An Effective Training Method for Convolutional Neural Network using Extreme Learning Machine

Bhurinuth Wongsrisakul
Atiruj Silnumkij
Sirapop Issariyodom

Bachelor of Engineering in Software Engineering
International College
King Mongkut's Institute of Technology Ladkrabang
Academic Year 2018

KMITL-2019-IC-B-003-001

Thesis - Academic Year 2018

Bachelor of Engineering in Software Engineering
International College
King Mongkut's Institute of Technology Ladkrabang

Title: An Effective Training Method for Convolutional Neural Network using Extreme Learning Machine

Authors:

1. Bhurinuth Wongsrisakul Student ID: 58090030
2. Sirapop Issariyodom Student ID: 58090041
3. Atiruj Silnumkij Student ID: 58090044

Approved for Submission

.....

(Dr. Ukrit Watchareeruetai)
Advisor

Date 21/6/2019

Acknowledgments

During the research, without the continuous support from our instructors, friends and families, this thesis would not succeed. First, We would like to acknowledge Dr. Ukrit Watchareeruetai, our thesis advisor, for the suggestions and support of our thesis, providing a place and equipment for our research. His contribution has considerably helped our research and some of the problem we faced during the time period.

We would like to thank all the academic and administrative staffs of the International College, for giving us all the knowledge and support. We would also like to thank Mr. Pavit Noinongyao and the other seniors in the computer vision lab for helping us in starting this thesis. He provided us with many advice, support, and time that led to the achievement of this thesis. Lastly, we would thank all our families and friends for all the support and motivation.

Abstract

In the field of object detection and recognition, convolutional neural network (CNN) is well-known for image classification and recognition and have been proven to be very effective. However, due to the backpropagation (BP) method that is used to train the CNN, the training process CNN requires a large amount of time. As a result, training CNN using BP is extremely slow. To mitigate this problem, Yoo and Oh presented a new algorithm CNN-ELM in 2016, where CNN is trained using extreme learning machine (ELM), a fast training algorithm. CNN-ELM have been proven to reduce the training time of CNN, but it is not quite effective in terms of accuracy. CNN-ELM average training time is significantly faster than the CNN trained using BP when a same number of hidden nodes are used, but the training accuracy is lower. To increase the effectiveness of CNN-ELM, in this thesis, we propose to apply the concept of iterative extreme learning machine (I-ELM) and constrained extreme learning machine (CELM).

Three models are proposed to improve the effectiveness of the CNN-ELM algorithm. On the first models, we have applied the concept of the I-ELM to the classification part of the CNN-ELM algorithm. Next, on the second models, the I-ELM was applied to both the feature extractor and the classification part of CNN-ELM. Lastly, on our third models, we will apply both the I-ELM and the CELM to the classification part of the CNN-I-ELM. The results are quite satisfactory. All of our three models showed an increased in the effectiveness of the CNN-ELM algorithm. However, due to the iterative process, the training speed was slightly slower than the original CNN-ELM algorithm. We can conclude that our models increased the effectiveness of the CNN-ELM algorithm with a slightly slower training time.

Table of contents

List of figures

List of tables

Chapter 1
Introduction

1.1 Problem description

Machine learning is a branch of artificial intelligence (AI) that gives the computer an ability to learn. Machine learning has been applied to fields of classification and recognition such as computer vision, speech recognition, and natural language processing. The basic idea of machine learning is to analyze the given data and use the information we have extracted or "learned" to be able to classify or distinguish the data into categories.

Deep learning is a subfield of machine learning that is based on multiple layers of nodes. Deep learning was inspired by the information processing and the communication pattern of the biological nervous system in the human brain. The idea of deep learning is to analyze the data and find hidden characteristics using multiple layers of artificial neural networks, which is a collection of nodes. One example of a deep learning model is the convolutional neural network (CNN) which was mainly used for image classification.

CNN is a model which was commonly used for the analysis of visual imagery [1]. CNN applies the convolutional operation to extract features from the input image. CNN performs nicely when there are a large amount of data but it requires an extremely long time to train. The main reason that CNN has a long training time is because it uses a gradient descent based learning algorithm, an algorithm that is used to minimize the errors, which usually requires a huge number of iterations to tune the parameters of the model.

ELM is a fast training algorithm for a single layer feed forward neural networks (SLFN), which is an artificial neural network which has only one single hidden layer where the connection between nodes move only in one direction, was proposed by

Huang et al. [2] in 2004. The way ELM train the network is by using only a single iteration to train the whole model which results in a very fast training speed. However, the original Huang's ELM in 2004 cannot be applied to train CNN directly because it was proposed as a training algorithm for SLFN, not for deep learning models, where there are many hidden layers.

In 2016, Yoo and Oh [3] proposed a new learning algorithm which could allow the CNN to be trained using ELM. This is called CNN-ELM. CNN-ELM is a combination of the CNN architecture and ELM representation learning. The resulting algorithm leads to a faster training of the CNN without sacrificing the classification performance. CNN-ELM has been proven to have a faster training time when comparing with the traditional CNN that has the same number of hidden nodes and was trained using a gradient descent based learning algorithm. However, the accuracy of CNN-ELM is lower than the traditional CNN causing the algorithm to be quite ineffective.

Therefore, in this thesis, we will try to improve the effectiveness of the CNN-ELM algorithm. We plan to apply the iterative ELM (I-ELM) which was proposed by Jiramaneepinit and Watchareeruetai in 2018 [4] and constrainted ELM (CELM) which was proposed by Zhu et al. in 2014 [5]. I-ELM is a training algorithm that will improve the performance of ELM by focusing more on the difficult instances. CELM is a training model which replaced the completely random connection weights from input layer to the hidden layer in ELM with a random subset of difference vectors of between-class samples. We believe that by using I-ELM and CELM. The overall effectiveness of CNN-ELM will be increased.

1.2 Objective

The objective of this thesis is to study the effects of using the modified concept of iterative ELM and constrained ELM to improve the effectiveness of the CNN-ELM learning algorithm.

1.3 Report structure

The remainder of this thesis is organized into different chapters as follows,

- Chapter 2 provides the background knowledges.
- Chapter 3 discusses about the related literatures.
- Chapter 4 introduces our proposed approaches.
- Chapter 5 explains the development and implementations.
- Chapter 6 presents our experimentation and results.
- Chapter 7 concludes this thesis.

Chapter 2
Background Knowledge

This chapter explains about the knowledge that is required to be able to understand our thesis. Starting with the architectures, single layer feed forward neural networks (SLFN) and convolutional neural network (CNN). After the architectures, gradient descent based learning and the extreme learning machine (ELM) will be introduced. Two additional learning algorithms, parallel ELM and iterative ELM is also introduced at the end of this chapter.

2.1 Single layer feedforward networks (SLFN)

SLFN is an artificial neural network that has one hidden layer which connections between nodes moves in one direction, from the input nodes through the hidden nodes and will end at the output nodes [6]. SLFN will use weights as a parameter to decide which path it will take to move to the next layer. To understand SLFN, let's first consider an input with only a single instance as shown in Fig. 2.1.

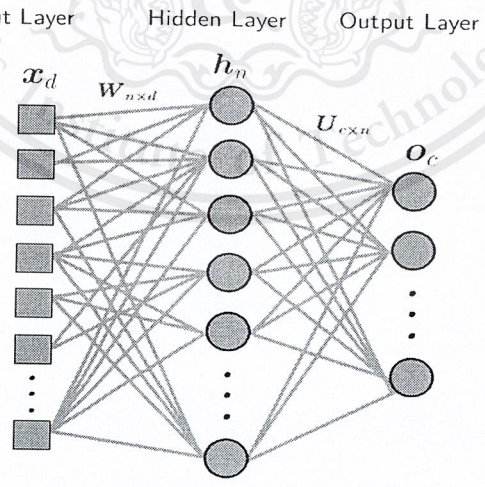

Figure 2.1: Single instance SLFN architecture

On the input side, we denote $\boldsymbol{x}$ as an input vector with d features. Then we denote $\boldsymbol{h}$ as a notation for a vector that contain n values which serve as our hidden nodes. The notation f is denoted as the activation function. We also need to declare a bias which can be denote as $\boldsymbol{b}$. For the output nodes, we use $\boldsymbol{o}$ to represent an output vector with c classes. Then we use $\boldsymbol{W}$, which is a matrix with a dimension of $n \times d$, to represent the weights from the input nodes to the hidden nodes and $\boldsymbol{U}$, which is a matrix with a dimension of $c \times n$, to represent weights from the hidden nodes to the output nodes. Now, the inputs are then feed directly to the output layer via the weights of each layer. Eq. (2.1). is used to calculate the value of $\boldsymbol{h}$. Finally, the Eq. (2.2) is used to obtain the weights of the output vector.

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{2.1}$$

$$\boldsymbol{o} = \boldsymbol{U}\boldsymbol{h} \tag{2.2}$$

In reality, this architecture was usually used to train multiple instances. There might be some changes in equations and notations into the correct representation, as shown in Fig. 2.2.

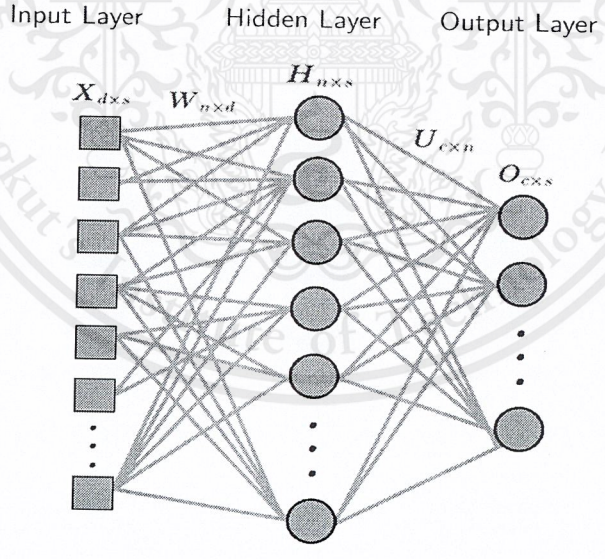

Figure 2.2: Multiple instances SLFN architecture

The notations are the same as the single instance SLFN, with the difference of the dimensions. In the single instance SLFN, we have used a vector as our representation of each instance, but for multiple instances, we have to use a matrix to represent our multiple instances. We use a capitalized variable to denote a matrix. First, we denote $\boldsymbol{X}$ as an input matrix with d features and s instances. For the hidden layer, we use $\boldsymbol{H}$ as a notation for a matrix that contain n values and s instances. Now, because $\boldsymbol{H}$ is a matrix, bias $\boldsymbol{b}$ must also be write in form of matrix. Eq. (2.3) is used to change bias $\boldsymbol{b}$, which is a vector, into a matrix without affecting the values. Finally, for the output layer, we denote $\boldsymbol{O}$ to represent an output matrix with c classes and s instances.

We still use $\boldsymbol{W}$, which is a matrix with a dimension of $n \times d$, to represent the weights from the input nodes to the hidden nodes and $\boldsymbol{U}$, which is a matrix with a dimension of $c \times n$, to represent the weights from the hidden node to the output nodes. The $\mathbf{1}$ in Eq. (2.3) denote as vector of ones with size of s. Now, Eq. (2.4) and Eq. (2.5) are used to calculate the weights between the hidden layer and output layer.

$$\boldsymbol{B} = (\boldsymbol{b})(\mathbf{1}_s^\intercal) \tag{2.3}$$

$$\boldsymbol{H} = f(\boldsymbol{W}\boldsymbol{X} + \boldsymbol{B}) \tag{2.4}$$

$$\boldsymbol{O} = \boldsymbol{U}\boldsymbol{H} \tag{2.5}$$

2.2 Convolutional neural network (CNN)

CNN [1][7] is a deep neural network architecture. It is being used in the area of image classification and recognition. It consists of many layers such as the convolutional layer, pooling layer, and fully-connected layer. The detailed explanation of each layer's action and methods are explained below.

2.2.1 Convolutional layer

The task of the convolutional layer is to use the convolutional operation to identify the features which are distinctive attributes of the input image. For example, a pen may have a long and slender shape, while an eraser may have a rectangular shape. These properties are known as features. To be able to identify the features, the convolutional

layer convolutes filters with an input image.

Consider the 5×5 and 3×3 matrices illustrated in Fig. 2.3. Let's assume that the 3×3 matrix is a filter matrix which will be used to convolute the image. This computation works by sliding the filter through the original image pixel-by-pixel then returns the convoluted values. The values can be calculated by taking the sum of the product of each values in the current region of the input image with the corresponding value of the filter. This process is shown in Fig. 2.3

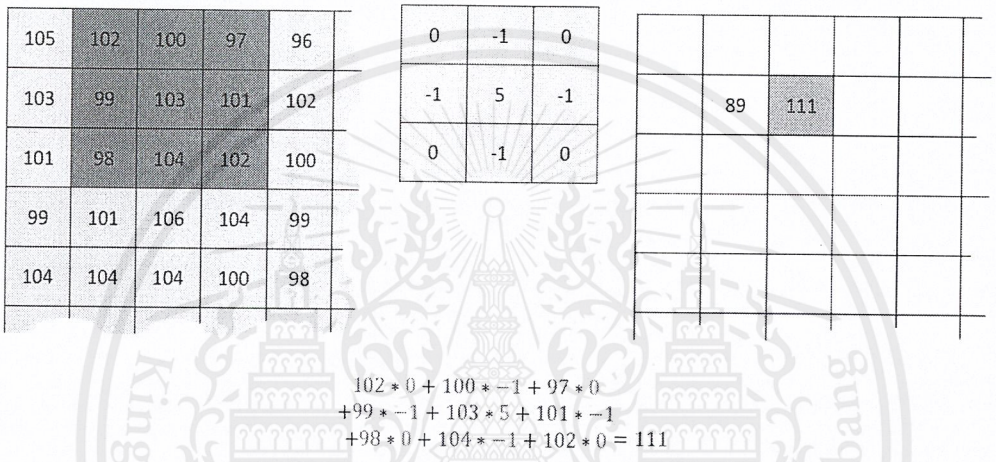

Figure 2.3: Convolutional operation

2.2.2 Pooling layer

This layer is also known as subsampling or down sampling. It is applied to the model after the convolutional layer. This layer takes an input image and reduces it size to make post-processing easier while keeping important data. In Fig. 2.4, we assume that the 4×4 matrix of the input image was chosen as an input, while filter size is 2×2.

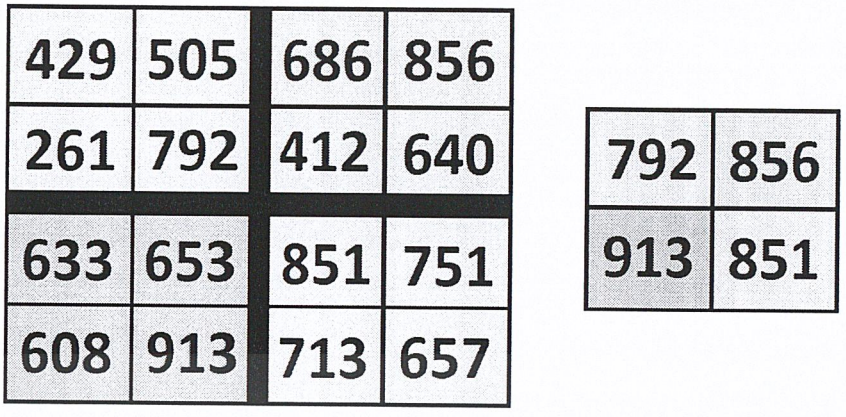

Figure 2.4: Pooling regions and results

We will then slide the filter according to the stride, which is the number of pixels shifts over the input matrix, through the input image. The stride is usually the same as the filter size. This is the same process as the convolutional operation. The calculation of the pooling operation, is to return the maximum value of the current region, in contrast to the convolutional operation which takes the sum of the product between the image and the filter. This is also known as the $max filter$ operation. The result output is going to be a smaller image where the distinctive attributes remains.

2.2.3 Fully-Connected layer

Architecture of SLFN have been used in this layer, where each neuron in each layer are connected to the next layer neurons (See Fig. 2.1). Now, we must flatten the output from pooling layer into the form of a 1-D vector to be used as an input instance of the SLFN. After passing through the SLFN, the input is now classified into different classes. In addition, there can be many fully-connected layer which must be connected together in series, but when multiple fully-connected layers are created, the last layer of the network must always be equal to the number of classes of the instances.

2.3 Backpropagation

Backpropagation is a technique that have been used in artificial neural networks to calculate a gradient that will be used in the optimization of the weights of the nodes in the network [8]. Backpropagation is in short form for the backward propagation of errors, since an error is computed at the output layer and distributed backwards throughout the network's layers. It looks for minimum value of the loss function, which used to evaluate the model from the given data, in weight space using gradient descent. The main idea of backpropagation is to use the chain rule to iteratively compute the gradients for each layer in multi-layered feedforward networks.

The algorithm begins after the input sample was passed into the network, the algorithm computes the loss which defined by $\boldsymbol{E}$ using Eq. (2.6) where the $\boldsymbol{S}$ is the number of total training samples, $\boldsymbol{o}$ is the expected output value of the i^{th} sample from the network, and $\boldsymbol{t}$ is the ground truth, which is the actual output value for the i^{th} sample. The learning can be achieved by using the loss value to update weights of the model. To update the weights, a partial derivative of the error function must be achieved, with respect to each weight value in the network.

$$E = \sum_{i=1}^{s} \left|\left| \boldsymbol{o}_i - \boldsymbol{t}_i \right|\right|^2 \tag{2.6}$$

According to Fig. 2.5, $w_{i,j}$ is the weight between the input node i and the hidden node j, and $u_{i,j}$ is the weight between the hidden node i and the output node j. The first step to calculate the gradient is to consider the weights $u_{i,j}$ according to loss function L by applying chain rule as shown in Eq. (2.7), where $\boldsymbol{y}$ is the output, which can be replaced by using Eq. (2.5), and k is the number of hidden nodes.

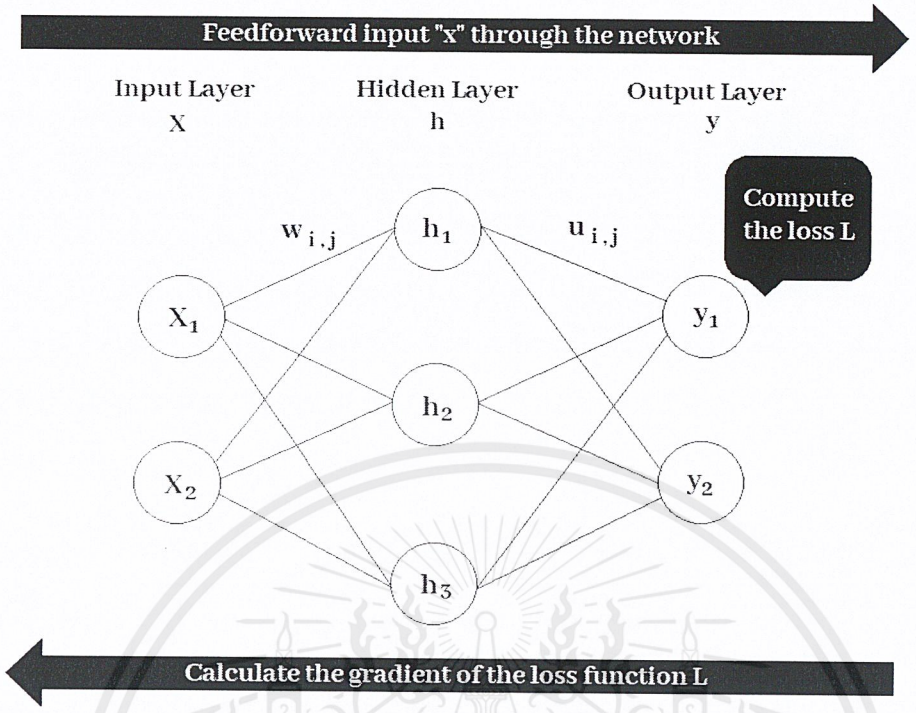

Figure 2.5: Backpropagation architecture

$$\begin{aligned}\frac{\partial L}{\partial u_{i,j}} &= \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial u_{i,j}} \\ &= \sum_{k}^{n} \frac{\partial L}{\partial y_k} \cdot \frac{\partial y_k}{\partial u_{i,j}} \\ &= \frac{\partial L}{\partial y_j} \cdot \frac{\partial y_j}{\partial u_{i,j}}\end{aligned} \tag{2.7}$$

From Eq. (2.7), the equation has been separated into 2 parts which are the part that it will differentiate y_j with L and another part which it differentiate $u_{i,j}$ with y_j. Then after Eq. (2.8) and Eq. (2.9) has been expand into the form that depends on y as shown in Eq. (2.8) and Eq. (2.9) respectively, it results as Eq. (2.10) which replaces both parts in Eq. (2.7) with Eq. (2.8) and Eq. (2.9). The next step is to consider the weights $w_{i,j}$ according to loss function L by also apply chain rule as shown in Eq. (2.11), where h_j is the value of hidden node j, which can be calculated by using Eq. (2.4).

$$\begin{aligned}\frac{\partial L}{\partial y_j} &= \frac{\partial \sum_k^n (t_k - y_k)^2}{\partial y_j} \\ &= \frac{\partial (t_j - y_j)^2}{\partial y_j} \\ &= 2(t_j - y_j)(-1) \\ &= -2(t_j - y_j)\end{aligned} \tag{2.8}$$

$$\begin{aligned}\frac{\partial y_j}{\partial u_{i,j}} &= \frac{\partial \sum_k^n h_k u_{k,j}}{\partial u_{i,j}} \\ &= \frac{\partial h_i u_{i,j}}{\partial u_{i,j}} \\ &= h_i\end{aligned} \tag{2.9}$$

$$\begin{aligned}\frac{\partial L}{\partial u_{i,j}} &= \frac{\partial L}{\partial y_j} \cdot \frac{\partial y_j}{\partial u_{i,j}} \\ &= -2(t_j - y_j)h_i\end{aligned} \tag{2.10}$$

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial h_j} \cdot \frac{\partial h_j}{\partial w_{i,j}} \tag{2.11}$$

The part where it differentiate y_j with L can be replaces by using Eq. (2.8), and the part where it differentiate $u_{i,j}$ with y_j can also be replaces by Eq. (2.9), which results as shown in Eq. (2.12). Next, let a_j denote the input to the hidden node j as shown in Eq. (2.13), and denote h_j as shown in Eq. (2.14), then use chain rule to differentiate $w_{i,j}$ with h_j and also using a_j, where g is the activation function.

$$\begin{aligned}\frac{\partial L}{\partial h_j} &= \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial h_j} \\ &= \sum_k^n \frac{\partial L}{\partial y_k} \cdot \frac{\partial y_k}{\partial h_j} \\ &= \sum_k^n [-2(t_k - y_k)u_{j,k}]\end{aligned} \tag{2.12}$$

$$a_j = \sum_k^n x_k W_{k,j} \tag{2.13}$$

$$h_j = g(a_j) = max(0, a_j) \tag{2.14}$$

$$\frac{\partial h_j}{\partial w_{i,j}} = \frac{\partial h_j}{\partial a_j} \cdot \frac{\partial a_j}{\partial w_{i.j}} = \begin{cases} x_i, & \text{if } a_j > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.15}$$

Eq. (2.15) can be obtained by using Eq. (2.16) and Eq. (2.17) which expand from the differentiation of $w_{i,j}$ with h_j. The last step is to combine Eq. (2.12) and Eq. (2.15) in order to get the equation for the differentiation of $w_{i,j}$ with loss function L as shown in Eq. (2.18) below.

$$\begin{aligned} \frac{\partial h_j}{\partial a_j} &= \begin{cases} \frac{\partial a_j}{\partial a_j}, & \text{if } a_j > 0 \\ \frac{\partial 0}{\partial a_j}, & \text{otherwise} \end{cases} \\ &= \begin{cases} 1, & \text{if } a_j > 0 \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{2.16}$$

$$\begin{aligned} \frac{\partial a_j}{\partial w_{i,j}} &= \frac{\partial \sum_k^n x_k w_{k,j}}{\partial w_{i,j}} \\ &= \frac{\partial x_i w_{i,j}}{\partial w_{i,j}} \\ &= x_i \end{aligned} \tag{2.17}$$

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial h_j} \cdot \frac{\partial h_j}{\partial w_{i,j}} = \begin{cases} x_i \sum_k^n [2 - (t_k - y_k) u_{j,k}, & \text{if } a_j > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.18}$$

2.4 Extreme learning machine (ELM)

Extreme learning machines (ELM) is a learning algorithm that is used for classification, regression, clustering, and feature learning for SLFN [10]. ELM was first proposed by Huang et al. in 2004 as a learning algorithm for SLFN [2]. The main idea of ELM is the randomization of the input weights, while the output weight will be calculated by using a generalized inverse operation. This process will be done using only a single iteration, which makes the training time much shorter when compared to the backpropagation method, which often requires an extremely large number of epochs.

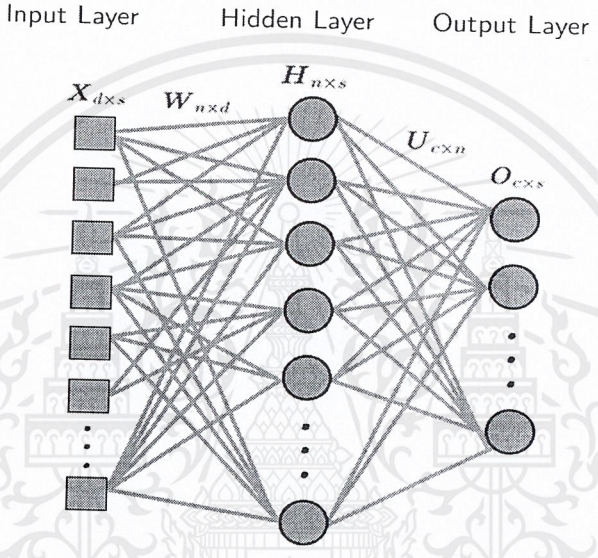

Figure 2.6: Multiple instances SLFN architecture with ELM

To train multiple instances using ELM as shown in Fig 2.6, the weights of each variables must be calculated. First we will be randomizing the weights of W. Next, using Eq. (2.4), we can find the values of H. Now, the values of the output O must be calculated. According to Eq. (2.5), we need the values from U.

Now to solve for U, we need to know the value of the output node. But we do not know the value of the output node yet, so we have to find the way to get the value of output node. Now, we want the value of output node to be the same or as close as possible to the value of target node, and we already know the value of target node, which is the value that is expected from the input, so we can assume that the value of the output node O will be equal to the value of the target node T. From Eq. (2.5) we can derive a new equation resulting in Eq. (2.19).

$$T_{c\times s} = U_{c\times n} H_{n\times s} \tag{2.19}$$

Using Eq. (2.19) to find the weight U we will have to eliminate H by multiplying an inverse matrix of H to both sides of the formula. This method will only work when matrix H is a square matrix, the accuracy for the target node when matrix H is a square matrix will be 100%. Unfortunately, most of the time, matrix H is not a square matrix. To get a better accuracy, the solution is to use a pseudoinverse which is a generalization of the inverse matrix to solve for the accuracy. The notation of pseudoinverse is depicted as $^+$ which is shown in Eq. (2.20). The value of U that we get is not the real value, but it is the value that gives the highest accuracy.

$$\hat{U}_{c\times n} = T_{c\times s} H^{+}_{s\times n} \tag{2.20}$$

Equation (2.20) that calculates for the approximation of U in ELM is the standard ELM since 2004. However, in 2012, regularization method has been introduced [11]. Regularization is the technique to improve generalization performance, which we can apply it to the calculation of U.

There are many types of regularization, one of them is called L2 regularization [11] which is widely used. In 2012, Huang et al. proposed another version of ELM, which supports feature that can include L2 regularization as shown in Eq. (2.21). Instance C is denoted as a constant. The algorithm for ELM is shown in Algorithm 1.

$$U = TH^{\intercal}\left(\frac{\mathbf{I}}{C} + HH^{\intercal}\right)^{-1} \tag{2.21}$$

Algorithm 1 Extreme learning machine (ELM)

1: Generate random weights for W and bias b
2: Calculate B from b using Eq. (2.3)
3: Calculate the value of the hidden nodes H using Eq. (2.4)
4: Calculate $\hat{U}$ using Eq. (biguhat)

2.5 Parallel ELM

Parallel ELM, proposed by Yoo and Oh in 2016, is a method to reduce resources that are required to store the training data in the local storage for efficient learning [3]. Using this algorithm, we can perform online learning, which is a method that we can

train the model separately and combine the result from each training. For example, if we have 500 data samples at first, we trained it and store the result in some variables, and in the future data samples become 600, we do not need to re-train the whole 600 data samples, but we only have to train the new 100 data samples and combine the result you get with the previous variables, which stores trained result for 500 data samples. The reason for dividing the training set into blocks is because some training data is too big to be processed at once. Here is where Yoo and Oh apply Eq. (2.21) to reduce the complexity of training set.

The parallel ELM technique performs the same calculation as the original ELM, but allows for parallel learning technique by using the results of Λ and Γ, which are the variables that will store the data of the training, to reduce memory usage without affecting the training speed. Λ and Γ can be calculated using Eq. (2.22) and Eq. (2.23) respectively, with s number of sample. We also denote t_i and h_i as a target node at the data sample I and hidden layer I respectively.

$$\begin{aligned}\Lambda &= \boldsymbol{T}\boldsymbol{H}^\intercal \\ &= \sum_{i=1}^{s} \boldsymbol{t}_i \boldsymbol{h}_i^\intercal \\ &= \boldsymbol{t}_1\boldsymbol{h}_1^\intercal + \boldsymbol{t}_2\boldsymbol{h}_2^\intercal + \ldots + \boldsymbol{t}_s\boldsymbol{h}_s^\intercal\end{aligned} \tag{2.22}$$

$$\begin{aligned}\Gamma &= \boldsymbol{H}\boldsymbol{H}^\intercal \\ &= \sum_{i=1}^{s} \boldsymbol{h}_i \boldsymbol{h}_i^\intercal \\ &= \boldsymbol{h}_1\boldsymbol{h}_1^\intercal + \boldsymbol{h}_2\boldsymbol{h}_2^\intercal + \ldots + \boldsymbol{h}_s\boldsymbol{h}_s^\intercal\end{aligned} \tag{2.23}$$

Eq. (2.24) and Eq. (2.25) has been used to update the value of Λ and Γ, and since Λ can be written in the form that shown in Eq. (2.22) and Γ can be written in the form that shown in Eq. (2.23). Eq. (2.21) can be modified as shown in Eq. (2.26). We also define $\boldsymbol{T}_{new}$ and $\boldsymbol{H}_{new}$ as the output of the new instances. The algorithm is shown in Algorithm 2.

$$\Lambda \leftarrow \Lambda + \boldsymbol{T}_{new}\boldsymbol{H}_{new}^\intercal \tag{2.24}$$

$$\Gamma \leftarrow \Gamma + \boldsymbol{H}_{new}\boldsymbol{H}_{new}^{\intercal} \tag{2.25}$$

$$\boldsymbol{U} = \Lambda\left(\frac{\mathbf{I}}{C} + \Gamma\right)^{-1} \tag{2.26}$$

Algorithm 2 Parallel extreme learning machine (Parallel-ELM)

1: Allocate Λ and Γ to the memory
2: Calculate $\boldsymbol{B}$ from $\boldsymbol{b}$ using Eq. (2.3)
3: Calculate the value of the hidden nodes $\boldsymbol{H}$ using Eq. (2.4)
4: Using Eq. (2.22) and Eq. (2.23), we update Λ and Γ
5: Calculate $\boldsymbol{U}$ using Eq. (2.26)

2.6 Iterative ELM

Iterative ELM is an effective method to improve the performance of the standard ELM for SLFN. This was proposed by Jiramaneepinit and Watchareeruetai in 2018 [4]. In iterative ELM, SLFN model is trained by standard ELM. The main idea of the iterative ELM is to iteratively update the output weight matrix $\boldsymbol{U}$ by using the incorrectly predicted instances. In the other words, iterative ELM forces the SLFN to pay more attention on difficult instances which were misclassified, and then tuned the weights accordingly.

To apply this concept, it starts by training SLFN model using standard ELM, then find the misclassified output and re-trains them. According to the concept of parallel learning, which uses Λ and Γ to allow the result from different trainings to be added, we can add the result of the new iteration that has trained using the misclassified instances of the previous iteration without having to re-training the whole network all over again. We can now update the Λ and Γ using the Eq. (2.27) and Eq. (2.28) where $\boldsymbol{T}_w$ and $\boldsymbol{H}_w$ are the target node and hidden node which respond to misclassified sample $\boldsymbol{w}$, respectively. This algorithm is shown in Algorithm 3.

$$\Lambda \leftarrow \Lambda + \boldsymbol{T}_w\boldsymbol{H}_w^{\intercal} \tag{2.27}$$

$$\Gamma \leftarrow \Gamma + \boldsymbol{H}_w\boldsymbol{H}_w^{\intercal} \tag{2.28}$$

Algorithm 3 Iterative extreme learning machine (I-ELM)

Allocate Λ and Γ to the memory
Generate random weights for $\boldsymbol{W}$ and $\boldsymbol{B}$
Using Eq. (2.22) and Eq. (2.23), we calculate Λ and Γ
for each iteration **do**
    Calculate $\boldsymbol{H}$ for the hidden layers using Eq. (2.4)
    Find the misclassified instances $\boldsymbol{X}_{wrong}$
    Update the current instances with $\boldsymbol{X}_{wrong}$
    Calculate $\boldsymbol{U}$ using Eq. (2.21)
    Update Λ and Γ using Eq. (2.27) and Eq. (2.28)
return $\boldsymbol{U}_w$

2.7 Constrained ELM

The completely random parameters in the hidden layer of ELM do not always represent discriminative features. Such unconstrained random parameters could cause the ELM to require many hidden nodes to meet desirable generalization performance. The larger number of hidden nodes, the more processing time.

To solve this problem, we can use the concept of Constrained ELM (CELM). Proposed by W. Zhu et al. [5], CELM will eliminate some hidden nodes that are ineffective by constraining the weight of the vector parameters of ELM to be randomly drawn from the closed set of difference vectors from between-class samples which will tackle the problem of discriminative hidden nodes.

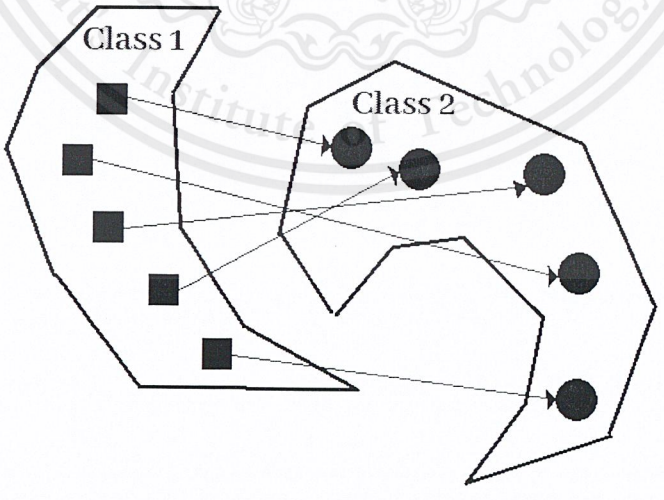

Figure 2.7: Difference vectors between two classes of samples

In Fig. 2.7, the blocks and circles represent the samples in class 1 and class 2, respectively. The difference vectors of between-class samples can map the samples to a higher discriminative feature space than ELM.

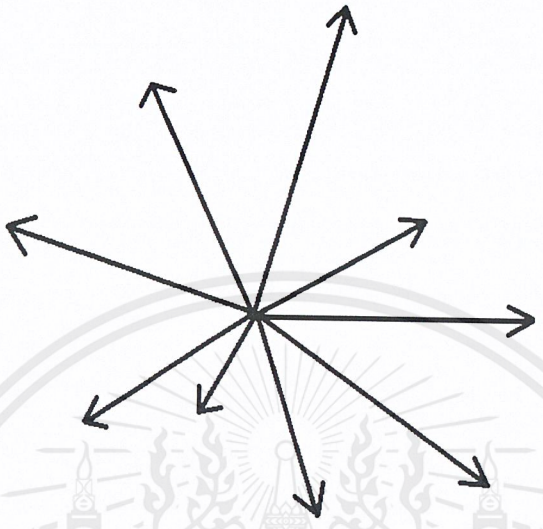

Figure 2.8: ELM weight vectors

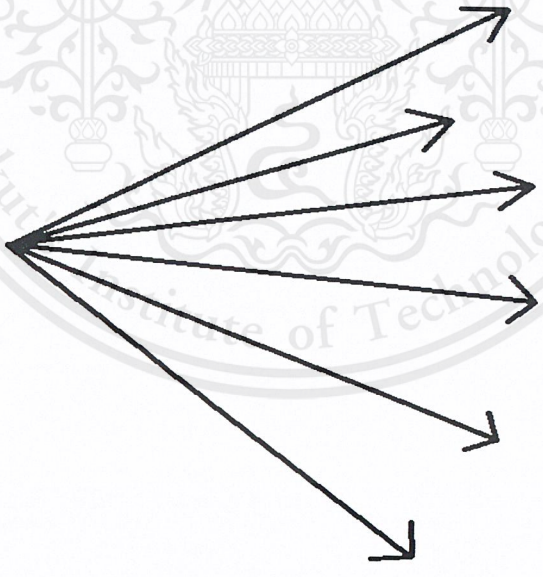

Figure 2.9: CELM difference vectors

As a comparison, the weight vectors from the input layer to the hidden nodes in ELMs are completely random without constraints, as shown in Fig. 2.8. It shows

that the weight vectors, which do not follow the direction from class 1 to class 2, are less discriminative for classification task, because directions of weight vectors shown in Fig. 2.8 are random, but to map the original samples into a discriminative feature space, directions of weight vectors should be close to the direction from class 1 to class 2 in Fig. 2.7. On the other hand, in Fig 2.9, directions of weight vectors are close to the direction from class 1 to class 2 in Fig 2.7, which are more discriminative for the classification tasks intuitively.

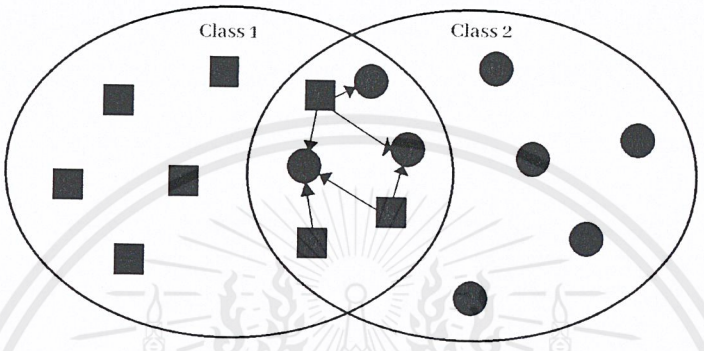

Figure 2.10: Noise difference vectors of between-class samples

There are two conditions that are used to delete noise and difference vectors for better generalization of CELM. The first method is to delete the small difference vectors. When the between-class samples are in the overlapped area of two classes as seen in Fig 2.10, the difference vectors are small and may contain non-discriminative or noise information. The second method is to delete the nearly parallel difference vectors. Nearly parallel vectors will make the data repeatedly projected to near points in the feature space, which may increase the number of hidden nodes, and may also increase the training time.

The aim of CELM is to split different classes samples into different areas in the feature space. We denote $\boldsymbol{x}_{c1}$ and $\boldsymbol{x}_{c2}$ as the samples drawn from two classes. Then the weight vector $\boldsymbol{W}$ from the input layer to one hidden neuron can be generated using Eq. (2.29) where α is the normalized factor. The data $\boldsymbol{X}$ is transformed to Eq. (2.30) by feature mapping where $\boldsymbol{b}$ is the bias with respect to the weight vector $\boldsymbol{W}$. We can obtain the normalization factor and bias as shown in Eq. (2.31) and Eq. (2.32) then, we use the vector $\boldsymbol{w}$ and bias b to construct the weight matrix $\boldsymbol{W}$ and bias vector $\boldsymbol{b}$. After this step, we will use the normal ELM to calculate the output weights U of the model. This algorithm is shown in Algorithm 4.

$$\alpha(\boldsymbol{x}_{c2} - \boldsymbol{x}_{c1}) \tag{2.29}$$

$$\boldsymbol{x}^{\intercal}\boldsymbol{w} + b = \alpha\boldsymbol{x}^{\intercal}(\boldsymbol{x}_{c2} - \boldsymbol{x}_{c1}) + b \tag{2.30}$$

$$\alpha = \frac{2}{\left|\left|\boldsymbol{x}_{c2} - \boldsymbol{x}_{c1}\right|\right|_2^2} \tag{2.31}$$

$$\frac{(\boldsymbol{x}_{c1} - \boldsymbol{x}_{c2})^{\intercal}(\boldsymbol{x}_{c1} - \boldsymbol{x}_{c2})}{\left|\left|\boldsymbol{x}_{c2} - \boldsymbol{x}_{c1}\right|\right|_2^2} \tag{2.32}$$

$$\boldsymbol{w} = \frac{2(\boldsymbol{x}_{c2} - \boldsymbol{x}_{c1})}{\left|\left|\boldsymbol{x}_{c2} - \boldsymbol{x}_{c1}\right|\right|_{L2}^2} \tag{2.33}$$

$$b = \frac{(\boldsymbol{x}_{c1} - \boldsymbol{x}_{c2})^{\intercal}(\boldsymbol{x}_{c1} - \boldsymbol{x}_{c2})}{\left|\left|\boldsymbol{x}_{c2} - \boldsymbol{x}_{c1}\right|\right|_{L2}^2} \tag{2.34}$$

Algorithm 4 Constrained extreme learning machine (CELM)

1: Randomly choose training samples $\boldsymbol{x}_{c1}$ and $\boldsymbol{x}_{c2}$ from any two classes
2: Generate difference vector $\boldsymbol{x}_{c1} - \boldsymbol{x}_{c2}$
3: **if** norm of vector is small enough **then**
4: delete and go back to step 1
5: **if** vector is nearly parallel with the previous generated vectors **then**
6: delete and go back to step 1
7: Normalize difference vector $\boldsymbol{w}$ using Eq. (2.33)
8: Calculate bias b using Eq. (2.34)
9: Use vector $\boldsymbol{w}$ and bias b to construct the weight matrix and bias vector
10: Calculate the hidden layer output matrix $\boldsymbol{H}$ using Eq. (2.4)
11: Calculate the hidden layer's output weight matrix $\boldsymbol{U}$ using Eq. (2.20)

Chapter 3
Related Works

This chapter discusses the related works of our thesis. It contains works that have applied the concept of ELM to train CNN for various purposes.

3.1 Fast training of CNN through ELM

CNN-ELM is a combination of the CNN architecture and ELM which leads its way to a rapid training of the CNN [3]. However, ELM learning takes a lot of memory space to store the entire training data in a very big matrix. To solve the problem, Yoo and Oh devised a new CNN-ELM training algorithm by employing parallel ELM learning, which resulted in a very efficient learning under limited memory resources. To train each convolution layer using ELM, the convolutional layer of CNN have would have to be a matrix but the convolutional operation does not produce a matrix. To produce a matrix that could be used with the ELM, the convolutional operator would have to be modified. Yoo and Oh modified the convolutional operations by applying various operation as shown in Fig. 3.1

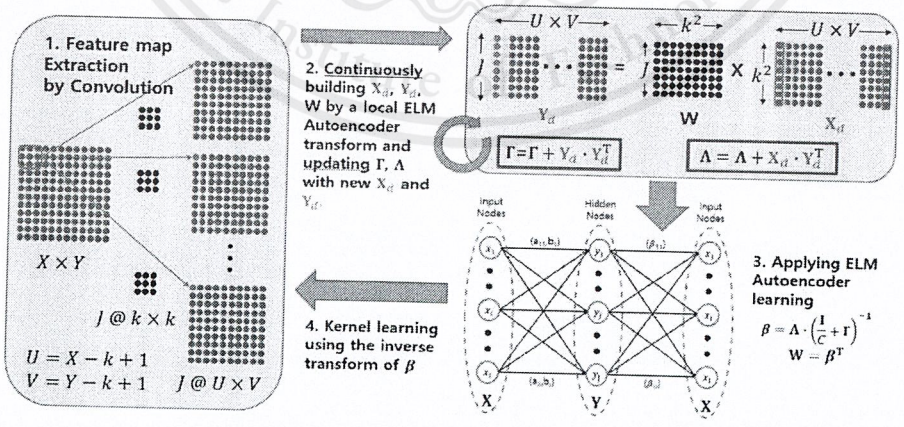

Figure 3.1: Yoo and Oh's CNN-ELM structure [3]

From Fig. 3.1 there are three steps to be taken,

1. The first step is to initialize the kernels by generating a random distribution function and perform mapping from the input feature map to the output feature map, which is called matrix X.

2. Next, the input feature map and output feature map, from the previous step is reshaped producing the big kernel matrix Y, which is a two-dimensional reshaped matrix combining all the individual convolutional kernels.

3. Finally, using ELM applied to the input and the output feature maps, the big kernel matrix will be learned by inversing the big kernel matrix into an individual convolution kernel to do the training of convolution layer. In the end, the feature map that was obtained from the convolution and pooling layer will be used in ELM to produce the class label of the input image.

In CNN-ELM training, the input feature matrix was computed from each training data via the local window based ELM auto-encoder, which requires huge memory resources due to the very nature of the convolution calculation. Auto-encoder is an unsupervised neural network model with its target output being the input itself as shown in Fig. 3.2. The main idea of auto-encoder seeks to find a representation, or a code, which best represents the data. An auto-encoder learns a representation by training the encoder and the decoder functions to minimize reconstruction error.

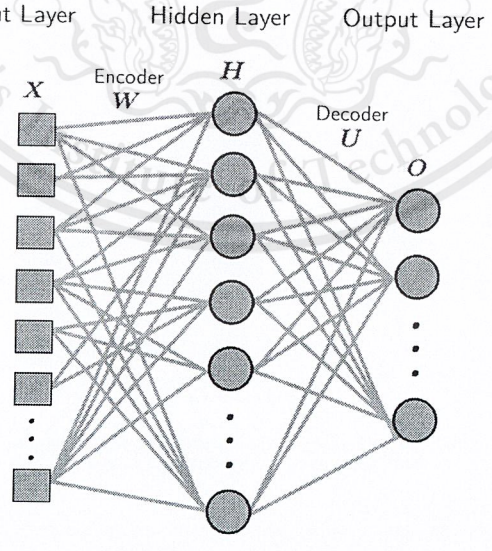

Figure 3.2: ELM auto-encoder

ELM auto-encoder, in this case, required a huge amount of memory because the training data sample was too big for it to be processed once as the total training data size k is much larger than the number of hidden nodes n. When k is more than n, the output weight, $\boldsymbol{U}$ is the desired output data with $\boldsymbol{C}$ being the ELM parameter, as shown in Eq. (3.1).

$$\boldsymbol{U} = \boldsymbol{T}\boldsymbol{H}^{\intercal}\left(\frac{\boldsymbol{I}}{C} + \boldsymbol{H}\boldsymbol{H}^{\intercal}\right)^{-1} \tag{3.1}$$

The parallel ELM, as previously discussed in section 2.5, performed the same calculation as the original ELM, but allowes for parallel training by using the intermediate results of Γ and Λ without affecting the training speed. The reason for using parallel ELM is because we would need to do the online learning for the training of each individual data of the whole dataset. The algorithm of CNN-ELM is listed in Algorithm 5 below.

Algorithm 5 CNN-ELM algorithm

1: Initialize kernel and j filters with size of k
2: Allocate Λ and Γ to the memory
3: Apply convolutional method to the input feature map with the size of $x \times y$
4: Perform mapping of input feature map into output feature map with the size of $u \times v$
5: Reshape input feature map and output feature map into an expanded form
6: Build the matrix $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{W}$ using local ELM-AE transformation
7: Update Λ and Γ with $\boldsymbol{X}$, $\boldsymbol{Y}$
8: Calculate $\boldsymbol{U}$ by applying ELM autoencoder learning using Eq. (3.1)
9: Apply the $maxpooling$ method to $\boldsymbol{X}$ and $\boldsymbol{Y}$
10: Assign $\boldsymbol{X}$ as $\boldsymbol{U}$
11: Allocate Λ and Γ to the memory
12: Calculate $\boldsymbol{B}$ from b using Eq. (2.3)
13: Calculate the value of the hidden nodes $\boldsymbol{H}$ using Eq. (2.4)
14: Using Eq. (2.22) and Eq. (2.23), we update Λ and Γ
15: Calculate $\boldsymbol{U}$ using Eq. (3.1)
16: **return** $\boldsymbol{U}$, $\boldsymbol{B}$, $\boldsymbol{H}$

In this paper, CNN-ELM has been tested on various famous dataset, such as MNIST [12], NORB, and CIFAR-10 [13], which CNN-ELM has shown to generalize well on the testing data. We have learned that the speed of CNN-ELM was much faster comparing to CNN-BP (CNN trained using back-propagation).

This indicates that the CNN features found by random weights are as powerful as or sometimes even better than those found by back propagation and furthermore, achieving this feat at a much greater speed. Comparing CNN-ELM specifically, the training speed of ELM still faster that CNN-ELM but what makes CNN-ELM better is an accuracy which is greatly better than ELM in a specific point of time. Finally, comparing to CNN-BP, CNN trained using back-propagation method, the CNN-ELM improved the miss rate and false positive curves by about 20 times.

3.2 CNN based on ELM for ship detection

Khellal et al. [14] tried to introduce new approach, also called CNN-ELM, to train CNN using ELM. To train CNN-ELM, the authors randomly initialized all the filters. Then, data are propagated through the network by directly computing the output of each block, which is the area of the filter that applied to the input image. For example, after training is performed in convolutional layer, the output can be evaluated from learning the convolutional kernel. However, to apply ELM to CNN, it needs more steps before training. The authors had to transform the image into a matrix form as shown in Fig. 3.3.

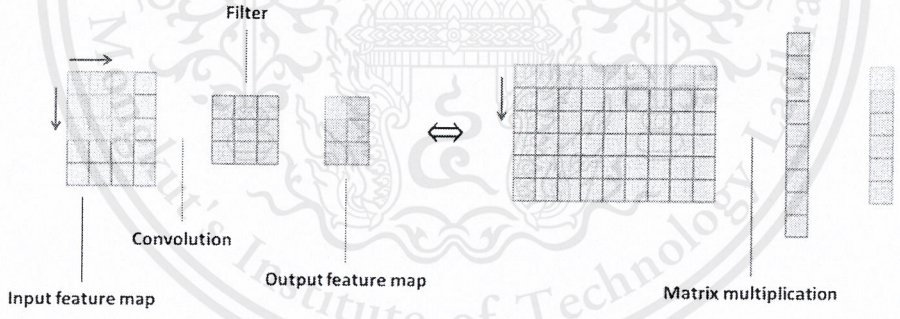

Figure 3.3: Khella et al. CNN-ELM model [14]

After the transformation, the convolutional operation, which is now a matrix where each local window is a vector, are all grouped into a single matrix. Then, the authors repeated this step on all filters in convolutional layer, then concatenate the vectorized version of all local windows respectively. The team then used the ELM based auto-encoder to train the convolutional filters.

This CNN-ELM algorithm is a layer-wise training procedure, which the team can stop at any desired layer and obtain the model. It is different from the backpropagation method, which the CNN will have to train the whole model. Moreover, it can apply to use with other models by changing the last layer to connect to other models.

In the experiment, the team used the VAIS dataset [15], which is a publicly available dataset formed by the long-wavelength infrared (LWIR) images and visible images of ships acquired from piers. The training set and test set each contains 539 and 703 samples respectively. Khella et al. approach is up to 950 times shorter than CNN which was trained using backpropagation. Moreover, the accuracy of CNN-ELM is also approximately 16% better.

3.3 Lane detection using CNN and ELM

Kim et al. [16] proposed a new algorithm, ELCNN, the mixture of the CNN and the concept of ELM. First, CNN is constructed by stacking the convolutional layer and pooling layer. After passing through several the stack, the data are then passed to a fully connected layer to finally classify the patterns with the desired value. In the conventional CNN with backpropagation, weights are optimized to minimize the error between the output and the given targets using optimization methods.

ELM replaces the iterative backpropagation method with a single-step matrix inversion function. The weight matrix from the input nodes to the hidden nodes is obtained by random initialization. With ELM, the input is assumed to a single hidden layer structure. The output from the hidden layer is computed using Eq. (3.2). In the equation, i and j denotes the row and column of the matrix.

$$\boldsymbol{h}_{ij} = g(\boldsymbol{w}_i \cdot \boldsymbol{x}_j + \boldsymbol{b}_i) = g\Big(\sum_{k=1}^{K} w_{i,k}\boldsymbol{x}_k + b_i\Big) \tag{3.2}$$

For the proposed CNN structure to have more than one hidden layer, there will be two sets of convolution and pooling layers and one fully connected Layer. The output of the fully connected layer is the target signal, so we can use Eq. (3.3) to find the optimal weight matrix, as depicted in Fig. 3.4.

$$\boldsymbol{\beta} = \boldsymbol{h}^{+}\boldsymbol{t} \tag{3.3}$$

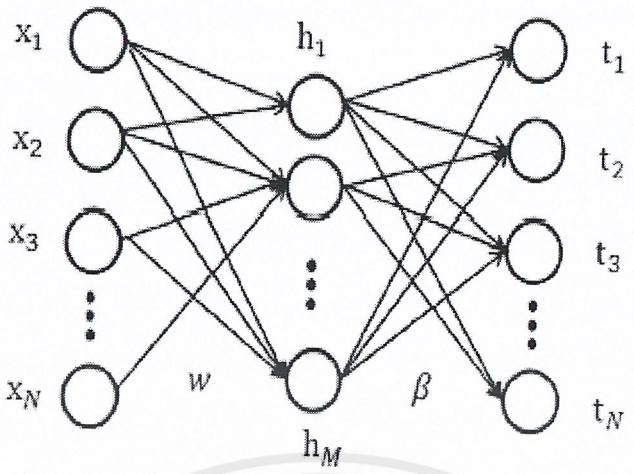

Figure 3.4: Kim et al. ELM structure [16]

In back propagation methods, the output error is propagated through a chained multiplication, so no exact target value is available in the next layer. To fix the problem, they borrow the values of the next convolutional layer as the target signals. Fig. 3.5 shows the structure of the proposed ELCNN.

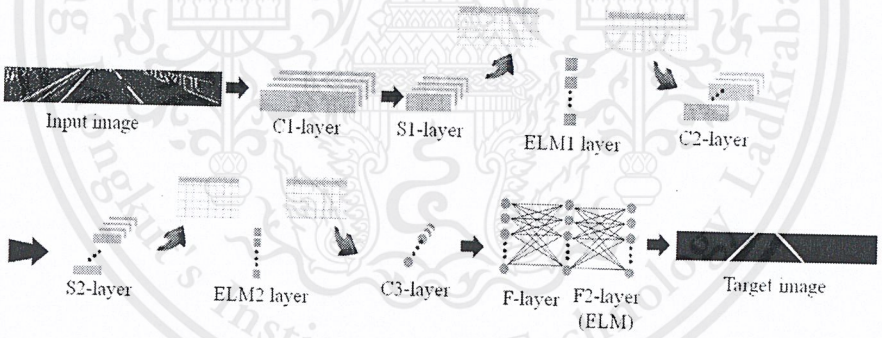

Figure 3.5: Kim et al. ELCNN model [16]

In the experimentation, the team used used the data from two different dataset KNU and Caltech. ELM alone produced the result that is far less superior to the general CNN but with the combination of CNN with ELM (ELCNN) produced satisfying results. For the training speed, the time required to train the data form the KNU dataset requires more than 2300 hours of training time, and the Caltech dataset required more than 800 hours. Both algorithms require more than 100,000 epochs to get the desired accuracy, but after using ELCNN, the time taken was reduced to 99-102 minutes and 21-22 minutes respectively. Even though the time required for ELCNN taken per epoch is

greater than the normal CNN, the number of epoch taken that is required for the desired accuracy is greatly reduced.

Chapter 4
Proposed Methods

CNN-ELM architecture which was proposed by Yoo and Oh [3], 2016, introduced in Section 3.1 was used as a basis our project. CNN-ELM is a CNN-based classifier that uses the ELM in the process of the convolution which will achieve a speedy learning.

Comparing between CNN-ELM and CNN-BP, using the same amount of hidden nodes, CNN-ELM training is faster but the effectiveness is lower. Iterative ELM and the constrained ELM both have been shown to have a better classification effectiveness, thus to improve the effectiveness of CNN-ELM, we now propose three new methods that uses iterative ELM and constrained ELM to train CNN-ELM.

4.1 Proposed method 1: CNN-I-ELM

The first method is called CNN-I-ELM. The architecture of this model is shown in Fig. 4.1. In this model iterative ELM is applied to the classifier, the fully connected layer. By applying iterative learning, the model will give more weights to the misclassified outputs. By giving more weights, the model would be able to more accurately classify the more weighted samples thus, the effectiveness should be increased. To be able to apply the iterative learning without the need to re-train the whole model, parallel ELM is used. The pseudocode for this method is shown in Algorithm 6.

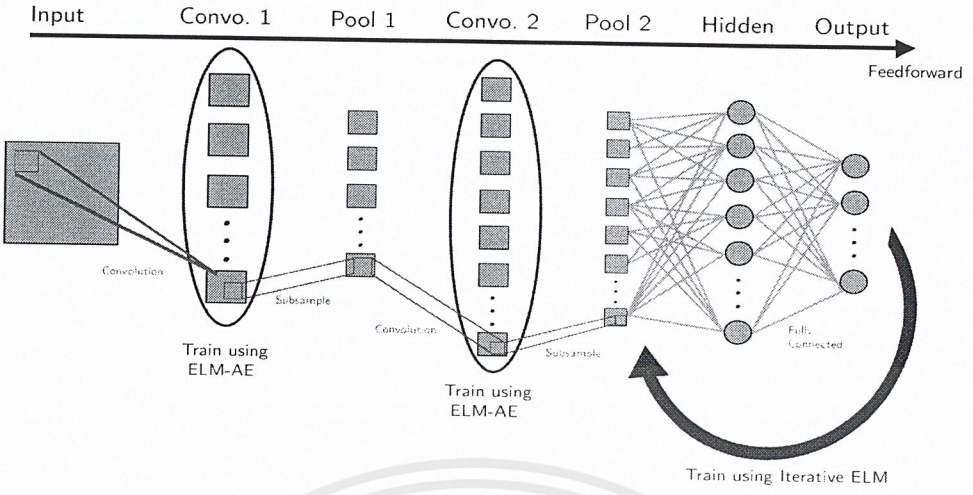

Figure 4.1: CNN-I-ELM architecture

Algorithm 6 CNN with iterative ELM (CNN-I-ELM)

1: Initialize kernel and j filters with size of k
2: Allocate Λ and Γ to the memory
3: Apply convolutional method to the input feature map with the size of $x \times y$
4: Perform mapping of input feature map into output feature map with the size of $u \times v$
5: Reshape input feature map and output feature map into an expanded form
6: Build the matrix $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{W}$ using local ELM-AE transformation
7: Update Λ and Γ_d with $\boldsymbol{X}$, $\boldsymbol{Y}$
8: Calculate $\boldsymbol{U}$ by applying ELM autoencoder learning using Eq. (3.1)
9: Apply the $maxpooling$ method to $\boldsymbol{X}$ and $\boldsymbol{Y}$
10: Assign $\boldsymbol{X}$ as $\boldsymbol{U}$
11: Allocate Λ and Γ to the memory
12: Generate random weights for $\boldsymbol{W}$ and $\boldsymbol{B}$
13: Using Eq. (2.22) and Eq. (2.23), we calculate Λ and Γ
14: **for** each iteration **do**
15: Calculate $\boldsymbol{H}$ for the hidden layers using Eq. (2.4)
16: Find the misclassified instances $\boldsymbol{X}_{wrong}$
17: Update the current instances with $\boldsymbol{X}_{wrong}$
18: Calculate $\boldsymbol{U}$ using Eq. (2.21)
19: Update Λ and Γ using Eq. (2.27) and Eq. (2.28)
20: **return** $\boldsymbol{U}$, $\boldsymbol{B}$, $\boldsymbol{H}$

4.2 Proposed method 2: I-CNN-I-ELM

Our second model, the I-CNN-I-ELM, shown in Fig. 4.2, is slightly different to our first model. In this model, the Iterative ELM is appled to both the feature extractor

(convolutional layer) and the classifier. Since the feature extractor of CNN-ELM uses ELM to train, iterative ELM could also be applied here to give more weights to the misclassified samples of the convolutional layer. The pseudocode for this method is shown in Algorithm 7.

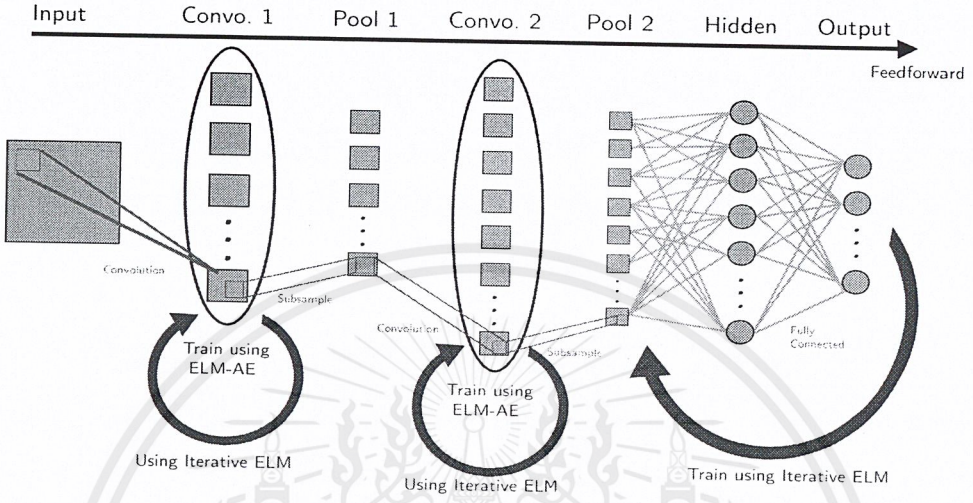

Figure 4.2: I-CNN-I-ELM architecture

Algorithm 7 Iterative CNN with iterative ELM (I-CNN-I-ELM)

1: Initialize kernel and j filters with size of k
2: Allocate Λ and Γ to the memory
3: Apply convolutional method to the input feature map with the size of $x \times y$
4: Perform mapping of input feature map into output feature map with the size of $u \times v$
5: Reshape input feature map and output feature map into an expanded form
6: **for** each iteration **do**
7: Build the matrix $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{W}$ using local ELM-AE transformation
8: Update Λ and Γ with $\boldsymbol{X}$, $\boldsymbol{Y}$
9: Calculate $\boldsymbol{U}$ by applying ELM autoencoder learning using Eq. (3.1)
10: Apply the $maxpooling$ method to $\boldsymbol{X}$ and $\boldsymbol{Y}$
11: Assign $\boldsymbol{X}$ as $\boldsymbol{U}$
12: Allocate Λ and Γ to the memory
13: Generate random weights for $\boldsymbol{W}$ and $\boldsymbol{B}$
14: Using Eq. (2.22) and Eq. (2.23), we calculate Λ and Γ
15: **for** each iteration **do**
16: Calculate $\boldsymbol{H}_w$ for the hidden layers using Eq. (2.4)
17: Find the misclassified instances $\boldsymbol{X}_{wrong}$
18: Update the current instances with $\boldsymbol{X}_{wrong}$
19: Calculate $\boldsymbol{U}$ using Eq. (2.21)
20: Update Λ and Γ using Eq. (2.27) and Eq. (2.28)
21: **return** $\boldsymbol{U}$, $\boldsymbol{B}$, $\boldsymbol{H}$

4.3 Proposed method 3: CNN-I-CELM

Our third method CNN-I-CELM, shown in Fig. 4.3, is also slightly different to our first model. The main difference is the weights of the classifier will be constrained. After the constraints, iterative ELM is also applied to let the model gives more weight to the misclassified samples. As most of the remaining hidden nodes are the effective nodes after the weight has been constrained, the algorithm would classify better. The pseudocode for this method is described in Algorithm 8.

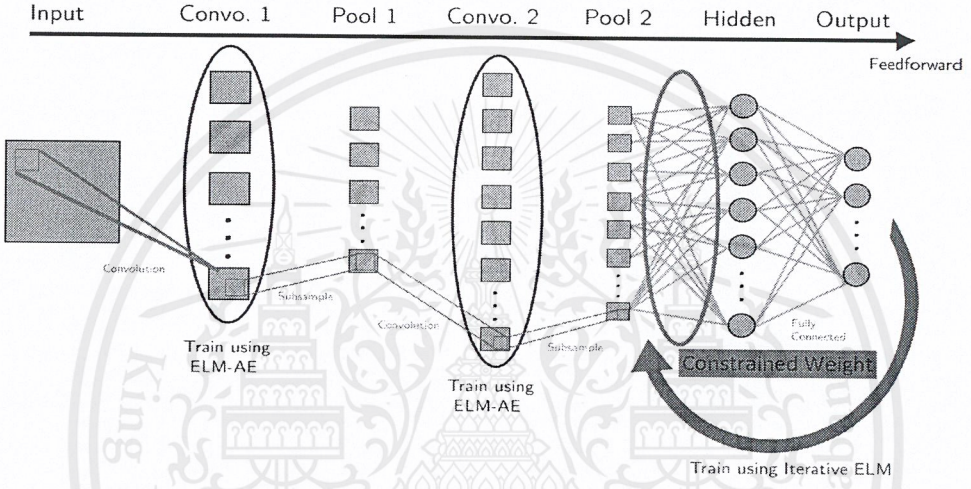

Figure 4.3: CNN-I-CELM architecture

Algorithm 8 CNN with iterative constrained ELM (CNN-I-CELM)

Initialize kernel and j filters with size of k
Allocate $\mathbf{\Lambda}$ and $\mathbf{\Gamma}$ to the memory
Apply convolutional method to the input feature map with the size of $x \times y$
Perform mapping of input feature map into output feature map with the size of $u \times v$
Reshape input feature map and output feature map into an expanded form
for each iteration **do**
    Build the matrix $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{W}$ using local ELM-AE transformation
    Update $\mathbf{\Lambda}$ and $\mathbf{\Gamma}$ with $\boldsymbol{X}$, $\boldsymbol{Y}$
    Calculate $\boldsymbol{U}$ by applying ELM autoencoder learning using Eq. (3.1)
    Apply the $maxpooling$ method to $\boldsymbol{X}$ and $\boldsymbol{Y}$
Assign $\boldsymbol{X}$ as $\boldsymbol{U}$
Allocate $\mathbf{\Lambda}$ and $\mathbf{\Gamma}$ to the memory
Generate random weights for $\boldsymbol{W}$ and $\boldsymbol{B}$
Using Eq. (2.22) and Eq. (2.23), we calculate $\mathbf{\Lambda}$ and $\mathbf{\Gamma}$
for each iteration **do**
    Randomly chooses training samples $\boldsymbol{x}_{c1}$ and $\boldsymbol{x}_{c2}$ from any two different classes
    Generate the difference vector $\boldsymbol{x}_{c1}$ and $\boldsymbol{x}_{c2}$
    Calculate the norm of vectors
    if Norm is small enough **then**
        return to step 16
    if the vectors are parallel **then**
        return to step 16
    Normalize difference vector $\boldsymbol{w}$ using Eq. (2.33)
    Calculate bias b using Eq. (2.34)
    Use the vector $\boldsymbol{w}$ and bias b to construct the weight matrix and bias vector
    Calculate $\boldsymbol{H}_w$ for the hidden layers using Eq. (2.4)
    Find the misclassified instances $\boldsymbol{X}_{wrong}$
    Update the current instances with $\boldsymbol{X}_{wrong}$
    Calculate $\boldsymbol{U}$ using Eq. (2.21)
    Update $\mathbf{\Lambda}$ and $\mathbf{\Gamma}$ using Eq. (2.27) and Eq. (2.28)
return $\boldsymbol{U}$, $\boldsymbol{B}$, $\boldsymbol{H}$

Chapter 5
Implementation

To begin with the implementation, all libraries and dependencies have to be initalized. This thesis is implemented using the Keras library which is a high-level neural network API of Tensorflow for Python. Python 3.7 and Tensorflow 1.12 are both required for this thesis. This chapter explains the implementation details of our models.

5.1 Proposed method 1: CNN-I-ELM

For our first model, we start by initializing every necessary component such as filter and some variables. The main process for convolution is in the train function. We pass the sample into the function as an input and apply the filter that we have generated to the sample to get the output feature map. TensorFlow is used to convolve and learn the big kernel. We applied parallel learning in order to train the big kernel. Next, we extract and stack the features into a TensorFlow array (tensor), and reshapes the input feature map and output feature map into the expanded input feature matrix in order to learn the big kernel.

We learned the big kernel matrix via ELM representation learning which we have applied to the input and the output feature map. Finally, we inversely transform the big kernel matrix into each individual convolution kernels. The last feature map that is obtained from the trained convolution layers and pooling layers will be passed as an input into ELM classifier.

After the convolutional layer is completed, the information passed into the pooling layer. For our pooling layer, we use Max-pooling2D function which uses the max pooling method of Keras and filter it with the dimension of 2×2. After pooling is done, we repeat the whole process for the second time, then the data passed into the fully connected layer by using ELM. In the Fully connected part, the weights from the input layer into the hidden layer was generated. In our implementation, we generate the weights by

randomizing the weights from the input to the output. This is the concept of ELM.

Next, the algorithm was trained by calculating the weights from the hidden nodes to the output nodes. In this training, we use the concept of iterative learning which requires us to develop the ELM training algorithm as a parallel learning. There is no need to re-train the whole algorithm for each iteration. For the iterative learning, first, we train the algorithm. After training, we use the training set to predict the outcome. We then isolate the incorrect outputs and return it. In the next iteration, we use the misclassified output and restart the training process of the fully connected layer. After all of this is done, it results a usable model of CNN-I-ELM, and we can test our algorithm with the test set of the dataset.

5.2 Proposed method 2: I-CNN-I-ELM

The process of I-CNN-I-ELM is very similar to the CNN-I-ELM method. The only difference is the convolutional process. In this method, we applied the iterative learning into the convolutional layer. By applying iterative learning into the convolutional layer, we have to find the error and feed it back as an input again into convolutional layer.

Since this layer is trained using ELM, we can evaluate the samples and generate a set of misclassified data. After we obtain the misclassified data, we passed the value to the next iteration and apply the parallel learning to add the misclassified output data into the final model.

After doing the iterative learning in the convolutional layer, next step is to do the fully connected layer the same way as CNN-I-ELM. When all process are done, it results a usable model which can be tested against the test set.

5.3 Proposed method 3: CNN-I-CELM

For this method, we have proceed with the convolutional process of the CNN-I-ELM method. After all of that is done, we applied the constrained method on the fully connected layer. For the constrained method, we declared the variables and other nec-

essary components. Next, parallel learning and the iterative concept have been applied.

After all the preparation is done, we have to constrain the weight of the ELM in order to decrease ineffective nodes. We done this by first divide the input into groups by paring the inputs. We pair the input by choosing the samples randomly. We then separate the input into a number of classes. After separating the input into different classes, we use an equation to find the different vectors of each classes.

Using the iterative and parallel process, we trained the network in the same manner as the CNN-I-ELM but using the constrained weights instead of the conventional randomized weights. After training, we proceed in testing our model, and pass the incorrect inputs to the next iteration until the desired amount of iteration has been passed.

Chapter 6
Experimentations

6.1 Datasets

In this thesis, we have conducted our experiments on three datasets. MNIST, Fashion MNIST, and CIFAR-10. The detailed explanation of each dataset is listed below.

1. MNIST [12] is a dataset of hand-written digits. It composes of 60,000 samples for training and 10,000 samples for testing. The samples are a 28×28 gray-scale images and each samples are categorized into 10 different classes. This dataset is shown in Fig. 6.1
2. Fashion MNIST [17] is a dataset that uses 10 different kinds of clothing accessories. In this dataset. There are 60,000 samples for training and 10,000 samples of testing. The samples are a 28×28 grayscale images, and 10 classes of each different types of clothing accessories. This dataset is also shown in Fig. 6.1.
3. CIFAR-10 [13] is a dataset which consists of 60,000 32×32 color images. It has 10 classes of different objects such as cats, dogs, airplanes, and trucks. There are 50,000 training images and 10,000 test images as shown in Fig. 6.2.

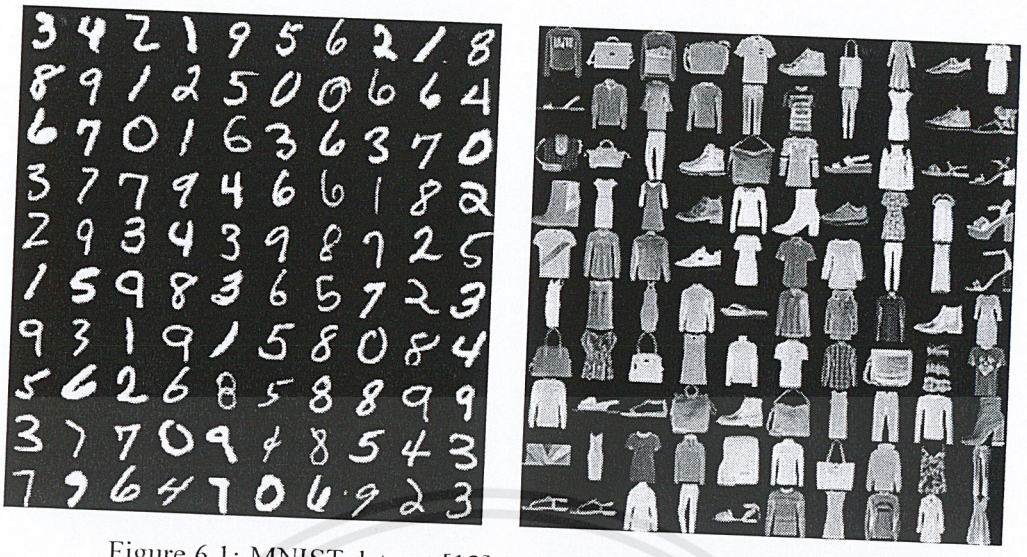

Figure 6.1: MNIST dataset [12] and Fashion MNIST dataset [17]

Figure 6.2: CIFAR-10 dataset [13]

6.2 Experimentation set-up

This experimentionl compares our proposed algorithm with CNN-BP, ELM (SLFN), and CNN-ELM. For each experiment, the number of hidden nodes together with the iterations or epochs of each algorithms was increased to see the overall trend. The specification of the computer that we have used for the experiments are as follows:

- CPU: Intel 1151 Core i5-8400 (2.80 GHz.)
- RAM: 24 GB of DDR4 (2400 MHz.)
- Storage: 500GB Segate Barracuda 7200 SATA3
- OS: Windows 10 Professional

This experimentation is done to test the effectiveness of our methods. The experimentation will be done with 1024, 2048, and 4096 hidden nodes, each on all three datasets. The parameter of each method will all be same, with the only exception of the input samples which is different for each datasets. The experimentation was done using 10 as a maximum number of iterations for each model.

6.3 Results and discussion

The results of each tests are shown in Table 6.1 (MNIST), Table 6.2 (Fashion MNIST), and Table 6.3 (CIFAR-10). In our experimentation, we have compared our model with CNN-BP, SLFN-ELM, and the CNN-ELM with the results as follows.

Table 6.1: Experimentation data using MNIST dataset

Method	1024 Nodes		2048 Nodes		4096 Nodes	
	Test Accuracy (%)	Training Time (Sec)	Test Accuracy (%)	Training Time (Sec)	Test Accuracy (%)	Training Time (Sec)
CNN-BP						
1 Epochs	94.15	204.89	96.12	369.11	94.01	704.25
2 Epochs	95.02	409.17	96.44	731.14	94.03	1,547.21
5 Epochs	96.12	1,004.52	97.13	1,795.55	94.54	3,714.03
10 Epochs	**97.08**	2,011.15	**98.11**	3,597.63	96.02	7,912.25
SLFN-ELM	92.03	**8.62**	94.63	**28.42**	96.41	108.04
CNN-ELM	93.68	70.06	94.29	74.38	94.51	**93.74**
CNN-I-ELM						
1 Iteration	94.56	74.70	94.82	78.81	94.98	101.19
2 Iterations	94.69	78.54	95.01	84.61	95.17	112.44
5 Iterations	95.15	89.00	95.35	101.69	95.65	123.52
10 Iterations	95.59	106.52	95.78	130.15	95.93	134.50
I-CNN-I-ELM						
1 Iteration	94.72	113.32	94.93	135.15	95.27	158.68
2 Iterations	94.91	116.95	95.13	141.04	95.45	170.21
5 Iterations	95.23	127.42	95.44	158.12	95.71	204.62
10 Iterations	95.64	145.22	95.83	186.82	95.95	261.50
CNN-I-CELM						
1 Iteration	96.34	78.29	96.88	86.84	97.41	114.58
2 Iterations	96.45	83.34	96.92	93.47	97.47	127.86
5 Iterations	96.53	98.92	97.01	113.48	97.51	167.59
10 Iterations	96.61	123.97	97.11	146.62	**97.61**	247.33

Table 6.2: Experimentation data using Fashion MNIST dataset

Method	1024 Nodes		2048 Nodes		4096 Nodes	
	Test Accuracy (%)	Training Time (Sec)	Test Accuracy (%)	Training Time (Sec)	Test Accuracy (%)	Training Time (Sec)
CNN-BP						
1 Epochs	38.41	205.35	37.82	361.35	27.15	679.01
2 Epochs	35.94	411.63	41.15	715.91	33.12	1,374.14
5 Epochs	89.17	1,017.15	79.35	1,785.56	55.14	3,372.36
10 Epochs	**89.52**	2,037.22	**88.50**	3,495.93	71.03	6,856.45
SLFN-ELM	81.17	**9.83**	82.11	**28.57**	82.43	108.11
CNN-ELM	80.85	67.81	82.41	74.35	83.50	**93.32**
CNN-I-ELM						
1 Iteration	81.86	69.55	83.30	76.62	84.41	105.12
2 Iterations	82.33	73.50	83.64	83.20	84.72	118.42
5 Iterations	82.49	85.14	83.98	102.51	84.96	157.33
10 Iterations	82.62	104.41	83.79	134.72	85.12	221.63
I-CNN-I-ELM						
1 Iteration	82.81	125.94	83.67	164.47	84.61	183.81
2 Iterations	83.58	129.73	84.07	170.88	84.77	195.46
5 Iterations	83.81	141.12	84.55	189.90	84.96	233.46
10 Iterations	84.01	159.71	84.84	221.36	85.06	295.64
CNN-I-CELM						
1 Iteration	82.67	73.26	83.90	81.36	84.73	135.87
2 Iterations	82.81	77.81	84.19	89.02	84.79	149.11
5 Iterations	83.35	91.94	84.33	119.31	85.09	188.44
10 Iterations	84.39	132.20	84.81	157.15	**85.11**	195.72

Table 6.3: Experimentation data using CIFAR-10 dataset

Method	1024 Nodes		2048 Nodes		4096 Nodes	
	Test Accuracy (%)	Training Time (Sec)	Test Accuracy (%)	Training Time (Sec)	Test Accuracy (%)	Training Time (Sec)
CNN-BP						
1 Epochs	64.10	224.87	64.50	394.19	65.14	736.14
2 Epochs	70.75	460.53	71.48	894.80	71.55	1,471.55
5 Epochs	70.83	1,215.52	71.84	1,968.51	72.35	3,686.39
10 Epochs	**71.07**	2,260.71	**71.63**	3,939.08	**72.60**	7,275.43
SLFN-ELM	40.79	**84.25**	40.81	**93.11**	41.14	**99.14**
CNN-ELM	40.63	93.26	41.02	97.14	41.52	104.35
CNN-I-ELM						
1 Iteration	41.06	100.58	41.44	104.67	41.93	119.09
2 Iterations	41.19	106.66	41.53	114.26	42.01	137.34
5 Iterations	41.59	127.73	41.79	142.51	42.15	191.77
10 Iterations	42.02	157.93	42.19	189.61	42.36	262.40
I-CNN-I-ELM						
1 Iteration	41.11	130.20	41.51	135.52	42.13	158.68
2 Iterations	41.34	136.39	41.72	145.21	42.32	170.21
5 Iterations	41.73	154.75	42.18	173.81	42.43	204.62
10 Iterations	42.11	185.42	42.48	221.14	42.53	261.50
CNN-I-CELM						
1 Iteration	45.40	107.14	45.90	112.31	46.01	130.84
2 Iterations	45.43	114.29	46.17	132.75	46.05	151.24
5 Iterations	45.52	134.80	46.36	161.44	46.28	205.06
10 Iterations	45.81	162.04	46.46	201.03	46.45	271.04

6.3.1 Experimentation 1: Our models vs. CNN-BP

Our results have shown that all of our three methods have achieved a slightly lower accuracy on both the MNIST and the Fashion MNIST, but on the CIFAR-10, our results have achieved a significantly lower accuracy. However, on all three datasets, our models has achieved a significantly faster training time. The training time of CNN-BP increases at an exponential rate while on all three of our models, the training time has increased at a linear rate.

6.3.2 Experimentation 2: Our models vs. SLFN-ELM

For this comparision, our results have shown to have a better accuracy on all three dataset. However, for the SLFN-ELM, since there is no iteration, we can only compare it with our first iteration. Our first iteration in all three models has achieved a better accuracy, but the training time is significantly slower than the SLFN-ELM. This is mainly because of the convolutional process of our models.

6.3.3 Experimentation 3: Our models vs. CNN-ELM

When comparing with CNN-ELM we have achieved a better results on all of the models tested on all three datasets. For our model, while we have achieved a better accuracy, the training time is slower. This is because of the many iterations which takes time. CNN-ELM while has a lower accuracy only train using one iteration so the training time is faster than our models. For each iteration, while the training time is getting slower, accuracy has increased.

6.3.4 Experimentation 4: CNN-I-ELM vs. I-CNN-I-ELM vs. CNN-I-CELM

CNN-I-ELM has achieved a comparable accuracy with the CNN-ELM but has a considerably lower time. When we add more iteration, it is shown that the accuracy of both datasets increased. I-CNN-I-ELM has also showed this trend. By looking at the results, I-CNN-I-ELM has shown that the accuracy has slightly increased when we applied the iterative process in the convoutional layer. This is because we have applied the iterative process to the convolutional part that was trained using ELM. From the results in the three tables, CNN-I-CELM model has the best accuracy and the training time is comparable to those of the CNN-ELM model.

6.3.5 Experimentation 5: Iterative ELM improvement

Table 6.4 shows the confusion matrix of the CNN-ELM algorithm on the test samples of the MNIST dataset, where the actual output is the row, predicted output is the column, and each number represents the number of misclassified instances. Harder misclassified instances, from our results, are the instances that are of a different class but are written to be quite similar to each other, as shown in Fig. 6.3. After we have combined the I-ELM to the CNN-ELM algorithm, CNN-I-ELM, which is our proposed method number 1, results have shown that the amount of the misclassified samples has reduced. Table 6.5, shows that after one iteration, the I-ELM has reduced the harder misclassified instances more than the easier misclassified instances.

Figure 6.3: Similar MNIST instances [13]

As an example, after 10 iterations, it was shown that the I-ELM has reduced the harder misclassified instances more than the easier ones, while some easier misclassified instances still remains. This is shown in Table 6.6. Table 6.7 shows the difference of the misclassified instances of the 0th iteration and the 10th iteration. According to the results, we can conclude that the I-ELM algorithms works better to reduce the harder misclassified instances, while having minimal effect on the already correctly predicted instances.

Table 6.4: Confusion Matrix of MNIST on CNN-ELM where actual output is represented as row, and predicted output is the column.

Confusion Matrix of MNIST on CNN-ELM										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**962**	0	12	3	2	10	9	3	4	8
1	0	**1121**	4	3	0	5	4	21	4	10
2	0	2	**949**	15	7	1	3	30	7	3
3	1	2	10	**938**	1	31	0	6	19	14
4	1	0	3	1	**879**	3	14	6	5	16
5	3	2	1	9	3	**802**	35	0	12	3
6	4	4	10	7	23	12	**888**	1	6	2
7	2	0	15	10	2	7	0	**935**	5	21
8	3	4	28	16	6	17	5	2	**898**	8
9	1	0	0	8	59	4	0	24	14	**924**

Table 6.5: Confusion Matrix of MNIST on CNN-I-ELM using 1 iteration where actual output is represented as row, and predicted output is the column.

Confusion Matrix of MNIST on 1st iteration										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**962**	0	11	2	2	9	9	3	4	8
1	0	**1121**	4	3	0	5	4	21	3	10
2	0	2	**951**	13	6	1	2	26	8	3
3	1	2	10	**944**	1	29	0	6	16	13
4	4	0	3	1	**887**	3	11	4	4	16
5	3	2	1	10	3	**805**	32	0	13	3
6	4	4	10	4	20	19	**895**	1	6	2
7	2	0	16	11	2	6	0	**941**	3	20
8	3	4	26	14	6	18	5	2	**906**	8
9	1	0	0	8	55	3	0	24	11	**926**

Table 6.6: Confusion Matrix of MNIST on CNN-I-ELM using 10 iterations where actual output is represented as row, and predicted output is the column.

Confusion Matrix of MNIST on 10th iterations										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**964**	0	9	1	2	8	8	2	4	7
1	0	**1121**	5	2	1	3	4	19	3	10
2	0	2	**970**	12	8	0	1	21	6	2
3	1	2	6	**953**	1	25	0	4	10	10
4	4	0	2	1	**907**	2	9	6	2	17
5	3	2	1	8	2	**815**	14	0	6	3
6	3	3	5	5	14	13	**917**	0	5	1
7	2	0	9	6	1	6	0	**954**	1	16
8	3	5	25	13	6	18	5	2	**930**	7
9	0	0	0	9	40	2	0	20	7	**936**

Table 6.7: Differences of 0th and 10th iterations where actual output is represented as row, and predicted output is the column.

Differences of 0th and 10th iterations										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**+2**	0	-3	-2	0	-2	-1	-1	0	-1
1	0	**0**	+1	-1	+1	-2	0	-2	-1	0
2	0	2	**+21**	-3	-1	-1	-2	-9	-1	-1
3	0	-2	-3	**+15**	0	-6	0	-2	-9	-4
4	+3	0	-1	0	**+28**	-1	-5	0	-3	+1
5	0	0	0	-1	-1	**+13**	-21	0	-6	0
6	-1	-1	-5	-2	-9	+1	**+29**	-1	-1	-1
7	0	0	-6	-4	-1	-1	0	**+19**	-4	-5
8	0	-1	-3	-3	0	+1	0	0	**+32**	-1
9	-1	0	0	-1	-19	-2	0	-4	-7	**+12**

6.3.6 Discussion

Fig. 6.4 - 6.12 shows the test results of the experiment on each different dataset. All figures have shown to have an increasing trend of the accuracy corresponding to the number of iterations. Overall, we can conclude that by using the concept of iterative ELM with the CNN-ELM, the effectiveness of the CNN-ELM algorithm will be increased, but when we have combine the concept of constrained ELM and iterative ELM, the accuracy has considerably increased when compared to CNN-ELM, thus, we can conclude that all of our models achieved a better effectiveness compared to the CNN-ELM.

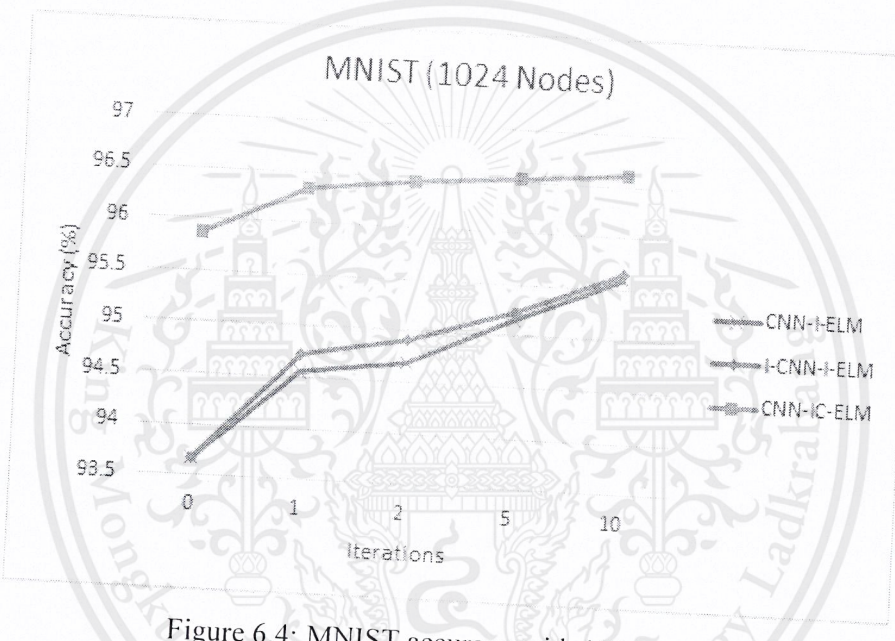

Figure 6.4: MNIST accuracy with 1024 nodes

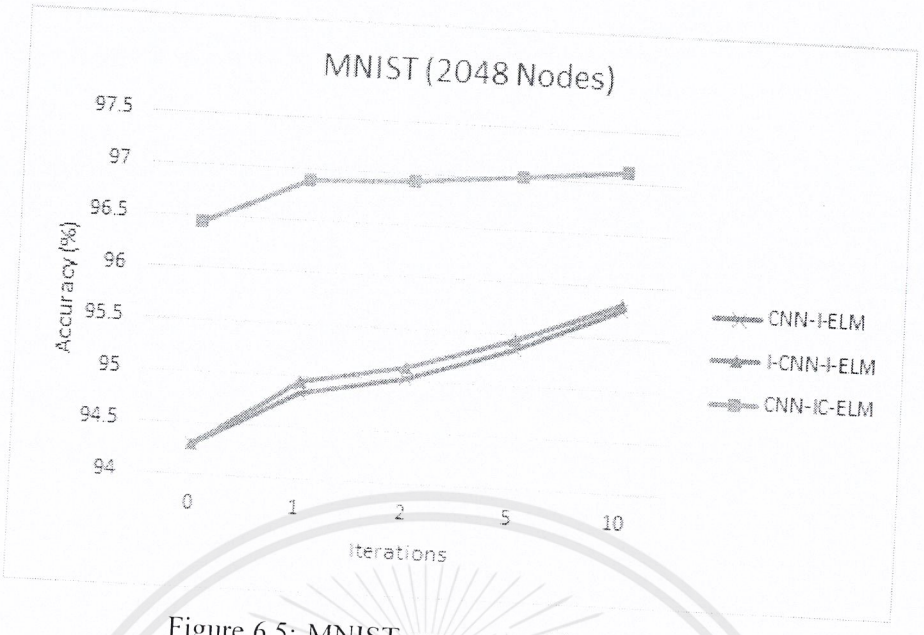

Figure 6.5: MNIST accuracy with 2048 nodes

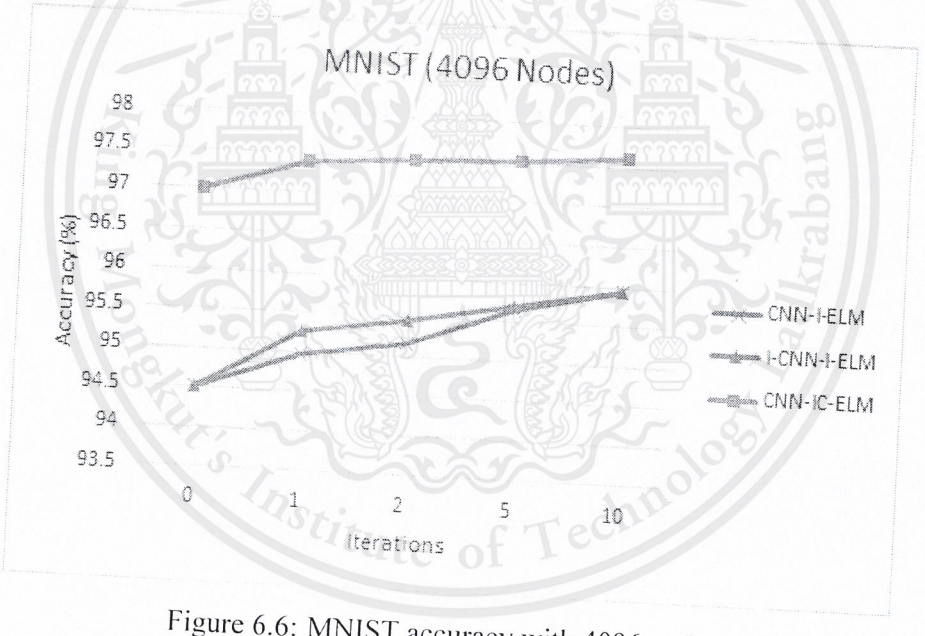

Figure 6.6: MNIST accuracy with 4096 nodes

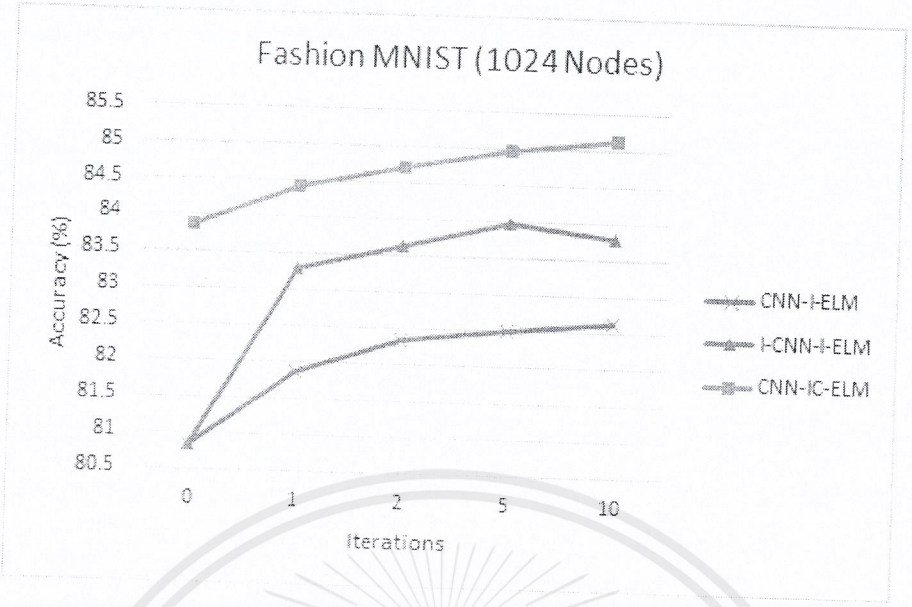

Figure 6.7: Fashion MNIST accuracy with 1024 nodes

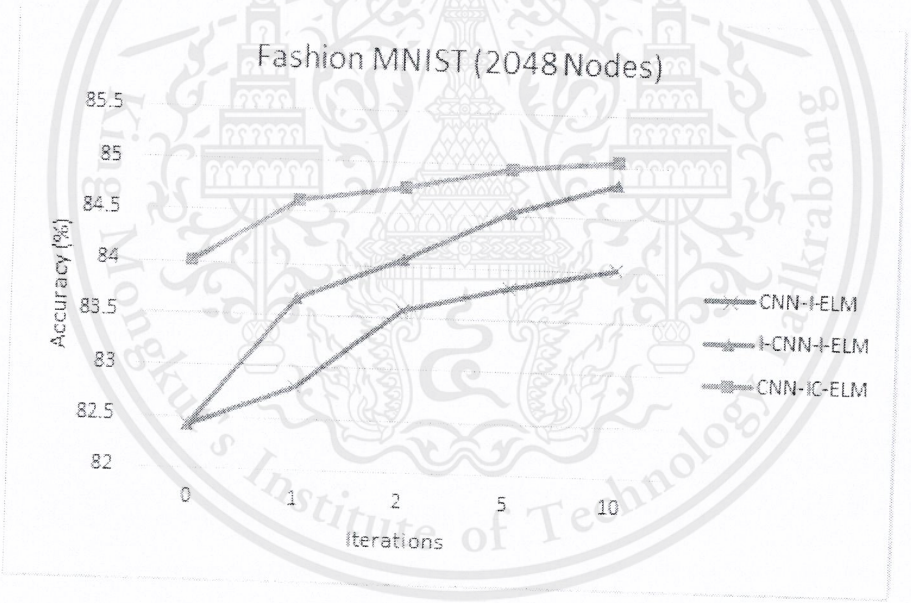

Figure 6.8: Fashion MNIST accuracy with 2048 nodes

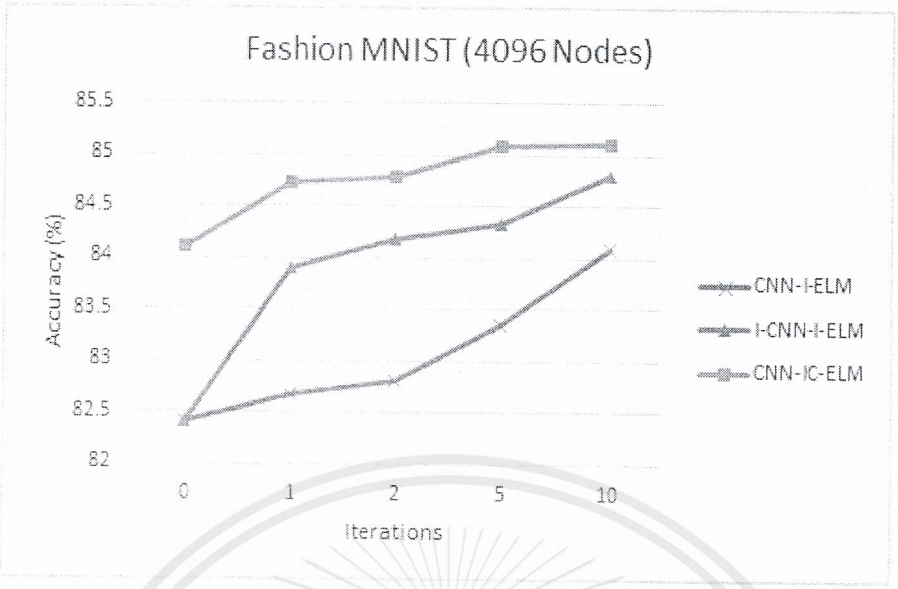

Figure 6.9: Fashion MNIST accuracy with 4096 nodes

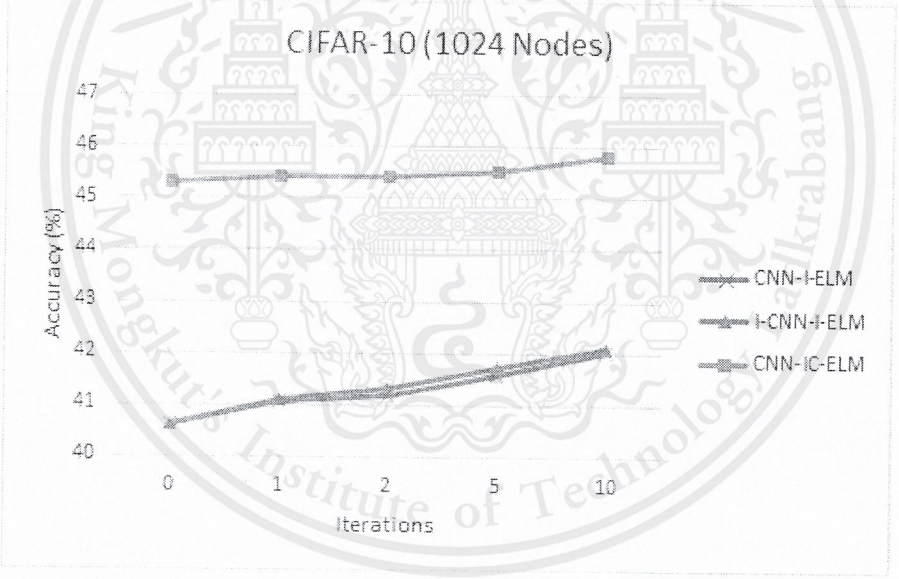

Figure 6.10: CIFAR-10 accuracy with 1024 nodes

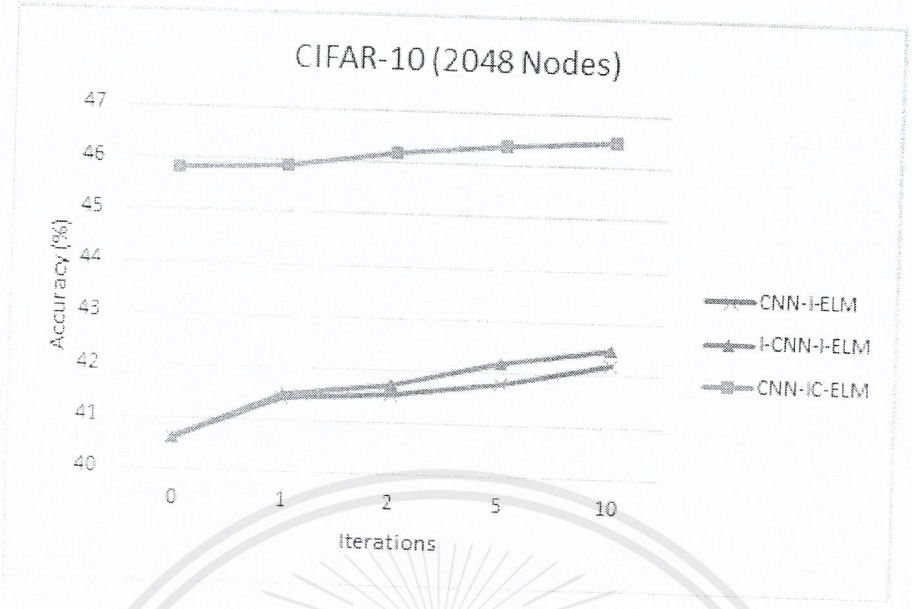

Figure 6.11: CIFAR-10 accuracy with 2048 nodes

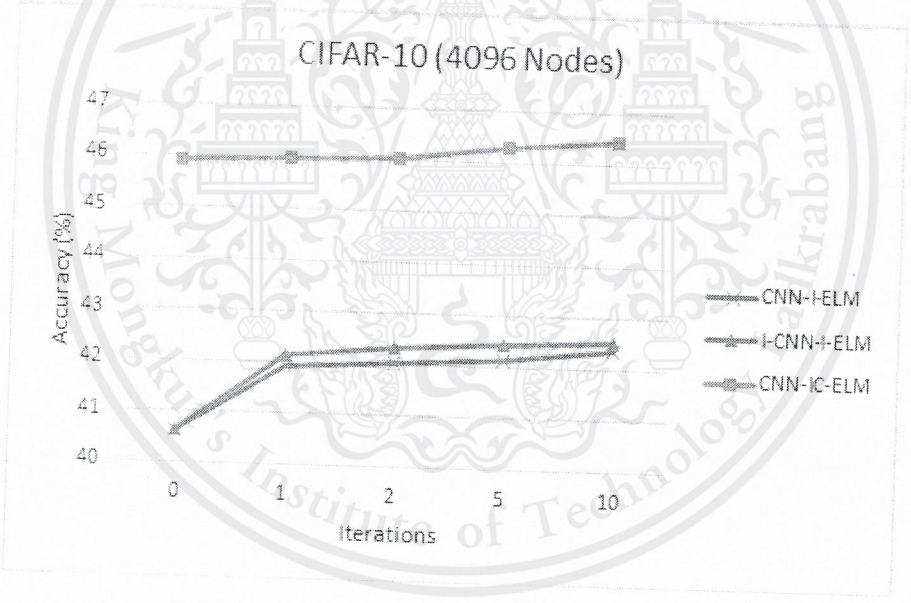

Figure 6.12: CIFAR-10 accuracy with 4096 nodes

Chapter 7
Conclusion

This thesis proposes a way to improve the training effectiveness of the CNN-ELM which was proposed by Yoo and Oh [3], in 2016. Three methods was proposed, CNN-I-ELM (Section 4.1), I-CNN-I-ELM (Section 4.2), and CNN-I-CELM (Section 4.3).

In CNN-I-ELM, we have applied the iterative ELM at the classification part. This application led to a higher accuracy at each increasing, but with a slower training speed, because each iteration increases the workload of the model.

For I-CNN-I-ELM, we have applied the iterative ELM to both the feature extractor and the classifier. This led to a higher accuracy when compared to the CNN-I-ELM, but the training speed is slower than CNN-I-ELM.

Finally, for CNN-I-CELM, we have added the constrained ELM from the CNN-I-ELM at the classification part. The randomized weights of the ELM will be constrained. This led to a higher accuracy than both previous models, with a comparable training speed when compared to the CNN-I-ELM.

Overall, all of our proposed models have achieved a better results when compared to the CNN-ELM algorithm that was a basis for our thesis. While the training speed was slower than CNN-ELM, the training accuracy was higher, making our models more effective than the CNN-ELM.

7.1 Future work

There are still rooms for further experimentations and developments. Models such as constraining both the classification and the feature extraction part could still be implemented and tested. Additionally, more datasets and methodologies can be used to test and improve our existing models to further improve the effectiveness of the CNN-ELM.

Bibliography

[1] Y. LeCun, P. Haffner, L. Bottou, Y. Bengio, "Object recognition with gradient-based learning," Shape, Contour, and Grouping in Computer Vision: Lecture notes in computer science, vol.1681, pp. 319, 1999.

[2] G.B. Huang, Q.Y. Zhu, and C.K. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," Proceedings of 2004 International Joint Conference on Neural Networks (IJCNN), vol.2, pp.985–990, 2004.

[3] Y. Yoo and S. Oh, "Fast training of convolutional neural network classifiers through extreme learning machines," 2016 International Joint Conference on Neural Networks (IJCNN), 2016.

[4] B. Jiaramaneepinit and U. Watchareeruetai, "Iterative extreme learning machine," The 22nd International Computer Science and Engineering Conference (ICSEC 2018), pp.156-161, 2018.

[5] W. Zhu, J. Mao, L. Qing, "Constrained extreme learning machine: A novel highly discriminative random feedforward neural network," Proceedings of 2014 International Joint Conference on Neural Networks (IJCNN), pp.800-807, 2014.

[6] D. Stutz, "Introduction to neural networks," Seminar on Selected Topics in Human Language Technology and Pattern Recognition, Technical report of Aachen University, 2014.

[7] M. Zeiler and R. Fergus "Visualizing and understanding convolutional networks," European Conference on Computer Vision (ECCV), vol 8689, pp. 818-833, 2014

[8] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning representations by backpropagating errors," Nature, vol.323, no.6088, pp.533-536, 1986.

[9] A. Cauchy, "Méthode générale pour la résolution des systemes d'équations simultanées," Proceedings of the Academy of sciences, pp.536–538, 1847.

[10] G.B. Huang, Q. Zhu and C. Siew, "Extreme learning machine: Theory and applications," Neurocomputing, vol.70, no.1-3, pp.489-501, 2006.

[11] G.B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol.42, no.2, pp.513–529, 2012.

[12] 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, 1998.

[13] A. Krizhevsky and G. Hinton, "Learning multiple Layers of features from tiny images," Technical report of University of Toronto, 2009.

[14] A. Khellal, H. Ma and Q. Fei, "Convolutional neural network based on extreme learning machine for Maritime Ships Recognition in Infrared Images." Sensors, vol.18, no.5, pp.1490, 2018.

[15] M. Zhang, J. Choi, K. Daniilidis, M. Wolf and C. Kanan. "VAIS: A dataset for recognizing maritime imagery in the visible and infrared spectrums." 2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2015.

[16] J. Kim, J. Kim, G. Jang and M. Lee, "Fast learning method for convolutional neural networks using extreme learning machine and its application to lane detection," Neural Networks, vol.87, pp.109-121, 2017.

[17] H. Xiao, K. Rasul and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," Technical Report of Cornell University, 2017.