A MODIFIED CONVOLUTIONAL NEURAL NETWORK USING PRE-TRAINED WEIGHTS AND EXTREME GRADIENT BOOSTING

SETTHANUN THONGSUWAN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE FACULTY OF SCIENCE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2019
KMITL-2019-SC-D-002-078

A MODIFIED CONVOLUTIONAL NEURAL NETWORK USING PRE-TRAINED WEIGHTS AND EXTREME GRADIENT BOOSTING

SETTHANUN THONGSUWAN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE FACULTY OF SCIENCE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2019

KMITL-2019-SC-D-002-078

Thesis Title	A Modified Convolutional Neural Network Using Pre-Trained Weights and Extreme Gradient Boosting
Student Name	Acting Sub Lt. Setthanun Thongsuwan
Student ID	56605009
Degree	Doctor of Philosophy (Computer Science)
Department	Computer Science
Year	2019
Thesis Advisor	Assistant Professor Dr. Saichon Jaiyen

Abstract

In this thesis describes a modified Convolutional Neural Network (CNN) for classification problems based on Pre-Trained weights and Extreme Gradient Boosting (XGBoost). As well as image data, this research works also supports general classification problems, with a data preprocessing module. This proposed model consists of several stacked convolutional layers to learn the features of the input and is able to learn features automatically, followed by XGBoost as the last layer to increase predictive efficiency for predicting the class labels. The model is simplified by reducing the number of parameters under appropriate conditions, since it is not necessary to re-adjust the weight values in a backpropagation cycle. Further, the model 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. Experiments on data sets, including images and general data sets, showed that the model proposed in this thesis, handled the classification problems well and that provided better accuracy than other models *e.g.* CNN, XGBoost, Logistic Regression (LR), Extra Trees Classifier (ETC), Gradient Boosting Classifier (GBC), Random Forest Classifier (RFC), Gaussian Naive Bayes (GNB), Decision Tree Classification (DTC), including Multilayer Perceptron (MLP), and Support Vector Classification (SVC) families.

Keywords : Convolutional Neural Network, Classification Algorithms, Deep Learning, Extreme Gradient Boosting, Feature Learning, Machine Learning, Pattern Recognition, Pre-trained Weights

Acknowledgements

The achievement of my study could not complete by my effort only, but also it can be finished with the help from many people and supporting from many facilities. First of all, I really appreciate to thank my advisor, Asst. Prof. Dr. Saichon Jaiyen, who give me the concept idea of research problems, help and support to develop this research title to be completed.

Secondly, I would really like to thank Prof. John Morris of KMITL Research and Innovation Services (KRIS). He helped me a lot to revise and review the manuscripts of my paper to be the best draft before submitting to the journal.

Thirdly, I also thank King Mongkut's Institute of Technology Ladkrabang (KMITL) for supporting the scholarship in my Ph.D. study.

Next, my research was supported by the Thailand Research Fund (TRF) under grant number RTA6080013.

Finally, I would like to thank my families. Especially, my mother, she gives love and stay with me all times of hardship, have an understanding, and always give me all supporting. I' m sure that I will get it forever.

Setthanun Thongsuwan

Table of Contents

Page

Abstract i
Acknowledgements ii
Table of Contents iii
List of Tables v
List of Figures vii
Chapter 1. Introduction 1
1.1 Background and Research Motivation 1
1.2 Objectives of the research 2
1.3 Scope of the research 2
1.4 Expected benefits 2
1.5 Overview of the thesis 3
Chapter 2. Theory and Related work 5
2.1 Convolutional Neural Network (CNN) 5
2.1.1 CNN Architecture 5
2.1.2 Convolution layer 6
2.1.3 Pooling layer 7
2.1.4 Fully Conected layer 7
2.2 Extreme Gradient Boosting (XGBoost) 7
Chapter 3. Research methodology 10
3.1 Convolutional Extreme Gradient Boosting (ConvXGB) 10
3.1.1 ConvXGB Architecture 10
3.1.2 Principles and Parameters 13
3.1.3 ConvXGB Learning Algorithm 15
3.1.4 Data Preprocessing 17
3.2 Pre-Trained weights and Extreme Gradient Boosting (PT-XGB) 18
3.2.1 PT-XGB Architecture 19
3.2.2 Pre-Trained weights 22
3.2.3 PT-XGB Training Algorithm 23
Chapter 4. Experimental and Numerical results 25
4.1 Experimental and results for ConvXGB 25
4.1.1 Data sets used to evaluate ConvXGB 25
4.1.2 Experimental Setup 25
4.1.3 Experimental Results 26

4.2 Experimental and results for PT-XGB 33
4.2.1 Data sets used to evaluate PT-XGB 33
4.2.2 Experimental Setup 34
4.2.3 Experimental Results 36
Chapter 5. Conclusions 46
References 48
Appendix 55
Appendix A 56
Appendix B 71
Author Biography 94

List of Tables

Table **Page**

4.1 Data sets used for ConvXGB 26
4.2 Parameters in each layer of the ConvXGB model 26
4.3 The properties of the models used in the performance comparison 27
4.4 Parameters for descriptions the time complexity 28
4.5 MNIST Data Set 28
4.6 Anuran Calls (MFCCs) Data Set 28
4.7 Breast Cancer Wisconsin (Original) Data Set 29
4.8 DrivFace Data Set 29
4.9 Parkinsons Data Set 29
4.10 QSAR Biodegradation Data Set 29
4.11 Sensorless Drive Diagnosis Data Set 30
4.12 Waveform Database Generator (ver. 2) Data Set 30
4.13 Our model compared with CNN, XGBoost and DTC 31
4.14 Our model compared with the MLP family 31
4.15 Our model compared with the SVC family 31
4.16 Data sets used to evaluate PT-XGB 33
4.17 Parameters in each layer of the PT-XGB model 34
4.18 Properties of models used in performance comparison 35
4.19 Run time of our model compared with CNN and ConvXGB 36
4.20 Balance Scale Data Set Data Set 37
4.21 Diabetic Retinopathy Debrecen 37
4.22 Image Segmentation Data Set 38
4.23 ISOLET Data Set 38
4.24 Letter Recognition Data Set 39
4.25 Optical Recognition of Handwritten Digits Data Set 39
4.26 Pen Based Recognition of Handwritten Digitsn Data Set 40
4.27 Seismic Bumps Data Set 40
4.28 Smartphone Based Recognition of Human Activities and Postural Transitions Data Set 41
4.29 Wine Data Set 41
4.30 Our model compared with ConvXGB, CNN and XGBoost 42
4.31 Our model compared with LR, ETC and GBC 42
4.32 Our model compared with RFC, GNB and DTC 43
4.33 Our model compared with the MLP family 43

4.34 Our model compared with the SVC family..... 44

List of Figures

Figure **Page**

2.1 Traditional Convolutional Neural Network (CNN) Architecture. 5

3.1 Architecture of the ConvXGB model. 11
3.2 The process of operation in part of predict the class labels consists of reshape layer, class prediction layer and output layer. 12
3.3 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 18
3.4 Overview of PT-XGB: (a) Pre-Trained weights, (b) Training and (c) Predicting 20
3.5 Architecture of PT-XGB 22

4.1 ConvXGB *vs* all models for the MNIST data set [1] 32
4.2 ConvXGB *vs* all models for the Anuran Calls (MFCCs) data set [2] 32
4.3 ConvXGB*vs* all models for the Breast Cancer Wisconsin (Original) data set [3]* 32
4.4 ConvXGB *vs* all models for the DrivFace data set [4] . 32
4.5 ConvXGB *vs* all models for the Parkinsons data set [5] 32
4.6 ConvXGB *vs* all models for the QSAR Biodegradation data set [6] 32
4.7 ConvXGB *vs* all models for the Sensorless Drive Diagnosis data set [2] 33
4.8 ConvXGB *vs* all models for the Waveform Database Generator (Ver. 2) data set [2] 33
4.9 PT-XGB *vs* all models for the Balance Scale data set [2] 44
4.10 PT-XGB *vs* all models for the Diabetic Retinopathy Debrecen data set [7] 44
4.11 PT-XGB *vs* all models for the Image Segmentation data set [2] 44
4.12 PT-XGB *vs* all models for the ISOLET data set [2] 44
4.13 PT-XGB *vs* all models for the Letter Recognition data set [2] 45
4.14 PT-XGB *vs* all models for the Optical Recognition of Handwritten Digits data set [2] 45
4.15 ConvXGB *vs* all models for the Pen-Based Recognition of Handwritten Digits data set [2] 45
4.16 ConvXGB *vs* all models for the Seismic Bumps data set [8] 45
4.17 ConvXGB *vs* all models for the Smartphone Based Recognition of Human Activities and Postural Transitions data set [9] 45
4.18 ConvXGB *vs* all models for the Wine data set [2] 45

Chapter 1

Introduction

1.1 Background and Research Motivation

Artificial intelligence (AI) has made great progress for the world. It unlocks the limitations of human thinking in the past and along with create advances in knowledge to develop new ideas and innovations that are constantly occurring. AI is a concept for building a machine capable human equivalent for solving common problems in theory and practice. For example, systems that have human-like thinking, and decision-making based on formal reasoning, systems that have actions that are equivalent to human beings with particular expertise in that particular knowledge or expert system [10, 11] *etc.*Machine Learning (ML) is the heart of AI research that is widely used today, which has been used to solve a variety of research areas, such as medical and clinical applications [12], agriculture [13], financial [14, 15, 16], and most of them are used in applications which involve data analysis, classification, and prediction.

Learning models are key to solving some ML problems. Therefore, in every problem must focus on learning models. Classification is a major topic for ML and appropriate learning criteria are essential for predicting a class labels - this is called supervised learning. Classification problems may be found in image processing [17, 18, 19, 20], document and text classification [21, 22, 23, 24], intrusion detection [25, 26, 27], medical diagnosis [28, 29, 30, 31], and pattern recognition [32, 33, 34, 35]: 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 [36, 37, 38] *etc.*Today, humans are affected by this data and significant knowledge and understanding may be found in it, if analyzed 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 [39, 40, 41] and classification problems [42, 43, 44], for example, Mironczuk and Protasiewicz have surveyed its application to text classification [23]. In solving machine learning problems, feature learning has become a driving force for success [45].

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 and deep learning, still use only a single model for training.

1.2 Objectives of the research

1) To apply pre-trained weights and extreme gradient boosting techniques for a modified convolutional neural network.
2) To develop an effective model for handling classification problems, which is not focused only on image processing, but also other general classification problems.
3) To compare other machine learning models, *i.e.* , Convolutional Neural Network (CNN), Extreme Gradient Boosting (XGBoost), Logistic Regression (LR), Extra Trees Classifier (ETC), Gradient Boosting Classifier (GBC), Random Forest Classifier (RFC), Gaussian Naive Bayes (GNB), Decision Tree Classification (DTC), Multilayer Perceptron (MLP), and Support Vector Classification (SVC).

1.3 Scope of the research

The scope of this thesis is focused on the study and modified model based on machine learning techniques.

1) To study the effective models for application to classification problems
2) To expand the scope of the model for classification problems, not limited to image, but including general information.
3) To study and modify techniques for automatic features learning and predicting class labels effectively

1.4 Expected benefits

1) Describe a new deep learning model – proposed for processing both image and general classification problems, it can be used to show its general applicability, with a data pre-processing module and the proposed method.
2) The model can reduce the process of weight adjusting in the training phase, it helps to get the weights that were the optimal training for the CNN model.
3) The pre-trained weights make it more effective than initiating a custom or randomly.

1.5 Overview of the thesis

The structure in the thesis consists of five chapters as follows:

Chapter 1: Introduction

Chapter 2: Theory and Related work

Chapter 3: Research methodology

Chapter 4: Experimental and Numerical results

Chapter 5: Conclusions

The details of each chapter in the thesis are described briefly as follow.

Chapter 1 present the background and research motivation of this thesis, including the literature review about state-of-the-art for artificial intelligence, machines learning challenge, and application in various fields *etc.*In addition, providing the objectives, scopes, benefits, and overview of the thesis will be displayed completely in this chapter.

Chapter 2 gives the fundamental concepts, definitions, and remarks that are necessary for both prototype models, convolutional neural network (CNN) and extreme gradient boosting (XGBoost). Including pre-trained weights techniques will be used in our model in the next chapter.

Chapter 3 describes the detail of the methodology used in the research. The first section is presented with the first model – Convolutional Extreme Gradient Boosting (ConvXGB), techniques behind the development of this model, and data preprocessing method. The second section, ConvXGB were developed at a higher level with pre-trained weights technique – Pre-Train Weights Extreme Gradient Boosting (PT-XGB). Furthermore, the learning algorithms of both models are also included in this explanation.

Chapter 4 presents the experimental results of the research, This experimental is divided into two part for both our model, ConvXGB and PT-XGB. We give numerical examples to support our methodology in the previous chapter. Including showing the data set used in the evaluation of our model. Our model was compared with other models *i.e.* Convolutional Neural Network (CNN), Extreme Gradient Boosting (XGBoost), Logistic Regression (LR), Extra Trees Classifier (ETC), Gradient Boosting Classifier (GBC), Random Forest Classifier (RFC), Gaussian Naive Bayes (GNB), Decision Tree Classification (DTC), Multilayer Perceptron (MLP), and Support Vector Classification (SVC).

Chapter 5 describes the conclusion of the thesis and the suggestions for further research, within this chapter contains: discussion, recommendations, limitations, and suggestions for further research.

In the end of this thesis, the appendices contain the articles published in the international journals. The first article is ”A Deep Single-Pass Learning for Recognition of

Handwritten Digits". For the second article is "A Deep One-Pass Learning based on Pre-Training Weights for Smartphone-Based Recognition of Human Activities and Postural Transitions". Also, the biography of the author is placed on the last page of the thesis.

Chapter 2

Theory and Related work

In this chapter, the theory and related work will be described, which consists of the fundamental concepts, definitions, and remarks that are necessary for both prototypes in the modeling of the proposed research: Convolutional Neural Network (CNN) [1], which is a class of deep learning that has been impressed by the efficiency, and accuracy in many research areas. The structure consists of many different layers *i.e.* convolution, pooling, and fully connected. The details will be shown in Section 2.1, and Extreme Gradient Boosting (XGBoost) [46] is a model developed from Gradient Boosting (GB), which has a structure consisting of many decision trees (or decision tree ensemble) to vote for the predicting class labels, more details will be explained in Section 2.2.

2.1 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN), first described by Lecun *et al.* [1], are described in detail by Goodfellow et al [47] and Stutz [48]. A short summary follows, but readers familiar with CNN may skip this section.

2.1.1 CNN Architecture

Traditional Convolutional Neural Network (CNN) architecture, generally consists of $\mathcal{L}$ convolutional layers, indexed by k, alternating with multiple pooling layers, which are responsible for learning the features of the training data. The last layer is usually a Fully Connected (FC) one see Figure 2.1.

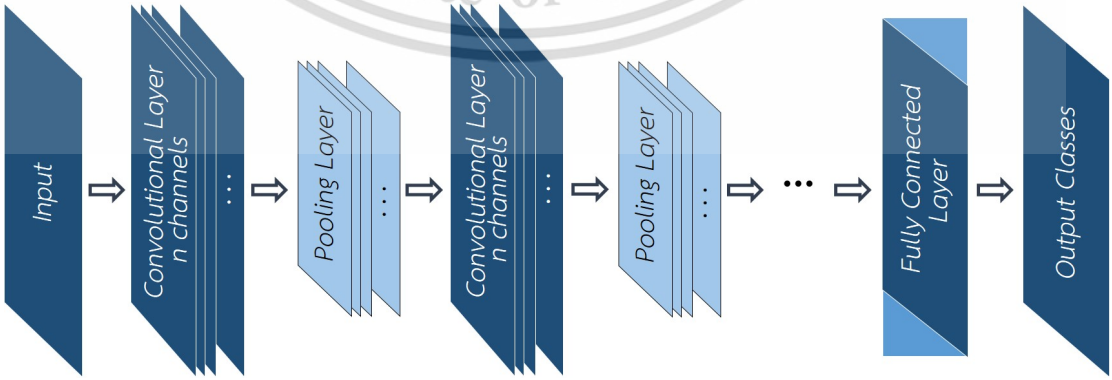


Figure 2.1 Traditional Convolutional Neural Network (CNN) Architecture.

2.1.2 Convolution layer

In each layer, a convolution operation, $\mathcal{K}^k$, is applied to the image which is fed the k^{th} layer. We assume a grayscale 'image', $\mathcal{Y}_j^k$, as the input to the k^{th} layer, (where $\mathcal{Y}_j^1 = \mathcal{I}_j$). The discrete convolution of $\mathcal{Y}_{j,m,n}^k$ and the kernel $\mathcal{K}^k$, with size $h \times d$ and indices $u \in [-h, h]$ and $v \in [-d, d]$, is computed as:

$$\mathcal{Y}_{j,m,n}^{k+1} = (\mathcal{Y}_j^k \otimes \mathcal{K})_{j,m,n} = \sum_{u=-h}^{h} \sum_{v=-d}^{d} \mathcal{K}_{u,v}^k \mathcal{Y}_{j,m+u,n+v}^k \tag{2.1}$$

where m, n ranges over the indices in the 'image' in the $k + 1^{th}$ layer, which is not always the same as the range in layer, k. For understanding, the calculation in each layer is forwarded to the next layer: the input to the first ($k = 1$) layer is the set of original images, but after convolution, the inputs to the next, and subsequent layers, are the F^k 'features' in layer k, $\mathcal{Y}_j^k | j \in [1..F^k]$, because they are transformations (the convolutions) of the previous inputs. Note that the number of feature maps in each layer, F^k, may vary, set by the user for each application. In convolution layer, $k \in [1..L]$:

Then, the j^{th} feature map for layer $k+1$, $\mathcal{Y}_j^{k+1}$, after a set of biases, $\mathcal{B}_j^k$, for each feature, j, in each layer, k, are added and the activation function, φ is applied:

$$\mathcal{Y}_j^{k+1} = \varphi(\mathcal{B}_j^k + \sum_{m=1}^{F^k} (\mathcal{K}_{m,j}^k \otimes \mathcal{Y}_m^k)) \tag{2.2}$$

Here a Rectified Linear Unit (ReLU) activation function was used for φ,

$$\varphi = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{otherwise.} \end{cases} \tag{2.3}$$

Thus, the elements of the output of layer, $k+1$, for the j^{th} feature map, $\mathcal{Y}_j^{k+1}$, at position (r, s) is:

$$(\mathcal{Y}_j^{k+1})_{r,s} = \varphi((\mathcal{B}_j^k)_{r,s} + \sum_{m=1}^{F^k} (\mathcal{K}_{m,j}^k \otimes \mathcal{Y}_m^k)_{r,s}) \tag{2.4}$$

2.1.3 Pooling layer

The pooling layer helps to reduce the size of the output (*i.e.* it may down-sample the input to that layer). Other function options could be used *e.g.* maximum, average, *etc.*. We followed common practice to use the maximum value on a local rectangular region (neighborhood). Let us assume $p \times p$ is size of the pooling window and sp is stride of the pooling, then $P(..)$ is a pooling function which acts on $\mathcal{Y}_j^{k+1}$, and pooling window of dimension is $((H-h)/sp+1) \times ((D-d)/sp+1)$. So that the output of a max-pooling function is:

$$P(\mathcal{Y}_j^{k+1})_{m,n} = max(\mathcal{Y}_j^{k+1})_{m,n} \tag{2.5}$$

2.1.4 Fully Conected layer

The FC layer, receives data from the previous convolutional and pooling layer. This layer is a classifier layer, the weights received in this layer in each iteration is fed back to adjust the weights in all previous layers. Generally, $softmax$ is the transformation function used in the feed back:

$$\mathcal{Y} = softmax(\mathbb{I} \otimes \mathbb{W} + \mathbb{B}) \tag{2.6}$$

Finally, we obtain as output, the predictions $\mathcal{Y}$, where $\mathbb{I}$. is the set of original images, and set, $\mathbb{W}$ of all weights, and set of biases $\mathbb{B}$.

2.2 Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting (XGBoost) is a machine learning model for classification and regression problems designed by Chen and Guestrin [46] and shown first in KDD Cup 2015. It is effective for a machine learning and data mining challenges and has been used extensively by data scientists.

A brief summary of Chen and Guestrin's work follows, but readers familiar with their work may skip to next Chapter. XGBoost is a highly scalable end-to-end tree boosting system. Its architecture consists of a tree, which is an ensemble of K classification and regression trees (CARTs). If x_i is the vector training set and y_i is the corresponding class labels of x_i . The output prediction, $\hat{y}_i$ is the sum of the prediction scores of K

trees:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i) \tag{2.7}$$

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

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{2.8}$$

where $\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} {w_j}^2$, with constants, γ and λ, which control the regularization degree, T is the set of leaves in the tree with the weight of each leaf denoted w. In addition, we can improve the efficiency in Equation (2.8) by expanding the loss function with a first or second order Taylor expansion. Therefore, at step t, we can calculate:

$$\begin{aligned}
\tilde{\mathcal{L}}^{(t)} &\simeq \sum_{i=1}^{n} [g_i f_i(x_i) + \frac{1}{2} h_i {f_i}^2(x_i)] + \Omega(f_t) \\
&= \sum_{i=1}^{n} [g_i f_i(x_i) + \frac{1}{2} h_i {f_i}^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} {w_j}^2 \\
&= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) {w_j}^2] + \gamma T
\end{aligned} \tag{2.9}$$

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

$$g_i = \frac{\partial l({\hat{y}_i}^{(t-1)}, y_i)}{\partial \hat{y}_i^{(t-1)}} \tag{2.10}$$

$$h_i = \frac{\partial^2 l({\hat{y}_i}^{(t-1)}, y_i)}{\partial (\hat{y}_i^{(t-1)})^2} \tag{2.11}$$

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

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \tag{2.12}$$

when calculating the weight in Equation (2.12), the final equation is the quality of a tree structure, q, computed as:

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T, \tag{2.13}$$

The XGBoost model calculates scores in each node in the tree structure for split decisions. For effective predictive, we realize the loss after the split and want to reduce it. If $I = I_L \cup I_R$, where I_L and I_R are the left and right nodes after the split, we compute:

$$\mathcal{L}_{split} = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} + \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma. \tag{2.14}$$

Chapter 3

Research methodology

3.1 Convolutional Extreme Gradient Boosting (ConvXGB)

This chapter is important for research because it has compiled details of the research process for the proposed model, Convolutional Extreme Gradient Boosting (ConvXGB). An overview of ConvXGB architecture shows in Section 3.1.1, the model is separated to describe each layer according to the steps of the process of operation, that includes the theoretical principles and parameters Section 3.1.2 that are used in each layer with different. In order to understand and simpler to study, we have described the ConvXGB by demonstrating the learning algorithm see Section 3.1.3. Furthermore, to expand the differences in the differences for data types, we have presented our data preprocessing see Section 3.1.4 to support our model performance to cover all data formats, which does not limit to image data.

3.1.1 ConvXGB Architecture

The architectural design of the proposed model, ConvXGB consist of six layers: 1) input layer, 2) data preprocessing layer, 3) convolutional layers, 4) reshape layer, 5) class prediction layer and 6) output layer see Figure 3.1. Each layer has different capabilities and responsibilities: these layers are the keys to the success of the model. ConvXGB architecture can be divided into two parts: one for feature learning and the other to predict the class labels, the types of layers are described in each part as follows.

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. Predeiction accuracy depends on effective feature learning. Details of each layer follow.

1) 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.

2) 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

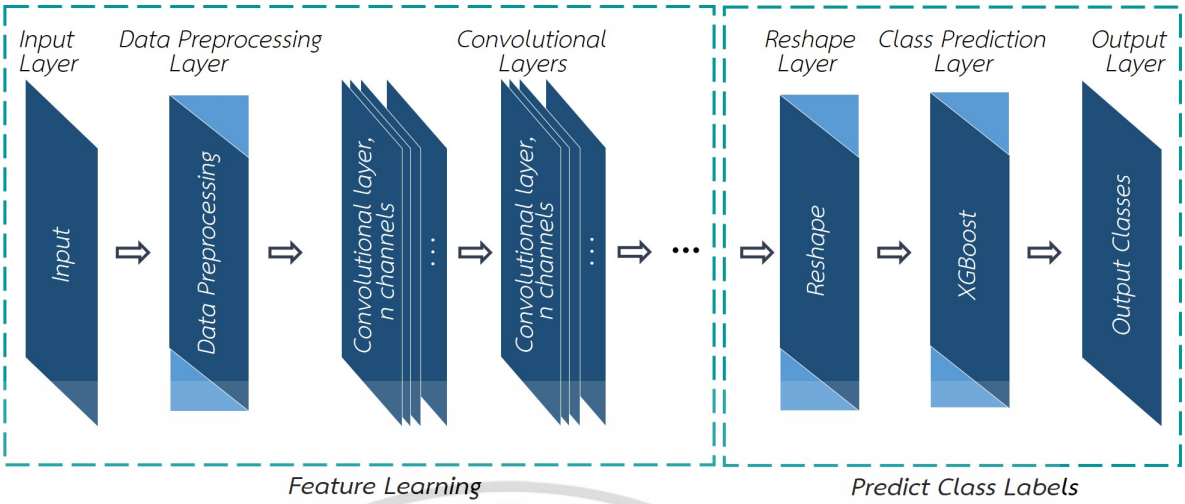


Figure 3.1 Architecture of the ConvXGB model.

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, the required format is further described in Section 3.1.4.

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 dinmensions, $(\sqrt{N}, \sqrt{N}, z^{(l)})$, see left of Figure 3.2, 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 $x = [x_1, x_2, ..., x_9]$ 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 2.4 in Section 2.1:

$$\mathcal{Y}_j^{k+1} = \varphi(\mathcal{B}_j^k + \sum_{m=1}^{F^k} (\mathcal{K}_{m,j}^k \otimes \mathcal{Y}_m^k))$$

The computational complexity is set out in Section 3.2.3.

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,

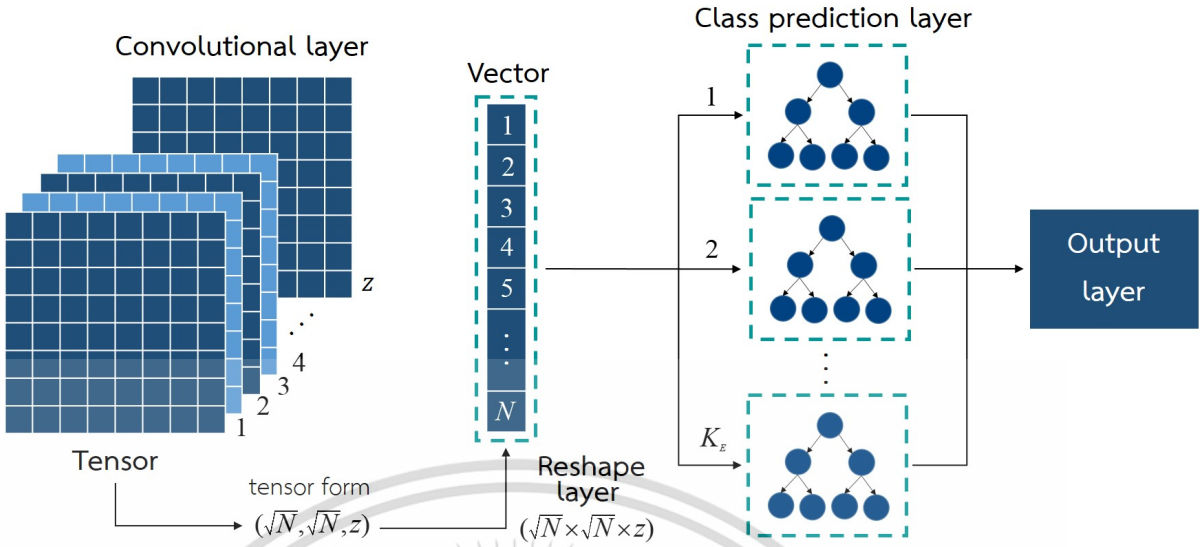


Figure 3.2 The process of operation in part of predict the class labels consists of reshape layer, class prediction layer and output layer.

the input must be in the form of a vector, so a housekeeping operation transforms the tensor to a vector in the reshape layer.

4) 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 - see middle of Figure 3.2.

5) 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 the performance. The quality of a tree structure can be scored as Equation 2.13:

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

where $I_j = \{i|q(x_i) = j\}$ denotes the instance set of leaf t and $g_i = \frac{\partial l(\hat{y}_i{}^{(t-1)}, y_i)}{\partial \hat{y}_i^{(t-1)}}$, $h_i = \frac{\partial^2 l(\hat{y}_i{}^{(t-1)}, y_i)}{\partial \hat{y}_i^{(t-1)}}$ 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 2.14. 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:

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

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).

6) 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.

3.1.2 Principles and Parameters

In this section, we will explain the parameters and the key equations of ConvXGB model. ConvXGB has two main parts, the ability to learn the features of the training set and efficiently predicting the class labels of the test set. The first part, before entering the prediction process, the training set will be learnt by deep feature learning, which consists of several convolutional layers, $\mathcal{Y}$, and the pooling layer, P, for reducing feature size. Generally, feature learning has a hierarchical structure, see a diagram of our model in Figure 3.1 showing the structre. The output of the convolution operation, $\mathcal{Y}^k$, in each layer k, is computed from the output of layer $k-1$. Therefore, the feature learning of the training data set follows this chain:

$$P(\mathcal{Y}_j^1) \to P(\mathcal{Y}_j^2) \to \cdots \to P(\mathcal{Y}_j^k) \to P(\mathcal{Y}_j^{k+1}) \to \mathcal{Z} \tag{3.1}$$

when $\mathcal{Z}$ is tensor output of the feature learning process. In each layer, the output of the previous layer is set to the input of the current layer. In the case where $L = k+1$, so the input, of $\mathcal{Y}_j^{k+1}$, in the $k+1$ layer for the j^{th} feature map, is derived from the output of the previous layer, $\mathcal{Y}_j^k$.

An additive bias is applied to each input so the input, $\mathcal{Y}_i^k$ of the k^{th} layer for the j^{th} feature map, is derived from the output of the previous layer, $\mathcal{Y}_j^{k-1}$, as set out previously in Equation 2.4 (repeated here).

$$\mathcal{Y}_j^{k+1} = \varphi(\mathcal{B}_j^k + \sum_{m=1}^{F^k} (\mathcal{K}_{m,j}^k \otimes \mathcal{Y}_m^k))$$

Remark 3.1. We used the capabilities of the convolutional layers from CNNs, the input must be in a tensor format. Therefore, the training set will be converted to the standard input format of convolutional operation as the following Definition 3.1.

Definition 3.1. 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:

$$\mathcal{T} = [\overline{a}_{j_1 j_2 \ldots j_M}^{i_1 i_2 \ldots i_M}]_{N'N'} \tag{3.2}$$

The output of convolution operation, $\mathcal{Y}$, caused from the tensor, $\mathcal{T}$, applied to the $h \times d$ kernel, $\mathcal{K}$. The kernel, $\mathcal{K}$, is slid through tensor, $\mathcal{T}$, by a stride, sk, and zero padding value.

In the second part, we use the decision rules in XGBoost to predict from the training set learned from the first part. We add the term of features learning $\mathcal{Z}$ (see in Equation 3.1) to predict classes. Therefore, the final prediction is the sum of the prediction scores for each tree E, as follows:

$$\hat{y}_i = \phi(\mathcal{Z}_i) = \sum_{k=1}^{K} f_k(\mathcal{Z}_i), \qquad f_k \in F, \tag{3.3}$$

For Equation (3.3) uses the ensemble of number of E classification and regression trees (CARTs), when $F = f(\mathcal{Z}) = w_{q(\mathcal{Y})}(q : \mathbb{R}^n \to T, w \in \mathbb{R}^T)$ is the set of all K scores for all CARTs, and f_k is the leaf score for the k^{th} tree corresponds to both the independent tree structure q and leaf weights w, where T is the number of leaves in the tree.

Remark 3.2. Since Equation (3.3) supports the training set $\mathcal{Z}$, data type is vector, but, the data type of $\mathcal{Z}$, after through the process of features learning is tensor. Therefore, it is necessary to convert the type of $\mathcal{Z}$ to vector see in Definition 3.2, which describes the standard and convert the input data.

Definition 3.2. The training set $\mathcal{Z}$ is a tensor size $[\overline{a}^{i_1 i_2 \ldots i_r}_{j_1 j_2 \ldots j_r}]_{n' n'}$, which r is the number of filters, and n' exhibit the number of rows and columns. Therefore, $\mathcal{Z}$ will be converted in the form of vector $\mathcal{V}$ of feature in $\mathbb{R}^b$ when $b = n' \times n' \times r$, and will be represented in the Equation (3.3) is:

$$\hat{y}_i = \phi(\mathcal{V}_i) = \sum_{k=1}^{K} f_k(\mathcal{V}_i), \qquad f_k \in F, \tag{3.4}$$

when $F = f(\mathcal{V}) = w_{q(\mathcal{V})}(q : \mathbb{R}^n \to T, w \in \mathbb{R}^T)$. The quality of a tree structure can be scored from Equation (2.13), we can set the number of trees, thus the size of the structure affects performance.

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

when $I_j = \{i | q(\mathcal{V}_i) = j\}$ denotes the instance set of leaf t and $g_i = \frac{\partial l(\hat{y}_i{}^{(t-1)}, y_i)}{\partial \hat{y}_i{}^{(t-1)}}$, $h_i = \frac{\partial^2 l(\hat{y}_i{}^{(t-1)}, y_i)}{\partial \hat{y}_i{}^{(t-1)}}$ are first and second order gradient statistics of the loss function, γ and λ are constants used to control the regularization degree. In addition, one of the key tasks is splitting into the best set of segments: we use the gain of the split in Equation (2.14). In each segment, we 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:

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

In our model, 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.

3.1.3 ConvXGB Learning Algorithm

Given a training set, $\mathbb{I} = \{\mathcal{I}_i, y_i | 1 \leq i \leq M\}$, consisting of M images, with labels, $y_i \in \mathbb{R}$, the label of image $\mathbf{x}_i$, when $\mathbf{x}_i \in \{\mathcal{T} \mid \mathcal{T} = [\overline{a}^{i_1 i_2 \ldots i_m}_{j_1 j_2 \ldots j_m}]_{n' n'}\}$ is an m-order tensor with n features ($\mathbb{R}^n$ or $\mathbb{R}^{n' \times n'}$, where $n' = \sqrt{n}$). The learning algorithm for the ConvXGB can be summarized as follows:

1. Initialize the training data set, $X = \{(x_j, y_j) | 1 \leq j \leq M\}$.
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, $N' \times N'$.
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 filter sizes, $K^{(l)}$, and
   (d) filter strides, $S_k^{(l)}$
5. For each layer, k, in $1..L$:
   calculate the convolutions to (left side of Figure 3.2) generate the $\mathcal{Y}_j^{k+1}$ for layer, $k+1$, and the j^{th} feature map:

$$\mathcal{Z} = \mathcal{Y}_j^{k+1} = \varphi(\mathcal{B}_j^k + \sum_{m=1}^{F^k} (\mathcal{K}_{m,j}^k \otimes \mathcal{Y}_m^k)),$$

6. Reshape $\mathcal{Z}$ to a vector of length $(n' \times n' \times r^{k+1})$ - $\mathcal{V}$ see in Definition 3.2.
7. Initialize parameters for the prediction step:
   (a) total number of trees, K
   (b) regularization parameters, γ and λ,
   (c) column subsampling parameter,
   (d) maximum tree depth, t and
   (e) learning rate
8. Determine the output class labels:

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

   where $F = f(\mathcal{V}) = w_{q(\mathcal{V})}(q : \mathbb{R}^n \to T, w \in \mathbb{R}^T)$
9. Calculate the optimal leaf weight for the best tree structure

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

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

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

    where T is the number of leaves in the tree

11. Calculate the best splitting points

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

12. Terminate

Our ConvXGB algorithm has time complexity:

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

where L is the number of layers, c 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, which reduces to $\mathcal{O}(Lc^2mnpq)$, because it dominates $r(Kt + \log B)$.

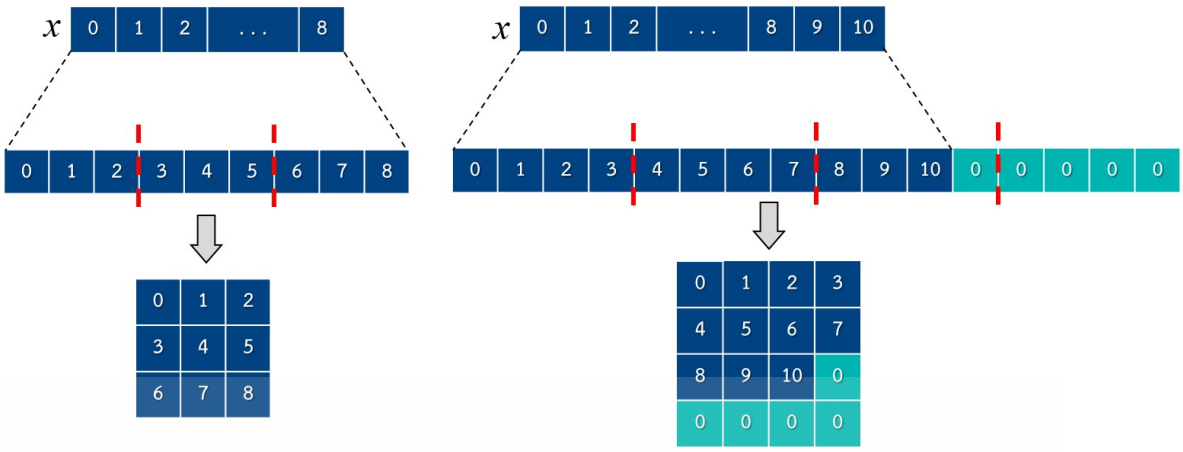
3.1.4 Data Preprocessing

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 $\mathcal{Q}$ - Definition 3.4.

Definition 3.3. Input: Let $\mathbb{I} = \{x_i, y_i | 1 \leq i \leq M\}$, be the training set of M vector, each vector x_i consists of N features, $x_i = \{p_j | 1 \leq j \leq N\}$, 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:

$$p_j \;\Rightarrow\; [p_0, p_1, \ldots, p_N]^T \Rightarrow \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1N'} \\ p_{21} & p_{22} & \cdots & p_{2N'} \\ \vdots & \vdots & \ddots & \vdots \\ p_{N'1} & p_{N'2} & \cdots & p_{N'N'} \end{bmatrix} \tag{3.5}$$

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, and show more examples in Figure 3.3 (a).



(a) *(b)*

Figure 3.3 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

$$p_j \Rightarrow [0,1,2,3,4,5,6,7,8]^T \Rightarrow \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$

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

Example 3.4. 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, and show more examples in Figure 3.3 (b).

$$\mathcal{Q}(p_j) \Rightarrow [0,1,2,3,4,5,6,7,8,9,10,0,0,0,0,0]^T$$
$$\Rightarrow \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

3.2 Pre-Trained weights and Extreme Gradient Boosting (PT-XGB)

Pre-Trained weights and Extreme Gradient Boosting (PT-XGB) extends and improves our previous work on Convolutional Extreme Gradient Boosting (ConvXGB), which has the same core structure, *i.e.* CNN combined with XGBoost. However, PT-XGB 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. We describe the details of our new model, PT-XGB, which combines elements of CNN and XGBoost, but emphasizes the key differences in PT-XGB. The primary difference is the Pre-Trained weights module, which is the first step in the overall structure, followed by the CNN and XGBoost modules. Each PT-XGB module is described in order in the following sections.

The main contributions of our model are:

- PT-XGB 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 predicts class labels by XGBoost. It's different from the traditional CNN since it doesn't have to adjust the weights and FC layer as replaced by XGBoost.
- Our model was one time only for feature learning data because we apply optimal weights from pre-trained. It does not need to rely backpropagation algorithm on back to re-adjustment of the weights. Therefore, it help to reduce the burden of the looping process for fined tuning weight in each a convolutional layer.
- We reduced the process of weight adjusting, it helped to get the weights that were the optimal training for CNN model. The time complexity for fined tuning weights is $\mathcal{O}(Ld^2mnpq) + \mathcal{O}(MNhce)$ see in Table 4.3. Pre-trained weights make it more effective than initiating a custom or randomly.
- Our model do not focus only on image processing. But also other general classification problems by our method, data preprocessing.

3.2.1 PT-XGB Architecture

The overall architecture of our model is divided into three sections according to the responsibility - arranged in a pipeline, see Figure 3.4 - Pre-Trained weights, Training, and Predicting. An overview of the function of module is described in the following list and then in detailed in the following sections and in Figure 3.5.

Pre-Trained weights: Pre-Trained Weights Figure 3.4 (a) is preprocessing module and is thus outside our model, but it is important because pre-training generates optimal weights 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 - see in Figure 3.4 (b); it is responsible for training with a training set. All parameters used here will be set as

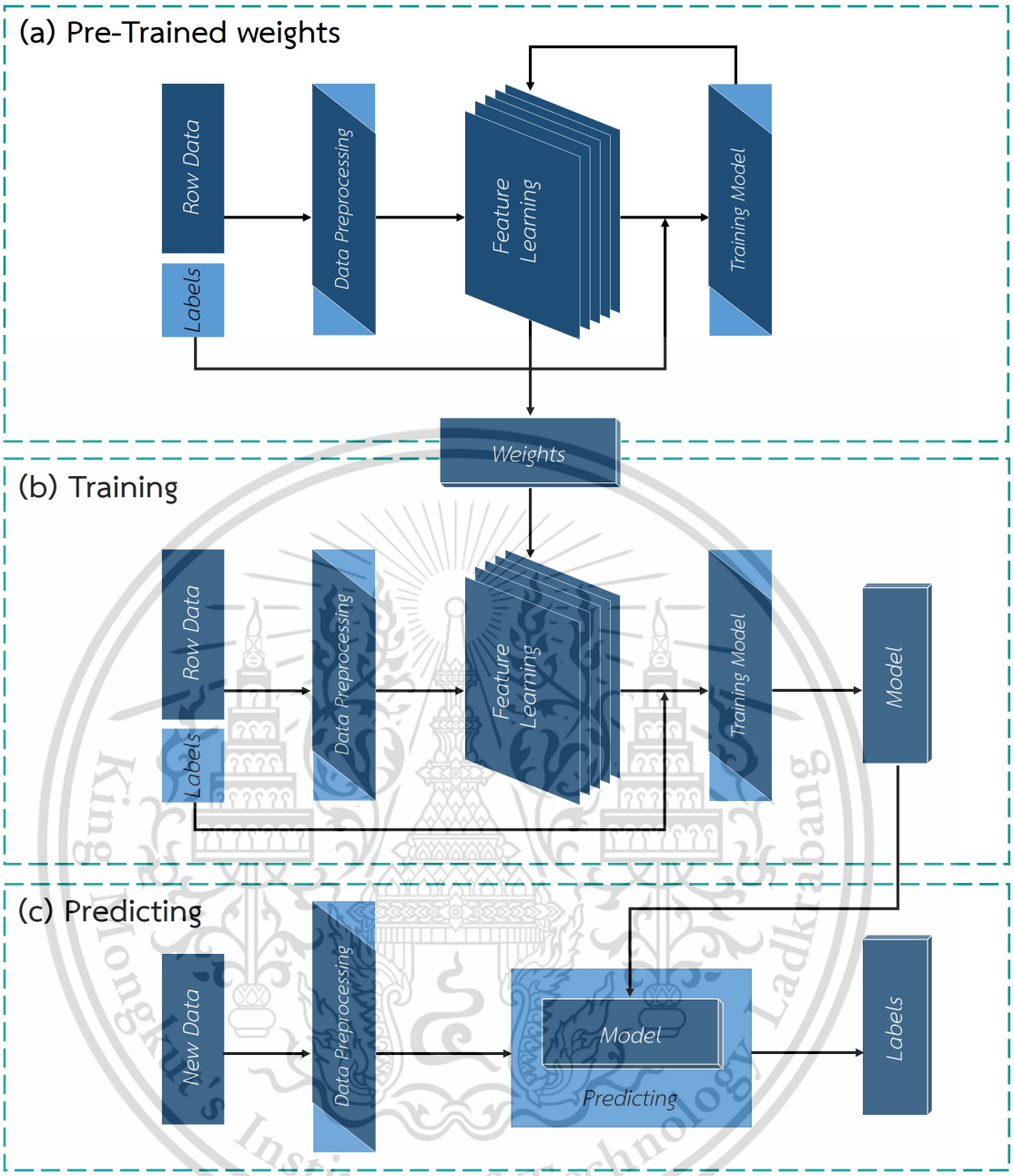


Figure 3.4 Overview of PT-XGB: (a) Pre-Trained weights, (b) Training and (c) Predicting

in the Pre-Training model, including weights in the feature learning step. The training module has seven layers - explained in further detail later - Section 3.2.1. Output from this modules is used for predicting the class labels (Section 3.2.1) for classification problems.

1. Input layer: The input layer is the first layer of the model for load training set, which may be image data or other types of data in the classification problems.

2. Data preprocessing layer: We used this module from ConvXGB, 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 3.1.4.

3. Convolutional layer: Feature are learnt in this layer using a convolutional operator. We set the number of layers, L, according to user needs. See Equation 2.4 in Section 2.1 for the equation of the convolutional operation.

   To determine the number of layers, L, we consider overall time complexity of the model (see Table 4.3, which contains L), computer power of the machine used, resulting in run time, and data adequacy since the number of features and instances that are too few, it may not be enough for convolutional layers. However, in PT-XGB, 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 3.2.2. This greatly reduces run time, see comparisons with CNN, ConvXGB and our model in Table 4.19 of Section 4.2.3.

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 (Figure 3.4 (c)) predicts the target class labels from the test set.

5. Reshape layer: The reshape layer re-adjusts the shape of the data (*e.g.* tensor) to the vector format required by the next layer, see in Figure 3.2 and full details in the description of ConvXGB.

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) [46]. We evaluate the quality of the tree by calculating the points (as in Equation 2.13):

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

   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 2.14) in Section 2.2.

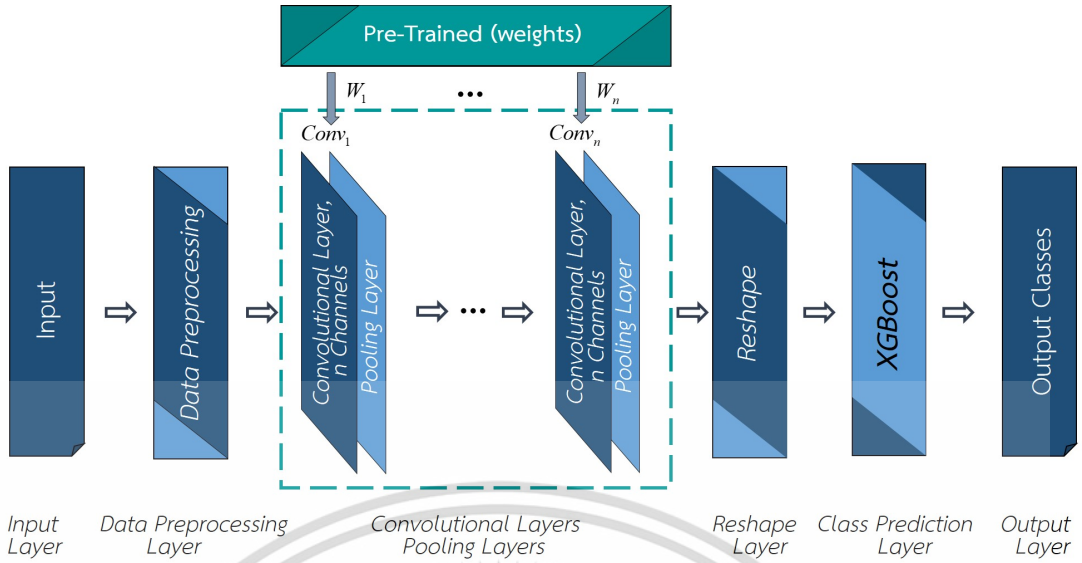


Figure 3.5 Architecture of PT-XGB

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

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 3.2.3.

3.2.2 Pre-Trained weights

Pre-training weights are commonly used for Visual Question Answering (VQA) tasks, using weights acquired from a library or public framework [49] (*e.g.* Theano [50], Lasagne [51], ImageNet [52], word2vec [53], AlexNet [54], GoogLeNet [55], and ResNet [56] *etc.*). There are many researches previously proposed using such techniques *i.e.* Jabri *et al.*[57] and Lioutas *et al.*[58]. We demonstrate using pre-training weights techniques for training models, by using the CNN structure as an example to make readers understand the work process more see Figure 2.1.

Pre-trained weights in a deep learning model avoid random weight initialization and use already pre-trained weights and avoid repeated training. Here, we used CNN for pre-training. Weights obtained from these models will be used in PT-XGB in the convolutional layer for feature learning.

This reduces the training time for the model, depending on the training set size - see in Table 4.19. Note that the weights used must be received from the pre-training model which has the same parameters as our model, *i.e.* number of convolutional layers, L, output depth, z, filter sizes, $K^{(l)}$, and filter strides, $S_k^{(l)}$, *etc.*

3.2.3 PT-XGB Training Algorithm

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

---

1. Initialize the training data set, $X = \{(x_j, y_j)|1 \leq j \leq M\}$.
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, $N' \times N'$.
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 filter sizes, $K^{(l)}$, and
   (d) filter strides, $S_k^{(l)}$
5. Determine the weights from the pre-trained model.
   (a) get weights, W_{PT} and
   (b) get bias, B_{PT}
6. For each layer, l, in $1..L$:
   (a) Calculate the convolutions (left side of Figure 3.2) to generate the $\mathcal{Y}_i(l)$ for layer, l:

$$\mathcal{Y}_i^{(l)} = \varphi(B_{PT}{}_i^{(l)} + \sum_{j=1}^{f^{(l-1)}} W_{PT}{}_{i,j}^{(l)} * \mathcal{Y}_j^{(l-1)}),$$

   (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 $(N' \times N' \times z^{(l)})$ - $\mathcal{YY}^{(l)}$
8. Initialize a new training data set for class prediction layer

$$X_{new} = \{(\mathcal{YY}_j, y_j)|1 \leq j \leq M\}.$$

9. Initialize paramers 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

10. Determine the class labels for output:

$$\hat{y}_i = \phi(YY_i) = \sum_{k=1}^{K_K} f_k(x_i), \qquad f_k \in F,$$

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

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

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

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

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

where T is the number of leaves in the tree

13. Calculate the best splitting points

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

14. Terminate

Our PT-XGB 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 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.

Chapter 4

Experimental and Numerical results

4.1 Experimental and results for ConvXGB

4.1.1 Data sets used to evaluate ConvXGB

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 [59]. 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 4.16.

4.1.2 Experimental Setup

In this experiment, we show the experiment using two types of data sets are image classification (MNIST Lecun1998) and general data (Anuran Calls (MFCCs) [2], Breast Cancer Wisconsin (Original) [3], DrivFace [4], Parkinsons [5], QSAR Biodegradation [6], Sensorless Drive Diagnosis citeDua2017, and Waveform Database Generator (ver. 2) [2]) see in Table 4.1.

Image classification data sets: we used five-fold cross-validation to train and test the models. Each data set was divided into three disjoint subsets. Then, four subsets were used as a training set and the other subset was used as a testing set. This process was repeated five times which each subsets was used exactly once as the testing set.

General classification data sets: Anuran Calls (MFCCs), Breast Cancer Wisconsin (Original), DrivFace, Parkinsons, QSAR Biodegradation, Sensorless Drive Diagnosis, Waveform Database Generator (ver. 2). We used three-fold cross-validation to train and test the models. In addition, the results from each test set are shown the averaged and standard deviation calculated.

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 3.2.3. 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 $n = 1, 2, ..., 10$. 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 available. The actual values chosen are shown in Table 4.2. We set

Table 4.1 Data sets used for ConvXGB

Data Set	Instances	Features	Features After Reshape	Classes	Filter Size	Output Depth
MNIST [1]	70,000	784	16,464	10	3×3	64
Anuran Calls (MFCCs) [2]	7,195	22	800	60	3×3	32
Breast Cancer Wisconsin (Original) [3]	699	10	512	2	4×4	32
DrivFace [4]	606	6,400	51,200	10	2×2	8
Parkinsons [5]	197	23	800	2	2×2	32
QSAR Biodegradation [6]	1,055	41	784	2	2×2	16
Sensorless Drive Diagnosis [2]	58,509	49	1,568	11	2×2	32
Waveform Database Generator (ver. 2) [2]	5,000	40	12,544	3	2×2	256

Table 4.2 Parameters in each layer of the ConvXGB model

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

na = not applicable

the filter size, $K = 2 \times 2$ and the stride of the filter $S_k = 1$ to enable small features to be recognized.

4.1.3 Experimental Results

The results of our ConvXGB model were compared to other models shown in Tables 4.30 - 4.32 including Convolutional Neural Network (CNN) [1], Extreme Gradient Boosting (XGBoost) [46], Decision Tree Classifier (DTC) [60], Multilayer Perceptron (MLP) [61] and the Support Vector Classification (SVC) [62]. We tried to configure the parameters in all cases to generate a fair comparison.

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 and the stride is set to 2, the numbers of neurons in the FC layer was 2^n, when $n = 6, 7, ..., 10$.

The XGBoost model was set similarly, with the same parameters as our model to fairly evaluate performance. Using these parameters, the accuracy of the two modelswas 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 was 2^n, when $n = 6, 7, ..., 10$, 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).

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 the properties are summarized in Table 4.3.

Table 4.3 The properties of the models used in the performance comparison

Models	Parameters Details	Ref.	Time Complexity
ConvXGB	No. of Convolutional layer $L = 2$ Pooling = No	–	$\mathcal{O}(Ld^2mnpq) + \mathcal{O}(r(Kt + \log B))$
CNN	No. of Convolutional layer $L = 2$, Pooling = Yes	[1]	$\mathcal{O}(Ld^2mnpq)$ + $\mathcal{O}(MNhce)$
XGBoost	Max_depth = 3, Objective = Binary: Logistic	[46]	$\mathcal{O}(r(Kt + \log B))$
DTC	Criterion = Gini, $minss = 2$	[60]	$\mathcal{O}(tM \log N)$
MLP1	Activation = Linear, Learning rate = 0.001	[61]	$\mathcal{O}(MNhce)$
MLP2	Activation = Sigmoid, Learning rate = 0.001	[61]	$\mathcal{O}(MNhce)$
MLP3	Activation = tanh, Learning rate = 0.001	[61]	$\mathcal{O}(MNhce)$
MLP4	Activation = ReLU, Learning rate = 0.001	[61]	$\mathcal{O}(MNhce)$
SVC1	Kernel = RBF, C = 1.0	[62]	$\mathcal{O}(M^3)$
SVC2	Kernel = Linear , C = 1.0	[62]	$\mathcal{O}(M^3)$
SVC3	Kernel = Poly, C = 1.0	[62]	$\mathcal{O}(M^3)$
SVC4	Kernel = Sigmoid, C = 1.0	[62]	$\mathcal{O}(M^3)$

Furthermore, Table 4.3 showed the time complexity of the models. We assume that parameters for descriptions the time complexity see Table 4.4

We showed the actual results from each run on the algorithms, to demonstrate the variation seen for different runs with different initial seeds. We used five-fold cross-validation to train and test the models in image classification data set (MNIST [1]) see Table 4.5, and general data (Anuran Calls (MFCCs) [2], Breast Cancer Wisconsin (Original) [3], DrivFace [4], Parkinsons [5], QSAR Biodegradation [6], Sensorless Drive Diagnosis [2], and Waveform Database Generator (ver. 2) [2]) see Tables 4.6 - 4.12.

Table 4.4 Parameters for descriptions the time complexity

Parameter	Descriptions
L	Number of layers
d	Various of input or output channels
$m \times n$	Data matrix size
$p \times q$	Filter size
$r = \|x\|$	Number of non-missing entries
K	Number of trees
t	Tree depth
B	Block length
M	Number of training sets
N	Number of features or dimensions
h	Number of hidden neurons
c	Number of classes
e	Number of epochs

Table 4.5 MNIST Data Set

Testing Fold	ConvXGB	CNN	XGBoost	DTC	MLP1	MLP2	MLP3	MLP4	SVC1	SVC2	SVC3	SCV4
1	99.08	97.83	96.18	86.99	90.89	96.50	96.43	96.70	94.14	91.57	83.42	92.51
2	99.12	98.23	96.50	86.60	90.43	96.55	96.46	96.32	94.17	92.00	83.00	92.58
3	99.02	98.18	96.21	86.37	90.43	96.33	96.33	96.68	94.06	91.77	82.85	92.42
4	99.12	98.50	96.21	86.23	90.40	96.13	96.18	96.73	93.55	91.49	82.77	92.07
5	99.14	98.08	96.07	86.94	90.06	95.90	96.38	96.43	93.65	91.19	83.18	91.87
Avg.	**99.10**	98.16	96.24	86.62	90.44	96.28	96.36	96.57	93.91	91.60	83.04	92.29
Stdv.	00.01	00.24	00.14	00.30	00.27	00.24	00.10	00.16	00.26	00.27	00.23	00.27

Table 4.6 Anuran Calls (MFCCs) Data Set

Testing Fold	ConvXGB	CNN	XGBoost	DTC	MLP1	MLP2	MLP3	MLP4	SVC1	SVC2	SVC3	SCV4
1	89.75	85.45	85.24	73.16	83.53	80.24	85.99	86.45	56.61	80.58	06.04	44.35
2	86.16	83.40	82.15	73.44	82.74	80.07	84.61	85.28	56.09	79.36	06.26	43.95
3	87.66	83.78	83.32	71.27	82.74	78.65	84.57	85.45	53.59	79.07	06.38	41.78
Avg.	**87.85**	84.21	83.57	72.62	83.00	79.65	85.06	85.73	55.43	79.67	06.23	43.36
Stdv.	01.30	00.89	01.27	00.96	00.38	00.71	00.66	00.52	01.32	00.65	00.14	01.13

Table 4.7 Breast Cancer Wisconsin (Original) Data Set

Testing Fold	ConvXGB	CNN	XGBoost	DTC	MLP1	MLP2	MLP3	MLP4	SVC1	SVC2	SVC3	SCV4
1	98.28	95.28	95.71	95.71	61.37	61.37	61.37	61.37	62.23	62.66	-	61.37
2	97.42	97.42	95.71	93.56	65.67	65.67	65.67	65.67	66.09	62.66	-	65.67
3	96.57	95.71	96.14	94.85	69.53	69.53	69.53	69.53	69.96	69.96	-	69.53
Avg.	**97.42**	96.14	95.85	94.71	65.52	65.52	65.52	65.52	66.09	65.09	-	65.52
Stdv.	00.70	00.93	00.20	00.88	03.33	03.33	03.33	03.33	03.15	03.44	-	03.33

Note: SVC3 is not supported for Breast Cancer Wisconsin (Original) data set - marked with a †.

Table 4.8 DrivFace Data Set

Testing Fold	ConvXGB	CNN	XGBoost	DTC	MLP1	MLP2	MLP3	MLP4	SVC1	SVC2	SVC3	SCV4
1	94.55	95.05	92.08	86.14	89.60	89.60	89.60	89.60	89.60	93.07	89.60	89.60
2	91.09	88.61	90.59	84.16	91.09	91.09	91.09	91.09	91.09	89.11	91.09	91.09
3	95.54	92.08	91.58	89.60	89.60	89.60	89.60	89.60	89.60	93.07	89.60	89.60
Avg.	**93.73**	91.91	91.42	86.63	90.10	90.10	90.10	90.10	90.10	91.75	90.10	90.10
Stdv.	01.91	02.63	00.62	02.25	00.70	00.70	00.70	00.70	00.70	01.87	00.70	00.70

Table 4.9 Parkinsons Data Set

Testing Fold	ConvXGB	CNN	XGBoost	DTC	MLP1	MLP2	MLP3	MLP4	SVC1	SVC2	SVC3	SCV4
1	95.38	89.23	100.0	95.38	76.92	78.46	81.54	80.00	73.85	81.54	87.69	67.69
2	96.92	90.77	92.31	83.08	81.54	87.69	89.23	81.54	83.08	87.69	87.69	81.54
3	96.92	92.31	92.31	98.46	78.46	84.62	83.08	76.92	78.46	84.62	90.77	76.92
Avg.	**96.41**	90.77	94.87	92.31	78.97	83.59	84.62	79.49	78.46	84.62	88.72	75.38
Stdv.	00.73	01.26	03.63	06.65	01.92	03.84	03.32	01.92	03.77	02.51	01.45	05.76

Table 4.10 QSAR Biodegradation Data Set

Testing Fold	ConvXGB	CNN	XGBoost	DTC	MLP1	MLP2	MLP3	MLP4	SVC1	SVC2	SVC3	SCV4
1	88.35	86.08	88.35	82.39	87.50	87.50	88.35	88.35	87.22	87.78	87.50	65.62
2	87.78	86.36	86.93	81.25	85.80	86.08	85.51	86.08	80.11	87.50	82.95	64.77
3	89.14	87.18	87.46	80.06	83.19	84.33	86.04	84.33	85.75	85.75	84.05	68.38
Avg.	**88.44**	86.54	87.58	81.23	85.50	85.97	86.63	86.25	84.36	87.01	84.83	66.26
Stdv.	00.57	00.47	00.59	00.95	01.77	01.30	01.23	01.65	03.06	00.90	01.94	01.54

Table 4.11 Sensorless Drive Diagnosis Data Set

Testing Fold	ConvXGB	CNN	XGBoost	DTC	MLP1	MLP2	MLP3	MLP4	SVC1	SVC2	SVC3	SCV4
1	99.56	93.85	98.88	98.20	87.63	93.49	94.48	89.80	25.30	87.00	81.26	08.77
2	99.58	95.38	99.03	98.33	87.03	94.37	92.83	91.46	24.53	87.63	80.59	08.87
3	99.57	95.38	98.98	98.28	88.99	93.51	93.40	90.08	24.85	86.95	80.80	09.27
Avg.	**99.57**	87.88	98.96	98.27	87.88	93.79	93.57	90.45	24.90	87.19	80.88	08.97
Stdv.	00.01	02.20	00.06	00.05	00.82	00.41	00.68	00.72	00.32	00.31	00.28	00.22

Table 4.12 Waveform Database Generator (ver. 2) Data Set

Testing Fold	ConvXGB	CNN	XGBoost	DTC	MLP1	MLP2	MLP3	MLP4	SVC1	SVC2	SVC3	SCV4
1	85.66	81.88	85.48	71.99	85.66	86.26	83.14	82.96	86.08	85.36	82.00	56.69
2	86.44	82.48	85.12	74.99	86.08	85.72	84.40	83.38	86.68	86.20	82.96	57.53
3	88.00	83.73	84.45	74.31	86.43	86.73	84.27	82.95	86.49	86.37	81.93	54.80
Avg.	**86.70**	82.70	85.02	73.76	86.06	86.24	83.94	83.10	86.42	85.92	82.30	56.34
Stdv.	00.97	00.77	00.43	01.28	00.32	00.41	00.57	00.20	00.25	00.42	00.47	01.14

In Table 4.3, we showed the accuracy of our system by comparing our model with others, using data sets in Table 4.1, and also display the results as bar charts, see Figures 4.1 - 4.8. In this experiment, we use a variety of data, both image, and general data, to expand the scope of the classification problems. Including demonstrating the ability of our model to support and handle for all classification data.

In Table 4.13, we compare our model and CNN, XGBoost, and DTC, for each data set, ConvXGB provided better accuracy than other models, including CNN and XGBoost - the prototypes for modeling - with an improvement by 0.9% to 11.7% (CNN) and 0.6% to 4.3% (XGBoost). In addition, we improved on DTC by 1.3% to 15.3%.

Our model was comapared with the MLP family (MLP1-MLP4) models in Table 4.14: ConvXGB performed better than all models in that family by 0.5% to 31.9%. Table 4.15 compares our models with those in the SVC family (SVC1-SVC4). Similarly, ConvXGB gave better results than models in SVC family by 0.3% to 31.3%.

We show the results as bar charts - see Figures 4.1 - 4.8, again, some data sets are not compatible with some model, *i.e.* in particular, Breast Cancer Wisconsin (Original) data set does not support the SVC3 model, leaving gaps in the bar charts. It is clear that our model, ConvXGB was effective in supporting all data sets (including general classification problems) and provided accuracy better than other models.

Table 4.13 Our model compared with CNN, XGBoost and DTC

Data Set	Improvement (Impv.)						
	ConvXGB	CNN		XGBoost		DTC	
	Acc.(%)	Acc.(%)	Impv.	Acc.(%)	Impv.	Acc.(%)	Impv.
MNIST	**99.1**±0.1	98.2±0.2	0.9%	96.2±0.1	2.9%	86.6±0.3	12.5%
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. 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 4.14 Our model compared with the MLP family

Data Set	ConvXGB	MLP1	MLP2	MLP3	MLP4	Improvement
	Acc.(%)	Acc.(%)	Acc.(%)	Acc.(%)	Acc.(%)	
MNIST	**99.1**±0.1	90.4±0.3	96.3±0.2	96.4±0.1	†96.6±0.1	2.5%
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. 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 4.15 Our model compared with the SVC family

Data Set	ConvXGB	SVC1	SVC2	SVC3	SVC4	Improvement
	Acc.(%)	Acc.(%)	Acc.(%)	Acc.(%)	Acc.(%)	
MNIST	**99.1**±0.1	†93.9±0.3	91.6±0.3	83.0±0.2	92.3±0.3	5.2%
Anuran Calls (MFCCs)	**87.9**±1.3	55.4±1.3	†79.7±0.7	06.2±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	09.0±0.2	12.4%
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 †.

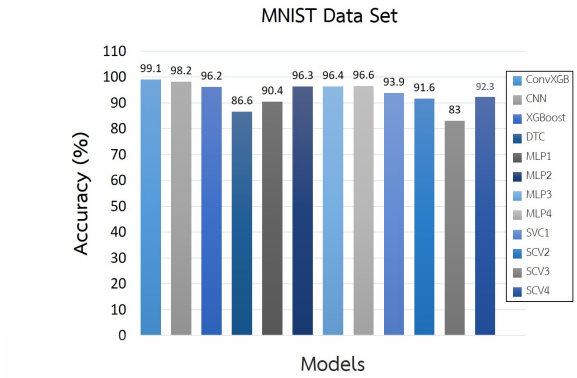


Figure 4.1 ConvXGB *vs* all models for the MNIST data set [1]

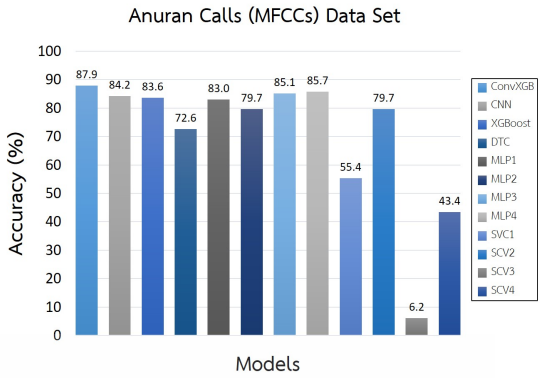


Figure 4.2 ConvXGB *vs* all models for the Anuran Calls (MFCCs) data set [2]

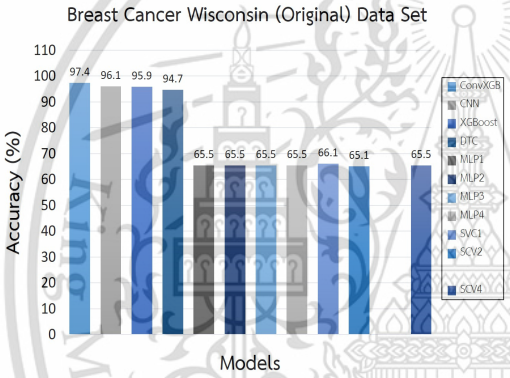


Figure 4.3 ConvXGB*vs* all models for the Breast Cancer Wisconsin (Original) data set [3]*

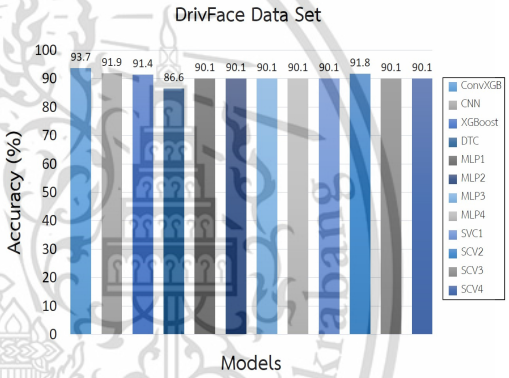


Figure 4.4 ConvXGB *vs* all models for the DrivFace data set [4]

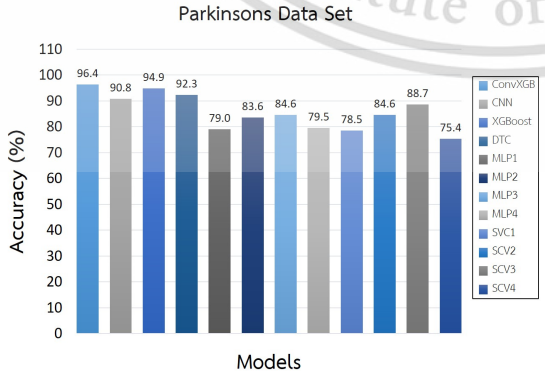


Figure 4.5 ConvXGB *vs* all models for the Parkinsons data set [5]

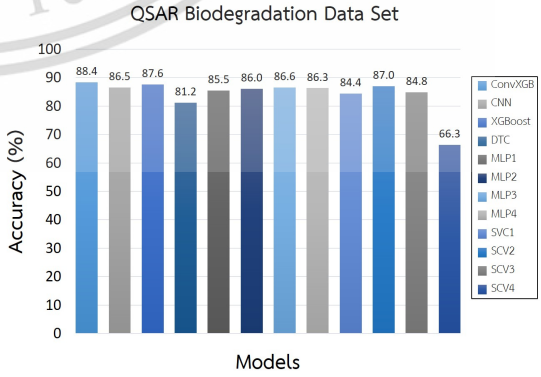


Figure 4.6 ConvXGB *vs* all models for the QSAR Biodegradation data set [6]

* SVC3 is not supported for Breast Cancer Wisconsin (Original) data set.

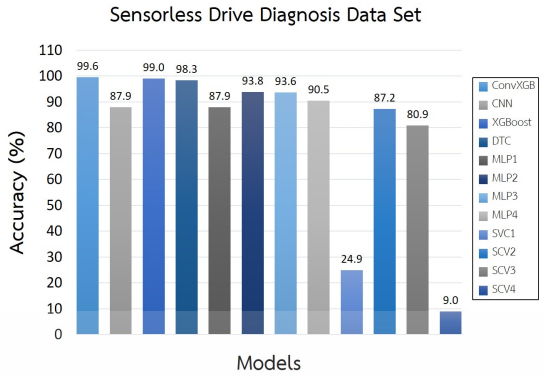


Figure 4.7 ConvXGB *vs* all models for the Sensorless Drive Diagnosis data set [2]

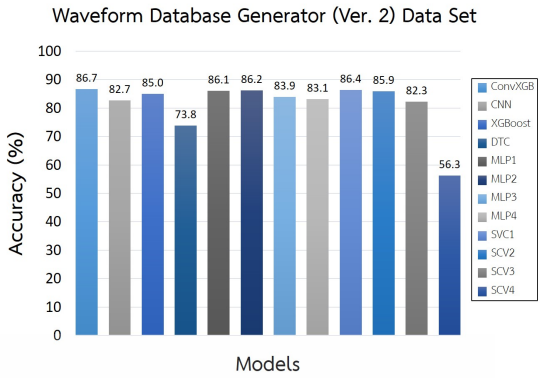


Figure 4.8 ConvXGB *vs* all models for the Waveform Database Generator (Ver. 2) data set [2]

4.2 Experimental and results for PT-XGB

4.2.1 Data sets used to evaluate PT-XGB

We evaluated performance of the model with 10 data sets for classification problems, comprising 10 data sets received from the University of California at Irvine (UCI) Repository of machine learning data sets [59]. Thus, we chose data sets from many areas, including data sets ranging from a few to very large numbers of instances.

Table 4.16 Data sets used to evaluate PT-XGB

Data Set	Instances	Features	Features After Reshape	Classes	Filter Size	Output Depth
Balance Scale [2]	625	4	32	3	3×3	32
Diabetic Retinopathy Debrecen [7]	1,151	20	16	2	3×3	16
Image Segmentation [2]	2,310	19	32	7	3×3	32
ISOLET [2]	7,797	617	512	26	3×3	32
Letter Recognition [2]	20,000	16	32	26	5×5	32
Optical Recognition of Handwritten Digits[2]	5,620	64	32	10	3×3	32
Pen-Based Recognition of Handwritten Digits [2]	10,992	16	32	10	3×3	32
Seismic Bumps [8]	2,584	19	32	2	3×3	32
Smartphone Based Recognition of Human Activities and Postural Transitions [9]	10,929	561	288	12	3×3	32
Wine [2]	178	13	32	3	3×3	32

4.2.2 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 3.2.3. 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 filter size, K, to 2×2 and 3×3, depending on the size of the training set. If 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 K is too large, accuracy may suffer (the filter size will not be larger than the training set). Additionally, a small stride of the filter $S_k = 1$ enables small features to be recognized. The parameters set in our model in each layer are shown in Table 4.17.

In this experiment, we used three-fold cross-validation to train and test the models. Each data set was divided into three disjoint subsets. Then, two subsets was 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 result from each testing set were averaged and a standard deviation was calculated.

Table 4.17 Parameters in each layer of the PT-XGB model

Layer Name	Type	Input	Filter	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	$(N', N', 1)$	$h_k \times d_k$	1	$(N', N', z^{(L-2)})$
L4*	Max-Pooling (Pool1)	$(N', N', z^{(L-2)})$	$h_p \times d_p$	$h_p \times d_p$	Pool1
L5	Convolutional	Pool1	$h_k \times d_k$	1	$(N', N', z^{(L-1)})$
L6*	Max-Pooling (Pool2)	$(N', N', z^{(L-1)})$	$h_p \times d_p$	$h_p \times d_p$	Pool2
L7	Convolutional	Pool2	$h_k \times d_k$	1	$(N', N', z^{(L)})$
L8*	Max-Pooling (Pool3)	$(N', N', z^{(L)})$	$h_p \times d_p$	$h_p \times d_p$	Pool3
L9	Reshape	Pool3	*na*	*na*	$N' \times N' \times z^{(L)}$
L10	Class Prediction	$N' \times N' \times z^{(L)}$	*na*	*na*	No. of Classes
L11	Output	*na*	*na*	*na*	No. of Classes

Notes: * Added for ConvXGB and *na* = not applicable

PT-XGB model was compared with other models *i.e.* , Convolutional Neural Network (CNN) [1], Extreme Gradient Boosting (XGBoost) [46], Logistic Regression (LR) [63, 64, 65], Extra Trees Classifier (ETC) [66], Gradient Boosting Classifier (GBC) [67, 68, 69], Random Forest Classifier (RFC) [70], Gaussian Naive Bayes (GNB) [71], Decision Tree Classifier (DTC) [60], Multilayer Perceptron (MLP) [61] and the Support Vector Classification (SVC) [62] - see Tables 4.30 - 4.32.

Table 4.18 Properties of models used in performance comparison

Models	Parameters Details	Ref.	Time Complexity
PT-XGB	No. of Conv. layer $L = 3$	-	$\mathcal{O}(Ld^2mnpq)$ + $\mathcal{O}(r(Kt + \log B))$
ConvXGB	No. of Conv. layer $L = 3$	-	$\mathcal{O}(Ld^2mnpq)$ + $\mathcal{O}(r(Kt + \log B))$
CNN*	No. of Conv. layer $L = 3$	[1]	$\mathcal{O}(Ld^2mnpq)$ + $\mathcal{O}(MNhce)$
XGBoost	Max_depth = 3	[46]	$\mathcal{O}(r(Kt + \log B))$
LR	Tol = 0.0001, C=1.0	[63, 64, 65]	$\mathcal{O}(MN)$ to $\mathcal{O}(MN^2)$
ETC	Criterion = Gini, MinSS = 2	[66]	$\mathcal{O}(MNK)$
GBC	Max_depth = 3, MinSS = 2	[67, 68, 69]	$\mathcal{O}(MNK)$
RFC	Criterion = Gini, MinSS = 2	[70]	$\mathcal{O}(KtMN \log N)$
GNB	Var_Smoothing = 1e-09	[71]	$\mathcal{O}(Mc)$
DTC	Criterion = Gini, MinSS = 2	[60]	$\mathcal{O}(tM \log N)$
MLP1	Activation = Linear	[61]	$\mathcal{O}(MNhce)$
MLP2	Activation = Sigmoid	[61]	$\mathcal{O}(MNhce)$
MLP3	Activation = tanh	[61]	$\mathcal{O}(MNhce)$
MLP4	Activation = ReLU	[61]	$\mathcal{O}(MNhce)$
SVC1	Kernel = RBF, C = 1.0	[62]	$\mathcal{O}(M^3)$
SVC2	Kernel = Linear , C = 1.0	[62]	$\mathcal{O}(M^3)$
SVC3	Kernel = Poly, C = 1.0	[62]	$\mathcal{O}(M^3)$
SVC4	Kernel = Sigmoid, C = 1.0	[62]	$\mathcal{O}(M^3)$

Notes: * CNN only handle image data and requires adaptation of the data our pre-processing module.

For the CNN model, the parameters were set to the same as those for our model, PT-XGB. 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 3.1.4) step to CNN to facilitate comparison with general data from UCI [59]. 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 evaluataed 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 17 models, including, LR, ETC, GBC, RFC, GNB and DTC. The effectiveness of our model and properties are summarized in Table 4.18

Furthermore, Table 4.18 shows the time complexity of the models. We assume that parameters for descriptions the time complexity see Table 4.4

Table 4.19 Run time of our model compared with CNN and ConvXGB

Data set	Run Time				
	PT-XGB	ConvXGB		CNN	
	time	time	Imp.	time	Imp.
Balance Scale [2]†	**0.2 s**	2.4×10^{2} s	8.3×10^{-4}	1.9×10^{4} s	1.1×10^{-5}
Diabetic Retinopathy Debrecen [7] †	**0.2 s**	1.5×10 s	1.3×10^{-2}	4.7×10^{2} s	4.3×10^{-4}
Image Segmentation [2] †	**1.3 s**	2.7×10^{3} s	4.8×10^{-4}	7.0×10^{2} s	1.9×10^{-3}
ISOLET [2] †	**1.0**$\times10^{2}$ **s**	2.4×10^{4} s	4.2×10^{-3}	4.3×10^{4} s	2.3×10^{-3}
Letter Recognition [2] †	**5.9**$\times10$ **s**	1.1×10^{5} s	5.4×10^{-4}	6.1×10^{4} s	9.7×10^{-4}
Optical Recognition of Handwritten Digits[2] †	**3.4 s**	7.0×10^{2} s	4.9×10^{-3}	2.7×10^{3} s	1.3×10^{-3}
Pen-Based Recognition of Handwritten Digits [2] †	**6.1 s**	4.1×10^{2} s	1.5×10^{-2}	4.4×10 s	1.4×10^{-1}
Seismic Bumps [8] †	**0.2 s**	2.0×10^{3} s	1.0×10^{-4}	9.0×10^{2} s	2.2×10^{-4}
Smartphone-Based Recognition of Human - Activities and Postural Transitions [9] †	**8.0**$\times10$ **s**	1.9×10^{4} s	4.2×10^{-3}	5.6×10^{4} s	1.4×10^{-3}
Wine [2] †	**0.2 s**	1.8×10 s	1.1×10^{-2}	1.6×10^{2} s	1.3×10^{-3}

Figures in the 'Imp.' column show the time for PT-XGB as a fraction of the other algorithm.

4.2.3 Experimental Results

PT-XGB, with pre-trained weights, was more efficient, because using weights obtained from pre-training instead of the loop to re-adjust weights to find the optimal weights. Training times (for deep learning model) for our model, ConvXGB and CNN are shown in Table 4.19.

PT-XGB was faster than both ConvXGB and CNN: it used only one epoch of training as with ConvXGB, while CNN used 1,000 epochs. However, the weights were derived from pre-trained model and it reduced the training data size in the pooling layer. Thus, relative time improvements *vs* PT-XGB as a fraction of the original time, ranged from 0.0001 to 0.015 for ConvXGB and 0.000011 to 0.14 for CNN. Although, PT-XGB has time complexity as ConvXGB, $\mathcal{O}(Ld^2mnpq) + \mathcal{O}(r(Kt + \log B))$ see Table 4.18. PT-XGB was several hundred times faster than ConvXGB, due to the factor of features after reshaped, PT-XGB reduce the size of the features after reshaped by downsampling in pooling layer. Therefore, the run time in training phase was less than ConvXGB. The accuracy was similarly improved, compare of PT-XGB with both CNN and ConvXGB - see Table 4.30.

We showed the actual results from each run on the algorithms, to demonstrate the variation seen for different runs with different initial seeds. We used three-fold cross-validation to train and test the models in general data (Balance Scale [2], Diabetic Retinopathy Debrecen [7], Image Segmentation [2], ISOLET [2], Letter Recognition [2] , Optical Recognition of Handwritten Digits[2], Pen-Based Recognition of Handwritten Digits [2], Seismic Bumps [8] Smartphone-Based Recognition of Human Activities and

†System Specifications– Processor: Intel® Core™ i5-4570 CPU @ 3.20GHz, Memory: 4.0 GiB, OS: Ubuntu 18.04.1LTS.

Postural Transitions [9], and Wine [2]) see Tables 4.20 - 4.29.

Table 4.20 Balance Scale Data Set Data Set

Testing Fold	PT-XGB	ConvXGB	CNN	XGBoost	LR	ETC	GBC	RFC	GNB	DTC
1	100.0	98.09	99.52	84.21	86.60	80.38	84.69	84.21	86.12	75.60
2	99.52	95.67	98.08	85.58	86.60	82.21	85.10	84.62	86.06	79.81
3	99.52	93.75	98.56	82.21	89.42	78.85	82.69	78.85	90.38	78.37
Avg.	**99.68**	95.84	98.72	84.00	87.54	80.48	84.16	82.56	87.52	77.92
Stdv.	00.27	01.77	00.73	01.69	01.63	01.68	01.29	03.22	02.48	02.14
Testing fold	**PT-XGB**	**MLP1**	**MLP2**	**MLP3**	**MLP4**	**SCV1**	**SCV2**	**SCV3**	**SCV4**	
1	100.0	86.12	87.56	84.69	89.95	89.47	92.34	100.0	43.54	
2	99.52	85.58	85.58	86.06	88.46	89.42	91.35	99.04	41.35	
3	99.52	89.42	89.90	89.90	91.35	87.50	91.35	99.52	39.42	
Avg.	**99.68**	87.04	87.68	86.88	89.92	88.80	91.68	99.52	41.44	
Stdv.	00.27	02.08	02.16	02.70	01.45	01.12	00.57	00.48	02.06	

Table 4.21 Diabetic Retinopathy Debrecen

Testing Fold	PT-XGB	ConvXGB	CNN	XGBoost	LR	ETC	GBC	RFC	GNB	DTC
1	92.97	74.74	76.04	68.75	73.18	67.45	66.41	65.36	60.42	60.68
2	88.77	75.26	80.99	65.62	73.70	65.36	69.01	63.54	60.16	56.77
3	94.79	74.67	76.50	66.84	75.72	64.49	71.54	67.62	58.22	61.36
Avg.	**92.18**	74.89	77.84	67.07	74.20	65.77	68.99	65.51	59.60	59.60
Stdv.	03.09	00.26	02.73	01.58	01.34	01.52	02.57	02.04	01.20	02.48
Testing fold	**PT-XGB**	**MLP1**	**MLP2**	**MLP3**	**MLP4**	**SCV1**	**SCV2**	**SCV3**	**SCV4**	
1	92.97	70.05	70.57	73.18	68.49	55.73	72.92	70.57	51.82	
2	88.77	71.61	70.57	72.14	71.35	56.77	74.74	64.58	52.60	
3	94.79	74.41	71.80	77.81	71.80	59.27	75.72	68.67	54.83	
Avg.	**92.18**	72.03	70.98	74.37	70.55	57.26	74.46	67.94	53.09	
Stdv.	03.09	02.21	00.71	03.02	01.80	01.82	01.42	03.06	01.56	

Table 4.22 Image Segmentation Data Set

Testing Fold	PT-XGB	ConvXGB	CNN	XGBoost	LR	ETC	GBC	RFC	GNB	DTC
1	99.87	96.75	98.05	96.75	90.00	97.14	96.62	96.62	78.70	95.71
2	98.83	97.79	98.70	98.18	91.82	96.88	97.79	97.27	81.04	96.10
3	99.48	97.01	98.83	97.92	92.08	96.88	97.92	96.49	79.22	96.23
Avg.	**99.39**	97.18	98.53	97.62	91.30	96.97	97.45	96.80	79.65	96.02
Stdv.	00.53	00.44	00.42	00.76	01.13	00.15	00.72	00.42	01.23	00.27
Testing fold	**PT-XGB**	**MLP1**	**MLP2**	**MLP3**	**MLP4**	**SCV1**	**SCV2**	**SCV3**	**SCV4**	
1	99.87	85.45	94.42	95.58	93.38	53.51	95.06	95.84	13.25	
2	98.83	89.74	94.55	94.81	95.32	54.94	95.71	95.84	13.12	
3	99.48	82.86	97.27	95.84	96.49	54.55	97.14	95.32	13.77	
Avg.	**99.39**	86.02	95.41	95.41	95.06	54.33	95.97	95.67	13.38	
Stdv.	00.53	03.47	01.61	00.54	01.57	00.74	01.06	00.30	00.34	

Table 4.23 ISOLET Data Set

Testing Fold	PT-XGB	ConvXGB	CNN	XGBoost	LR	ETC	GBC	RFC	GNB	DTC
1	99.12	94.77	96.54	95.19	96.61	89.65	92.19	89.73	84.49	80.57
2	99.38	95.23	96.69	94.00	95.92	88.42	91.84	89.15	82.11	78.68
3	99.19	95.50	96.61	95.04	95.42	89.42	92.88	89.19	82.38	81.80
Avg.	**99.23**	95.17	96.61	94.74	95.99	89.16	92.30	89.35	82.99	80.35
Stdv.	00.13	00.30	00.08	00.65	00.60	00.65	00.53	00.32	01.30	01.57
Testing fold	**PT-XGB**	**MLP1**	**MLP2**	**MLP3**	**MLP4**	**SCV1**	**SCV2**	**SCV3**	**SCV4**	
1	99.12	96.27	96.81	96.42	96.38	95.11	96.84	90.30	93.88	
2	99.38	96.23	96.46	96.65	96.31	95.00	96.08	88.42	94.00	
3	99.19	95.04	95.77	95.73	95.19	94.73	95.77	87.92	93.46	
Avg.	**99.23**	95.84	96.34	96.27	95.96	94.95	96.23	88.88	93.78	
Stdv.	00.13	00.70	00.53	00.48	00.67	00.20	00.55	01.25	00.28	

Table 4.24 Letter Recognition Data Set

Testing Fold	PT-XGB	ConvXGB	CNN	XGBoost	LR	ETC	GBC	RFC	GNB	DTC
1	99.75	95.35	97.47	88.06	76.11	93.84	91.74	93.49	63.88	86.49
2	99.67	94.86	97.91	87.67	75.46	93.63	91.03	92.74	64.57	86.79
3	99.67	95.12	98.22	88.31	76.18	93.95	91.76	92.77	64.82	87.10
Avg.	**99.70**	95.11	97.87	88.02	75.92	93.81	91.51	93.00	64.43	86.79
Stdv.	00.05	00.04	00.38	00.32	00.40	00.16	00.42	00.42	00.49	00.31
Testing fold	**PT-XGB**	**MLP1**	**MLP2**	**MLP3**	**MLP4**	**SCV1**	**SCV2**	**SCV3**	**SCV4**	
1	99.75	75.25	92.02	93.30	91.26	96.96	85.59	94.44	03.69	
2	99.67	75.60	91.92	93.70	90.30	96.67	84.72	94.53	03.63	
3	99.67	75.49	92.29	92.71	90.79	97.06	84.97	94.57	03.53	
Avg.	**99.70**	75.45	92.08	93.23	90.78	96.90	85.09	94.51	03.61	
Stdv.	00.05	00.18	00.19	00.50	00.48	00.20	00.45	00.07	00.08	

Table 4.25 Optical Recognition of Handwritten Digits Data Set

Testing Fold	PT-XGB	ConvXGB	CNN	XGBoost	LR	ETC	GBC	RFC	GNB	DTC
1	100.0	98.56	98.88	97.44	95.78	96.85	97.01	95.57	75.19	88.63
2	100.0	98.08	98.72	97.12	97.12	97.22	97.54	96.10	79.93	89.8
3	99.95	97.97	98.88	97.76	96.05	96.21	97.70	95.94	84.09	89.32
Avg.	**99.98**	98.20	98.83	97.44	96.32	96.76	97.42	95.87	79.73	89.25
Stdv.	00.03	00.26	00.09	00.26	00.71	00.51	00.36	00.27	04.45	00.59
Testing fold	**PT-XGB**	**MLP1**	**MLP2**	**MLP3**	**MLP4**	**SCV1**	**SCV2**	**SCV3**	**SCV4**	
1	100.0	95.41	97.92	97.49	97.17	65.37	97.65	98.72	08.86	
2	100.0	96.58	98.24	98.18	98.34	65.67	97.92	98.99	09.45	
3	99.95	96.37	98.02	97.97	98.13	60.01	97.97	98.77	08.81	
Avg.	**99.98**	96.12	98.06	97.88	97.88	63.68	97.85	98.83	09.04	
Stdv.	00.03	00.62	00.16	00.35	00.62	03.18	00.17	00.14	00.36	

Table 4.26 Pen Based Recognition of Handwritten Digitsn Data Set

Testing Fold	PT-XGB	ConvXGB	CNN	XGBoost	LR	ETC	GBC	RFC	GNB	DTC
1	99.92	99.34	99.67	97.79	93.78	99.04	98.36	98.25	86.05	95.14
2	99.89	99.43	99.62	98.58	93.18	98.77	98.99	98.69	85.07	96.40
3	99.92	99.43	99.65	98.58	94.32	98.80	98.58	98.77	85.75	95.96
Avg.	**99.91**	99.40	99.65	98.32	93.76	98.87	98.64	98.57	85.63	95.83
Stdv.	00.02	00.04	00.03	00.46	00.57	00.15	00.32	00.28	00.50	00.64
Testing fold	**PT-XGB**	**MLP1**	**MLP2**	**MLP3**	**MLP4**	**SCV1**	**SCV2**	**SCV3**	**SCV4**	
1	99.92	91.54	98.99	98.94	98.83	10.04	97.63	99.45	09.53	
2	99.89	90.67	99.43	98.83	98.50	12.75	97.68	99.21	10.34	
3	99.92	92.69	99.29	99.10	98.74	11.46	97.46	99.40	10.07	
Avg.	**99.91**	91.63	99.24	98.95	98.69	11.42	97.59	99.35	09.98	
Stdv.	00.02	01.01	00.22	00.14	00.17	01.36	00.12	00.13	00.41	

Table 4.27 Seismic Bumps Data Set

Testing Fold	PT-XGB	ConvXGB	CNN	XGBoost	LR	ETC	GBC	RFC	GNB	DTC
1	95.24	90.95	91.42	91.07	90.37	91.07	90.72	91.07	88.75	87.35
2	95.24	94.54	95.24	95.59	91.29	93.50	94.31	94.54	91.17	88.04
3	95.24	92.68	93.73	93.26	89.90	91.64	92.80	92.22	90.82	87.92
Avg.	**95.24**	92.72	93.46	93.31	90.52	92.07	92.61	92.61	90.25	87.77
Stdv.	00.00	01.47	01.92	02.26	00.71	01.27	01.80	01.77	01.31	00.37
Testing fold	**PT-XGB**	**MLP1**	**MLP2**	**MLP3**	**MLP4**	**SCV1**	**SCV2**	**SCV3**	**SCV4**	
1	95.24	91.30	91.30	91.30	89.68	91.30	91.30	-	91.30	
2	95.24	79.33	95.24	95.24	95.01	95.24	95.01	-	95.24	
3	95.24	93.61	93.73	93.73	93.50	93.73	93.61	-	93.73	
Avg.	**95.24**	88.08	93.42	93.42	92.73	93.42	93.31	-	93.42	
Stdv.	00.00	07.67	01.99	01.99	02.75	01.99	01.87	-	01.99	

Note: SVC3 is not supported for Seismic Bumps data set - marked with a †.

Table 4.28 Smartphone Based Recognition of Human Activities and Postural Transitions Data Set

Testing Fold	PT-XGB	ConvXGB	CNN	XGBoost	LR	ETC	GBC	RFC	GNB	DTC
1	99.56	97.31	97.61	97.04	96.24	93.99	96.76	94.26	70.22	90.53
2	99.26	97.20	97.80	97.09	96.60	93.91	96.93	94.54	73.43	90.42
3	99.20	97.34	98.11	97.06	96.95	93.96	96.87	94.84	73.92	90.89
Avg.	**99.34**	97.28	97.84	97.06	96.60	93.95	96.85	94.55	72.52	90.61
Stdv.	00.19	00.06	00.25	00.03	00.36	00.04	00.09	00.29	02.01	00.25
Testing fold	**PT-XGB**	**MLP1**	**MLP2**	**MLP3**	**MLP4**	**SCV1**	**SCV2**	**SCV3**	**SCV4**	
1	99.56	96.10	96.79	96.71	96.62	92.48	97.15	89.13	89.76	
2	99.26	96.21	97.01	96.68	95.94	91.82	97.20	88.09	88.77	
3	99.20	96.73	96.76	97.06	96.10	92.86	97.39	88.25	89.46	
Avg.	**99.34**	96.35	96.85	96.82	96.22	92.39	97.25	88.49	89.33	
Stdv.	00.019	00.34	00.14	00.21	00.36	00.53	00.13	00.56	00.51	

Table 4.29 Wine Data Set

Testing Fold	PT-XGB	ConvXGB	CNN	XGBoost	LR	ETC	GBC	RFC	GNB	DTC
1	100.0	93.33	100.0	69.56	93.33	95.00	96.67	96.67	96.67	90.00
2	100.0	91.53	91.53	69.37	93.22	94.92	91.53	96.61	98.31	91.53
3	100.0	93.22	100.0	68.76	93.22	91.53	91.53	98.31	98.31	93.22
Avg.	**100.0**	92.69	97.18	69.23	93.26	93.81	93.24	97.19	97.76	91.58
Stdv.	00.00	00.83	04.89	00.42	00.06	01.98	02.97	00.96	00.95	01.61
Testing fold	**PT-XGB**	**MLP1**	**MLP2**	**MLP3**	**MLP4**	**SCV1**	**SCV2**	**SCV3**	**SCV4**	
1	100.0	40.00	93.33	91.67	38.33	38.33	93.33	95.00	38.33	
2	100.0	42.37	94.92	94.92	32.20	52.54	94.92	98.31	44.07	
3	100.0	37.29	93.22	94.92	32.20	42.37	94.92	93.22	37.29	
Avg.	**100.0**	39.89	93.82	93.83	34.25	44.42	94.39	95.51	39.90	
Stdv.	00.00	02.54	00.95	01.88	03.54	07.32	00.92	02.58	03.65	

We evaluated the PT-XGB performance by comparing its accuracy with 17 models on 10 test data sets - see Tables 4.30 - 4.34. Data restrictions, noted in the table, prevented comparison for some models.

Table 4.30 showed the comparison of our model with ConvXGB, CNN and XGBoost. These models are our previous work (ConvXGB) and as prototypes for our model (CNN and XGBoost). However, evaluating the performance all data set. Our models provided higher accuracy than all models, with accuracy higher than ConvXGB (0.5% to 17.3%), CNN (0.2% to 14.4%), and XGBoost (1.8% - 30.8%).

Table 4.31 showed the comparison of our model with LT, ETC, and GBC and with RFC, GNB, and DTC in Table 4.32. Again, our model provided accuracy higher than LT (2.7% - 23.8%), ETC (1.0% - 26.4%), GBC (1.3% - 25.2%), LT (1.3% - 56.7%), GNB (2.2% - 35.3%) and DTC (3.4% - 32.6%).

Tables 4.33 - 4.34 showed the comparison of our model with the MLP family and the SVC family in Table 4.33, our model provides accuracy higher than the MLP family (0.7% - 17.8%) and Table 4.34 the SVC family (0.2% - 17.7%).

Finally, Figures 4.9 - 4.18, we showed the accuracy of our system by comparing our model with others, using data sets in Table 4.16, and also display the results as bar charts.

Table 4.30 Our model compared with ConvXGB, CNN and XGBoost

Data set	Improvement (Imp.)						
	PT-XGB	ConvXGB		CNN		XGBoost	
	Acc.(%)	Acc.(%)	Imp.	Acc.(%)	Imp.	Acc.(%)	Imp.
Balance Scale	**99.7**±0.3	95.8±1.8	3.9%	98.7±0.7	1%	84.0±1.7	15.7%
Diabetic Retinopathy Debrecen	**92.2**±3.1	74.9±0.3	17.3%	77.8±2.7	14.4%	67.1±1.6	25.1%
Image Segmentation	**99.4**±0.5	97.2±0.4	2.2%	98.5±0.4	0.9%	97.6±0.8	1.8%
ISOLET	**99.2**±0.1	95.2±0.3	4%	96.6±0.1	2.6%	94.7±0.7	4.5%
Letter Recognition	**99.7**±0.1	95.1±0.2	4.1%	97.9±0.4	1.8%	88.0±0.3	11.7%
Optical Recognition of Handwritten Digits	**100.0**±0.0	98.2±0.3	1.8%	98.8±0.1	1.2%	97.4±0.3	2.6%
Pen-Based Recognition of Handwritten Digits	**99.9**±0.0	99.4±0.0	0.5%	99.7±0.0	0.2%	98.3±0.5	1.6%
Seismic Bumps	**95.2**±0.0	92.7±1.5	2.5%	93.5±0.9	1.8%	93.3±2.3	1.9%
Smartphone-Based Recognition of Human - Activities and Postural Transitions	**99.3**±0.2	97.3±0.1	2%	97.8±0.3	1.5%	97.1±0.0	2.2%
Wine	**100.0**±0.0	92.7±0.8	7.3%	97.2±4.9	2.8%	68.2±0.4	30.8%

Note: Improvement *vs* PT-XGB are shown in the 'Imp.' column.

Table 4.31 Our model compared with LR, ETC and GBC

Data set	Improvement (Imp.)						
	PT-XGB	LR		ETC		GBC	
	Acc.(%)	Acc.(%)	Imp.	Acc.(%)	Imp.	Acc.(%)	Imp.
Balance Scale	**99.7**±0.3	87.5±1.6	12.2%	80.5±1.7	19.2%	84.2±1.3	15.5%
Diabetic Retinopathy Debrecen	**92.2**±3.1	74.2±1.3	18.0%	65.8±1.5	26.4%	67.0±2.6	25.2%
Image Segmentation	**99.4**±0.5	91.3±1.1	8.1%	97.0±0.2	2.4%	97.5±0.7	1.9%
ISOLET	**99.2**±0.1	96.0±0.6	3.2%	89.2±0.7	10.0%	92.3±0.5	6.9%
Letter Recognition	**99.7**±0.1	75.9±0.4	23.8%	93.8±0.2	5.9%	91.5±0.4	8.2%
Optical Recognition of Handwritten Digits	**100.0**±0.0	96.3±0.7	3.7%	96.8±0.5	3.2%	97.4±0.4	2.6%
Pen-Based Recognition of Handwritten Digits	**99.9**±0.0	93.8±0.6	6.1%	98.9±0.2	1.0%	98.6±0.3	1.3%
Seismic Bumps	**95.2**±0.0	90.5±0.7	4.7%	92.1±1.3	3.1%	92.6±1.8	2.6%
Smartphone-Based Recognition of Human - Activities and Postural Transitions	**99.3**±0.2	96.6±0.4	2.7%	94.0±0.0	5.3%	96.9±0.1	2.4%
Wine	**100.0**±0.0	93.3±0.1	6.7%	93.8±2.0	6.2%	93.2±3.0	6.8%

Note: Improvement to PT-XGB show in 'Imp.' column.

Table 4.32 Our model compared with RFC, GNB and DTC

Data set	Improvement (Imp.)						
	PT-XGB	RFC		GNB		DTC	
	Acc.(%)	Acc.(%)	Imp.	Acc.(%)	Imp.	Acc.(%)	Imp.
Balance Scale	**99.7**±0.3	82.6±3.2	17.1%	87.5±2.5	12.2%	77.9±2.1	21.8%
Diabetic Retinopathy Debrecen	**92.2**±3.1	65.5±2.0	56.7%	59.6±1.2	32.6%	59.6±2.5	32.6%
Image Segmentation	**99.4**±0.5	96.8±0.4	2.6%	79.7±1.2	19.7%	96.0±0.3	3.4%
ISOLET	**99.2**±0.1	89.4±0.3	9.8%	83.0±1.3	16.2%	80.4±1.6	18.8%
Letter Recognition	**99.7**±0.1	93.0±0.4	6.7%	64.4±0.5	35.3%	86.8±0.3	12.9%
Optical Recognition of Handwritten Digits	**100.0**±0.0	95.9±0.3	4.1%	79.7±4.5	20.3%	89.3±0.6	10.7%
Pen-Based Recognition of Handwritten Digits	**99.9**±0.0	98.6±0.3	1.3%	85.6±0.5	14.3%	95.8±0.6	4.1%
Seismic Bumps	**95.2**±0.0	92.6±1.8	2.6%	90.3±1.3	4.9%	87.8±0.4	7.4%
Smartphone-Based Recognition of Human - Activities and Postural Transitions	**99.3**±0.2	94.6±0.3	4.7%	72.5±2.0	26.8%	90.6±0.3	8.7%
Wine	**100.0**±0.0	97.2±1.0	2.8%	97.8±1.0	2.2%	91.6±1.6	8.4%

Note: Improvement *vs* PT-XGB shown in 'Imp.' column.

Table 4.33 Our model compared with the MLP family

Data set	PT-XGB	MLP1	MLP2	MLP3	MLP4	Improvement
	Acc.(%)	Acc.(%)	Acc.(%)	Acc.(%)	Acc.(%)	
Balance Scale	**99.7**±0.3	87.0±2.0	87.7±2.2	86.9±2.7	†89.9±1.5	9.8%
Diabetic Retinopathy Debrecen	**92.2**±3.1	72.0±2.2	71.0±0.7	†74.4±3.0	70.6±1.8	17.8%
Image Segmentation	**99.4**±0.5	86.0±3.5	†95.4±1.6	†95.4±0.5	95.1±1.6	4.0%
ISOLET	**99.2**±0.1	95.8±0.7	†96.3±0.5	†96.3±0.5	96.0±0.7	2.9%
Letter Recognition	**99.7**±0.1	75.5±0.2	92.1±0.2	†93.2±0.5	90.8±0.5	6.5%
Optical Recognition of Handwritten Digits	**100.0**±0.0	96.1±0.6	†98.1±0.2	97.9±0.4	97.9±0.6	1.9%
Pen-Based Recognition of Handwritten Digits	**99.9**±0.0	91.6±1.0	†99.2±0.2	99.0±0.1	98.7±0.2	0.7%
Seismic Bumps	**95.2**±0.0	88.1±7.7	†93.4±2.0	†93.4±2.0	92.7±2.8	1.8%
Smartphone-Based Recognition of Human - Activities and Postural Transitions	**99.3**±0.2	96.4±0.3	†96.9±0.1	96.8±0.2	96.2±0.4	2.4%
Wine	**100.0**±0.0	39.9±2.5	93.8±1.0	†93.8±1.9	34.3±3.5	6.2%

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

Table 4.34 Our model compared with the SVC family

Data set	PT-XGB Acc.(%)	SVC1 Acc.(%)	SVC2 Acc.(%)	SVC3 Acc.(%)	SVC4 Acc.(%)	Improvement
Balance Scale	**99.7**±0.3	88.8±1.1	91.7±0.6	†99.5±0.5	41.4±2.1	0.2%
Diabetic Retinopathy Debrecen	**92.2**±3.1	57.3±1.8	†74.5±1.4	67.9±3.1	53.1±1.6	17.7%
Image Segmentation	**99.4**±0.5	54.3±0.7	†96.0±1.0	95.7±0.3	13.4±0.3	3.4%
ISOLET	**99.2**±0.1	95.0±0.2	†96.2±0.6	88.9±1.3	93.8±0.3	3.0%
Letter Recognition	**99.7**±0.1	†96.9±0.2	85.1±0.5	94.5±0.1	03.6±0.1	2.8%
Optical Recognition of Handwritten Digits	**100.0**±0.0	63.7±3.2	97.9±0.2	†98.8±0.1	09.0±0.4	1.2%
Pen-Based Recognition of Handwritten Digits	**99.9**±0.0	11.4±1.4	97.6±0.1	†99.4±0.1	10.0±0.1	0.5%
Seismic Bumps	**95.2**±0.0	†93.4±2.0	93.3±1.9	-	†93.4±2.0	1.8%
Smartphone-Based Recognition of Human - Activities and Postural Transitions	**99.3**±0.2	92.4±0.5	†97.3±0.1	88.5±0.6	89.3±0.5	2.0%
Wine	**100.0**±0.0	44.4±7.3	94.4±0.9	†95.5±2.6	39.9±3.7	4.5%

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

SVC3 is not supported for Seismic Bumps data set.

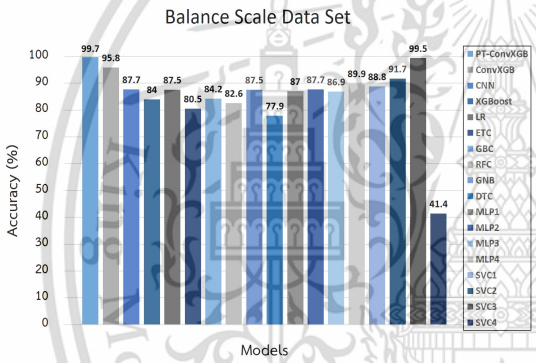


Figure 4.9 PT-XGB *vs* all models for the Balance Scale data set [2]

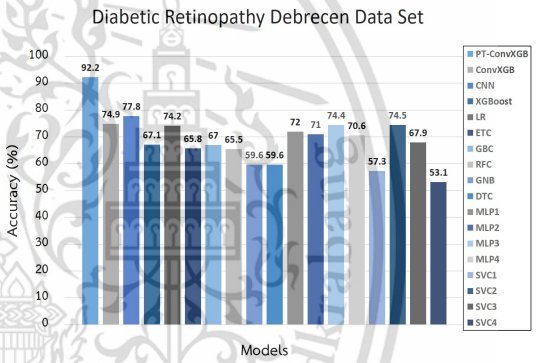


Figure 4.10 PT-XGB *vs* all models for the Diabetic Retinopathy Debrecen data set [7]

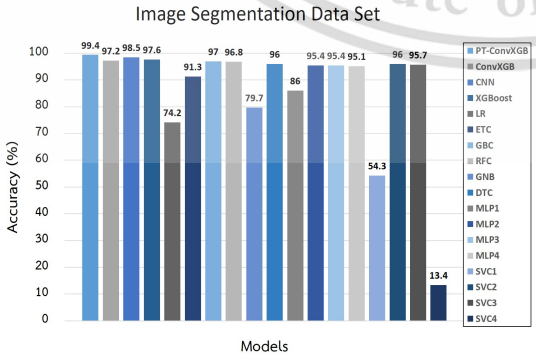


Figure 4.11 PT-XGB *vs* all models for the Image Segmentation data set [2]

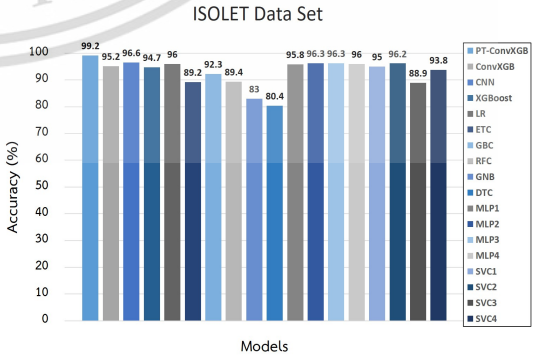


Figure 4.12 PT-XGB *vs* all models for the ISOLET data set [2]

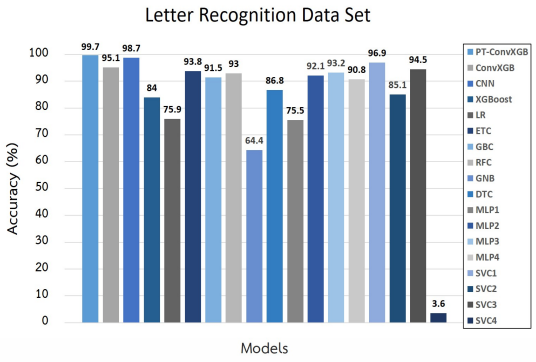


Figure 4.13 PT-XGB *vs* all models for the Letter Recognition data set [2]

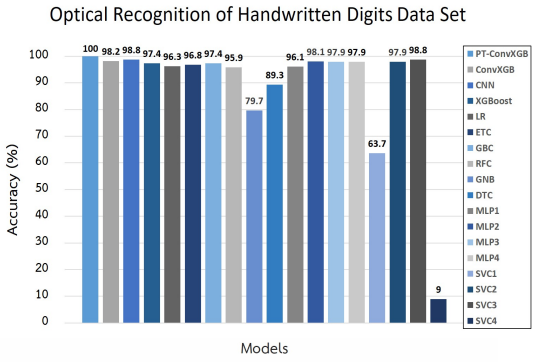


Figure 4.14 PT-XGB *vs* all models for the Optical Recognition of Handwritten Digits data set [2]

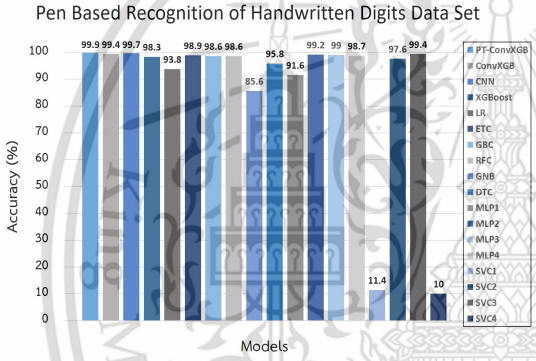


Figure 4.15 ConvXGB *vs* all models for the Pen-Based Recognition of Handwritten Digits data set [2]

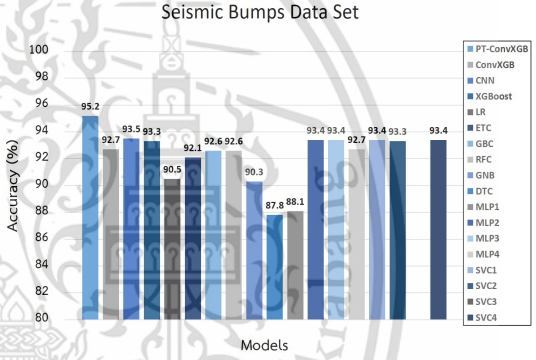


Figure 4.16 ConvXGB *vs* all models for the Seismic Bumps data set [8]

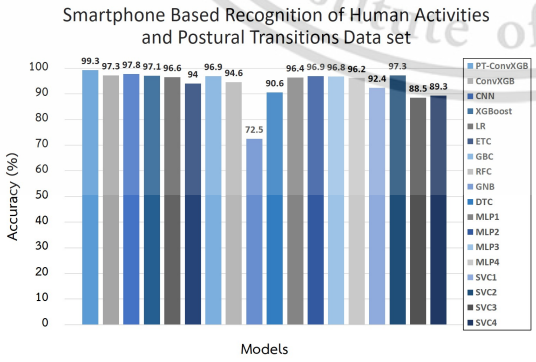


Figure 4.17 ConvXGB *vs* all models for the Smartphone Based Recognition of Human Activities and Postural Transitions data set [9]

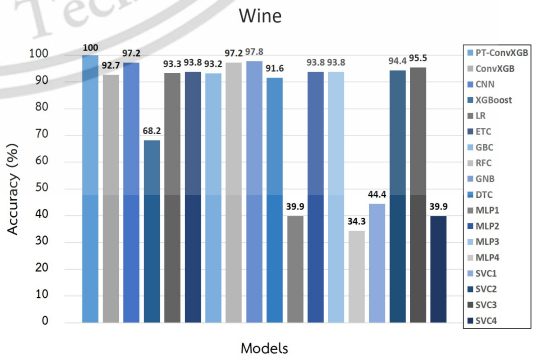


Figure 4.18 ConvXGB *vs* all models for the Wine data set [2]

Chapter 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.

PT-XGB extended and improved our previous work on Convolutional Extreme Gradient Boosting (ConvXGB) for classification problems; it covers both general and image data as in ConvXGB. Pre-trained weights, obtained from the pre-trained module, meant that PT-XGB did not need to re-adjust the weights to find optimal weights in the training process. Therefore, the training time was significantly reduced compared to ConvXGB and conventional CNN. In addition, accuracy also increased for PT-XGB.

Overall, results from several classification problems, found in common benchmark data sets, showed that our model improved on our previous work (ConvXGB) and models used as prototypes for our model (CNN and XGBoost) as well as a selection of other commonly used models, *e.g.* LR, ETC, GBC, RFC, GNB, and DTC, including MLP and SVC families.

A survey and review state of the art and research challenges for human activity recognition, our pre-training weight method has not been explained, which challenges the further development of the research in the future. Therefore, our pre-training weight method is a new thing for other general classification problems. It opens the issue of challenging problems for future development and research improvement. PT-XGB has expanded the potential of deep learning techniques. Pre-training context, we obtain a set of weights from the pre-training model, CNN was used as a prototype. Note that the weights used must be received from the pre-training model whose parameters are set as same our model, *i.e.* number of convolutional layers, L, *etc.*

In addition, PT-XGB has been described as a limitation. The number of convo-

lutional layers, L in training procedure and number of tree, K in prediction procedure, PT-XGB can be increased or decreased with some constraints. However, it cannot be arbitrarily increased. L and K increasing affects overall time complexity of the model, computer power of the machine used. In addition, the number of features and large instances that affect the run time of the system. Therefore, we would like to suggest it would be best for considering to estimate the run time, cost of the system, and accuracy required. We should consider these things before designing or developing models.

Finally, we are looking for techniques for increasing the flexibility of our model mentioned above, which will be a challenge and developed in future research.

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[2] D. Dheeru and E. Karra Taniskidou, "Uci machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[3] W. H. Wolberg and O. L. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 87, no. 23, pp. 9193–9194, 1991.

[4] K. Diaz-Chito, A. Hernández-Sabaté, and A. M. López, "A reduced feature set for driver head pose estimation," *Appl. Soft Comput.*, vol. 45, no. C, pp. 98–107, aug 2016. [Online]. Available: http://dx.doi.org/10.1016/j.asoc.2016.04.027

[5] M. A Little, P. Mcsharry, S. Roberts, D. A E Costello, and I. M Moroz, "Exploiting nonlinear recurrence and fractal scaling properties for voice disorder detection," *Biomedical engineering online*, vol. 6, pp. 1–19, 02 2007.

[6] M. Kamel, T. Ringsted, D. Ballabio, R. Todeschini, and V. Consonni, "Quantitative structure-activity relationship models for ready biodegradability of chemicals," *Journal of chemical information and modeling*, vol. 53, pp. 867–878, 03 2013.

[7] B. Antal and A. Hajdu, "An ensemble-based system for automatic screening of diabetic retinopathy," *Knowledge-Based Systems*, vol. 60, 04 2014.

[8] M. Sikora and . Wróbel, "Application of rule induction algorithms for analysis of data collected by seismic hazard monitoring systems in coal mines," *Archives of Mining Sciences*, vol. 55, pp. 91–114, 01 2010.

[9] J.-L. Reyes-Ortiz, L. Oneto, A. Samà, X. Parra, and D. Anguita, "Transition-aware human activity recognition using smartphones," *Neurocomput.*, vol. 171, no. C, pp. 754–767, jan 2016. [Online]. Available: https://doi.org/10.1016/j.neucom.2015.07.085

[10] N. Nikzad–Khasmakhi, M. Balafar, and M. R. Feizi–Derakhshi, "The state-of-the-art in expert recommendation systems," *Engineering Applications of Artificial Intelligence*, vol. 82, pp. 126–147, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197619300703

[11] I. Portugal, P. Alencar, and D. Cowan, "The use of machine learning algorithms in recommender systems: A systematic review," *Expert Systems*

with Applications, vol. 97, pp. 205–227, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417417308333

[12] J. M. Mateos-Pérez, M. Dadar, M. Lacalle-Aurioles, Y. Iturria-Medina, Y. Zeighami, and A. C. Evans, "Structural neuroimaging as clinical predictor: A review of machine learning applications," *NeuroImage: Clinical*, vol. 20, pp. 506–522, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2213158218302602

[13] J. G. A. Barbedo, "Detection of nutrition deficiencies in plants using proximal images and machine learning: A review," *Computers and Electronics in Agriculture*, vol. 162, pp. 482 –492, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169918318957

[14] B. M. Henrique, V. A. Sobreiro, and H. Kimura, "Literature review: Machine learning techniques applied to financial market prediction," *Expert Systems with Applications*, vol. 124, pp. 226–251, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095741741930017X

[15] H. Ghoddusi, G. G. Creamer, and N. Rafizadeh, "Machine learning in energy economics and finance: A review," *Energy Economics*, vol. 81, pp. 709–727, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140988319301513

[16] W. Bao, N. Lianju, and K. Yue, "Integration of unsupervised and supervised machine learning algorithms for credit risk assessment," *Expert Systems with Applications*, vol. 128, pp. 301–315, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417419301472

[17] Y. Yao, F. Shen, J. Zhang, L. Liu, Z. Tang, and L. Shao, "Extracting privileged information for enhancing classifier learning," *IEEE Transactions on Image Processing*, vol. 28, no. 1, pp. 436–450, Jan 2019.

[18] Y. Guo, X. Jia, and D. Paull, "Effective sequential classifier training for svm-based multitemporal remote sensing image classification," *IEEE Transactions on Image Processing*, vol. 27, no. 6, pp. 3036–3048, June 2018.

[19] Z. Zhang, W. Jiang, J. Qin, L. Zhang, F. Li, M. Zhang, and S. Yan, "Jointly learning structured analysis discriminative dictionary and analysis multiclass classifier," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 8, pp. 3798–3814, Aug 2018.

[20] Y. Wang, S. Liu, C. Chen, and B. Zeng, "A hierarchical approach for rain or snow removing in a single color image," *IEEE Transactions on Image Processing*, vol. 26, no. 8, pp. 3936–3950, Aug 2017.

[21] E. L. Park, S. Cho, and P. Kang, "Supervised paragraph vector: Distributed representations of words, documents and class labels," *IEEE Access*, vol. 7, pp. 29 051–29 064, 2019.

[22] Y. Ma, P. Zhang, and J. Ma, "An ontology driven knowledge block summarization approach for chinese judgment document classification," *IEEE Access*, vol. 6, pp. 71 327–71 338, 2018.

[23] M. M. Mirończuk and J. Protasiewicz, "A recent overview of the state-of-the-art elements of text classification," *Expert Systems with Applications*, vol. 106, pp. 36–54, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095741741830215X

[24] D. Isa, L. H. Lee, V. P. Kallimani, and R. RajKumar, "Text document preprocessing with the bayes formula for classification using the support vector machine," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 9, pp. 1264–1272, Sept 2008.

[25] H. H. Pajouh, R. Javidan, R. Khayami, A. Dehghantanha, and K. R. Choo, "A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in iot backbone networks," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 2, pp. 314–323, April 2019.

[26] H. Sadreazami, A. Mohammadi, A. Asif, and K. N. Plataniotis, "Distributed-graph-based statistical approach for intrusion detection in cyber-physical systems," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 1, pp. 137–147, March 2018.

[27] M. H. Ali, B. A. D. A. Mohammed, A. Ismail, and M. F. Zolkipli, "A new intrusion detection system based on fast learning network and particle swarm optimization," *IEEE Access*, vol. 6, pp. 20 255–20 261, 2018.

[28] M. Adam, E. Y. Ng, J. H. Tan, M. L. Heng, J. W. Tong, and U. R. Acharya, "Computer aided diagnosis of diabetic foot using infrared thermography: A review," *Computers in Biology and Medicine*, vol. 91, pp. 326–336, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010482517303566

[29] H. Müller and D. Unay, "Retrieval from and understanding of large-scale multi-modal medical datasets: A review," *IEEE Transactions on Multimedia*, vol. 19, no. 9, pp. 2093–2104, Sept 2017.

[30] A. Carè, F. A. Ramponi, and M. C. Campi, "A new classification algorithm with guaranteed sensitivity and specificity for medical applications," *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 393–398, July 2018.

[31] H. Zhu, X. Liu, R. Lu, and H. Li, "Efficient and privacy-preserving online medical pre-diagnosis framework using nonlinear svm," *IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 3, pp. 838–850, May 2017.

[32] W. Hu, B. Wu, P. Wang, C. Yuan, Y. Li, and S. Maybank, "Context-dependent random walk graph kernels and tree pattern graph matching kernels with applications to action recognition," *IEEE Transactions on Image Processing*, vol. 27, no. 10, pp. 5060–5075, Oct 2018.

[33] A. A. Adewuyi, L. J. Hargrove, and T. A. Kuiken, "An analysis of intrinsic and extrinsic hand muscle emg for improved pattern recognition control," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 24, no. 4, pp. 485–494, April 2016.

[34] Y. Geng, Y. Ouyang, O. W. Samuel, S. Chen, X. Lu, C. Lin, and G. Li, "A robust sparse representation based pattern recognition approach for myoelectric control," *IEEE Access*, vol. 6, pp. 38 326–38 335, 2018.

[35] E. Tu, N. Kasabov, and J. Yang, "Mapping temporal variables into the neucube for improved pattern recognition, predictive modeling, and understanding of stream data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 6, pp. 1305–1317, June 2017.

[36] L. D. W. Thomas and A. Leiponen, "Big data commercialization," *IEEE Engineering Management Review*, vol. 44, no. 2, pp. 74–90, Second 2016.

[37] F. Liang, W. Yu, D. An, Q. Yang, X. Fu, and W. Zhao, "A survey on big data market: Pricing, trading and protection," *IEEE Access*, vol. 6, pp. 15 132–15 154, 2018.

[38] N. Chawla and D. Davis, "Bringing big data to personalized healthcare: A patient-centered framework," *Journal of general internal medicine*, vol. 28(suppl 3), pp. 1–7, 2013.

[39] N. Kruger, P. Janssen, S. Kalkan, M. Lappe, A. Leonardis, J. Piater, A. J. Rodriguez-Sanchez, and L. Wiskott, "Deep hierarchies in the primate visual cortex: What can we learn for computer vision?" *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1847–1871, Aug 2013.

[40] A. Brunetti, D. Buongiorno, G. F. Trotta, and V. Bevilacqua, "Computer vision and deep learning techniques for pedestrian detection and tracking: A survey," *Neurocomputing*, vol. 300, pp. 17–33, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092523121830290X

[41] J. Thevenot, M. B. López, and A. Hadid, "A survey on computer vision for assistive medical diagnosis from faces," *IEEE Journal of Biomedical and Health Informatics*, vol. 22, no. 5, pp. 1497–1511, Sept 2018.

[42] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *ArXiv e-prints*, pp. 01–32, aug 2017.

[43] J. Choo and S. Liu, "Visual analytics for explainable deep learning," *ArXiv e-prints*, pp. 01–10, apr 2018.

[44] 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*, vol. 6, no. 2, pp. 48–56, April 2017.

[45] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, Aug 2013.

[46] 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*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

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

[48] D. Stutz, "Understanding convolutional neural networks," Fakultät für Mathematik, Informatik und Naturwissenschaften Lehr-und Forschungsgebiet Informatik VIII Computer Vision, Seminar Report, 2014.

[49] 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*, vol. 163, pp. 21–40, language in Vision. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314217300772

[50] T. D. Team, "Theano: A python framework for fast computation of mathematical expressions," *CoRR*, vol. abs/1605.02688, pp. 1–19, 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[51] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, and et al., "Lasagne: First release." aug 2015. [Online]. Available: https://doi.org/10.5281/zenodo.27878

[52] S. Gross and M. Wilber, "Training and investigating residual nets," 2016.

[53] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," pp. 1–12, 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[54] 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*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[55] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," pp. 1–9, 2015.

[56] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognitions," pp. 770–778, 2016.

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

[58] V. Lioutas, N. Passalis, and A. Tefas, "Explicit ensemble attention learning for improving visual question answering," *Pattern Recognition Letters*, vol. 111, pp. 51–57, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865518301600

[59] A. Asuncion and D. Newman, "Uci machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/$\sim$mlearn/{MLR}epository.html

[60] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[61] G. E. Hinton, "Connectionist learning procedures," *Artificial Intelligence*, vol. 40, no. 1, pp. 185–234, 1989. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0004370289900490

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

[63] 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*, vol. 9, pp. 1871–1874, jun 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1390681.1442794

[64] H.-F. Yu, F.-L. Huang, and C.-J. Lin, "Dual coordinate descent methods for logistic regression and maximum entropy models," *Machine Learning*, vol. 85, no. 1, pp. 41–75, Oct 2011. [Online]. Available: https://doi.org/10.1007/s10994-010-5221-8

[65] 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, vol. abs/1407.0202, pp. 1–1, 2014. [Online]. Available: http://arxiv.org/abs/1407.0202

[66] P. Geurts and L. Ernst, Damien & Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, Apr 2006. [Online]. Available: https://doi.org/10.1007/s10994-006-6226-1

[67] J. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, 11 2000.

[68] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367 – 378, 2002, nonlinear Methods and Data Mining. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167947301000652

[69] T. Hastie, R. Tibshirani, and J. Friedman, "The elements of statistical learning," *Springer*, vol. 1, 01 2009.

[70] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[71] 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*. Heidelberg: Physica-Verlag HD, 1982, pp. 30–41.

Appendix

Appendix A

1^{st} journal

Published by Thai Journal of Mathematics

“A Deep Single-Pass Learning for Recognition of Handwritten Digits”

Reprint submitted to **T**hai **J**ournal of **M**athematics

http://thaijmath.in.cmu.ac.th
Online ISSN 1686-0209

A Deep Single-Pass Learning for Recognition of Handwritten Digits

Setthanun Thongsuwan and Saichon Jaiyen [1]

Advanced Artificial Intelligence (AAI) Research Laboratory,
Department of Computer Science,
King Mongkut's Institute of Technology Ladkrabang,
Bangkok 10520, Thailand
e-mail : tsetthanun@gmail.com, saichon.ja@kmitl.ac.th

Abstract : We describe a deep learning model - Deep Single-Pass Learning (DSPL) - that can learn a data set, with a single pass for recognition, and predict with high accuracy, when evaluated for visual recognition of handwritten digits. DSPL consists of several stacked convolutional layers to learn features automatically and Extreme gradient boosting (XGBoost) was set as the last layer for predicting class labels. The learning time complexity is $\mathcal{O}(Lc^2mnpq)$, or less than the learning time of deep learning - Convolutional Neural Networks (CNNs). The network does not need iteration to re-adjust weights during feature learning. Tests showed that our model provided better accuracy than other models *i.e.* CNNs, XGBoost, LR, ETC, GBC, RFC, GNB, and DTC, including MLP and SVC families: in the worst case, DSPL provided 99.95% accuracy.

Keywords : deep single-pass learning; handwriting recognition; pattern recognition; convolutional neural networks; xgboost

2010 Mathematics Subject Classification : 68T10; 68T30; 62H30; 62H35

---

[0]Thanks! This research was supported by the Thailand Research Fund (TRF) under grant number RTA6080013

[1]Corresponding author email: saichon.ja@kmitl.ac.th, tsetthanun@gmail.com

1 Introduction

Handwriting recognition (HWR) is one of the challenging problems for research in artificial intelligence (AI) and has been extensively investigated. Handwriting patterns are specific to a person and these characteristics can be used in various areas of pattern recognition, *e.g.* writer identification [1, 2, 3]. It also includes the analysis of personal behavior: these unique features can be used to analyze various diseases, *e.g.* [4, 5, 6]. However, the ambiguity and lack of clarity of handwriting is still a problem in learning and pattern recognition of systems, as well as optical recognition of handwritten digits [7], described in more detail in Section 2.1, therefore, effective methods are still being developed and presented continuously to achieve higher accuracy results.

In recent years, deep learning technology has gained significant attention in AI. It caused a revolution in state-of-the-art developments in all fields of research, with the ability to effectively learn a network, that solves a problem. Especially, automatic features learning, from the training set, allows a system to discover the representations needed for predictions. Currently, there are many alternative learning models for deep learning used in HWR problems. Convolutional Neural Networks (CNNs) [8] belong to a class of deep learning techniques, which has become widely used by researchers. All deep learning models still to learn repeatedly to adjust the weights, that causes the system to have high computational complexity (including CNNs). Furthermore, these models become slow when a very large data set must be learned and the models use many epochs in the learning step, because of the optimization techniques added.

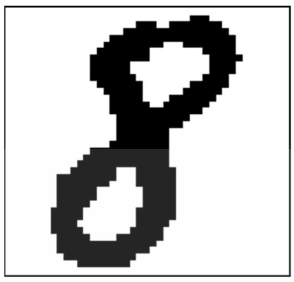
We describe a new deep learning model - Deep Single-Pass Learning (DSPL) - to learn a data set in a single pass. Our model has three convolutional layers, where XGBoost[11] is the last layer, but the structure of the convolution layer is flexible and the number of layers can be increased or decreased, depending on available computational power and data set size.

The rest of this paper is organized as follows. Materials and methods are introduced in Section 2. In Section 3, we describe the deep single-pass learning (DSPL) for parameter computation and learning algorithm. The model evaluations follow in Section 4. Finally, Section 5 concludes.

2 Materials and Methods

2.1 Handwritten Digits Data Set

The handwritten digits data set used to evaluate the performance of our model, was built by Garris *et al.*[12] from handwritten digits, on a preprinted form, from 43 people, with 30 for the training set and 13 others for the test set. The National



0	0	1	8	13	13	2	0
0	4	16	8	1	12	4	0
0	7	13	3	10	13	0	0
0	3	15	15	14	15	0	0
0	0	13	10	0	10	6	0
0	0	11	5	0	9	8	0
0	0	7	11	4	14	2	0
0	0	1	13	12	4	0	0

(a) (b)

Figure 1: Examples of (a) original images and (b) digital representations used in our evaluations.

Institute of Standards and Technology (NIST) preprocessed Garris *et al.*'s images to reduce dimensionality and extract normalized bitmaps - see Figure 1.

An input matrix of 8×8 was be created and each element is an integer in the range 0..16, by dividing into non-overlapping blocks of 4×4 from 32×32 bitmaps where the number of on pixels is counted in each block. Readers can download the data sets from the University of California at Irvine (UCI) Repository of machine learning data sets [7]. This data set is provided in the form of vectors, each vector consists of classes label (in the last column) see Definition 2.2.

Definition 2.1. *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:*

$$\mathcal{I} = \{p_{jk} | 1 \leq j \leq H, 1 \leq k \leq D\}, \tag{2.1}$$

Definition 2.2. *Training set: Let $\mathbb{I} = \{\mathcal{I}_i, y_i | 1 \leq i \leq M\}$, is the training set of M images and y_i is the label of image $\mathbf{x}_i$ in $\mathbb{R}$*

This data set was found in the UCI repository [7], included 5,620 images, reduced to 64 real-valued pixels. Thus for the test data set: $H = D = 8$ and $M = 5{,}620$.

2.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs), first described by Lecun *et al.* [8], are described in detail by Goodfellow et al [9] and Stutz [10]. A short summary follows, but readers familiar with CNNs may skip this section.

CNNs have an architecture, generally consisting of $\mathcal{L}$ convolutional layers, indexed by k, alternating with multiple pooling layers, which are responsible for learning the features of the training data. The last layer is usually a Fully Connected (FC) one. In each layer, a convolution operation, $\mathcal{K}^k$, is applied to the 'image' which the input to the k^{th} layer.

We assume a grayscale 'image', $\mathcal{Y}_j^k$, as the input to the k^{th} layer, (where $\mathcal{Y}_j^1 = \mathcal{I}_j$). The discrete convolution of $\mathcal{Y}_{j,m,n}^k$ and the kernel $\mathcal{K}^k$, with size $h \times d$ and indices $u \in [-h, h]$ and $v \in [-d, d]$, is computed as

$$\mathcal{Y}_{j,m,n}^{k+1} = (\mathcal{Y}_j^k \otimes \mathcal{K})_{j,m,n} = \sum_{u=-h}^{h} \sum_{v=-d}^{d} \mathcal{K}_{u,v}^k \mathcal{Y}_{j,m+u,n+v}^k \tag{2.2}$$

where m, n ranges over the indices in the 'image' in the $k + 1^{th}$ layer, which is not always the same as the range in layer, k. For understanding, the calculation in each layer is forwarded to the next layer: the input to the first ($k = 1$) layer is the set of original images, but after convolution, the inputs to the next, and subsequent layers, are the F^k 'features' in layer k, $\mathcal{Y}_j^k[j \in [1..F^k]$, because they are transformations (the convolutions) of the previous inputs. Note that the number of feature maps in each layer, F^k, may vary, set by the user for each application. In convolution layer, $k \in [1..L]$:

Then, the j^{th} feature map for layer $k + 1$, $\mathcal{Y}_j^{k+1}$, after a set of biases, $\mathcal{B}_j^k$, for each feature, j, in each layer, k, are added and the activation function, φ is applied:

$$\mathcal{Y}_j^{k+1} = \varphi(\mathcal{B}_j^k + \sum_{m=1}^{F^k} (\mathcal{K}_{m,j}^k \otimes \mathcal{Y}_m^k)) \tag{2.3}$$

Here a Rectified Linear Unit (ReLU) activation function was used for φ,

$$\varphi = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{otherwise.} \end{cases} \tag{2.4}$$

Thus, the elements of the output of layer, $k+1$, for the j^thfeature map, $\mathcal{Y}_j^{k+1}$, at position (r, s) is:

$$(\mathcal{Y}_j^{k+1})_{r,s} = \varphi((\mathcal{B}_j^k)_{r,s} + \sum_{m=1}^{F^k} (\mathcal{K}_{m,j}^k \otimes \mathcal{Y}_m^k)_{r,s}) \tag{2.5}$$

The pooling layer helps to reduce the size of the output (*i.e.* it may downsample the input to that layer). Other function options could be used *e.g.* maximum, average, *etc.*. We followed the common practice to use the maximum value on a local rectangular region (neighborhood). Let us assume $p \times p$ is size of the pooling window and sp is stride of the pooling, then $P(..)$ is a pooling function which acts on $\mathcal{Y}_j^{k+1}$, and pooling window of dimension is $((H - h)/sp + 1) \times ((D - d)/sp + 1)$. So that the output of a max-pooling function is:

$$P(\mathcal{Y}_j^{k+1})_{m,n} = max(\mathcal{Y}_j^{k+1})_{m,n} \tag{2.6}$$

The FC layer receives data from the previous convolutional and pooling layer. This layer is a classifier layer, the weights received in this layer in each iteration

is fed back to adjust the weights in all previous layers. Generally, $softmax$ is the transformation function used in the feed back:

$$\mathcal{Y} = softmax(\mathcal{I} \otimes \mathcal{W} + \mathcal{B}) \tag{2.7}$$

Finally, we obtain as output, the predictions $\mathcal{Y}$, where $\mathbb{I}.$ is the set of original images, and set, $\mathbb{W}$ of all weights, and set of biases $\mathbb{B}$.

2.3 Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting (XGBoost) is a machine learning model for classification and regression problems designed by Chen and Guestrin [11] and shown first in KDD Cup 2015. It is effective for machine learning and data mining challenges and has been used extensively by data scientists.

A brief summary of Chen and Guestrin's work follows, but readers familiar with their work may skip to Section 3.1. XGBoost is a highly scalable end-to-end tree boosting system. Its architecture consists of a tree, which is an ensemble of K classification and regression trees (CARTs). If x_i is the vector training set and y_i is the corresponding class labels of x_i . The output prediction, $\hat{y}_i$ is the sum of the prediction scores of K trees:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i) \tag{2.8}$$

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 scoring function. The output prediction, $\hat{y}_i$ were compared between the target based on a loss function, $l(\hat{y}_i, y_i)$, with the addition of an Ω term to the model for prevent overfitting, calculated:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{2.9}$$

where $\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} {w_j}^2$, with constants, γ and λ, which control the regularization degree, T is the set of leaves in the tree with the weight of each leaf denoted w. In addition, we can improve the efficiency in Equation (2.9) by expanding the loss function with a first or second order Taylor expansion. Therefore, at step t, we can calculate :

$$\begin{aligned}
\tilde{\mathcal{L}}^{(t)} &\simeq \sum_{i=1}^{n} [g_i f_i(x_i) + \frac{1}{2} h_i {f_i}^2(x_i)] + \Omega(f_t) \\
&= \sum_{i=1}^{n} [g_i f_i(x_i) + \frac{1}{2} h_i {f_i}^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} {w_j}^2 \\
&= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) {w_j}^2] + \gamma T
\end{aligned} \tag{2.10}$$

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

$$g_i = \frac{\partial l(\hat{y_i}^{(t-1)}, y_i)}{\partial \hat{y}_i^{(t-1)}} \tag{2.11}$$

$$h_i = \frac{\partial^2 l(\hat{y_i}^{(t-1)}, y_i)}{\partial (\hat{y}_i^{(t-1)})^2} \tag{2.12}$$

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

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \tag{2.13}$$

when calculating the weight in Equation (2.13), the final equation is the quality of a tree structure, q, computed as:

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T, \tag{2.14}$$

The XGBoost model calculates scores in each node in the tree structure for split decisions. For effective predictive, we realize the loss after the split and want to reduce it. If $I = I_L \cup I_R$, where I_L and I_R are the left and right nodes after the split, we compute:

$$\mathcal{L}_{split} = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} + \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma. \tag{2.15}$$

3 Deep Single-Pass Learning (DSPL)

3.1 Structure and Parameter Computation

In this section, we will explain the parameters and the key equations of a Deep Single-Pass Learning (DSPL) model. DSPL has two main parts, the ability to learn the features of the training set and efficiently predicting the class labels of the test set. The first part, before entering the prediction process, the training set will be learnt by deep feature learning, which consists of several convolutional layers, $\mathcal{Y}$, and the pooling layer, P, for reducing feature size. Generally, feature learning has a hierarchical structure, see a diagram of our model in Figure 2 showing the

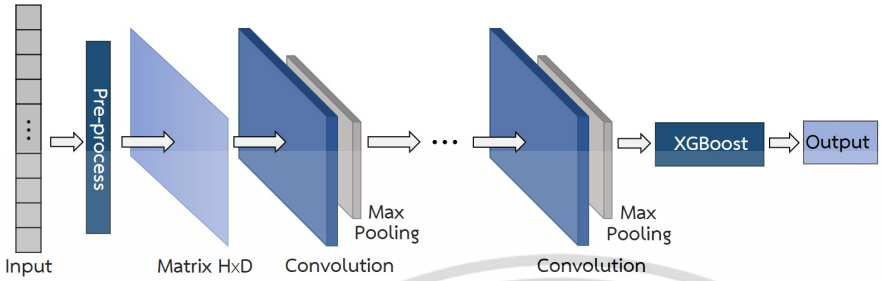


Figure 2: The structure overview of the DSPL.

structre. The output of the convolution operation, $\mathcal{Y}^{\|}$, in each layer k, is computed from the output of layer $k-1$. Therefore, the feature learning of the training data set follows this chain:

$$P(\mathcal{Y}_j^1) \to P(\mathcal{Y}_j^2) \to \cdots \to P(\mathcal{Y}_j^k) \to P(\mathcal{Y}_j^{k+1}) \to \mathcal{Z} \tag{3.1}$$

when $\mathcal{Z}$ is tensor output of the feature learning process. In each layer, the output of the previous layer is set to the input of the current layer. In the case where $L = k + 1$, so the input, of $\mathcal{Y}_j^{k+1}$, in the $k + 1$ layer for the j^{th} feature map, is derived from the output of the previous layer, $\mathcal{Y}_j^k$.

An additive bias is applied to each input So the input, $\mathcal{Y}_i^k$, of the k^{th} layer for the j^{th} feature map, is derived from the output of the previous layer, $\mathcal{Y}_j^{k-1}$, as set out previously in Equation 2.5 (repeated here)

$$\mathcal{Y}_j^{k+1} = \varphi(\mathcal{B}_j^k + \sum_{m=1}^{F^k} (\mathcal{K}_{m,j}^k \otimes \mathcal{Y}_m^k))$$

Remark 3.1. *We used the capabilities of the convolutional layers from CNNs, the input must be in a tensor format, see Definition 2.1. Therefore, the training set (in Section 2.1) will be converted to the standard input format of convolutional operation as the following Definition 3.2.*

Definition 3.2. *Training Set: Let is the set of training sets, when $[x_j^i]_{n1}$ 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, a general tensor, $\mathcal{T}$ of m-order can be created as follows:*

$$\mathcal{T} = [\overline{a}_{j_1 j_2 \cdots j_m}^{i_1 i_2 \cdots i_m}]_{n'n'} \tag{3.2}$$

The output of convolution operation, $\mathcal{Y}$, caused from the tensor, $\mathcal{T}$, applied to the $h \times d$ kernel, $\mathcal{K}$. The kernel, $\mathcal{K}$, is slid through tensor, $\mathcal{T}$, by a stride, sk, and zero padding value.

In the second part, we use the decision rules in XGBoost to predict from the training set learned through from the first part. We addition term of features learning $\mathcal{Z}$ (see in Equation 3.1) to predict classes. Therefore, the final prediction is the sum of the prediction scores for each tree E, as follows:

$$\hat{y}_i = \phi(\mathcal{Z}_i) = \sum_{k=1}^{K} f_k(\mathcal{Z}_i), \quad f_k \in F, \tag{3.3}$$

For Equation (3.3) uses an ensemble of number of E classification and regression trees (CARTs), when $F = f(\mathcal{Z}) = w_{q(\mathcal{Y})}(q : \mathbb{R}^n \to T, w \in \mathbb{R}^T)$ is is the set of all K scores for all CARTs, and f_k is the leaf score for the k^{th} tree corresponds to both the independent tree structure q and leaf weights w, where T is the number of leaves in the tree.

Remark 3.3. *Since Equation (3.3) supports the training set $\mathcal{Z}$, data type is vector, but, the data type of $\mathcal{Z}$, after through the process of features learning is tensor. Therefore, it is necessary to convert the type of $\mathcal{Z}$ to vector see in Definition 3.4, which describes the standard and convert the input data.*

Definition 3.4. *The training set $\mathcal{Z}$ is tensor size as $[\bar{a}^{i_1 i_2 \ldots i_r}_{j_1 j_2 \ldots j_r}]_{n' n'}$, which r is the number of filters, and n' exhibit the number of rows and columns. Therefore, $\mathcal{Z}$ will be converted in the form of vector $\mathcal{V}$ of feature in $\mathbb{R}^b$ when $b = n' \times n' \times r$, and will be represented in the Equation (3.3) is:*

$$\hat{y}_i = \phi(\mathcal{V}_i) = \sum_{k=1}^{K} f_k(\mathcal{V}_i), \quad f_k \in F, \tag{3.4}$$

when $F = f(\mathcal{V}) = w_{q(\mathcal{V})}(q : \mathbb{R}^n \to T, w \in \mathbb{R}^T)$. The quality of a tree structure can be scored from Equation (2.14), we can set the number of trees, thus the size of the structure affects performance.

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

when $I_j = \{i | q(\mathcal{V}_i) = j\}$ denotes the instance set of leaf t and $g_i = \frac{\partial l(\hat{y_i}^{(t-1)}, y_i)}{\partial \hat{y_i}^{(t-1)}}$, $h_i = \frac{\partial^2 l(\hat{y_i}^{(t-1)}, y_i)}{\partial \hat{y_i}^{(t-1)}}$ are first and second order gradient statistics of the loss function, γ and λ are constants to control the regularization degree. In addition, one of the key tasks is splitting into the best set of segments: we use the gain of the split in Equation (2.15). 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:

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

In our model, 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.

3.2 DSPL Algorithm

Given a training set, $\mathbb{I} = \{\mathcal{I}_i, y_i | 1 \leq i \leq M\}$, consisting of M images, with labels, $y_i \in \mathbb{R}$, the label of image $\mathbf{x}_i$, when $\mathbf{x}_i \in \{\mathcal{T} \mid \mathcal{T} = [\overline{a}^{i_1 i_2 \ldots i_m}_{j_1 j_2 \ldots j_m}]_{n' n'}\}$ is an m-order tensor with n features ($\mathbb{R}^n$ or $\mathbb{R}^{n' \times n'}$, where $n' = \sqrt{n}$). The learning algorithm for the DSPL can be summarized as follows:

---

1. Initialize the training set, $\mathbb{I} = \{\mathcal{I}_i, y_i | 1 \leq i \leq M\}$
2. Set the parameters of the convolutions for learning features
   (a) number of convolutional layers, L
   (b) for each layer, set the filter sizes, $\mathcal{K}^k$, and
   (c) kernel strides, sk^k
3. For each layer, k, in $1..L$:
   calculate the convolutions to generate the $\mathcal{Y}_j^{k+1}$ for layer, $k+1$, and the j^{th} feature map:
$$\mathcal{Z} = \mathcal{Y}_j^{k+1} = \varphi(\mathcal{B}_j^k + \sum_{m=1}^{F^k} (\mathcal{K}_{m,j}^k \otimes \mathcal{Y}_m^k)),$$
4. Reshape $\mathcal{Z}$ to a vector of length $(n' \times n' \times r^{k+1})$ - $\mathcal{V}$ see in Definition 3.4.
5. Initialize parameters for the prediction step:
   (a) total number of trees, K
   (b) regularization parameters, γ and λ,
   (c) column subsampling parameter,
   (d) maximum tree depth, t and
   (e) learning rate
6. Determine the output class labels:
$$\hat{y}_i = \phi(\mathcal{V}_i) = \sum_{k=1}^{K} f_k(\mathcal{V}_i), \quad f_k \in F,$$
   where $F = f(\mathcal{V}) = w_{q(\mathcal{V})}(q : \mathbb{R}^n \to T, w \in \mathbb{R}^T)$
7. Calculate the optimal leaf weight for the best tree structure
$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda},$$

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

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

where T is the number of leaves in the tree

9. Calculate the best splitting points

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

10. Terminate

Our DSPL algorithm has time complexity:

$$\mathcal{O}(Lc^2mnpq) + \mathcal{O}(x(Kt + \log B))$$

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

4 Model Evaluations

4.1 Experimental Setup

We chose the experimental setup carefully to balance the resources used with good performance, guided by the time complexity of our model - see Section 3.2. Initial parameters were to: number of the convolutional layers or number of maps, $L = 3$, and the output depth of the convolutional layer, $r = 32$. The filter size was set as: $\mathcal{K} = 3 \times 3$: if $\mathcal{K}$ is small, accuracy will be high, but the convolution operation will be repeated many times and the computation will be slow. On the other hand, if $\mathcal{K}$ is too large, accuracy willy suffer. Additionally, choosing a small stride of the filter, $sk = 1$, enables small features to be recognized. The parameters set of our model in 10 layers - see in Table 1.

Our model was compared with with other models which has shown good performance: Convolutional Neural Networks (CNNs) [8], Extreme Gradient Boosting (XGBoost) [11], Logistic Regression (LR) [13], Extra Trees Classifier (ETC) [14], Gradient Boosting Classifier (GBC) [15], Random Forest Classifier (RFC) [16], Gaussian Naive Bayes (GNB) [17], Decision Tree Classifier (DTC) [18], Multilayer Perceptron (MLP) [19] and the Support Vector Classification (SVC) [20] - see Table 3.

For the CNNs model, parameters were set to the same as those for our model, DSPL. The number of neurons in the FC class was 2^n, where $n = 8, 9$ and 10. The

Table 1: Structure of each layer for the DSPL model

Layer	Type	Input	Kernel	Stride	Output
L1	Input	$[n', n', m]$	na	na	$[n', n', m]$
L2	Convolutional (Conv1)	$[n', n', m]$	$h \times d$	1	$[n', n', r^{(L-2)}]$
L3	Max-Pooling (Pool1)	Conv1	$p \times p$	1	Pool1
L4	Convolutional (Conv2)	Pool1	$h \times d$	1	$[n', n', r^{(L-1)}]$
L5	Max-Pooling (Pool2)	Conv2	$p \times p$	1	Pool2
L6	Convolutional (Conv3)	Pool2	$h \times d$	1	$[n', n', r^{(L)}]$
L7	Max-Pooling (Pool3)	Conv3	$p \times p$	1	Pool3
L8	Reshape	Pool3	na	na	$n' \times n' \times r^{(L)}$
L9	Class Prediction	$n' \times n' \times r^{(L)}$	na	na	No. of Classes
L10	Output	na	na	na	No. of Classes

Notes: na = not applicable.

XGBoost model was set similarly, with parameters, matching those in the class prediction layer of our model, to fairly evaluate performance.

The MLP family model was evaluated with different activation functions: linear (MLP1), sigmoid (MLP2), tanh (MLP3), and ReLU (MLP4). The numbers of neurons were 2^n, where $n = 8, 9$ and 10 and the learning rate was set to 0.001. Similarly, the SVC family model was 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 is summarized in Table 2.

Table 2: Properties of models used in performance comparison

Model	Parameters details	Ref.	Time Complexity
Deep Single-Pass Learning (DSPL)	No. of Conv. layer $L = 3$	-	$\mathcal{O}(Lc^2mnpq) + \mathcal{O}(x(Kt + \log B))$
Convolutional Neural Networks (CNNs)	No. of Conv. layer $L = 3$	[8]	$\mathcal{O}(Lc^2mnpq) + \mathcal{O}(mnhge)$
eXtreme Gradient Boosting (XGBoost)	Max_depth = 3	[11]	$\mathcal{O}(x(Kt + \log B))$
Logistic Regression (LR)	Tol = 0.0001, C=1.0	[13]	$\mathcal{O}(mn)$ to $\mathcal{O}(mn^2)$
Extra Trees Classifier (ETC)	Criterion = Gini, MinSS = 2	[14]	$\mathcal{O}(nmK)$
Gradient Boosting Classifier (GBC)	Max_depth = 3, MinSS = 2	[15]	$\mathcal{O}(mnK)$
Random Forest Classifier (RFC)	Criterion = Gini, MinSS = 2	[16]	$\mathcal{O}(Ktmn \log n)$
Gaussian Naive Bayes (GNB)	Var_Smoothing = 1e-09	[17]	$\mathcal{O}(Mg)$
Decision Tree Classifier (DTC)	Criterion = Gini, MinSS = 2	[18]	$\mathcal{O}(tm \log n)$
Multi-layer Perceptron Classifier (MLP)			
MLP1	Activation = Linear	[19]	$\mathcal{O}(mnhge)$
MLP2	Activation = Sigmoid	[19]	$\mathcal{O}(mnhge)$
MLP3	Activation = tanh	[19]	$\mathcal{O}(mnhge)$
MLP4	Activation = ReLU	[19]	$\mathcal{O}(mnhge)$
Support Vector Classification (SVC)			
SVC1	Kernel = RBF, $C = 1.0$	[20]	$\mathcal{O}(m^3)$
SVC2	Kernel = Linear , $C = 1.0$	[20]	$\mathcal{O}(m^3)$
SVC3	Kernel = Poly, $C = 1.0$	[20]	$\mathcal{O}(m^3)$
SVC4	Kernel = Sigmoid, $C = 1.0$	[20]	$\mathcal{O}(m^3)$

Table 2 also shows the time complexity of the models. We assumed that: L is the number of layers, c is the various of input or output channels, the data matrix has size $m \times n$, the filter has size $p \times q$, $x = \|x\|$ is the number of non-missing entries, K is the number of trees, t is the tree depth and B is the block length,

Table 3: Properties of models used in performance comparison.

Model	Worst (%)	Best (%)	Improvement
Deep Single-Pass Learning (DSPL)	99.95	100.0	-
Convolutional Neural Networks (CNNs)	98.72	98.88	1.2%
Extreme Gradient Boosting (XGBoost)	97.12	97.76	2.8%
Logistic Regression (LR)	95.78	97.12	4.2%
Extra Trees Classifier (ETC)	96.21	97.22	3.7%
Gradient Boosting Classifier (GBC)	97.01	97.70	2.9%
Random Forest Classifier (RFC)	95.57	96.10	4.4%
Gaussian Naive Bayes (GNB)	75.19	84.09	24.8%
Decision Tree Classifier (DTC)	88.63	89.80	11.3%
Multi-layer Perceptron Classifier (MLP)			
MLP1	95.41	96.58	4.5%
MLP2	97.92	98.24	2.0%
MLP3	97.49	98.18	2.5%
MLP4	97.17	98.34	2.8%
Support Vector Classification (SVC)			
SVC1	60.01	65.67	39.9%
SVC2	97.65	97.97	2.3%
SVC3	98.72	98.99	1.2%
SVC4	08.81	09.45	91.1%

Note: Improvement *vs* DSPL are shown in the 'Improvement' column.

m is the number of training sets, n is the number of features or dimensions, h is number of hidden neurons, g is the number of classes and e is the number of epochs.

4.2 Experimental Results

DSPL was evaluated on the visual recognition of handwritten digits in the [7] data set - see Section 2.1- a multiclass classification problems and compared with the other 16 models in Table 2. 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 the testing set. This was repeated three times, with each of the three subsets used exactly once as the testing set. Overall, our model efficiently provided worst case accuracy of 99.95% - see Table 3.

However, evaluating the performance for visual recognition of handwritten digits. Our model provided higher accuracy than all models, with accuracy higher than worst vs worst here .. CNNs (1.2%), XGBoost (2.8%), LR (4.2%), ETC (3.7%), GBC (2.9%), RFC (4.4%), GNB (24.8%), and DTC (11.3%). In addition, compares our model with the MLP family and the SVC family, which shows that our model provides higher accuracy than the MLP family (2.0% to 4.5%) and the SVC family (1.2% to 91.14%).

5 Conclusion

We designed a deep single-pass learning model (DSPL): it used only one epoch to learn to recognize handwritten digits. In addition, it can learn data set without needing to repeatedly adjust the weight. The number of the convolutional layer can be increased or decreased with some conditions, and the data set in the features learning process will be passed into XGBoost which is the last layer of the model. A DSPL trained model was very effective, on tests with 5620 images, its worst case accuracy was 99.95% (and sometimes reached 100%). DSPL's worst case was 1.2% better than CNNs, the next best performer, our tests. The very low number of misclassifications, in the worst case, showed that our algorithm would be effective in practice.

Acknowledgement : We thank Prof. John Morris of the KMITL Research and Innovation Services (KRIS) for editing the final manuscript.

References

[1] A. Rehman, S. Naz, M. I. Razzak, I. A. Hameed, Automatic Visual Features for Writer Identification: A Deep Learning Approach, IEEE Access, 2019, 17149 - 17157.

[2] C. Adak, B. B. Chaudhuri, M. Blumenstein, An Empirical Study on Writer Identification and Verification From Intra-Variable Individual Handwriting, IEEE Access, 2019, 24738 - 24758.

[3] V. Venugopal, S. Sundaram, Online Writer Identification With Sparse Coding-Based Descriptors, IEEE Transactions on Information Forensics and Security, 2018, 2538 - 2552.

[4] D. Impedovo, G. Pirlo, Dynamic Handwriting Analysis for the Assessment of Neurodegenerative Diseases: A Pattern Recognition Perspective, IEEE Reviews in Biomedical Engineering, 2019, 209 - 220.

[5] D. Impedovo, Velocity-Based Signal Features for the Assessment of Parkinsonian Handwriting, IEEE Signal Processing Letters, 2019, 632 - 636.

[6] C. Kahindo, M. A. El-Yacoubi, S. G. Salicetti, A. S. Rigaud, V. C. Lacroix, Characterizing Early-Stage Alzheimer Through Spatiotemporal Dynamics of Handwriting, IEEE Signal Processing Letters, 2018, 1136 - 1140.

[7] D. Dua, C. Graff, UCI Machine Learning Repository, Irvine, CA: University of California, School of Information and Computer Science, https://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/.

[8] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE, 1998, 2278-2324.

[9] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[10] D. Stutz, Understanding Convolutional Neural Networks, Seminar Report, Fakultät für Mathematik, Informatik und Naturwissenschaften Lehr-und Forschungsgebiet Informatik VIII Computer Vision, 2014, 1-23.

[11] C. Tianqi, G. Carlos, XGBoost: A Scalable Tree Boosting System, Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, 2016, 785-794.

[12] M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C. L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469, 1994, 1-63.

[13] R. E. Fan, K.W. Chang, C. J. Hsieh, X. R. Wang, C. J. Lin, LIBLINEAR: A Library for Large Linear Classification, Journal of Machine Learning Research, 2008, 1871-1874.

[14] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Machine Learning, 2006, 3-42.

[15] J. Friedman, Greedy Function Approximation: A Gradient Boosting Machine, The Annals of Statistics, 2000.

[16] L. Breiman, Random Forests, Machine Learning, 2001, 5-32.

[17] T.F. Chan, G. H. Golub, R. J. LeVeque, Updating Formulae and a Pairwise Algorithm for Computing Sample Variances, COMPSTAT 1982 5th Symposium held at Toulouse 1982, Physica-Verlag HD, 1982, 30-41.

[18] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, Classification and Regression Trees, Wadsworth and Brooks, Monterey, CA, 1984.

[19] , G. E. Hinton, Connectionist learning procedures, Artificial Intelligence, 185-234, 1989.

[20] C. C. Chang, C. J Lin, LIBSVM: A Library for Support Vector Machines, ACM Transactions on Intelligent Systems and Technology, 2011, 1-27.

(Received xx xx xx)
(Accepted xx xx xx)

Appendix B

2^{nd} journal

Published by Communications in Mathematics and Applications

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

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 [1,*]

[1]Advanced Artificial Intelligence (AAI) Research Laboratory, Department of Computer Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand (tsetthanun@gmail.com (S.T.); saichon.ja@kmitl.ac.th (S.J.))
[2]Department of Mathematics, Anand International College of Engineering, Jaipur 303012, India (goyal.praveen2011@gmail.com)
[*]**Corresponding author:** saichon.ja@kmitl.ac.th, tsetthanun@gmail.com

Preprint submitted to RGN Publications on Day/Month/Year

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.

2010 AMS Classification: 68T10

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

Article type: Research article

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 [1, 2] and various industrial areas [3]. Human Activity Recognition (HAR) is a key research topic for human behavior classification problems [4, 5, 6], 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 [7], transportation [8], manufacturing systems [9], consumer devices and services [10], and including previous research on recognising human activities in smart homes [11] etc.

Convolutional Neural Network (CNN) [12] is one of the DL techniques. In addition, to good ability to learn data, it can handle complex HAR tasks well. Wang *et al.*[13] have surveyed recent advances of deep learning in activity recognition. CNN has been used for sensor-based activity recognition [14, 15, 16, 17], it has expanded the scope in and state of the art and for many research challenges. Nweke *et al.*[18] 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 [19, 20, 21]. 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 [22]. Our model combines the performance of a CNN and Extreme Gradient Boosting (XGBoost) [23], 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 [7, 8, 9, 10, 23].

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 [24, 25, 26]), Extra Trees Classifier (ETC [27]), Gradient Boosting Classifier (GBC [28, 29, 30]), Random Forest Classifier (RFC [31]), Gaussian Naive Bayes (GNB [32]), Decision Tree Classification (DTC [33]), Multilayer Perceptron (MLP [34]) and Support Vector Classification (SVC [35]). In addition, we evaluated the performance of other research CNNs [14, 15, 16, 17], previously reported for HRA tasks, listed in Wang *et al.*'s survey [9].

DOPL differs from previous work of Jabri *et al.*[36] and Lioutas *et al.*[37], in that Pre-training weights are commonly used for Visual Question Answering (VQA) tasks, using weights acquired from a library or public framework [38] (*e.g.* Theano [39], Lasagne [40], ImageNet [41], word2vec [42], AlexNet [43], GoogLeNet [44], and ResNet [45] *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.*[22] 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 classifid 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 [46] – UCI smartphone: It is commonly used in research for high-level activity understanding [13]: 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* $\mathbb{I} = \{x_i, y_i | 1 \leq i \leq M\}$*, be the training set of* M *vectors, each vector* x_i *consists of* N *features,* $x_i = \{p_j | 1 \leq j \leq N\}$*, and* y_i *a label,* $\mathbf{x}_i$ *in* $\mathbb{R}$*.*

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

Table 1: Activity Labels characteristic

Class ID	Activity Labels	Description	Number of Instances
1	WALKING	walking	1722
2	WALKING_UPSTAIRS	walking upstairs	1544
3	WALKING_DOWNSTAIRS	walking downstairs	1407
4	SITTING	sitting	1801
5	STANDING	standing	1979
6	LYING_DOWN	lying	1958
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

2.2 Convolutional Neural Network

A Convolutional Neural Network (CNN) [12] is described in detail by Goodfellow et al [47]. 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, $\mathcal{I}$, 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:*

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

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

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:

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

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 [23]. 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:

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

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:

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

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.

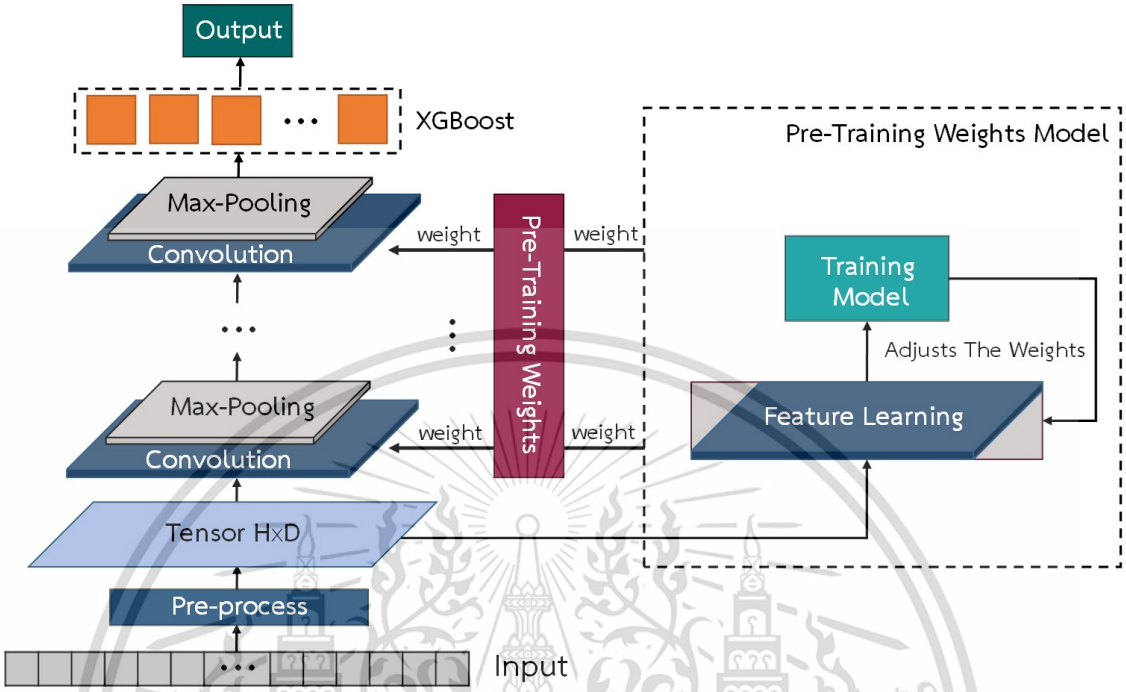


Figure 1: Overview of DOPL: Training and prediction (left side), and Pre-Trained weights (right side).

- 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. 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.

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) [23]. We evaluate the quality of the tree by calculating the points (as in Equation 9):

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

   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

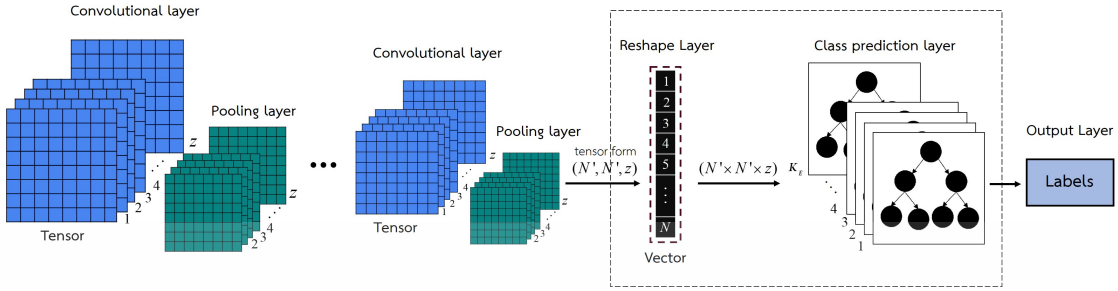


Figure 2: The reshape layer re-adjusts the shape of the data (tensor) to the vector format

Section 2.3.

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

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 [36, 37]. 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 [39], Lasagne [40], ImageNet [41], word2vec [42], AlexNet [43], GoogLeNet [44], ResNet [45] *etc.*However, pre-training was not mentioned in HAR tasks, investigated by Wang *et al.*[13] and Nweke *et al.*[18]. 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 [14, 16, 17]. 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 $\mathcal{Q}$ - Definition 3.4.

Definition 3.2 *Input: Let* $\mathbb{I} = \{x_i, y_i | 1 \leq i \leq M\}$, *be the training set of* M *vector, each vector* x_i *consists of* N *features,* $x_i = \{p_j | 1 \leq j \leq N\}$, *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 zeroes,* $\mathcal{Q}$, *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 zeroes, 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:*

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

3.4 DOPL Algorithm

Let $\mathbb{I} = \{x_i, y_i | 1 \leq i \leq M\}$, be the training set of M vector, each vector x_i consists of N feature $x_i = \{p_j | 1 \leq j \leq N\}$, 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, $\mathbb{I} = \{x_i, y_i | 1 \leq i \leq M\}$
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

$$\mathbb{I}_{new} = \{(\mathcal{Z}_i, y_i) | 1 \leq i \leq M\}.$$

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

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

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

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

13. Calculate the best splitting points

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

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 [48], *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 [49]. 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.

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.

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.

Our model was compared with other models *i.e.* , Convolutional Neural Network (CNN) [12, 14, 15, 16, 17], Extreme Gradient Boosting (XGBoost) [23], Logistic Regression (LR) [24, 25, 26], Extra Trees Classifier (ETC) [27], Gradient Boosting Classifier (GBC) [28, 29, 30], Random Forest Classifier (RFC) [31], Gaussian Naive Bayes (GNB) [32], Decision Tree Classifier (DTC) [33], Multilayer Perceptron (MLP) [34] and the Support Vector Classification (SVC) [35] - 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

Table 4: Parameters in each layer of the DOPL model

Layer Name	Descriptions	Parameters
L1	Input: raw data size	10929×561
	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

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.

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 $L = 3$	[12]	$\mathcal{O}(Ld^2mnpq) + \mathcal{O}(MNhce)$
Extreme Gradient Boosting (XGBoost)	Max_depth = 3	[23]	$\mathcal{O}(r(Kt + \log B))$
Logistic Regression (LR)	Tol = 0.0001, C=1.0	[24, 25, 26]	$\mathcal{O}(MN)$ to $\mathcal{O}(MN^2)$
Extra Trees Classifier (ETC)	Criterion = Gini, MinSS = 2	[27]	$\mathcal{O}(MNK)$
Gradient Boosting Classifier (GBC)	Max_depth = 3, MinSS = 2	[28, 29, 30]	$\mathcal{O}(MNK)$
Random Forest Classifier (RFC)	Criterion = Gini, MinSS = 2	[31]	$\mathcal{O}(KtMN \log N)$
Gaussian Naive Bayes (GNB)	Var_Smoothing = 1e-09	[32]	$\mathcal{O}(Mc)$
Decision Tree Classifier (DTC)	Criterion = Gini, MinSS = 2	[33]	$\mathcal{O}(tM \log N)$
Multi-layer Perceptron Classifier (MLP)			
MLP1	Activation = Linear	[34]	$\mathcal{O}(MNhce)$
MLP2	Activation = Sigmoid	[34]	$\mathcal{O}(MNhce)$
MLP3	Activation = tanh	[34]	$\mathcal{O}(MNhce)$
MLP4	Activation = ReLU	[34]	$\mathcal{O}(MNhce)$
Support Vector Classification (SVC)			
SVC1	Kernel = RBF, $C = 1.0$	[35]	$\mathcal{O}(M^3)$
SVC2	Kernel = Linear , $C = 1.0$	[35]	$\mathcal{O}(M^3)$
SVC3	Kernel = Poly, $C = 1.0$	[35]	$\mathcal{O}(M^3)$
SVC4	Kernel = Sigmoid, $C = 1.0$	[35]	$\mathcal{O}(M^3)$

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

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 [22] †	**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.

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.

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

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

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 [14] *	**95.18**	-	-	-	4.0%
CNN3 [15] *	**94.79**	-	-	-	4.4%
CNN4 [16] *	**90.00**	-	-	-	9.2%
CNN5 [17] *	**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.

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.

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 author declares that he has no competing interests.

Authors' Contributions

The author wrote, read and approved the final manuscript.

References

[1] P. Zham, D. K. Kumar, P. Dabnichki, S. Poosapadi Arjunan, and S. Raghav, "Distinguishing different stages of parkinson's disease using composite index of speed and pen-pressure of sketching a spiral," *Frontiers in Neurology*, vol. 8, p. 435, 2017.

[2] 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*, vol. 66, pp. 365–372, Feb 2019.

[3] H. Liu and L. Wang, "Gesture recognition for human-robot collaboration: A review," *International Journal of Industrial Ergonomics*, vol. 68, pp. 355–367, 2018.

[4] E. Kim, S. Helal, and D. Cook, "Human activity recognition and pattern discovery," *IEEE Pervasive Computing*, vol. 9, pp. 48–53, Jan 2010.

[5] 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*, vol. 65, pp. 243–252, May 2019.

[6] Z. Zhang, S. Shan, Y. Fang, and L. Shao, "Deep learning for pattern recognition," *Pattern Recognition Letters*, vol. 119, pp. 1–2, 2019. Deep Learning for Pattern Recognition.

[7] M. Wainberg, D. Merico, A. Delong, and B. J. Frey, "Deep learning in biomedicine," *Nature Biotechnology*, vol. 36, pp. 829–838, 2018.

[8] H. Nguyen, L. Kieu, T. Wen, and C. Cai, "Deep learning methods in transportation domain: a review," *IET Intelligent Transport Systems*, vol. 12, no. 9, pp. 998–1004, 2018.

[9] J. Wang, Y. Ma, L. Zhang, R. X. Gao, and D. Wu, "Deep learning for smart manufacturing: Methods and applications," *Journal of Manufacturing Systems*, vol. 48, pp. 144–156, 2018. Special Issue on Smart Manufacturing.

[10] 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*, vol. 6, pp. 48–56, April 2017.

[11] D. Liciotti, M. Bernardini, L. Romeo, and E. Frontoni, "A sequential deep learning application for recognising human activities in smart homes," *Neurocomputing*, 2019.

[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[13] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognition Letters*, vol. 119, pp. 3–11, 2019. Deep Learning for Pattern Recognition.

[14] 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, (New York, NY, USA), pp. 1307–1310, ACM, 2015.

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

[16] 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), p. 858–860, 2015.

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

[18] 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*, vol. 105, pp. 233–261, 2018.

[19] S. Lee, T. Chen, L. Yu, and C. Lai, "Image classification based on the boost convolutional neural network," *IEEE Access*, vol. 6, pp. 12755–12768, 2018.

[20] P. He, X. Jiang, T. Sun, and H. Li, "Computer graphics identification combining convolutional and recurrent neural networks," *IEEE Signal Processing Letters*, vol. 25, pp. 1369–1373, Sep. 2018.

[21] G. Liang, H. Hong, W. Xie, and L. Zheng, "Combining convolutional neural network with recursive neural network for blood cell image classification," *IEEE Access*, vol. 6, pp. 36188–36197, 2018.

[22] J.-L. Reyes-Ortiz, L. Oneto, A. Samà, X. Parra, and D. Anguita, "Transition-aware human activity recognition using smartphones," *Neurocomput.*, vol. 171, pp. 754–767, Jan. 2016.

[23] 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, (New York, NY, USA), pp. 785–794, ACM, 2016.

[24] 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*, vol. 9, pp. 1871–1874, jun 2008.

[25] H.-F. Yu, F.-L. Huang, and C.-J. Lin, "Dual coordinate descent methods for logistic regression and maximum entropy models," *Machine Learning*, vol. 85, pp. 41–75, Oct 2011.

[26] 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*, vol. abs/1407.0202, pp. 1–1, 2014.

[27] P. Geurts and L. Ernst, Damien & Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, pp. 3–42, Apr 2006.

[28] J. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, 11 2000.

[29] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367 – 378, 2002. Nonlinear Methods and Data Mining.

[30] T. Hastie, R. Tibshirani, and J. Friedman, "The elements of statistical learning," *Springer*, vol. 1, 01 2009.

[31] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, Oct 2001.

[32] 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*, (Heidelberg), pp. 30–41, Physica-Verlag HD, 1982.

[33] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[34] G. E. Hinton, "Connectionist learning procedures," *Artificial Intelligence*, vol. 40, no. 1, pp. 185–234, 1989.

[35] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 1–27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

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

[37] V. Lioutas, N. Passalis, and A. Tefas, "Explicit ensemble attention learning for improving visual question answering," *Pattern Recognition Letters*, vol. 111, pp. 51–57, 2018.

[38] 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*, vol. 163, pp. 21–40. Language in Vision.

[39] T. D. Team, "Theano: A python framework for fast computation of mathematical expressions," *CoRR*, vol. abs/1605.02688, pp. 1–19, 2016.

[40] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, and et al., "Lasagne: First release.," aug 2015.

[41] S. Gross and M. Wilber, "Training and investigating residual nets," 2016.

[42] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[43] 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, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[44] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2015.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognitions," 2016.

[46] D. Dua and C. Graff, "UCI machine learning repository," 2015. `https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones`.

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

[48] 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," 2015. Software available from tensorflow.org.

[49] 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*, vol. 12, pp. 2825–2830, 2011.

Author Biography

Name	Acting Sub Lt. Setthanun Thongsuwan
Date of Birth	10^{th} February 1984
Address	34/4, Saensuk, Muang, Chonburi, 20130, Thailand.
Education	(2013) Master of Science in Computer Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand. (2008) Bachelor of Science in Computer Science, Burapha University, Chonburi, Thailand.

Academic Publications

1. S. Thongsuwan and S. Jaiyen, "A Deep Single-Pass Learning for Recognition of Handwritten Digits", Thai Journal of Mathematics , 2019.
2. S. Thongsuwan, P. Agarwal, and S. Jaiyen, "A Deep One-Pass Learning based on Pre-Training Weights for Smartphone-Based Recognition of Human Activities and Postural Transitions", Communications in Mathematics and Applications, 2019.