Thai Journal of **Math**ematics
Volume 20 Number 1 (2022)
Pages 293–304

http://thaijmath.in.cmu.ac.th

ISSN 1686-0209

A Deep Single-Pass Learning for Recognition of Handwritten Digits

Setthanun Thongsuwan and Saichon Jaiyen*

Advanced Artificial Intelligence (AAI) Research Laboratory, Department of Computer Science, King Mongkuts Institute of Technology Ladkrabang, Bangkok 10520, Thailand
e-mail : tsetthanun@gmail.com (S. Thongsuwan); saichon.jai@sit.kmutt.ac.th (S. Jaiyen)

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.

MSC: 68T10; 68T30; 62H30; 62H35
Keywords: deep single-pass learning; handwriting recognition; pattern recognition; convolutional neural networks; xgboost

Submission date: 29.05.2019 / Acceptance date: 07.07.2019

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 [1–3]. It also includes the analysis of personal behavior: these unique features can be used to analyze various diseases, *e.g.* [4–6]. 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

*Corresponding author.

Published by The Mathematical Association of Thailand.

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.

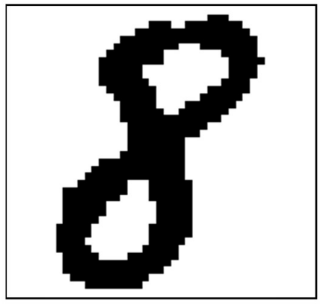
We describe a new deep learning model - Deep Single-Pass Learning (DSPL) - to learn a data set in a single pass. Our model has three convolutional layers, where XGBoost [9] is the last layer, but the structure of the convolution layer is flexible and the number of layers can be increased or decreased, 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.* [10] 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 Institute of Standards and Technology (NIST) preprocessed Garris *et al.*'s images to reduce dimensionality and extract normalized bitmaps - see Figure 1.



(a)

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

(b)

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

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 [11] and Stutz [12]. 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 [9] 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 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:

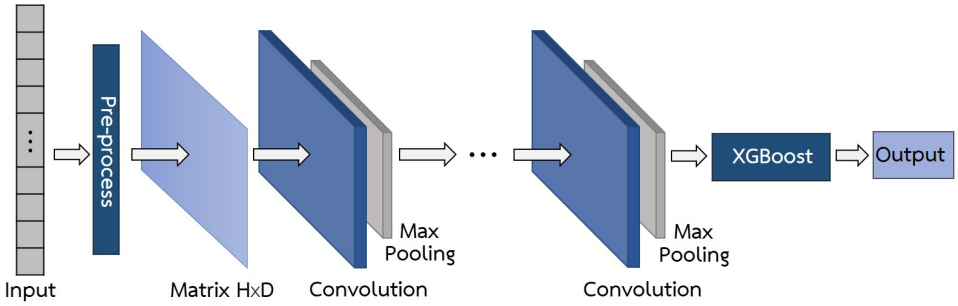


FIGURE 2. The structure overview of the DSPL.

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

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:

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

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 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 $[\overline{a}^{i_1 i_2 \ldots i_r}_{j_1 j_2 \ldots j_r}]_{n' n'}$, 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:

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

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.

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.

Our model was compared with with other models which has shown good performance: Convolutional Neural Networks (CNNs) [8], Extreme Gradient Boosting (XGBoost) [9], 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 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	[9]	$\mathcal{O}(x(Kt + \log B))$
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, 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.

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.

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.

Acknowledgements

This research was supported by the Thailand Research Fund (TRF) under grant number RTA6080013. 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 7 (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 7 (2019) 24738–24758.

[3] V. Venugopal, S. Sundaram, Online writer identification with sparse coding-based descriptors, IEEE Trans. Inf. Forensics Secur. 13 (10) (2018) 2538–2552.

[4] D. Impedovo, G. Pirlo, Dynamic handwriting analysis for the assessment of neurodegenerative diseases: A pattern recognition perspective, IEEE Rev. Biomed. Eng. 12 (2019) 209–220.

[5] D. Impedovo, Velocity-based signal features for the assessment of parkinsonian handwriting, IEEE Signal Process. Lett. 26 (4) (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 Process. Lett. 25 (8) (2018) 1136–1140.

[7] D. Dua, C. Graff, UCI Machine Learning Repository, Irvine, CA: University of California, School of Information and Computer Science.

[8] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.

[9] C. Tianqi, G. Carlos, XGBoost: A scalable tree boosting system, Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2016) 785–794.

[10] M.D. Garris, J.L. Blue, G.T. Candela, D.L. Dimmick, J. Geist, P.J. Grother, S.A. Janet, C.L. Wilson, NIST form-based handprint recognition system, NISTIR-5469 (1994) 1–63.

[11] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press; Cambridge, 2016.

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

[13] R.E. Fan, K.W. Chang, C.J. Hsieh, X.R. Wang, C.J. Lin, LIBLINEAR: A library for large linear classification, J. Mach. Learn. Res. (9) (2008) 1871–1874.

[14] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Machine Learning (63) (2006) 3–42.

[15] J.H. Friedman, Greedy function approximation: A gradient boosting machine, Ann. Statist. 29 (5) (2001) 1189–1232.

[16] L. Breiman, Random forests, Machine Learning (45) (2001) 5–32.

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

[18] L. Breiman, J. H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, CRC Press; Taylor & Francis Group LLC, New York, 1984.

[19] G.E. Hinton, Connectionist learning procedures, Artificial Intelligence 40 (1989) 185–234.

[20] C.C. Chang, C.J Lin, LIBSVM: A library for support vector machines, ACM Trans. Intell. Syst. Technol. (2011) 1–27.