}$, and y_i a label, $\mathbf{x}_i$ in $\mathbb{R}$.

Example 2.2. The expression: If $M = 10,929$ and $N = 561$, then $\mathbb{I} = \{(x_i, y_i) | 1 \le i \le 10,929\}$, (x_i, y_i) be the training set of 10,929 vectors, each vector x_i consists of 561 feature $x_i = \{p_j | 1 \le j \le 561\}$.

Table 1. Activity Labels characteristic

Class ID	Activity Labels	Description	Number of Instances
1	WALKING	walking	1,722
2	WALKING_UPSTAIRS	walking upstairs	1,544
3	WALKING_DOWNSTAIRS	walking downstairs	1,407
4	SITTING	sitting	1,801
5	STANDING	standing	1,979
6	LYING_DOWN	lying	1,958
7	STAND_TO_SIT	postural transition: standing - to - sitting	70
8	SIT_TO_STAND	postural transition: sitting - to - standing	33
9	SIT_TO_LIE	postural transition: sitting - to - lying	107
10	LIE_TO_SIT	postural transition: lying - to - sitting	85
11	STAND_TO_LIE	postural transition: standing - to - lying	139
12	LIE_TO_STAND	postural transition: lying - to - standing	84

Table 2. HAR data set

Characteristic	Activity Labels
Type	Activities of Daily Living (ADL)
The number of Subject	30
Sensor Rate	50 Hz
Sensor	Accelerometer, Gyroscope
Sensor Modality	Body-worn
Activity	6
The number of Activity Labels	12
The number of Instances	10,929
The number of Attributes	561
The number of Training/Testing set	7,286/3,643

2.2 Convolutional Neural Network

A *Convolutional Neural Network* (CNN) [24] is described in detail by Goodfellow *et al*. [14]. A short summary follows, but readers familiar with CNN may skip this section. We assume the input training data to be a grey scale image, $\mathbb{J}$, by Definition 2.3.

Definition 2.3. Image: An input $H \times D$ grayscale image, $\mathcal{I} \in \mathbb{R}^{H \times D}$, when $p_{jk} \in \mathbb{R}$ is the intensity of the pixel represented as:

$$\mathcal{I} = \{p_{jk} | 1 \leq j \leq H, 1 \leq k \leq D\}, \tag{1}$$

A CNN architecture usually consists of several convolutional layers, alternating with multiple pooling layers, which are responsible for feature learning from the training data. A fully Connected (FC) will be the last layer of net (*i.e.*, Conv,Pool/Conv,Pool/.../FC). For understanding, the calculation for each layer is described in order in th following paragraphs.

The convolutional layer applies a $\mathcal{I} \otimes \mathcal{K}$, in which a dot product operation between image and a $h_k \times d_k$ kernel, $\mathcal{K}$, is slid through every element, with stride S_k. The output from layer, l, $\mathcal{Y}_i$, where i^{th} feature map, with a bias, $\mathcal{B}$, added is sent through the Rectified Linear Unit (ReLU) activation function, φ, where $\varphi = \{(0,x)|\ 0 \text{ if } x < 0, x \text{ if } x \geq 0\}$. So the normalization formula for calculating is:

$$\mathcal{Y}_i^{(l)} = \varphi\left(\mathcal{B}_i^{(l)} + \sum_{j=1}^{f^{(l-1)}} \mathcal{K}_{i,j}^{(l)} * \mathcal{Y}_j^{(l-1)}\right), \tag{2}$$

where l^{th} is index layer, than $\mathcal{Y}_i^{(l)}$ is the output of l^{th} layer for the i^{th} feature map and $\mathcal{Y}_j^{(l-1)}$ is the output of the previous layer for the j^{th} feature map by index of a feature map is $f \in [1..F^k]$, because they are transformations (the convolutions) of the previous inputs. Note that the number of feature maps in each layer may vary, set by the user for each application, F^k. In convolution layer, $k \in [1..L]$.

The pooling layer will help reduce the number of the output (downsamples or downscaled). Function options are set to be used as *e.g.*maximum, average, *etc.*, commonly used maximum value on a local rectangular region (neighborhood). Let us assume $h_p \times d_p$ is size of the pooling windows, then $P(..)$ is a pooling function which acts on $\mathcal{Y}_i^{(l)}$ by the output of a max-pooling function is:

$$P(\mathcal{Y}_i^{(l)})_{m,n} = \max(\mathcal{Y}_i^{(l)})_{m,n}. \tag{3}$$

The FC layer, received training data from the previous convolutional and pooling layer. In this layer is a classifier layer, the weights received in this layer each iteration is taken back to adjust the weights in the previous layer to the first layer (the convolutional layer). We know that the MLP algorithm is behind the process of predicting probabilities of class labels. Mostly, *softmax* is the transformation function, the formula is

$$Y = softmax(\mathcal{I}.W + B). \tag{4}$$

Finally, we obtain the output predictions, denote Y, where $\mathcal{I}$ is the set of images, W are weights and B, biases of net.

2.3 Extreme Gradient Boosting Model

Extreme Gradient Boosting (XGBoost) is a machine learning model for classification and regression problems. This model is effective for machine learning and data mining challenges and used extensively. Chen and Guestrin's XGBoost was first used in the KDD Cup 2015 [6]. A brief summary of their work follows, but readers familiar with their work may skip

to Section 3. The architecture of the model has a tree, which is an ensemble of K classification and regression trees (CARTs). Let x_i are the vector training set and y_i are the class labels of x_i in order of i. The output prediction, $\hat{y}_i$ are the sum of the prediction scores of k^{th} trees:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \tag{5}$$

where $f_k \in F$ then f_k is the leaf score for the k^{th} tree and F is the set of all K scores. The output prediction, $\hat{y}_i$, was compared between the target evaluated with a loss function, $l(\hat{y}_i, y_i)$, and an added Ω term to prevent overfitting, calculated from:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k), \tag{6}$$

where $\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} {w_j}^2$, then γ and λ are constants to control the regularization degree, T is the set of leaves in the tree and weight of each leaf denote w. In addition, we can also improve the efficiency in Equation (6) by expanding the loss function with a first and second order Taylor expansion. Therefore, we can calculate at step t:

$$\begin{aligned}
\tilde{\mathcal{L}}^{(t)} &\simeq \sum_{i=1}^{n} \left[g_i f_i(x_i) + \frac{1}{2} h_i {f_i}^2(x_i) \right] + \Omega(f_t) \\
&= \sum_{i=1}^{n} \left[g_i f_i(x_i) + \frac{1}{2} h_i {f_i}^2(x_i) \right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} {w_j}^2 \\
&= \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) {w_j}^2 \right] + \gamma T,
\end{aligned} \tag{7}$$

where $g_i = \frac{\partial l(\hat{y}_i^{(t-1)}, y_i)}{\partial \hat{y}_i^{(t-1)}}$ and $h_i = \frac{\partial^2 l(\hat{y}_i^{(t-1)}, y_i)}{\partial (\hat{y}_i^{(t-1)})^2}$ are the first and second order gradient statistics of the loss function respectively, $I_j = \{i | q(x_i) = j\}$ is the number of set in leaf, t and w_j^* are the optimal weight value of leaf, j. The weight is calculated:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{8}$$

when calculating the weight in Equation (8), the final equation is the quality of a tree structure, q can be computed:

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i \right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \tag{9}$$

The XGBoost model is based on calculating scores in each node in the tree structure for split decisions. For effective prediction, we realize the loss after the split process and want to reduce it. Let $I = I_L \cup I_R$, where I_L is the left and I_R is the right node after the split, $\mathcal{L}_{split}$ can be evaluated from:

$$\mathcal{L}_{split} = \frac{1}{2} \left[\frac{\left(\sum_{i \in I_L} g_i \right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i \right)^2}{\sum_{i \in I_R} h_i + \lambda} + \frac{\left(\sum_{i \in I} g_i \right)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma. \tag{10}$$

3. Deep One-Pass Learning (DOPL) Model

In this section, we describe the details of our new model, DOPL, which combines elements of CNN and XGBoost, but emphasizes the key differences in DOPL. The primary difference is the Pre-Trained weight module, which is the first step in the overall structure, followed by the CNN and XGBoost modules. Each DOPL module is described in order in the following sections.

The main contributions of our model are:

- DOPL is powerful, auto feature learning by convolutional layer and optimal weights from pre-trained. In addition, we have a scalable end-to-end tree boosting for predicting class labels by XGBoost. It differs from traditional CNN, since it does not have to adjust the weights and the FC layer was replaced by XGBoost.
- Our model uses one step only for learning feature data, because we apply optimal weights from pre-training. It does not need to rely on a backpropagation step to re-adjust weights. Therefore, it reduces the cost of an iterative loop for fine tuning weights in each a convolutional layer.
- We removed the re-adjusting weights step, weights were optimal after pre-training with the CNN model. The time complexity for fine tuning weights is $\mathcal{O}(Ld^2mnpq)+\mathcal{O}(MNhce)$ (see in Table 5). Pre-trained weights make it more effective, than initiating a custom or randomly generated set of weights.
- Our model does not focus only on image processing, but also other general classification problems.

3.1 DOPL Architecture

The overall architecture of our model see in Figure 1.

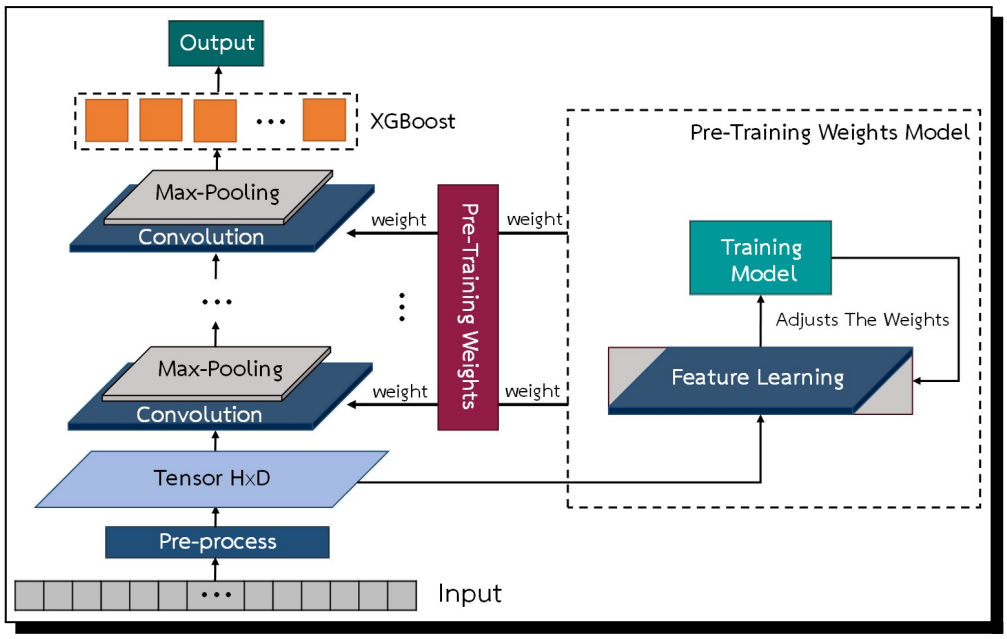

Figure 1. Overview of DOPL: Training and prediction (left side), and Pre-Trained weights (right side)

An overview of the function of each module is described in the following list:

Pre-Trained weights: Pre-Trained Weights (right side of Figure 1) is the preprocessing module and is outside our model, but it is important, because pre-training generates optimal weights for use in our model and improves the performance of our model. CNN is used to train weights: we used a backpropagation method to adjust weights repeatedly until the end of the epoch (or predefined number of cycles) or optimal weights were found by measuring the model accuracy. These optimal weights are used in the next module.

Training: This module is a key part of our model (left side of Figure 1); it is responsible for training with a training set. All parameters used here will be set as in the Pre-Training model, including weights in the feature learning step. The training module has seven layers - explained in further detail later - Section 3.1. Output from this module is used for predicting class labels (Section 3.1) for classification problems.

1. *Input layer*: The input layer is the first layer of the model for loading the training set - vector format data in the HAR problem. Remark 3.1 has affected this layer:

   Remark 3.1. We use the capabilities of the convolutional layers from CNN, the input must be a tensor format see in Definition 2.3. Therefore, the training set (in Section 2.1) will be converted to the standard input format for convolutional operation, as described in the following Section 3.3.

2. *Data preprocessing layer*: We used this module from Section 3.3, because it enables our system to handle a wide variety of problems appearing in quite different formats. It defines a common data format for the model, supporting a variety of data types, so that our model can process general data and is not limited to image data. Details of the common format, which is sent to the next step, are shown in Section 3.3.

3. *Convolutional layer*: Features are learnt in this layer using a convolutional operator. We set the number of layers, L, according to user needs. See Equation (2) in Section 2.2 for the equation of the convolutional operation. To determine the number of layers, L, we consider overall time complexity of the model (see Table 5, which contains L), computer power of the machine used, to estimate the run time, and data adequacy, because if the number of features and instances is low, it may not be enough for convolutional layers

   However, in this layer, we will process only once without re-adjusting weights, since weights used in this layer are obtained from the Pre-Trained weights module, see Section 3.2. This greatly reduces run time, see comparisons with CNN and our model in Table 6 of Section 4.

4. *Pooling layer*: The pooling layer helps reduce the size of the previous convolutional layer: the Max operation used in this layer is max pooling. Thus, the number of layers for pooling and convolution will be the same in our model.

   Predicting: After receiving the training model, the prediction module (left side of Figure 1) predicts the target class labels from the test set.

5. *Reshape layer*: After a sequence of pairs of (convolution:pooling layers) which form the feature learning stage, the output is a tensor, $\mathcal{Z}$. The reshape layer, shown in the dashed box in Figure 2, reformats this final output to a hierarchical structure.

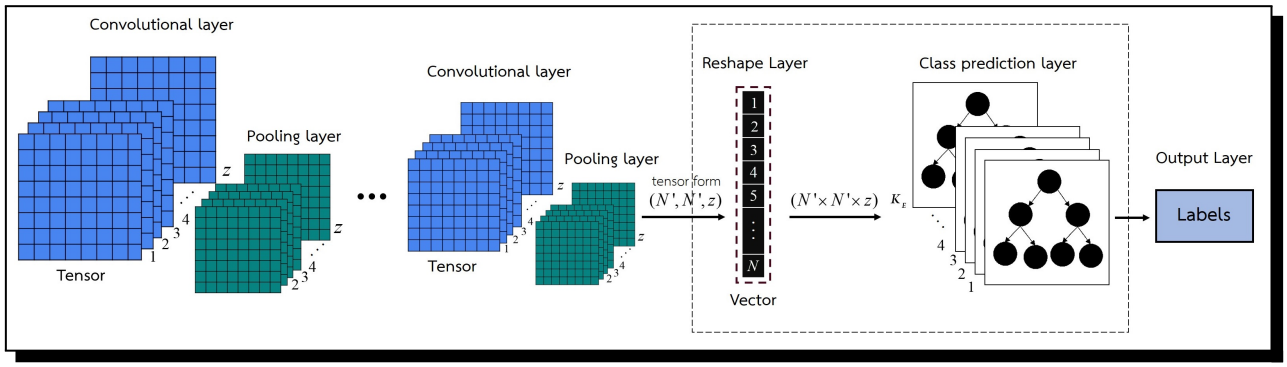

Figure 2. The reshape layer re-adjusts the shape of the data (tensor) to the vector format

6. *Class prediction layer*: The class prediction layer is a key layer of the model and influences accuracy of the class prediction. Behind the prediction, efficiency is driven by ensemble tree gradient boosting, Extreme Gradian Boosting (XGBoost) [6]. We evaluate the quality of the tree by calculating the points (as in Equation (9)):

$$\widetilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\frac{\left(\sum\limits_{i\in I_j} g_i\right)^2}{\sum\limits_{i\in I_j} h_i + \lambda} + \gamma T$$

and candidate split will be considered based on scores of the left and right nodes of the instance, after a split, in order to reduce loss in the split operation (as in Equation (10)) in Section 2.3.

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum\limits_{i\in I_L} g_i\right)^2}{\sum\limits_{i\in I_L} h_i + \lambda} + \frac{\left(\sum\limits_{i\in I_R} g_i\right)^2}{\sum\limits_{i\in I_R} h_i + \lambda} + \frac{\left(\sum\limits_{i\in I} g_i\right)^2}{\sum\limits_{i\in I} h_i + \lambda}\right] - \gamma\,.$$

7. *Output layer*: The output layer is the last layer and generates the predicted class - the final output. All the steps of the model are set out in Section 3.4.

3.2 Pre-Trained weights

Pre-training weights in a deep learning model avoid random weight initial settings and repeated training for weight adjustment. It reduces the training time, when comparing our models with traditional CNN (see in Table 6). In general, pre-training has been commonly used in Visual Question Answer (VQA) problems [20, 29]. It is used to simplify model training. The model can predict class labels, without computing the weights itself, but uses weights shared from a public service provider, *e.g.* Theano [42], Lasagne [8], ImageNet [15], word2vec [32], AlexNet [23], GoogLeNet [41], ResNet [18] *etc.* However, pre-training was not mentioned in HAR tasks, investigated by Wang *et al.* [44] and Nweke *et al.* [34]. We were interested here, to assess the effectives of pre-training for predicting class labels and avoid adjustment of training weights and its computational cost.

Our DOPL system has two modules, the training module and the recognition module (see Figure 1), where the pre-training module is shown on the right in the dashed box. Although not explicitly shown, the pre-trained weights, central vertical box in Figure 1 can be saved and used in a second recognition task (the system on the left). CNN was chosen as a prototype of the pre-training model because it is effective and is commonly used in HAR tasks [21, 39, 40]. The CNN parameters were set to be appropriate for our model, including the number of convolutional layers, L, output depth, z, kernel sizes, $\mathcal{K}^{(l)}$, and kernel strides, $S_k^{(l)}$, *etc.*

3.3 Data Preprocessing

As noted before, this section is an important housekeeping stage to allow our system to handle data from multiple sources in multiple formats. Basically, we pad out the data to generate square tensors, as necessary, by adding zeroes $\mathcal{Q}$ (see Definition 3.4).

Definition 3.2. *Input*: Let $\mathbb{I} = \{x_i, y_i | 1 \le i \le M\}$, be the training set of M vector, each vector x_i consists of N features, $x_i = \{p_j | 1 \le j \le N\}$, and y_i is the label of image $\mathbf{x}_i$ in $\mathbb{R}$, if $p_j = [p_0, p_1, \ldots, p_N]^T$ is a set of N feature vectors in $\mathbb{R}^N$, since $N' = \sqrt{N}$ is integral, we convert to input data to be a square matrix with dimension N' as follows:

$$p_j \Rightarrow [p_0, p_1, \ldots, p_N]^T \Rightarrow \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1N'} \\ p_{21} & p_{22} & \cdots & p_{2N'} \\ \vdots & \vdots & \ddots & \vdots \\ p_{N'1} & p_{N'2} & \cdots & p_{N'N'} \end{bmatrix}. \tag{11}$$

Example 3.3. If $N = 9$, then $N' = 3$ the vector $[0,1,2,3,4,5,6,7,8]^T$ is a set of data vectors in $\mathbb{R}^9$, this vector matches the standard criterion, so is just copied a 3×3 matrix as follows:

$$p_j \Rightarrow [p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8]^T \Rightarrow \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}.$$

Definition 3.4. *Adding Zeros*: If N is not integral, we pad the feature vector with zeros, $\mathcal{Q}$, and copy the padded vector to a $N' \times N'$ square matrix.

Example 3.5. For example adding zeros when we assume a vector $[0,1,2,\ldots,10]^T$ in $\mathbb{R}^{11}$, so five zeroes, $[0,0,0,0,0]^T$, are added will lead to a 4×4 matrix as follows:

$$\mathcal{Q}(p_j) \Rightarrow [p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, 0, 0, 0, 0, 0]^T$$
$$\Rightarrow \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

For input in Section 2.1, Let $N = 561$, when $N' = \sqrt{561}$ is not integral, a vector $[0,1,2,\ldots,561]^T$ in $\mathbb{R}^{561}$, so 15 zeros, are added will leading to a vector $[0,1,2,\ldots,576]^T$ in $\mathbb{R}^{576}$, then $N' = \sqrt{576} = 24$ is arranged in a 24×24 matrix.

Definition 3.6. *Tensor*: Let M is the training set, when $[x_j^i]_N$ be a set of N feature vectors in $\mathbb{R}^N$ then $N' = \sqrt{N}$ is integral, it is defined as dimension for a square matrix notation $[a_j^i]_{N'N'}$. Therefore, the general tensor, $\mathcal{T}$ of M-order can be created as follows:

$$\mathcal{T} = [\overline{a}^{i_1 i_2 \ldots i_M}_{j_1 j_2 \ldots j_M}]_{N'N'}. \tag{12}$$

3.4 DOPL Algorithm

Let $\mathbb{I} = \{x_i, y_i | 1 \le i \le M\}$, be the training set of M vector, each vector x_i consists of N feature $x_i = \{p_j | 1 \le j \le N\}$, be a set of N feature vectors in $\mathbb{R}^N$ or $\mathbb{R}^{N' \times N'}$, where $N' = \sqrt{N}$ and y_i is the label of image $\mathbf{x}_i$ in $\mathbb{R}$. The learning algorithm for the DOPL can be summarized as follows:

---

1. Initialize the training data set, $\mathbb{I} = \{x_i, y_i | 1 \le i \le M\}$
2. If necessary, pad the N elements of each training data item, x_i, so that new data item can be formed into a square matrix of dimensions, $N' \times N'$.
3. Convert x_j tensor format, $(N', N', z^{(l)})$
4. Set the parameters of the convolutions for learning features:
   (a) number of convolutional layers, L
   (b) convolutional layer output depth, z
   (c) for each layer, set the kernel sizes, $\mathcal{K}^{(l)}$, and
   (d) kernel strides, $S_k^{(l)}$
5. Determine the weights from the pre-trained model.
   (a) get weights, $\mathcal{W}_{PT}$ and
   (b) get bias, $\mathcal{B}_{PT}$
6. For each layer, l, in 1..L:
   (a) Calculate the convolutions to generate the $\mathcal{Y}_i^{(l)}$ for layer, l:
   $$\mathcal{Y}_i^{(l)} = \varphi(\mathcal{B}_{PT_i}{}^{(l)} + \sum_{j=1}^{f^{(l-1)}} \mathcal{W}_{PT}{}_{i,j}^{(l)} * \mathcal{Y}_j^{(l-1)}).$$
   (b) Calculate the pooling, replaces the output with the maximum value.
   $$P(\mathcal{Y}_i^{(l)}) = \max(\mathcal{Y}_i^{(l)}).$$
7. Reshape $P(\mathcal{Y}_i^{(l)})$ to a vector of length $(N' \times N' \times z^{(l)}) - \mathcal{Z}^{(l)}$.
8. Initialize a new training data set for class prediction layer
   $$\mathbb{I}_{new} = \{(\mathcal{Z}_i, y_i) | 1 \le i \le M\}.$$
9. Initialize parameters for the prediction step, set
   (a) total number of trees, K
   (b) regularization parameters, γ and λ,
   (c) number of leaves in the tree, T
   (d) column subsampling parameter,

(e) maximum tree depth and

(f) learning rate

10. Determine the class labels for output:

$$\hat{y}_i = \phi(\mathcal{Z}_i) = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in F,$$

where $F = f(\mathcal{Z}_i) = w_{q(\mathcal{Z})}(q : \mathbb{R}^N \to T, w \in \mathbb{R}^T)$.

11. Calculate the optimal leaf weight for the best tree structure

$$w_j^* = -\frac{\sum\limits_{i \in I_j} g_i}{\sum\limits_{i \in I_j} h_i + \lambda}.$$

12. Calculate the quality of the tree structure, q, using the scoring function

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \frac{\left(\sum\limits_{i \in I_j} g_i\right)^2}{\sum\limits_{i \in I_j} h_i + \lambda} + \gamma T.$$

13. Calculate the best splitting points

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum\limits_{i \in I_L} g_i\right)^2}{\sum\limits_{i \in I_L} h_i + \lambda} + \frac{\left(\sum\limits_{i \in I_R} g_i\right)^2}{\sum\limits_{i \in I_R} h_i + \lambda} + \frac{\left(\sum\limits_{i \in I} g_i\right)^2}{\sum\limits_{i \in I} h_i + \lambda}\right] - \gamma.$$

14. Terminate

Our DOPL algorithm has overall time complexity:

$$\mathcal{O}(Ld^2mnpq) + \mathcal{O}(r(Kt + \log B))$$

which reduces to $\mathcal{O}(Ld^2mnpq)$, where L is the number of layers, d is the number of input or output channels, the data matrix has size $m \times n$, the kernel has size $p \times q$, $r = \|x\|$ is the number of non-missing entries, K is the number of trees, t is the tree depth and B is the block length.

4. Model Evaluations

4.1 Experimental Setup

We set the experimental parameters carefully to balance the resources used while achieving good performance, guided by the time complexity of our model (see Section 3.4). Initially, we set the number of the convolutional layers or number of maps, $L = 3$, and the output depth of the convolutional layer, $z = 2^n$, where $n = 1, 2, ..., 5$. The chosen z value sets a balance between performance and use of resources. We set the kernel size, $\mathcal{K}$, is 3×3, if $\mathcal{K}$ is small, accuracy will be high, but the convolution operation will be repeated many times and consume computation time. On the other hand, if $\mathcal{K}$ is too large, accuracy may suffer (the kernel size will not be larger than the training set). Additionally, a small stride of the kernel $S_k = 1$ enables small features to be recognized. The parameters set in our model in each layer are shown in Table 3.

We implemented and tested our and other models with Python (3.6.4) and functions from the TensorFlow library [1], *e.g.* the convolutional operation (also used in CNN), together with the XGBoost python package for the prediction step. For other model testing, we used the machine learning library from scikit-learn [35]. Our experiments used Linux (Ubuntu 18.04.1LTS) on a system containing an Intel® Xeon® CPU E5-2620 0 @ 2.00GHz×12, 24.0 GB.

Table 3. Architecture in each layer of the DOPL model

Layer Name	Type	Input	Kernel	Stride	Output
L1	Input	$N' \times N'$	*na*	*na*	$N' \times N'$
L2	Data Preprocessing	$N' \times N'$	*na*	*na*	$[N', N', 1]$
L3	Convolutional (Conv1)	$[N', N', 1]$	$h_k \times d_k$	1	$[N', N', z^{(L-2)}]$
L4	Max-Pooling (Pool1)	Conv1	$h_p \times d_p$	$h_p \times d_p$	Pool1
L5	Convolutional (Conv2)	Pool1	$h_k \times d_k$	1	$[N', N', z^{(L-1)}]$
L6	Max-Pooling (Pool2)	Conv2	$h_p \times d_p$	$h_p \times d_p$	Pool2
L7	Convolutional (Conv3)	Pool2	$h_k \times d_k$	1	$[N', N', z^{(L)}]$
L8	Max-Pooling (Pool3)	Conv3	$h_p \times d_p$	$h_p \times d_p$	Pool3
L9	Reshape	Pool3	*na*	*na*	$N' \times N' \times z^{(L)}$ or $\mathcal{Z}$
L10	Class Prediction	$\mathcal{Z}$	*na*	*na*	Predicted Class
L11	Output	*na*	*na*	*na*	Predicted Class

Notes: *na* = not applicable.

We used three-fold cross-validation to train and test the models. Each data set was divided into three disjoint subsets. Then, two subsets were used as a training set and the other subset was used as a testing set. This was repeated three times: each subset was used exactly once as the testing set. The results from each testing set were averaged and a standard deviation calculated.

Our model was compared with other models *i.e.*, Convolutional Neural Network (CNN) [21, 24, 38–40], Extreme Gradient Boosting (XGBoost) [6], Logistic Regression (LR) [7, 10, 47], Extra Trees Classifier (ETC) [13], Gradient Boosting Classifier (GBC) [11, 12, 16], Random Forest Classifier (RFC) [3], Gaussian Naive Bayes (GNB) [4], Decision Tree Classifier (DTC) [2], Multilayer Perceptron (MLP) [19] and the Support Vector Classification (SVC) [5] (see Table 5).

Parameter settings for DOPL and details for each layer are shown in Table 4. For the CNN model, the parameters were set to be the same as those for our model, DOPL. The number of neurons in the FC class was 2^n, when $n = 8, 9, 10$. In general, CNN supports image data only, but for our experiments, we added our data preprocessing (see Section 3.3) step to CNN to facilitate comparison for training sets of vectors in HAR data set (Section 2.1). The XGBoost model was set similarly, with parameters matching those in the class prediction layer of our model, to fairly evaluate performance.

In addition, we evaluated four variants of the MLP model, with different activation functions: linear (MLP1), sigmoid (MLP2), tanh (MLP3), and ReLU (MLP4). The numbers of neurons was 2^n, when $n = 8, 9, 10$ and the learning rate was set to 0.001. Similarly, four variants of the SVC model were tested with differing kernel functions: RBF (SVC1), linear (SVC2), polynomial (SVC3) and sigmoid (SVC4). In total, there were 16 models, including, LR, ETC, GBC, RFC, GNB and DTC. The effectiveness of our model and properties are summarized in Table 5.

Table 4. Parameters in each layer of the DOPL model

Layer Name	Descriptions	Parameters
L1	Input: raw data size	$10,929 \times 561$
	Number of classes	12
L2	Data Preprocessing Section 3.3, input size	24×24
	Number of convolutional layers, L	3
	Learning rate	0.001
L3	Convolutional (First layer)	
	Kernel size	32×32
	Kernel stride	1
L4	Max-Pooling (First layer)	
	Kernel size	2×2
	Kernel stride	1
L5	Convolutional (Second layer)	
	Kernel size	32×32
	Kernel stride	1
L6	Max-Pooling (Second layer)	
	Kernel size	2×2
	Kernel stride	1
L7	Convolutional (Third layer)	
	Kernel size	32×32
	Kernel stride	1
L8	Max-Pooling (Third layer)	
	Kernel size	2×2
	Kernel stride	1
L9	Reshape: re-adjusts data to vector format - see Figure 2	
L10	Class Prediction	
	Total number of trees, K	100
	Maximum tree depth	3
L11	Output: class ID of activity label Table 1	1-12

Furthermore, Table 5 shows the time complexity of the models. We assume that: L is the number of layers, d is the number of input or output channels, the data matrix has size $m \times n$, the kernel has size $p \times q$, $r = \|x\|$ is the number of non-missing entries, K is the number of trees, t is the tree depth and B is the block length, M is the number of training sets, N is the number of features or dimensions, h is the number of hidden neurons, c is the number of classes and e is the number of epochs.

Table 5. Properties of models used to compare performance

Models	Parameters details	Ref.	Time Complexity
Deep One-Pass Learning (DOPL)	No. of Conv. layer $L = 3$	-	$\mathcal{O}(Ld^2mnpq) + \mathcal{O}(r(Kt + \log B))$
Convolutional Neural Network (CNN)*	No. of Conv. layer $L = 3$	[24]	$\mathcal{O}(Ld^2mnpq) + \mathcal{O}(MNhce)$
Extreme Gradient Boosting (XGBoost)	Max_depth = 3	[6]	$\mathcal{O}(r(Kt + \log B))$
Logistic Regression (LR)	Tol = 0.0001, C=1.0	[7, 10, 47]	$\mathcal{O}(MN)$ to $\mathcal{O}(MN^2)$
Extra Trees Classifier (ETC)	Criterion = Gini, MinSS = 2	[13]	$\mathcal{O}(MNK)$
Gradient Boosting Classifier (GBC)	Max_depth = 3, MinSS = 2	[11, 12, 16]	$\mathcal{O}(MNK)$
Random Forest Classifier (RFC)	Criterion = Gini, MinSS = 2	[3]	$\mathcal{O}(KtMN \log N)$
Gaussian Naive Bayes (GNB)	Var_Smoothing = 1e-09	[4]	$\mathcal{O}(Mc)$
Decision Tree Classifier (DTC)	Criterion = Gini, MinSS = 2	[2]	$\mathcal{O}(tM \log N)$
Multi-layer Perceptron Classifier (MLP)			
MLP1	Activation = Linear	[19]	$\mathcal{O}(MNhce)$
MLP2	Activation = Sigmoid	[19]	$\mathcal{O}(MNhce)$
MLP3	Activation = tanh	[19]	$\mathcal{O}(MNhce)$
MLP4	Activation = ReLU	[19]	$\mathcal{O}(MNhce)$
Support Vector Classification (SVC)			
SVC1	Kernel = RBF, C = 1.0	[5]	$\mathcal{O}(M^3)$
SVC2	Kernel = Linear , C = 1.0	[5]	$\mathcal{O}(M^3)$
SVC3	Kernel = Poly, C = 1.0	[5]	$\mathcal{O}(M^3)$
SVC4	Kernel = Sigmoid, C = 1.0	[5]	$\mathcal{O}(M^3)$

Notes: * CNN requires adaptation to the data our pre-processing module.

4.2 Experimental Results

The performance of the DOPL was evaluated on the smartphone-based recognition of human activities and postural transitions data set (see Section 2.1), a multiclass classification problem. The results were compared with 16 models in Table 5. DOPL, with pre-trained weights, was more efficient, because, using weights from pre-training, instead of a loop to re-adjust weights, found the optimal weights faster. Training times (for the deep learning model) for our model and CNN are shown in Table 6. DOPL is faster than CNN and it uses a single pass training. Further, the weights were derived from the pre-trained model and it reduced the training data size in the pooling layer.

Table 6. Training time of our model compared with CNN

Data set	Training Time (sec.)		Improvement
	DOPL	CNN	
Smartphone-Based Recognition of Human Activities and Postural Transitions [37] †	**80**	5.6×10^4	1.4×10^{-3}

Figures in the 'Improvement' column show the time for DOPL as a fraction of that for CNN.

†System Specifications – Processor: Intel® Xeon® CPU E5-2620 0 @ 2.00GHz×12, Memory: 24.0 GiB, OS: Ubuntu 18.04.1LTS

DOPL and CNN separates very clearly pre-training and running, DOPL used only 1 epoch of training, while CNN used 1,000 epochs. Thus, DOPL ran in 80 s, a fraction of the original time, 1.4×10^{-3} s for CNN. The accuracy was similarly improved (see Table 7). We also compared the accuracy of DOPL with 16 models: it showed better accuracy than all others - a worst (or 'safe') case of 99.20%. Table 7 compares our model with other models as mentioned above. We have highlighted (bold face), the worst case for three runs of each model and the standard deviation. Thus we show that our model **always** performs better than others, as reflected in better worst case numbers. The standard deviation is a measure of run-to-run variation.

Table 7. Accuracy of DOPL *vs* other algorithms

Model	Testing Fold (%)			σ	Imp.
	1	2	3		
Deep One-Pass Learning (DOPL)	99.56	99.26	**99.20**	±0.19	-
Convolutional Neural Network (CNN)					
CNN1	**97.61**	97.80	98.11	±0.25	1.6%
CNN2 [21] *	**95.18**	-	-	-	4.0%
CNN3 [38] *	**94.79**	-	-	-	4.4%
CNN4 [39] *	**90.00**	-	-	-	9.2%
CNN5 [40] *	**94.61**	-	-	-	4.6%
Extreme Gradient Boosting (XGBoost)	**97.04**	97.09	97.06	±0.02	2.2%
Logistic Regression (LR)	**96.24**	96.60	96.95	±0.29	3.0%
Extra Trees Classifier (ETC)	93.99	**93.91**	93.96	±0.03	5.3%
Gradient Boosting Classifier (GBC)	**96.76**	96.93	96.87	±0.07	2.4%
Random Forest Classifier (RFC)	**94.26**	94.54	94.84	±0.24	4.9%
Gaussian Naive Bayes (GNB)	**70.22**	73.43	73.92	±1.64	29.0%
Decision Tree Classifier (DTC)	90.53	**90.42**	90.89	±0.20	8.8%
Multi-layer Perceptron Classifier (MLP)					
MLP1	**96.10**	96.21	96.73	±0.28	3.1%
MLP2	96.79	97.01	**96.76**	±0.11	2.4%
MLP3	96.71	**96.68**	97.06	±0.18	2.5%
MLP4	96.62	**95.94**	96.10	±0.29	3.3%
Support Vector Classification (SVC)					
SVC1	92.48	**91.82**	92.86	±0.43	7.4%
SVC2	**97.15**	97.20	97.39	±0.11	2.1%
SVC3	89.13	**88.09**	88.25	±0.46	11.1%
SVC4	89.76	**88.77**	89.46	±0.41	10.4%

Note: Improvements *vs* DOPL are shown in the 'Imp.' column.

* From there source, only an average value was available.

CNN and XGBoost are representative of state-of-the-art models, but DOPL provided higher accuracy than both models, 1.6% improvement compared to the best of five implementations of CNN and 2.2% compared to XGBoost. For the MLP variants, performance was in the same

range as CNN, but DOPL also provided more accurate results. SVC2 was also in the same range as the CNN, but worse than DOPL.

5. Conclusion

We designed a *deep one-pass learning model* (DOPL) based on pre-trained weights: this model used only one epoch to recognize smartphone-based human activities and postural transitions. In addition, it can learn a data set without the need to repeatedly adjust the weights. The number of convolutional layers can be increased or decreased with some constraints, and the data set in the features learning process will be passed into XGBoost, which is the last layer of the model. Pre-training context, we obtain a set of weights from the pre-training model, CNN was used as a prototype and training model with HAR data set in Section 2.1. The model trained by our algorithm learnt the data set effectively. Experimental results show that our model outperformed state-of-the-art models (CNN and XGBoost) used as prototypes for designing our model, as well as a selection of other commonly used models.

Finally, DOPL has expanded the potential of deep learning techniques. We reduced the time in recognition: our model was faster than traditional CNN: 80 s *vs* 56,000 s, or several hundred times faster. In addition, a survey and review state-of-the-art, and research challenges for human activity recognition, our pre-training weights method has not been explained, which challenges the further development of the research in the future. Therefore, our pre-training weights method is a new thing for human activity recognition tasks. It opens the issue of challenging problems for future development and research improvement.

Acknowledgement

This work is supported by the Thailand Research Fund (TRF) under grant number RTA6080013. In addition, we thank Prof. John Morris of the KMITL Research and Innovation Services (KRIS) for editing the final manuscript.

Competing Interests

The authors declare that they have no competing interests.

Authors' Contributions

All the authors contributed significantly in writing this article. The authors read and approved the final manuscript.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, software available from `http://tensorflow.org/` (2015).

[2] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone, in *Classification and Regression Trees*, Wadsworth and Brooks, Monterey, CA (1984), https://www.bibsonomy.org/bibtex/27f293aa2bdfd10960ef36928f2795f1d/machinelearning.

[3] L. Breiman, Random forests, *Machine Learning*, **45**(1) (2001), 5–32, DOI: 10.1023/A:1010933404324.

[4] T. F. Chan, G. H. Golub and R. J. LeVeque, Updating formulae and a pairwise algorithm for computing sample variances, in *COMPSTAT 1982 5th Symposium held at Toulouse 1982*, Physica-Verlag HD, Heidelberg, 30–41 (1982).

[5] C.-C. Chang and C.-J. Lin, LIBSVM: A library for support vector machines, *ACM Transactions on Intelligent Systems and Technology*, **2**(3) (2011), 1–27, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[6] T. Chen and C. Guestrin, XGBoost: A scalable tree boosting system, in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '16), San Francisco, California, USA, 785–794, ACM, New York, USA (2016), DOI: 10.1145/2939672.2939785.

[7] A. Defazio, F. R. Bach and S. Lacoste-Julien, SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives, *Advances in Neural Information Processing Systems*, *abs/1407.0202* (2014), 1–1, http://arxiv.org/abs/1407.0202, retrieved on 13 August 2018.

[8] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. De Fauw, M. Heilman, diogo149, B. McFee, H. Weideman, takacsg84, peterderivaz, Jon, instagibbs, K. Rasul, CongLiu, Britefury, and J. Degrave, *Lasagne: First Release* (2015), DOI: 10.5281/zenodo.27878.

[9] D. Dua and C. Graff, UCI Machine Learning Repository, https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones, University of California, Irvine, School of Information and Computer Sciences (2015).

[10] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang and C.-J. Lin, LIBLINEAR: A Library for Large Linear Classification, *Journal of Machine Learning Research*, **9** (2008), 1871–1874, http://dl.acm.org/citation.cfm?id=1390681.1442794.

[11] J. Friedman, Greedy function approximation: A gradient boosting machine, *The Annals of Statistics* **29** (2000), DOI: 10.1214/aos/1013203451.

[12] J. H. Friedman, Stochastic gradient boosting, *Computational Statistics & Data Analysis* **38**(4) (2002), 367–378, DOI: 10.1016/S0167-9473(01)00065-2.

[13] P. Geurts, D. Ernst and L. Wehenkel, Extremely randomized trees, *Machine Learning* **63**(1) (2006), 3–42, DOI: 10.1007/s10994-006-6226-1.

[14] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, http://www.deeplearningbook.org (2016).

[15] S. Gross and M. Wilber, *Training and Investigating Residual Nets*, http://torch.ch/blog/2016/02/04/resnets.html (2016).

[16] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, **1** (2009), *Springer*.

[17] P. He, X. Jiang, T. Sun and H. Li, Computer graphics identification combining convolutional and recurrent neural networks, *IEEE Signal Processing Letters* **25**(9) (2018), 1369–1373, DOI: 10.1109/LSP.2018.2855566.

[18] K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognitions, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778 (2016).

[19] G. E. Hinton, Connectionist learning procedures, *Artificial Intelligence* **40**(1) (1989), 185–234, DOI: 10.1016/0004-3702(89)90049-0.

[20] A. Jabri, A. Joulin and L. van der Maaten, Revisiting visual question answering baselines, in *Computer Vision – ECCV 2016*, Springer International Publishing, Cham., 727–739 (2016).

[21] W. Jiang and Z. Yin, Human activity recognition using wearable sensors by deep convolutional neural networks, in *Proceedings of the 23rd ACM International Conference on Multimedia*, MM'15 series, 2015, Brisbane, Australia, 1307–1310, ACM, New York, USA, DOI: 10.1145/2733373.2806333.

[22] E. Kim, S. Helal and D. Cook, Human activity recognition and pattern discovery, *IEEE Pervasive Computing* **9**(1) (2010), 48–53, DOI: 10.1109/MPRV.2010.7.

[23] A. Krizhevsky, I. Sutskever and G. E. Hinton, ImageNet classification with deep convolutional neural networks, in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, Lake Tahoe, Nevada, 1097–1105, `http://dl.acm.org/citation.cfm?id=2999134.2999257`, Curran Associates Inc., USA (2012).

[24] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, Gradient-based learning applied to document recognition, *Proceedings of the IEEE* **86**(11) (1998), 2278–2324, DOI: 10.1109/5.726791.

[25] S. Lee, T. Chen, L. Yu and C. Lai, Image classification based on the boost convolutional neural network, *IEEE Access* **6** (2018), 12755–12768, DOI: 10.1109/ACCESS.2018.2796722.

[26] J. Lemley, S. Bazrafkan and P. Corcoran, Deep learning for consumer devices and services: pushing the limits for machine learning, artificial intelligence, and computer vision, *IEEE Consumer Electronics Magazine* **6**(2) (2017), 48–56, DOI: 10.1109/MCE.2016.2640698.

[27] G. Liang, H. Hong, W. Xie and L. Zheng, Combining convolutional neural network with recursive neural network for blood cell image classification, *IEEE Access* **6** (2018), 36188–36197, DOI: 10.1109/ACCESS.2018.2846685.

[28] D. Liciotti, M. Bernardini, L. Romeo and E. Frontoni, A sequential deep learning application for recognising human activities in smart homes, *Neurocomputing* (2019), DOI: 10.1016/j.neucom.2018.10.104.

[29] V. Lioutas, N. Passalis and A. Tefas, Explicit ensemble attention learning for improving visual question answering, *Pattern Recognition Letters* **111** (2018), 51–57, DOI: 10.1016/j.patrec.2018.04.031.

[30] H. Liu and L. Wang, Gesture recognition for human-robot collaboration: a review, *International Journal of Industrial Ergonomics* **68** (2018), 355–367, DOI: 10.1016/j.ergon.2017.02.004.

[31] Z. Lu, K. Tong, X. Zhang, S. Li and P. Zhou, Myoelectric pattern recognition for controlling a robotic hand: a feasibility study in stroke, *IEEE Transactions on Biomedical Engineering* **66**(2) (2019), 365–372, DOI: 10.1109/TBME.2018.2840848.

[32] T. Mikolov, K. Chen, G. Corrado and J. Dean, Efficient estimation of word representations in vector space, (2013), 1–12, `https://arxiv.org/abs/1301.3781`.

[33] H. Nguyen, L. Kieu, T. Wen and C. Cai, Deep learning methods in transportation domain: a review, *IET Intelligent Transport Systems* **12**(9) (2018), 998–1004, DOI: 10.1049/iet-its.2018.0064.

[34] H. F. Nweke, Y. W. Teh, M. A. Al-Garadi and U. R. Alo, Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: state of the art and research challenges, *Expert Systems with Applications* **105** (2018), 233–261, DOI: 10.1016/j.eswa.2018.03.056.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, Scikit-learn: machine learning in Python, *Journal of Machine Learning Research* **12** (2011), 2825–2830.

[36] C. N. Phyo, T. T. Zin and P. Tin, Deep learning for recognizing human activities using motions of skeletal joints, *IEEE Transactions on Consumer Electronics* **65**(2) (2019), 243–252, DOI: 10.1109/TCE.2019.2908986.

[37] J.-L. Reyes-Ortiz, L. Oneto, A. Samà, X. Parra and D. Anguita, Transition-aware human activity recognition using smartphones, *Neurocomput.* **171**(C) (2016), 754–767, DOI: 10.1016/j.neucom.2015.07.085.

[38] C. A. Ronao and S.-B. Cho, Deep convolutional neural networks for human activity recognition with smartphone sensors, in *Neural Information Processing*, 46–53 (2015), Springer International Publishing, Cham.

[39] C. A. Ronao and S.-B. Cho, Evaluation of deep convolutional neural network architectures for human activity recognition with smartphone sensors, in *Proceedings of the KIISE Korea Computer Congress*, Korea, 858860 (2015).

[40] C. A. Ronao and S.-B. Cho, Human activity recognition with smartphone sensors using deep learning neural networks, *Expert Syst. Appl.* **59** (2016), 235–244.

[41] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, Going deeper with convolutions, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9 (2015), DOI: 10.1109/CVPR.2015.7298594.

[42] Theano Development Team, Theano: A Python framework for fast computation of mathematical expressions, *CoRR* (2016), 1–19, `http://arxiv.org/abs/1605.02688`, 13 August 2018.

[43] M. Wainberg, D. Merico, A. Delong and B. J. Frey, Deep learning in biomedicine, *Nature Biotechnology* **36** (2018), 829–838, DOI: 10.1038/nbt.4233.

[44] J. Wang, Y. Chen, S. Hao, X. Peng and L. Hu, Deep learning for sensor-based activity recognition: a survey, *Pattern Recognition Letters* **119** (2019), 3–11, DOI: 10.1016/j.patrec.2018.02.010.

[45] J. Wang, Y. Ma, L. Zhang, R. X. Gao and D. Wu, Deep learning for smart manufacturing: methods and applications, *Journal of Manufacturing Systems* (Special Issue on Smart Manufacturing) **48** (2018), 144–156, DOI: 10.1016/j.jmsy.2018.01.003.

[46] Q. Wu, D. Teney, P. Wang, C. Shen, A. Dick and A. van den Hengel, Visual question answering: a survey of methods and datasets, *Computer Vision and Image Understanding* **163** (2017), 21–40, DOI: 10.1016/j.cviu.2017.05.001.

[47] H.-F. Yu, F.-L. Huang and C.-J. Lin, Dual coordinate descent methods for logistic regression and maximum entropy models, *Machine Learning* **85**(1) (2011), 41–75, DOI: 10.1007/s10994-010-5221-8.

[48] P. Zham, D. K. Kumar, P. Dabnichki, S. A. Poosapadi and S. Raghav, Distinguishing different stages of Parkinsons disease using composite index of speed and pen-pressure of sketching a spiral, *Frontiers in Neurology* **8** (2017), 435, DOI: 10.3389/fneur.2017.00435.

[49] Z. Zhang, S. Shan, Y. Fang and L. Shao, Deep learning for pattern recognition, *Pattern Recognition Letters* **119** (2019), 1–2, DOI: 10.1016/j.patrec.2018.10.028.