Nuclear Engineering and Technology 53 (2021) 522–531

Contents lists available at ScienceDirect

Nuclear Engineering and Technology

journal homepage: www.elsevier.com/locate/net

Original Article

ConvXGB: A new deep learning model for classification problems based on CNN and XGBoost

Setthanun Thongsuwan [a], Saichon Jaiyen [a], Anantachai Padcharoen [b], Praveen Agarwal [c, *]

[a] Advanced Artificial Intelligence (AAI) Research Laboratory, Department of Computer Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok, 10520, Thailand
[b] Department of Mathematics, Faculty of Science, Rambhai Barni Rajabhat University, Chanthaburi, 22000, Thailand
[c] Department of Mathematics, Anand International College of Engineering, Jaipur, 303012, India

ARTICLE INFO

Article history:
Received 8 October 2019
Received in revised form
28 March 2020
Accepted 7 April 2020
Available online 2 August 2020

Keywords:
Convolutional neural network (CNN)
Classification algorithms
Deep learning
Extreme gradient boosting
XGBoost
Machine learning
Pattern recognition

ABSTRACT

We describe a new deep learning model - Convolutional eXtreme Gradient Boosting (ConvXGB) for classification problems based on convolutional neural nets and Chen et al.'s XGBoost. As well as image data, ConvXGB also supports the general classification problems, with a data preprocessing module. ConvXGB consists of several stacked convolutional layers to learn the features of the input and is able to learn features automatically, followed by XGBoost in the last layer for predicting the class labels. The ConvXGB model is simplified by reducing the number of parameters under appropriate conditions, since it is not necessary re-adjust the weight values in a back propagation cycle. Experiments on several data sets from UCL Repository, including images and general data sets, showed that our model handled the classification problems, for all the tested data sets, slightly better than CNN and XGBoost alone and was sometimes significantly better.

1. Introduction

Learning models are key to solving some machine learning problems, so, we must focus on learning models. Classification is a major topic for machine learning and we must use the appropriate learning criteria for predicting a class label — this is called supervised learning. Classification problems may be found in image processing [1,2], text and document classification [3,4], intrusion detection [5,6], medical diagnosis [7–10] and pattern recognition [11–14]: these problems may be either binary or multi-class. Currently, the world is in an enormous store of information and much of it is very complex with data having many features [15–17]. We are all affected by this data and significant knowledge and understanding may be found in it, if analysed appropriately. In recent years, deep learning has become widely used as a learning model and has led to striking advances in fields such as computer vision [18–20] and classification problems [21–23], for example, Mironczuk and Protasiewicz have surveyed its application to text classification [3]. In solving machine learning problems, feature learning has become a driving force for success [24]. Generally, the mass of information in the real world has made it difficult for models to discover the key features in any particular problem. For this reason, common methods rely on manual feature engineering, which leads to features that may not aid the best solution, because the external environment may bias some features and lead to inappropriate feature additions or deletions. Automatic feature learning can solve this problem; it allows the system to automatically discover key features from raw data. Therefore, we need to realize the importance of effective models and also the ability of automatic feature learning to find a complete model. However, most models, including the traditional shallow models [25] and deep learning [26–28], still use only a single model for training.

We based our work on two hypotheses:

1. The capabilities of a single model may not be sufficient to meet the accuracy required for various problems.
2. If we analyze the model carefully and fairly, then each model has its own advantages and disadvantages, if we can discover them,

* Corresponding author.
E-mail addresses: tsetthanun@gmail.com (S. Thongsuwan), saichon.ja@kmitl.ac.th (S. Jaiyen), anantachai.p@rbru.ac.th (A. Padcharoen), goyal.praveen2011@gmail.com (P. Agarwal).

https://doi.org/10.1016/j.net.2020.04.008

we can take advantage of them to develop better models, and avoid the disadvantages. Thus, we look for those advantages.

We describe a Convolutional eXtreme Gradient Boosting (ConvXGB) algorithm as a new deep learning model for classification problems. ConvXGB combines the performance of a Convolutional Neural Network (CNN) [29] and eXtreme Gradient Boosting (XGBoost) [30], which, as we will show, leads to high accuracy, and a state-of-the-art performance. A key feature of ConvXGB is a systematic strategy for choosing these two models:

1. XGBoost, commonly used by data scientists, is a scalable machine learning system for tree boosting which avoids overfitting. It performs well on its own and have been shown to be successful in many machine learning competitions. However, we observe that this model is still unclear for feature learning.
2. We add clarity by adding automatic feature learning with CNN, a class of deep learning, containing hierarchical learning in several different layers.

To evaluate ConvXGB and show that it will meet our goal to not only solve image classification problems, we included some additional general classification problems applicable to other tasks.

The major contributions of this paper are:

– A new deep learning model for classification problems called "ConvXGB" based on combine between CNN and XGBoost.
– The ConvXGB architecture consists of a net with several stacked convolutional layers and with XGBoost as the last layer of the model. It differs from the traditional CNN, because there is neither a pooling layer nor a Fully Connected (FC) layer: this introduces simplicity and reduces the number of calculation parameters, since it is not necessary to bring weights from the FC layers back to re-adjust weights in the previous layers.
– We measured the performance of ConvXGB with both image processing and general classification problems to show its general applicability.
– ConvXGB uses auto feature learning effectively and predicts class labels, with higher accuracy than the two individual models, which are the current prototypes for modeling, and other extant models, e.g. Decision Tree Classification (DTC [31]), Multilayer Perceptron (MLP [32]) and Support Vector Classification (SVC [33]).

The remainder of this paper is organized as follows: in Section 2, we briefly review CNN and the XGBoost model. In Section 3, we describe our ConvXGB model in detail. The experiments and results are presented in Section 4. Finally, we discuss the result and conclude.

2. Related work

2.1. Convolutional Neural Network (CNN)

In this section, we overview the CNN formulation and explain the mathematical theory behind it [29,34–36]. We assume an input $W \times H$, grey scale image, $\mathscr{I} \in R^{W \times H}$, represented as:

$$\mathscr{I} = \{x(m,n) | 1 \le m \le W, 1 \le n \le H\}, \tag{1}$$

when $x(m,n)$ is the intensity of the pixel at (m,n) and given the $w_k \times h_k$ filters (or kernels), $\mathscr{K}$, the convolution produces a feature map, $\mathscr{Y}$, from the image, $\mathscr{I}$, by applying the filter, $\mathscr{K}$. The filter $\mathscr{K}$ is slid through the image, $\mathscr{I}$ by a stride, S_k, and zero padding value. We can define the discrete convolution as:

$$(\mathscr{I} \otimes \mathscr{K})_{m,n} = \sum_{u=-w_k}^{w_k} \sum_{v=-h_k}^{h_k} K_{u,v} I_{m+u,j+v}, \tag{2}$$

In each convolutional layer, indexed by l, a convolution operation and an additive bias will be applied to the input, for a feature map indexed by $f \in \{1, \ldots, f(l)\}$. So the output, $\mathscr{Y}_i^{(l)}$, of the l^{th} layer for the i^{th} feature map, is derived from the output of the previous layer, $\mathscr{Y}_i^{(l-1)}$, by:

$$\mathscr{Y}_i^{(l)} = \phi\left(B_i^{(l)} + \sum_{j=1}^{f^{(l-1)}} K_{i,j}^{(l)} * \mathscr{Y}_j^{(l-1)}\right), \tag{3}$$

where ϕ is Rectified Linear Unit (ReLU) activation function, $\mathscr{B}_i^{(l)}$ is a bias matrix, $\mathscr{K}_{i,j}^{(l)}$ is the filter of size $2w_k + 1 \times 2h_k + 1$.

Thus, the elements of the output of layer, l, for feature map, i, $\mathscr{Y}_i^{(l)}$, at position (m,n) is:

$$\begin{aligned}\left(\mathscr{Y}_i^{(l)}\right)_{m,n} &= \phi\left(\left(\mathscr{B}_i^{(l)}\right)_{m,n} + \sum_{j=1}^{f^{(l-1)}} \left(K_{i,j}^{(l)} \otimes \mathscr{Y}_j^{(l-1)}\right)_{m,n}\right) \\ &= \phi\left(\left(B_i^{(l)}\right)_{m,n} + \sum_{j=1}^{f^{(l-1)}} \sum_{u=-w_k^{(l)}}^{w_k^{(l)}} \sum_{v=-h_k^{(l)}}^{h_k^{(l)}} \left(\mathscr{K}_{i,j}^{(l)}\right)_{m,n} \left(\mathscr{Y}_j^{(l-1)}\right)_{m+u,n+v}\right)\end{aligned} \tag{4}$$

A pooling layer further modifies the layer output: it downsamples and avoids overfitting of the output. It replaces the output with the maximum or average value within a rectangular neighborhood. For example, if $(\mathscr{Y}_i^{(l)})_{m,n}$ is an output of the previous layer, with ϕ activation function, then $P(\cdot)$ is a pooling function which acts on $\mathscr{Y}_i^{(l)}$ by passing it through a pooling process with stride, S_p, and $w_p \times h_p$ pooling window. In general, pooling operates by placing windows at non-overlapping positions in each feature map and keeping one value per window so that the feature maps are subsampled. Two types of pooling are commonly used: average pooling and max pooling: max pooling, in which the maximum value of each window is taken, is usually applied. Thus the output of a max-pooling function becomes:

$$P\left(\mathscr{Y}_i^{(l)}\right)_{m,n} = max\left(\mathscr{Y}_i^{(l)}\right)_{m,n}, \tag{5}$$

where the *max* function is applied to the max-pooling window of dimension:

$$dP\left(\mathscr{Y}_i^{(l)}\right)_{m,n} = ((W - w_k)/S_p + 1) \times ((H - h_k)/S_p + 1), \tag{6}$$

A Fully Connected (FC) layer is the last layer of the CNN architecture: the output from the previous pooling layer will be stretched to a single column vector and become the input of this layer. In the FC layer, all neurons in the previous layer are connected to every neuron in the following another. Generally, in FC layers, we can apply the well-known equations for multilayer perceptrons. Let L be a number of FC layers, $f_1^{(l)}$ is the number feature maps of size $f_2^{(l)} \times f_3^{(l)}$ (following the notation used by Stutz [35]) and the i^{th} feature map in layer l, is computed

$$\left(\mathscr{Y}_i^{(l)}\right)_{m,n} = \phi\left(\sum_{p=1}^{f_1^{(l-1)}}\sum_{q=1}^{f_2^{(l-1)}}\sum_{r=1}^{f_3^{(l-1)}} w_{i,p,q,r}^{(l)}\left(\mathscr{Y}_j^{(l-1)}\right)_{q,r}\right). \tag{7}$$

where $w_{i,p,q,r}^{(l)}$ is the weight connecting the unit at (m,n) in feature map, i, in layer, l, to the one at the position (q,r) in feature map, p, in layer $(l-1)$. In practice, *softmax* is the transformation function for multi-class prediction, and dropout regularization is used to decreases number of neurons in the FC layer to avoid overfitting.

2.2. Extreme Gradient Boosting (XGBoost)

The Extreme Gradient Boosting (XGBoost) of Chen and Guestrin is a highly scalable end-to-end tree boosting system - a machine learning technique for classification and regression problems [30]. XGBoost uses an ensemble of K classification and regression trees (CARTs), each of which has $K_E^i \middle| i \in 1..K$ nodes. The final prediction is the sum of the prediction scores for each tree:

$$\hat{y}_i = \varphi(x_i) = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in F, \tag{8}$$

where the x_i are members of the training set and y_i are the corresponding class labels, f_k is the leaf score for the k^{th} tree and F is the set of all K scores for all CARTs. Regularization is applied to improve the final result:

$$\mathscr{L}(\varphi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{9}$$

The first term, l, represents the differentiable loss function, which measures the difference between target y_i and the prediction $\hat{y}_i$. The second term avoids over-fitting: Ω penalizes the complexity of the model:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{10}$$

where γ, λ are constants controlling the regularization degree, T is the number of leaves in the tree and w is the weight of each leaf. Gradient boosting (GB) is effective in regression and classification problems. GB was used with the loss function, extended by a second order Taylor expansion, with the constant term removed to produce a simplified objective at step t, as follows:

$$\begin{aligned}\tilde{\mathscr{L}}^{(t)} &\simeq \sum_{i=1}^{n}\left[g_i f_i(x_i) + \frac{1}{2}h_i f_i^2(x_i)\right] + \Omega(f_t) \\ &= \sum_{i=1}^{n}\left[g_i f_i(x_i) + \frac{1}{2}h_i f_i^2(x_i)\right] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right) w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right) w_j^2\right] + \gamma T\end{aligned} \tag{11}$$

where $I_j = \{i | q(x_i) = j\}$ denote the instance set of leaf t, and

$$g_i = \frac{\partial l(\hat{y}_i^{(t-1)}, y_i)}{\partial \hat{y}_i^{(t-1)}} \tag{12}$$

$$h_i = \frac{\partial^2 l(\hat{y}_i^{(t-1)}, y_i)}{\partial\left(\hat{y}_i^{(t-1)}\right)^2} \tag{13}$$

Are first and second order gradient statistics of the loss function. The optimal weight w_j^* of leaf j and the quality of a tree structure q, for a given tree structure $q(x_i)$ can be computed:

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda}, \tag{14}$$

$$\tilde{\mathscr{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum_{i\in I_j} g_i\right)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T. \tag{15}$$

In practice, the evaluating for split candidates by utilized the score in the instance sets of left I_L and right I_R nodes after the split, where $I = I_R \cup I_L$, then the loss reduction after the split is:

$$\mathscr{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} + \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{16}$$

3. ConvXGB

In this section, the ConvXGB model is described in detail. The architecture has two main parts: each part is composed of several different layers-see Fig. 1. In Section 3.2, we describe the training algorithm. The data sets used are listed in Table 1.

3.1. ConvXGB architecture

The architecture of ConvXGB is shown in Fig. 1. The model has six layers: 1) input layer, 2) data preprocessing layer, 3) convolutional layers, 4) reshape layer, 5) class prediction layer and 6) output layer. Each layer has different capabilities and responsibilities: these layers are the keys to the success of the model. Each layer can be divided into two parts: one for feature learning and the other to predict the class labels. The types of layers are described next.

3.1.1. Feature learning part

This part learns the key features from the training data set. This part has three layers: the input, data preprocessing convolutional layers. Prediction accuracy depends on effective feature learning. Details of each layer follow.

Input layer: The input layer is the first layer and is responsible for the input of the model. We assume a training data set, X, consists of a set of tuples, (x_j, y_j), where j is the index of the data set. x_j is a $\sqrt{N} \times \sqrt{N}$ feature matrix and y_j is the class label assigned to vector, x_j. If the training set is in this required format, it will be passed to directly to the convolutional layers of the feature learning section, otherwise, it will be formatted in the data preprocessing layer - described next.

Data preprocessing layer: To make our system flexible and able to handle data coming from many sources, we decided to use a common square matrix format for the tensors in the convolution sections. In this layer, if the input data is not our standard square form, with dimensions, $\sqrt{N} \times \sqrt{N}$, we convert it, by padding as necessary, to the common form. Different data types are also converted in this layer.

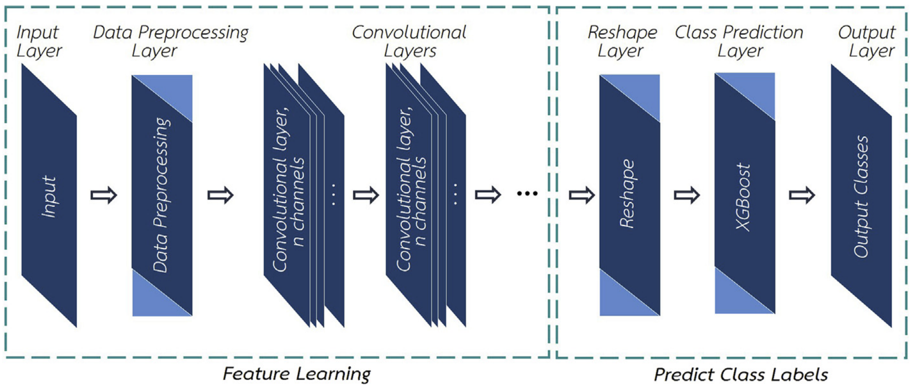

Fig. 1. Architecture of the ConvXGB model.

Table 1
Data sets used: taken from UCI Repository [37].

Data sets	Area	Instances	Features	Features after reshape	Classes	Output depth
Anuran Calls (MFCCs) [38]	Life	7195	22	800	60	32
Breast Cancer Wisconsin (Original) [39]	Life	699	10	512	2	32
DrivFace [40]	Computer	606	6400	51,200	10	8
Parkinsons [41]	Life	197	23	800	2	32
QSAR Biodegradation [42]	N/A	1055	41	784	2	16
Sensorless Drive Diagnosis [38]	Computer	58,509	49	1568	11	32
Waveform Database Generator (ver. 1) [38]	Physical	5000	40	800	3	32
Waveform Database Generator (ver. 2) [38]	Physical	5000	40	12,544	3	256

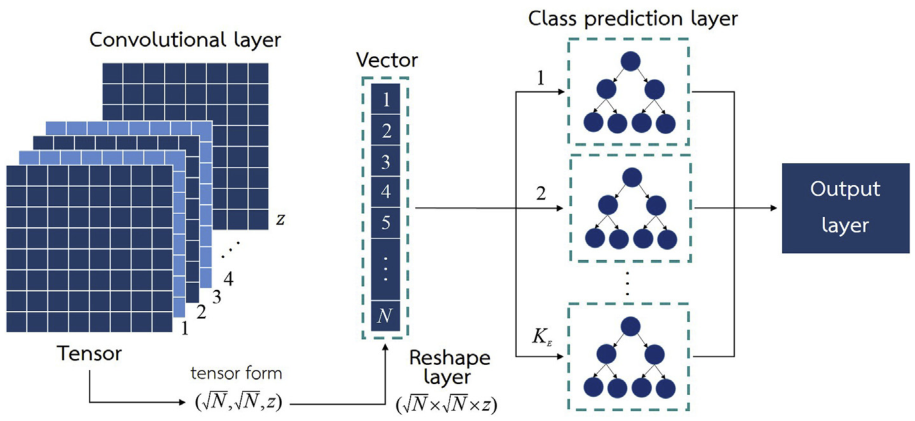

Fig. 2. The process of operation in part of predict the class labels consists of reshape layer, class prediction layer and output layer.

The required format is further described in Section 3.3.

Convolutional layer: The convolutional layers are the core layers in this part; they are responsible for feature learning and apply a convolution and an additive bias to the input data. The data is logically a tensor, with dimensions, $(\sqrt{N},\sqrt{N},z^{(l)})$, see left of Fig. 2, where $z^{(l)}$ is the number of filters (or output depth) in the l layer. For example, with 9 feature vectors and 64 output convolution channels, $N = 9$ and $x = [x_1, x_2, \ldots, x_9]$ is a set of feature vectors in $\mathbb{R}^9$ or $\mathbb{R}^{\sqrt{9}\times\sqrt{9}}$ and $d = 64$ and the output of each convolutional layer is a tensor with $3 \times 3 \times 64$ elements.

We can set the number of convolutional layers as required. However, we must be careful when increasing the number of the convolutional layers, because, as the number of layer increases, the computation time will increase. Adding too many layers may outweigh any advantage and we must balance carefully any increase in accuracy with the cost incurred, including the availability of a machine with sufficient power to support the necessary computation. From Equation (3) in Section 2.1:

$$Y_i^{(l)} = \phi\left(B_i^{(l)} + \sum_{j=1}^{f^{(l-1)}} K_{i,j}^{(l)} * Y_j^{(l-1)}\right)$$

The computational complexity is set out in Section 3.2.

3.1.2. *Predicting the class labels*

This part predicts the class labels from training data through feature learning in the convolutional layers of the previous part. Before input to the prediction part, the input must be in the form of a vector, so a housekeeping operation transforms the tensor to a vector in the reshape layer - see middle of Fig. 2.

Reshape layer: This layer just runs some housekeeping operations to convert the (logical) tensors output from the convolutional layers to the vector required by the next layer.

Class prediction layer: The main task in this layer is to predict the class using XGBoost as the driving force. XGBoost uses a tree structure and we can set the number of trees, thus the size of the structure affects performance. The quality of a tree structure can be scored as:

$$\tilde{\mathscr{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum_{i\in I_j} g_i\right)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T \tag{17}$$

where $I_j = \{i|q(x_i) = j\}$ denotes the instance set of leaf t and $g_i = \frac{\partial l(\widehat{y}_i^{(t-1)}, y_i)}{\partial \widehat{y}_i^{(t-1)}}$, $h_i = \frac{\partial^2 l(\widehat{y}_i^{(t-1)}, y_i)}{\partial \widehat{y}_i^{(t-1)}}$ are first and second order gradient statistics of the loss function, γ and λ are constants to control the regularization degree, and T is the number of leaves in the tree. One of the key tasks for this layer is splitting into the best set of segments: we use the gain of the split in Equation (16). In each segment, sort the data according to feature values and visit the data will be implemented as a first step in sorted order to accumulate the gradient statistics.

Let I_R, I_L are the left and right instance sets and $I = I_R \cup I_L$ is their union, then the loss after the split is:

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

In practice, this formula is used for evaluating candidate splits by using the scores of the instance sets of the left and right child nodes after the split. In addition, the model will speed up the training process and reduce the number of samples that are used by ignoring sparse inputs - 0 features or missing values).

Output layer: Output layer will get class predictable by class prediction layer. Finally, these classes are evaluated for accuracy, which represents the learning performance of the model.

3.2. *ConvXGB learning algorithm*

Let $X = \{(x_j, y_j)\big| 1 \le j \le M\}$, where M is the size of training data set, $x_j = [x_1, x_2, \ldots, x_N]$ be a set of N feature vectors in $\mathbb{R}^N$ or $\mathbb{R}^{\sqrt{N}\times\sqrt{N}}$ and y_j is the label of vector x_j. The learning algorithm for the ConvXGB can be summarized as follows:

1. Initialize the training data set, $X = \{(x_j, y_j)\big| 1 \le j \le M\}$.
2. If necessary, pad the N elements of each training data item, x_j, so that new data item can be formed into a square matrix of dimensions, $\sqrt{N}\times\sqrt{N}$.
3. Convert x_j tensor format, $(\sqrt{N}, \sqrt{N}, z^{(l)})$.
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 filter sizes, $K^{(l)}$, and
   (d) filter strides, $S_k^{(l)}$
5. For each layer, l, in $1\ldots L$: Calculate the convolutions (left side of Fig. 2) to generate the $\mathscr{Y}_i(l)$ for layer, l:

$$\mathscr{Y}_i^{(l)} = \phi\left(B_i^{(l)} + \sum_{j=1}^{f^{(l-1)}} K_{i,j}^{(l)} * \mathscr{Y}_j^{(l-1)}\right),$$

6. Reshape $\mathscr{Y}_i^{(l)}$ to a vector of length $(\sqrt{N}\times\sqrt{N}\times z^{(l)})$ - $\mathscr{Y}\mathscr{Y}^{(l)}$
7. Initialize a new training data set for class prediction layer

$$X_{new} = \left\{\left(\mathscr{Y}\mathscr{Y}_j, y_j\right)\Big| 1 \le j \le M\right\}.$$

8. Initialize parameters for the prediction step, set
   (a) total number of trees, K_K
   (b) regularization parameters, γ and λ,
   (c) column subsampling parameter,
   (d) maximum tree depth and
   (e) learning rate
9. Determine the class labels for output:

$$\widehat{y}_i = \varphi(YY_i) = \sum_{k=1}^{K_K} f_k(x_i), \quad f_k \in F,$$

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

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

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

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

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

where T is the number of leaves in the tree.

12. Calculate the best splitting points

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

13. Terminate

Our ConvXGB algorithm has time complexity: $\mathscr{O}Ld^2mnpq + \mathscr{O}r(Kt + \log B)$ which reduces to $\mathscr{O}Ld^2mnpq$, 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 filter 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.

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 by adding zeroes as necessary. For example, If the training data set, $X = \{(x_j, y_j) \big| 1 \le j \le M\}$ has M training data items, $x_j = [x_1, x_2, \ldots, x_N]$ be 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'. For example, the vector $[0,1,2,3,4,5,6,7,8]$ 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 $\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$ and shown in Fig. 3(a) However, if $\sqrt{N}$ is not integral, we pad the feature vector with zeroes and copy the padded vector to a square matrix. For example, as shown in Fig. 3(b), we have a vector $[0,1,2,3,4,5,6,7,8,9,10]$ in $\mathbb{R}^{11}$, so five zeroes, $[0,0,0,0,0]$, are added will leading to a 4×4 matrix (see Fig. 4) $\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$.

4. Experimental methods and results

The data sets used to evaluate the model performance for classification problems were collected from the University of California at Irvine (UCI) Repository of machine learning data sets [37]. The data sets were chosen from several areas, including data sets with few to very large numbers of instances and also sets where the number of attributes was much greater than the number of instances - see Table 1.

In this experiment, we used threefold 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 process 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 (see Fig. 5, Fig. 6, Fig. 7, Fig. 8, Fig. 9, Fig. 10, and Fig. 11).

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 3.2. Initially, we set the number of the convolutional layers or number of maps, $L = 2$, and the output depth of the convolutional layer, $z = 2^n$, where $n = 1, 2, \ldots, 10$. The chosen z value sets a balance between performance and use of resources: for example, if we choose the number of maps, z, as too small, then there will be insufficient to represent all the features in the data, but, if we choose it too large, then the data will be overfitted and the computation will need more memory than the computer has available. The actual values chosen are shown in Table 2. We set the filter size, $K = 2 \times 2$ and the stride of the filter $S_k = 1$ to enable small features to be recognized.

The results of our ConvXGB model were compared with other models shown in Tables 5(a) - 5(c) including Convolutional Neural Network (CNN) [29], eXtreme Gradient Boosting (XGBoost) [30], Decision Tree Classifier (DTC) [31], Multilayer Perceptron (MLP) [32] and the Support Vector Classification (SVC) [33]. We tried to configure the parameters in all cases to generate a fair comparison.

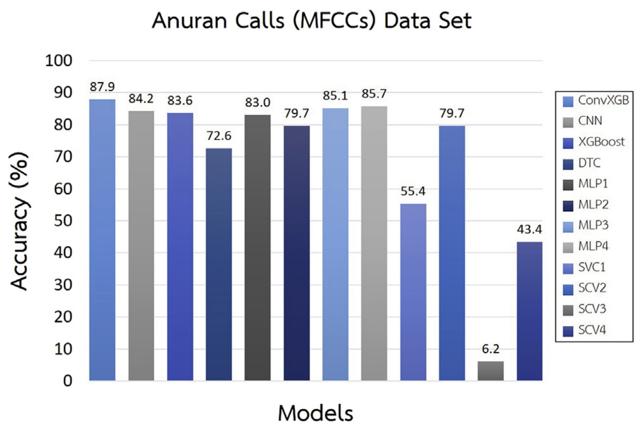

Fig. 4. ConvXGB *vs* all models for the Anuran Calls (MFCCs) data set [38].

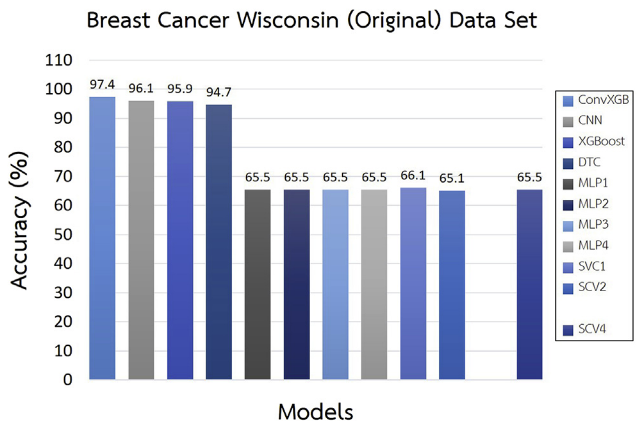

Fig. 5. ConvXGB *vs* all models for the Breast Cancer Wisconsin (Original) data set [39]. *SVC3 is not supported for Breast Cancer Wisconsin (Original) data set.

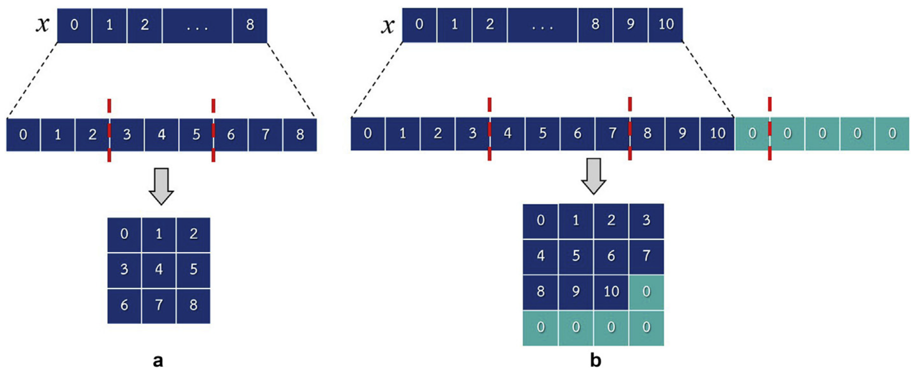

Fig. 3. Example of converting data to the standard criterion: (a) Describes of a ‘square’ feature vector which can be simply copied whereas, in (b), zeroes are added so that the feature vector is square.

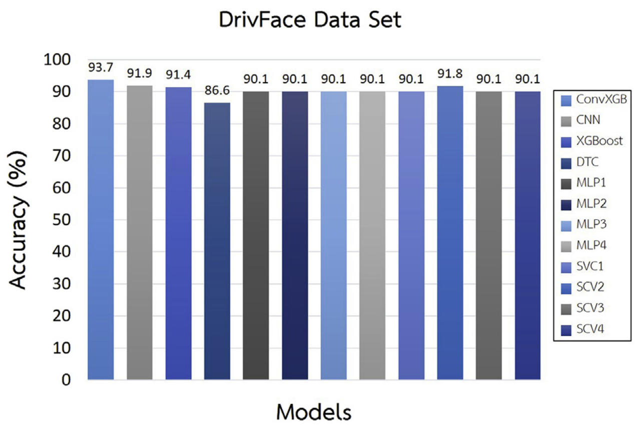

Fig. 6. ConvXGB *vs* all models for the DrivFace data set [40].

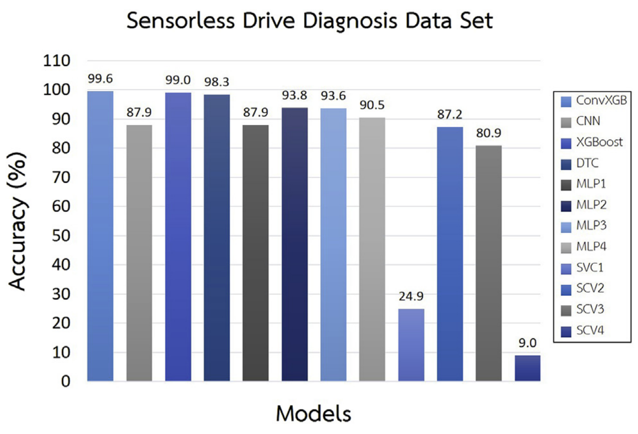

Fig. 9. ConvXGB *vs* all models for the Sensorless Drive Diagnosis data set [38].

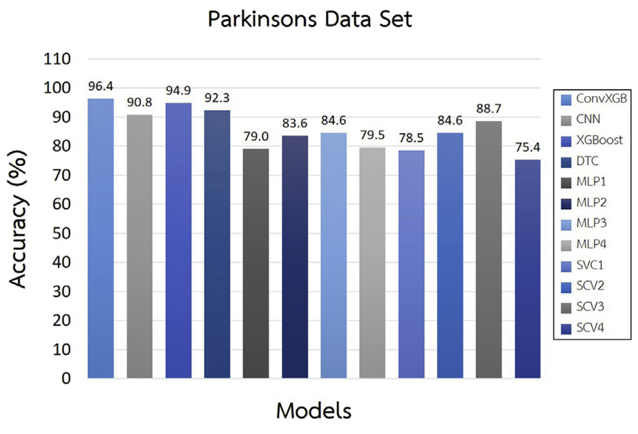

Fig. 7. ConvXGB *vs* all models for the Parkinsons data set [41].

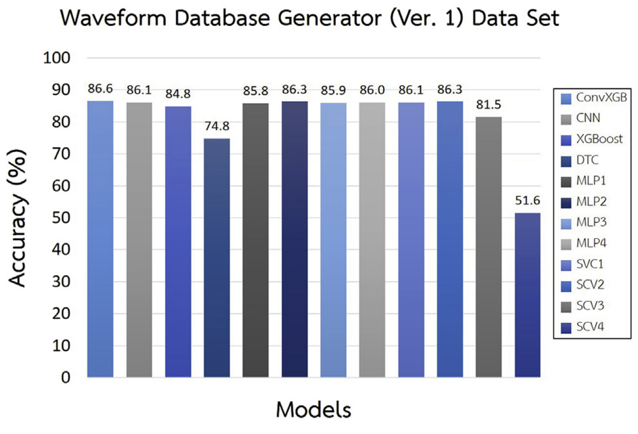

Fig. 10. ConvXGB *vs* all models for the Waveform Database Generator (Ver. 1) data set [38].

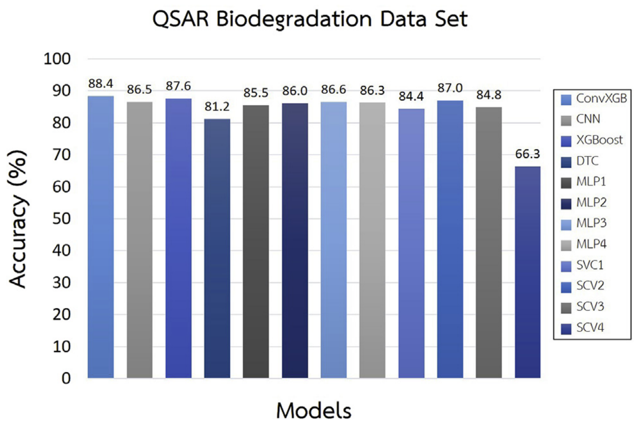

Fig. 8. ConvXGB *vs* all models for the QSAR Biodegradation data set [42].

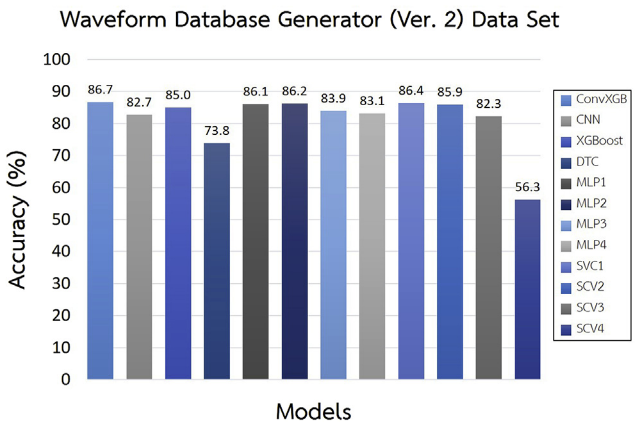

Fig. 11. ConvXGB *vs* all models for the Waveform Database Generator (Ver. 2) data set [38].

In the CNN model, the parameters in the convolutional layers were set according to our model with the difference that CNN had an added pooling layer of size 2×2 pool and stride 2, numbers of neurons in the FC layer was 2^n, when $n = 6, 7, \ldots, 10$. The XGBoost model was set similarly, with parameters set matching our model to fairly evaluate performance. Using these parameters, the accuracy of the two models was compared based on the same underlying resources. In the DTC model, the maximum depth of the tree was expanded until all leaves are pure or until all leaves contain less than $minss = 2$, where $minss$ (minimum samples split) is the minimum number of samples required to split an internal node. The Gini impurity was used as a criterion for the function to measure the quality of a split.

For the MLP model, the numbers of neurons is 2^n, when $n = 6, 7, \ldots, 10$, the learning rate was set to 0.001 and we used four variants, using different activation functions: linear (MLP1), sigmoid (MLP2), tanh (MLP3), and ReLU (MLP4).

Table 2
Parameters in each layer of the ConvXGB model.

Layer	Type	Input	Kernel(K)	stride (S_k)	Output
1	Input	$\sqrt{N}\times\sqrt{N}$	*na*	*na*	$\sqrt{N}\times\sqrt{N}$
2	Data Preprocessing	$\sqrt{N}\times\sqrt{N}$	*na*	*na*	$\sqrt{N},\sqrt{N},1$
3	Convolutional	$\sqrt{N},\sqrt{N},1$	2×2	1	$\sqrt{N},\sqrt{N},z^{(l-1)}$
4	Convolutional	$\sqrt{N},\sqrt{N},z^{(l-1)}$	2×2	1	$\sqrt{N},\sqrt{N},z^{(l)}$
5	Reshape	$\sqrt{N},\sqrt{N},z^{(l)}$	*na*	*na*	$\sqrt{N}\times\sqrt{N}\times z^{(l)}$
6	Class Prediction	$\sqrt{N}\times\sqrt{N}\times z^{(l)}$	*na*	*na*	No. of Classes
7	Output	*na*	*na*	*na*	No. of Classes

na = not applicable.

Table 3
The properties of the models used in the performance comparison.

Models	Parameters details	Ref.	Time Complexity
CNN	No. of Convolutional layer l = 2, Pooling = Yes	[29]	$\mathcal{O}Ld^2mnpq + \mathcal{O}MNhce$
XGBoost	Max_depth = 3, Objective = Binary: Logistic	[30]	$\mathcal{O}r(Kt + \log B)$
DTC	Criterion = Gini, *minss* = 2	[31]	$\mathcal{O}tMlogN$
MLP1	Activation = Linear, Learning rate = 0.001	[32]	$\mathcal{O}MNhce$
MLP2	Activation = Sigmoid, Learning rate = 0.001	[32]	$\mathcal{O}MNhce$
MLP3	Activation = tanh, Learning rate = 0.001	[32]	$\mathcal{O}MNhce$
MLP4	Activation = ReLU, Learning rate = 0.001	[32]	$\mathcal{O}MNhce$
SVC1	Kernel = RBF, C = 1.0	[33]	$\mathcal{O}M^3$
SVC2	Kernel = Linear, C = 1.0	[33]	$\mathcal{O}M^3$
SVC3	Kernel = Poly, C = 1.0	[33]	$\mathcal{O}M^3$
SVC4	Kernel = Sigmoid, C = 1.0	[33]	$\mathcal{O}M^3$

The SVC model similarly was divided into four variants with differing kernel functions: RBF (SVC1), linear (SVC2), polynomial (SVC3) and sigmoid (SVC4). In each variant, the penalty parameter of the error term was set to $C = 1.0$. Consequently, a total of 11 models were used: the properties are summarized in Table 3.

Furthermore, Table 3 shows the time complexity of the models. We assume that: L is the number of layers, d is the various of input or output channels, the data matrix has size $m \times n$, the filter 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 number of hidden neurons, c is the number of classes and e is the number of epochs.

We implemented and tested our and other models with Python (3.6.4) and functions from the TensorFlow library [43], the convolutional operation (also used in CNN), together with the XGBoost python package for the prediction step. For other models testing, we used the machine learning library from scikit-learn [44]. Our experiments used Linux (Ubuntu 18.04.1LTS) on a system specifications — Processor: Intel® Core™ i5-4570 CPU @ 3.20 GHz, Memory: 4.0 GiB, OS: Ubuntu 18.04.1LTS. Training times (for deep learning model) for our model, ConvXGB and CNN, and other models XGBoost, DTC, MLP, and SVC, show the run time of each model. ConvXGB is slower than XGBoost, DTC, MLP, and SVC. However, when compared to the deep learning model, CNN. ConvXGB is faster than CNN: it uses a one pass training - see Table 4.

5. Conclusions

We developed a new deep learning model for classification problems. ConvXGB, This model has two parts: one for feature learning and one to predict the class labels. We assessed ConvXGB, not only on image data, but also on some general data sets, which used our data preprocessing module. ConvXGB was simplified by reducing the number of needed parameters and did not require back propagation in the fully connected layer. The feature learning was also automatic. In addition, the number of the convolution layers can be increased, depending on the data.

ConvXGB was based on CNN and XGBoost, but our experimental results show that it was always slightly better and generally significantly better than these two models and also other models, which are often used as modeling prototypes *e.g.* the traditional DTC, MLP and SVC families. CNN also generally performed well and approached our results for some data sets, *e.g.* the Breast Cancer data set. XGBoost showed the next best result: in one instance, it returned 100% for the Parkinsons data set, but its average was slightly worse than our ConvXGB.

Table 4
Run time of our model compared with other models.

Data set	Run Time (sec.)					
	ConvXGB	CNN	XGBoost	DTC	MLP	SVC
Anuran Calls (MFCCs)	**43** s	1.6×10^6 s	5.4 s	13 s	17 s	23 s
Breast Cancer Wisconsin (Original)	**2.7** s	5.2×10^3 s	8.1×10^{-1} s	9.3×10^{-1} s	9.8×10^{-1} s	2.1×10^{-2} s
DrivFace	**2.2** $\times10^2$ s	2.4×10^4 s	7.4×10^{-1} s	3.1 s	4.7 s	1.4×10^{-2} s
Parkinsons	**1.2** s	1.5×10^3 s	4.5×10^{-1} s	3.6×10^{-1} s	6.3×10^{-1} s	4.7×10^{-1} s
QSAR Biodegradation	**6.2** s	7.9×10^3s	2.1 s	2.3 s	3 s	2.7 s
Sensorless Drive Diagnosis	**6.7** $\times10^2$ s	2.4×10^6 s	3.8×10^2 s	1.3×10^2 s	3.3×10^2 s	1.2×10^4 s
Waveform Database Generator (ver. 1)	**57** s	5.6×10^4 s	8.8 s	9.7 s	16 s	13 s
Waveform Database Generator (ver. 2)	**4.5** $\times10^2$ s	6.1×10^4 s	14 s	11 s	21 s	18 s

Table 5a
Our model compared with CNN, XGBoost and DTC.

Data set	Improvement (Impv.)						
	ConvXGB	CNN		XGBoost		DTC	
	Acc. (%)	Acc. (%)	Impv.	Acc. (%)	Impv.	Acc. (%)	Impv.
Anuran Calls (MFCCs)	**87.9**±1.3	84.2±0.9	3.7%	83.6±1.3	4.3%	72.6±1.0	15.3%
Breast Cancer Wisconsin (Original)	**97.4**±0.7	96.1±0.9	1.3%	95.9±0.2	1.5%	94.7±0.9	2.7%
DrivFace	**93.7**±1.9	91.9±2.6	1.8%	91.4±0.6	2.3%	86.6±2.3	7.1%
Parkinsons	**96.4**±0.7	90.8±1.3	5.6%	94.9±3.6	1.5%	92.3±6.7	4.1%
QSAR Biodegradation	**88.4**±0.6	86.5±0.5	1.9%	87.6±0.6	0.8%	81.2±1.0	7.2%
Sensorless Drive Diagnosis	**99.6**±0.0	87.9±2.2	11.7%	99.0±0.1	0.6%	98.3±0.1	1.3%
Waveform Database Generator (ver. 1)	**86.6**±0.4	86.1±0.5	0.5%	84.8±0.9	1.8%	74.8±0.1	11.8%
Waveform Database Generator (ver. 2)	**86.7**±1.0	82.7±0.8	4.0%	85.0±0.4	1.7%	73.8±1.3	12.9%

Note: Improvement to ConvXGB show in Impv. column.

Table 5b
Our model compared with the MLP family.

Data set	ConvXGB	MLP1	MLP2	MLP3	MLP4	Improvement
	Acc. (%)	Acc. (%)	Acc. (%)	Acc. (%)	Acc. (%)	
Anuran Calls (MFCCs)	**87.9**±1.3	83.0±0.4	79.7±0.7	85.1±0.7	†85.7±0.5	2.2%
Breast Cancer Wisconsin (Original)	**97.4**±0.7	65.5±3.3	65.5±3.3	65.5±3.3	65.5±3.3	31.9%
DrivFace	**93.7**±1.9	90.1±0.7	90.1±0.7	90.1±0.7	90.1±0.7	3.6%
Parkinsons	**96.4**±0.7	79.0±1.9	83.6±3.8	†84.6±3.3	79.5±1.9	11.8%
QSAR Biodegradation	**88.4**±0.6	85.5±1.8	86.0±1.3	†86.6±1.2	86.3±1.7	1.8%
Sensorless Drive Diagnosis	**99.6**±0.0	87.9±0.8	†93.8±0.4	93.6±0.7	90.5±0.7	5.8%
Waveform Database Generator (Ver. 1)	**86.6**±0.4	85.8±0.6	†86.3±0.8	85.9±0.6	86.0±0.5	0.3%
Waveform Database Generator (Ver. 2)	**86.7**±1.0	86.1±0.3	†86.2±0.4	83.9±0.4	83.1±0.2	0.5%

Note: Improvement shown as improvement of ConvXGB *vs* the best of MLP family -marked with a †.

Table 5c
Our model compared with the SVC family.

Data set	ConvXGB	SVC1	SVC2	SVC3	SVC4	Improvement
	Acc. (%)	Acc. (%)	Acc. (%)	Acc. (%)	Acc. (%)	
Anuran Calls (MFCCs)	**87.9**±1.3	55.4±1.3	†79.7±0.7	6.2±0.1	43.4±1.1	8.2%
Breast Cancer Wisconsin (Original)	**97.4**±0.7	†66.1±3.2	65.1±3.4	–	65.5±3.3	31.3%
DrivFace	**93.7**±1.9	90.1±0.7	†91.8±1.9	90.1±0.7	90.1±0.7	1.9%
Parkinsons	**96.4**±0.7	78.5±3.8	84.6±2.5	†88.7±1.5	75.4±5.8	7.7%
QSAR Biodegradation	**88.4**±0.6	84.4±3.1	†87.0±0.9	84.8±1.9	66.3±1.5	1.4%
Sensorless Drive Diagnosis	**99.6**±0.0	24.9±0.3	†87.2±0.3	80.9±0.3	9.0±0.2	12.4%
Waveform Database Generator (Ver. 1)	**86.6**±0.4	86.1±0.6	†86.3±0.2	81.5±0.9	51.6±1.6	0.3%
Waveform Database Generator (Ver. 2)	**86.7**±1.0	†86.4±0.3	85.9±0.4	82.3±0.5	56.3±1.1	0.3%

Note: Improvement shown as improvement of ConvXGB *vs* the best of SVC family - marked with a †.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

S. Thongsuwana and S. Jaiyen would like to thank the Thailand Research Fund (TRF) under grant number RTA6080013 for supporting.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.net.2020.04.008.

References

[1] Y. Guo, X. Jia, D. Paull, Effective sequential classifier training for multitemporal remote sensing image classification, IEEE Trans. Image Process. (2018), https://doi.org/10.1109/TIP.2018.2808767.
[2] Y. Wang, S. Liu, C. Chen, B. Zeng, A hierarchical approach for rain or snow removing in a single color image, IEEE Trans. Image Process. 26 (8) (2017) 3936–3950, https://doi.org/10.1109/TIP.2017.2708502.
[3] M.M. Mirończuk, J. Protasiewicz, A recent overview of the state-of-the-art elements of text classification, Expert Syst. Appl. 106 (2018) 36–54, https://doi.org/10.1016/j.eswa.2018.03.058. http://www.sciencedirect.com/science/article/pii/S095741741830215X.
[4] D. Isa, L.H. Lee, V.P. Kallimani, R. RajKumar, Text document preprocessing with the bayes formula for classification using the support vector machine, IEEE Trans. Knowl. Data Eng. 20 (9) (2008) 1264–1272, https://doi.org/10.1109/TKDE.2008.76.
[5] H. Sadreazami, A. Mohammadi, A. Asif, K.N. Plataniotis, Distributed-graph-based statistical approach for intrusion detection in cyber-physical systems, IEEE Trans. Signal Info. Process. Over Networks 4 (1) (2018) 137–147, https://doi.org/10.1109/TSIPN.2017.2749976.
[6] M.H. Ali, B.A.D.A. Mohammed, A. Ismail, M.F. Zolkipli, A new intrusion detection system based on fast learning network and particle swarm optimization, IEEE Access 6 (2018) 20255–20261, https://doi.org/10.1109/ACCESS.2018.2820092.
[7] M. Adam, E.Y. Ng, J.H. Tan, M.L. Heng, J.W. Tong, U.R. Acharya, Computer aided diagnosis of diabetic foot using infrared thermography: a review, Comput. Biol. Med. 91 (2017) 326–336, https://doi.org/10.1016/j.compbiomed.2017.10.030. http://www.sciencedirect.com/science/article/pii/S0010482517303566.
[8] H. Müller, D. Unay, Retrieval from and understanding of large-scale multi-modal medical datasets: A review, IEEE Trans. Multimed. 19 (9) (2017) 2093–2104, https://doi.org/10.1109/TMM.2017.2729400.

[9] A. Carè, F.A. Ramponi, M.C. Campi, A new classification algorithm with guaranteed sensitivity and specificity for medical applications, IEEE Control Syst. Lett. 2 (3) (2018) 393–398, https://doi.org/10.1109/LCSYS.2018.2840427.
[10] H. Zhu, X. Liu, R. Lu, H. Li, Efficient and privacy-preserving online medical prediagnosis framework using nonlinear SVM, IEEE J. Biomed. Health Info. 21 (3) (2017) 838–850, https://doi.org/10.1109/JBHI.2016.2548248.
[11] W. Hu, B. Wu, P. Wang, C. Yuan, Y. Li, S. Maybank, Context-dependent random walk graph kernels and tree pattern graph matching kernels with applications to action recognition, IEEE Trans. Image Process. 27 (10) (2018) 5060–5075, https://doi.org/10.1109/TIP.2018.2849885.
[12] A.A. Adewuyi, L.J. Hargrove, T.A. Kuiken, An analysis of intrinsic and extrinsic hand muscle EMG for improved pattern recognition control, IEEE Trans. Neural Syst. Rehabil. Eng. 24 (4) (2016) 485–494, https://doi.org/10.1109/TNSRE.2015.2424371.
[13] Y. Geng, Y. Ouyang, O.W. Samuel, S. Chen, X. Lu, C. Lin, G. Li, A robust sparse representation based pattern recognition approach for myoelectric control, IEEE Access 6 (2018) 38326–38335, https://doi.org/10.1109/ACCESS.2018.2851282.
[14] E. Tu, N. Kasabov, J. Yang, Mapping temporal variables into the neucube for improved pattern recognition, predictive modeling, and understanding of stream data, IEEE Trans. Neural Networks Learning Syst. 28 (6) (2017) 1305–1317, https://doi.org/10.1109/TNNLS.2016.2536742.
[15] L.D.W. Thomas, A. Leiponen, Big data commercialization, IEEE Eng. Manag. Rev. 44 (2) (2016) 74–90, https://doi.org/10.1109/EMR.2016.2568798.
[16] F. Liang, W. Yu, D. An, Q. Yang, X. Fu, W. Zhao, A survey on big data market: Pricing, trading and protection, IEEE Access 6 (2018) 15132–15154, https://doi.org/10.1109/ACCESS.2018.2806881.
[17] N. Chawla, D. Davis, Bringing big data to personalized healthcare: A patient-centered framework, J. Gen. Intern. Med. 28 (suppl 3) (2013) 1–7, https://doi.org/10.1007/s11606-013-2455-8.
[18] N. Kruger, P. Janssen, S. Kalkan, M. Lappe, A. Leonardis, J. Piater, A.J. Rodriguez-Sanchez, L. Wiskott, Deep hierarchies in the primate visual cortex: What can we learn for computer vision? IEEE Trans. Pattern Anal. Mach. Intell. 35 (8) (2013) 1847–1871, https://doi.org/10.1109/TPAMI.2012.272.
[19] A. Brunetti, D. Buongiorno, G.F. Trotta, V. Bevilacqua, Computer vision and deep learning techniques for pedestrian detection and tracking: A survey, Neurocomputing 300 (2018) 17–33, https://doi.org/10.1016/j.neucom.2018.01.092. http://www.sciencedirect.com/science/article/pii/S092523121830290X.
[20] J. Thevenot, M.B. López, A. Hadid, A survey on computer vision for assistive medical diagnosis from faces, IEEE J. Biomed. Health Info. 22 (5) (2018) 1497–1511, https://doi.org/10.1109/JBHI.2017.2754861.
[21] T. Young, D. Hazarika, S. Poria, E. Cambria, Recent Trends in Deep Learning Based Natural Language Processing, 1708, 02709. ArXiv e-printsarXiv.
[22] J. Choo, S. Liu, Visual Analytics for Explainable Deep Learning, 1804, 02527. ArXiv e-printsarXiv.
[23] J. Lemley, S. Bazrafkan, P. Corcoran, Deep learning for consumer devices and services: Pushing the limits for machine learning, artificial intelligence, and computer vision, IEEE Consumer Electron. Mag. 6 (2) (2017) 48–56, https://doi.org/10.1109/MCE.2016.2640698.
[24] Y. Bengio, A. Courville, P. Vincent, Representation learning: A review and new perspectives, IEEE Trans. Pattern Anal. Mach. Intell. 35 (8) (2013) 1798–1828, https://doi.org/10.1109/TPAMI.2013.50.
[25] H.A. Gohel, H. Upadhyay, L. Lagos, K. Cooper, A. Sanzetenea, Predictive maintenance architecture development for nuclear infrastructure using machine learning, Nucl. Eng. Technol. (2019), https://doi.org/10.1016/j.net.2019.12.029. http://www.sciencedirect.com/science/article/pii/S1738573319306783.
[26] Y.D. Koo, Y.J. An, C.-H. Kim, M.G. Na, Nuclear reactor vessel water level prediction during severe accidents using deep neural networks, Nucl. Eng. Technol. 51 (3) (2019) 723–730, https://doi.org/10.1016/j.net.2018.12.019. http://www.sciencedirect.com/science/article/pii/S1738573318307861.
[27] K. Malik, M. Żbikowski, A. Teodorczyk, Detonation cell size model based on deep neural network for hydrogen, methane and propane mixtures with air and oxygen, Nucl. Eng. Technol. 51 (2) (2019) 424–431, https://doi.org/10.1016/j.net.2018.11.004. http://www.sciencedirect.com/science/article/pii/S1738573318305953.
[28] J. Park, S.-J. Han, N. Munir, Y.-T. Yeom, S.-J. Song, H.-J. Kim, S.-G. Kwon, MRPC eddy current flaw classification in tubes using deep neural networks, Nucl. Eng. Technol. 51 (7) (2019) 1784–1790, https://doi.org/10.1016/j.net.2019.05.011. http://www.sciencedirect.com/science/article/pii/S1738573319302414.
[29] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324, https://doi.org/10.1109/5.726791.
[30] T. Chen, C. Guestrin, Xgboost: A Scalable Tree Boosting System, 2016. ArXiv e-printsarXiv:1603.02754.
[31] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Wadsworth and Brooks, Monterey, CA, 1984.
[32] G.E. Hinton, Connectionist learning procedures, Artif. Intell. 40 (1) (1989) 185–234, https://doi.org/10.1016/0004-3702(89)90049-0. http://www.sciencedirect.com/science/article/pii/0004370289900490.
[33] C.-C. Chang, C.-J. Lin, LIBSVM: A library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (2011) 27, 1–27:27, software available at: http://www.csie.ntu.edu.tw/~cjlin/libsvm.
[34] P. Murugan, Feed Forward and Backward Run in Deep Convolution Neural Network, 2019. ArXiv e-printsarXiv:1711.03278.
[35] D. Stutz, Understanding convolutional neural networks, Seminar report, Fakultät für Mathematik, Informatik und Naturwissenschaften Lehr-und Forschungsgebiet Informatik VIII Computer Vision (2014).
[36] V. Papyan, Y. Romano, J. Sulam, M. Elad, Theoretical foundations of deep learning via sparse representations: A multilayer sparse model and its connection to convolutional neural networks, IEEE Signal Process. Mag. 35 (4) (2018) 72–89, https://doi.org/10.1109/MSP.2018.2820224.
[37] A. Asuncion, D. Newman, Uci Machine Learning Repository, 2007. http://www.ics.uci.edu/&dollar;&bsol;sim&dollar;mlearn/{MLR}epository.html.
[38] D. Dheeru, E. Karra Taniskidou, Uci Machine Learning Repository, 2017. http://archive.ics.uci.edu/ml.
[39] W.H. Wolberg, O.L. Mangasarian, Multisurface method of pattern separation for medical diagnosis applied to breast cytology, Proceed. Natl. Acad. Sci. U. S. A. 87 (23) (1991) 9193–9194, https://doi.org/10.1073/pnas.87.23.9193.
[40] K. Diaz-Chito, A. Hernández-Sabaté, A.M. López, A reduced feature set for driver head pose estimation, Appl. Soft Comput. 45 (C) (2016) 98–107, https://doi.org/10.1016/j.asoc.2016.04.027.
[41] M. A Little, P. Mcsharry, S. Roberts, D.A.E. Costello, I. M Moroz, Exploiting nonlinear recurrence and fractal scaling properties for voice disorder detection, Biomed. Eng. Online 6 (2007) 23, https://doi.org/10.1186/1475-925X-6-23.
[42] M. Kamel, T. Ringsted, D. Ballabio, R. Todeschini, V. Consonni, Quantitative structure-activity relationship models for ready biodegradability of chemicals, J. Chem. Inf. Model. 53 (2013) 867–878, https://doi.org/10.1021/ci4000213.
[43] 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, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015 software available from: tensorflow.org, http://tensorflow.org/.
[44] 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, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.