IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 8, 2021, accepted January 20, 2022, date of publication January 31, 2022, date of current version February 8, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3147869

A Layer-Wise Theoretical Framework for Deep Learning of Convolutional Neural Networks

HUU-THIET NGUYEN, SITAN LI, AND CHIEN CHERN CHEAH, (Senior Member, IEEE)
School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798

Corresponding author: Chien Chern Cheah (eccheah@ntu.edu.sg)

This work was supported by the Agency for Science, Technology and Research of Singapore (A*STAR), under the National Robotics Program—Robotics Domain Specific, under Grant 1922200001.

ABSTRACT As research attention in deep learning has been focusing on pushing empirical results to a higher peak, remarkable progress has been made in the performance race of machine learning applications in the past years. Yet deep learning based on artificial neural networks still remains difficult to understand as it is considered as a black-box approach. A lack of understanding of deep learning networks from the theoretical perspective would not only hinder the employment of them in applications where high-stakes decisions need to be made, but also limit their future development where artificial intelligence is expected to be robust, predictable and trustable. This paper aims to provide a theoretical methodology to investigate and train deep convolutional neural networks so as to ensure convergence. A mathematical model based on matrix representations for convolutional neural networks is first formulated and an analytic layer-wise learning framework for convolutional neural networks is then proposed and tested on several common benchmarking image datasets. The case studies show a reasonable trade-off between accuracy and analytic learning, and also highlight the potential of employing the proposed layer-wise learning method in finding the appropriate number of layers in actual implementations.

INDEX TERMS Deep learning, CNNs, layer-wise learning, explainable AI, trust in AI.

I. INTRODUCTION

Convolutional neural networks (CNNs) have been successfully utilized for various applications with image inputs such as image classification, pattern recognition, object detection, image segmentation. Numerous CNN structures have been proposed over the years in attempts to obtain better empirical results for different applications. From a simple structure with just several convolutional layers in LeNet [1], deeper and larger convolutional neural networks have been constructed over time, such as AlexNet [2], and VGG [3]. The learning of these deep networks, deep learning (DL), is largely based on backpropagation (BP) of the gradient information of the loss function [4], where the training requires 2 passes: forward and backward. In the forward pass, the data is propagated from the input layer to the output layer to produce an output error, and then in the backward pass, the gradient of the loss function based on the error is propagated backward to the front layers to adjust the weights. Although CNN models trained by BP have achieved great successes, majority of the achievements are from the empirical perspective. The theoretical understanding of these successes still remains unknown, and the deep models trained by BP algorithm are well known to be black-box [5] and thus difficult to explain. In high-stake decision making, humans cannot put their trust in something of which they have no knowledge. Improving the theoretical understanding of deep networks and their learning algorithms is crucial for their robustness and trustworthiness. While gaining understanding of deep learning networks, some trade-offs could be acceptable such as the decrease of the model accuracy [6]. Developing a theoretical framework for a better understanding of deep learning would pave the way for artificial intelligence (AI) to be deployed in an extensive scale and in higher-stakes applications in the future.

Besides black-box property, some inherent issues of backpropagation have been known over the years such as backward locking (the weights in a specific layer have to wait for the signal to propagate through other layers before they can be updated) [7], memory reuse problem [8], and

The associate editor coordinating the review of this manuscript and approving it for publication was Nikhil Padhi.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

biologically implausibility [9]. Recently there is an increasing interest in an alternative of BP algorithm, layer-wise learning of deep neural networks, in which the large black boxes are dissected into smaller pieces. The idea was originally used as a pre-training method called greedy layer-wise pre-training [10], where the whole network still needs fine-tuning using backpropagation of global errors after pre-trained. The method as a complete learning algorithm without fine-tuning of whole networks has attracted more interest in the machine learning (ML) community recently under different concepts such as forward thinking [11], or learning using local errors [9], [12]. The approach is believed to produce fairly comparable accuracy to global backpropagation in a number of applications. However, despite the reasonably good performance, the theoretical analysis was not sufficiently considered, and there was no convergence analysis for these algorithms [9]–[12].

The problem of convergence analysis in deep networks is well-known to be very challenging. The optimization problems in deep networks are high-dimensional, highly non-convex and thus very complicated to analyze. This is one of the reasons why convergence issue has not received sufficient considerations from ML scientists. A learning algorithm without ensuring convergence could pose a risk when employed in systems that requires stability and robustness, such as in control and robotics [13]–[15]. The ignorance of convergence issue can also hinder the future development and deployment of deep learning. Thus, more efforts have been made recently in analyzing the convergence of the gradient descent method in network learning.

For ease of analysis, most results have considered only shallow networks with one or two hidden layers and different techniques have been used to prove the convergence in learning of the shallow networks [16]–[20]. In [16], the convergence of a two-layer network was analyzed by adding an identity mapping to the standard structure. For networks with standard structures, most studies have considered over-parameterization networks [17], [18] where the widths of the hidden layers are assumed to be very large so that the problem can be analyzed mathematically. Moderating the over-parameterization property was the aim of some recent studies [19], [20] but the results are still limited to shallow networks.

Recently, few works have been devoted to analyzing the convergence issue in deep networks [21]–[23]. In [21], an analytic layer-wise learning framework with guarantee of convergence was developed for multilayer fully connected networks. The learning algorithm can be applied to both classification and regression problems in offline and online robotic applications such as real-time robot control. However, the framework is limited to fully connect networks (FCNs), which therefore cannot be used for full CNNs whose structure is different from the FCNs. In [22], [23], over-parameterized networks were analyzed for deep learning by assuming a huge width in each inner layer. Although these works have contributed towards the theoretical understanding of deep learning networks trained by gradient descent, there is still a huge gap between these theoretical analyses and practical experiments. In particular, the theoretical assumption of having huge width in deep networks has made it impractical. As a result, there were also no experimental results to support the theories given in those studies [22], [23].

In this paper, we develop a layer-wise theoretical framework for learning deep convolutional neural networks. As compared to fully connected networks with only dense weight matrices, the weight sharing in convolutional filters and the presence of pooling layers in CNNs create a unique problem which cannot be directly solved by techniques used in FCNs. The convolutional neural networks are in fact a more general model for image classifications as they consist of convolutional layers, pooling layers and also fully connected layers. In this paper, we explicitly introduce the matrix representations for different types of layers in CNNs to derive a general model of deep CNNs. The weight sharing property of the convolutional layers is clearly formulated in the model. Based on the model, a layer-wise learning algorithm for CNNs is proposed and the convergence is analyzed. The proposed method does not require the assumption of over-parameterization. The method is then tested on several common image datasets and the results show a reasonable trade-off between test accuracy and analytic learning. Though there is a trade-off in test accuracies in some case studies, the results also show that some deep CNNs may not need as many convolutional layers as in their original structure to achieve reasonable accuracies. This demonstrates the possibility of using the layer-wise learning method as an indicator to determine the appropriate number of layers in final implementations of the models.

The rest of this paper is organized as follows. Section II presents the problem formation where the maxtrix representations of the mathematical operations taking place in CNNs are introduced and the full equation of CNN models is derived. Section III presents the learning algorithms where the theoretical analysis is provided to prove the convergence of the proposed algorithms. Section IV presents the case studies where different CNN structures and databases are employed to demonstrate the efficacy of the proposed method.

II. PROBLEM FORMATION

We consider the problem of supervised learning with a dataset containing input images and their labels. The aim is to approximate the true mapping between the input images and their labels for predicting the labels of unknown images. Denoting $\boldsymbol{X} \in \mathbb{R}^{n\times n\times n_c}$ as an input image (where n is the width and also the height, and n_c is the number of channels (or depth) of the image), and $\boldsymbol{y} \in \mathbb{R}^p$ as the output variable representing the image's label (where p is the number of classes in the classification problem), the true mapping $\boldsymbol{f}$ can be described as

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{X}) \tag{1}$$

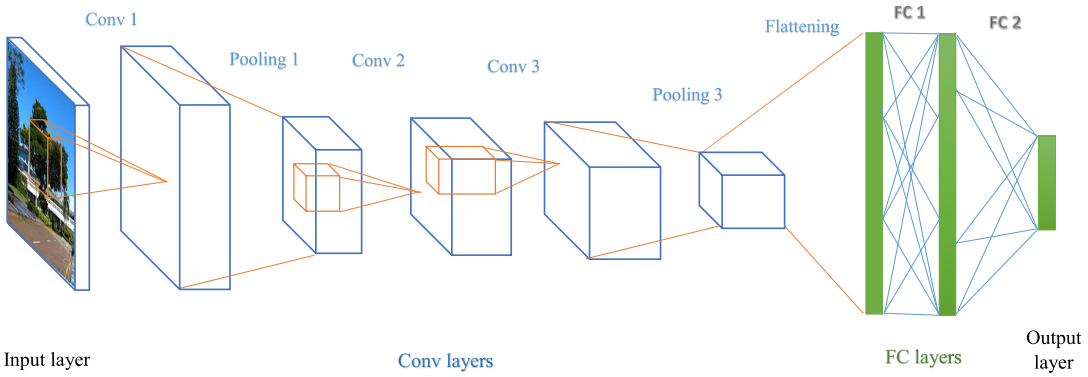


FIGURE 1. An example of a CNN with 3 convolutional layers and 2 fully connected layers.

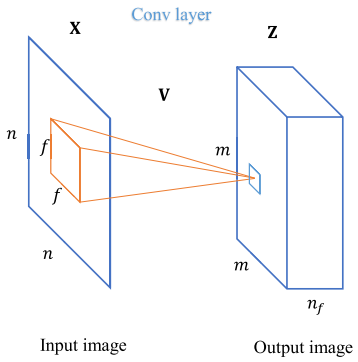


FIGURE 2. Convolution with 2D Images.

The mapping $\boldsymbol{f}$ in (1) is to be approximated by a CNN. An example of CNNs is shown in Fig. 1. Normally, a CNN can be divided into 2 main parts: convolutional (Conv) part which includes convolutional layers and pooling layers, and fully connected (FC) part which includes fully connected (or dense) layers. Between the 2 parts, there is a flattening operation which reshapes the 3D volume of neurons into a 1D list of neurons. For simplicity, the pooling layers in our work are accompanied with the preceding Conv layers and referred to as pooling operations within the Conv layers. The Conv part can thus be referred to as Conv layers, and the FC part can also be referred to as FC layers.

In this section, we introduce the matrix representations of various operations in a CNN, including convolution and pooling in a Conv layer, and flattening at the end of the Conv part.

A. MATRIX REPRESENTATIONS

1) CONVOLUTION WITH 2D IMAGES

Before considering the more complex 3D inputs, we first study the case of 2D inputs where $n_c = 1$. Examples of 2D inputs are black and white images which can be found in MNIST [24] or Fashion MNIST datasets [25]. The convolution operation with a 2D image is illustrated in Fig 2.

Given an input image of size $n \times n$ (pixels) with n being the width (and also the height) of the image and a filter with a dimension of $f \times f$ as shown in Fig. 3a and Fig. 3b respectively.

Convolving the input image with the filter results in an $m \times m$ convolution output where $m = (n - f + 2p)/s + 1$ with p being the size of zero padding and s being the stride of the convolution operation. Fig. 3c shows the 2D convolution output with size of $m \times m$. For simplicity of presentation, we present the case of no padding and stride 1, which means $m = n - f + 1$. Despite that, the matrix representations for the general case are still the same. The resulting output elements of the convolution operation can be computed as follows

$$\begin{aligned} z_{11} &= v_{11}x_{11} + v_{12}x_{12} + \cdots + v_{1f}x_{1f} \\ &\quad + v_{21}x_{21} + v_{22}x_{22} + \cdots + v_{2f}x_{2f} \\ &\quad + \cdots \\ &\quad + v_{f1}x_{f1} + v_{f2}x_{f2} + \cdots + v_{ff}x_{ff} \end{aligned} \tag{2}$$

$$\begin{aligned} z_{12} &= v_{11}x_{12} + v_{12}x_{13} + \cdots + v_{1f}x_{1(f+1)} \\ &\quad + v_{21}x_{22} + v_{22}x_{23} + \cdots + v_{2f}x_{2(f+1)} \\ &\quad + \cdots \\ &\quad + v_{f1}x_{f2} + v_{f2}x_{f3} + \cdots + v_{ff}x_{f(f+1)} \end{aligned} \tag{3}$$

$$\vdots$$

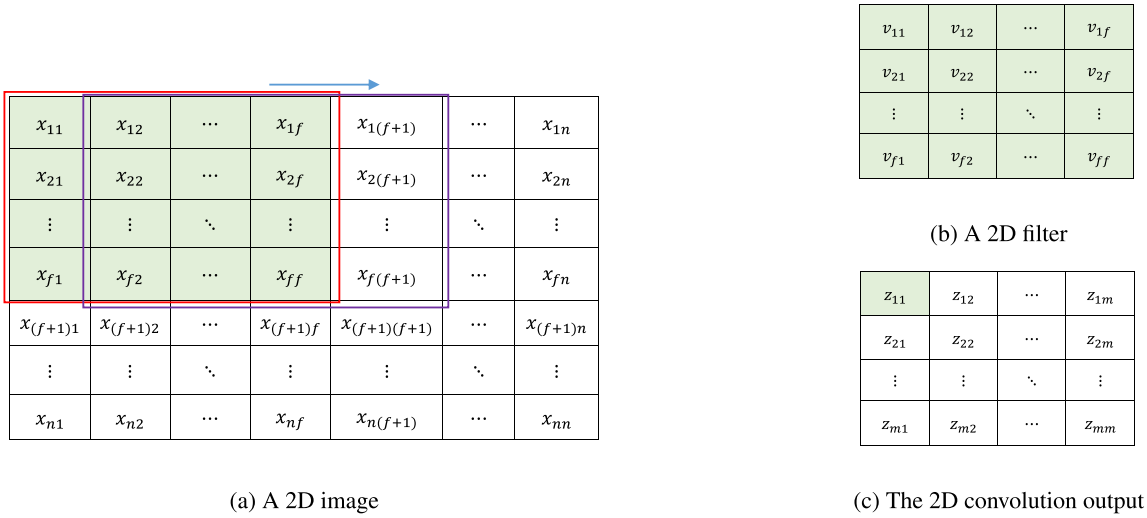
$$\begin{aligned} z_{1m} &= v_{11}x_{1m} + v_{12}x_{1(m+1)} + \cdots + v_{1f}x_{1n} \\ &\quad + v_{21}x_{2m} + v_{22}x_{2(m+1)} + \cdots + v_{2f}x_{2n} \\ &\quad + \cdots \\ &\quad + v_{f1}x_{fm} + v_{f2}x_{f(m+1)} + \cdots + v_{ff}x_{fn} \end{aligned} \tag{4}$$

$$\vdots$$

$$\begin{aligned} z_{mm} &= v_{11}x_{mm} + v_{12}x_{m(m+1)} + \cdots + v_{1f}x_{mn} \\ &\quad + v_{21}x_{(m+1)m} + v_{22}x_{(m+1)(m+1)} + \cdots + v_{2f}x_{(m+1)n} \\ &\quad + \cdots \\ &\quad + v_{f1}x_{nm} + v_{f2}x_{n(m+1)} + \cdots + v_{ff}x_{nn} \end{aligned} \tag{5}$$

Equation (2) can be rewritten as

$$z_{11} = \boldsymbol{x}_{11}^{f}\boldsymbol{v} \tag{6}$$



(a) A 2D image

(b) A 2D filter

(c) The 2D convolution output

FIGURE 3. The input, filter and output in convolution with 2D images.

where $\boldsymbol{x}_{11}^f = [x_{11}, x_{12}, \cdots, x_{1f}, x_{21}, x_{22}, \cdots, x_{2f}, \cdots, x_{f1}, x_{f2}, \cdots, x_{ff}] \in \mathbb{R}^{1\times f^2}$, and

$$\boldsymbol{v} = [v_{11}, v_{12}, \cdots, v_{1f}, v_{21}, v_{22}, \cdots, v_{2f}, , \cdots, v_{f1}, v_{f2}, \cdots, v_{ff}]^T \in \mathbb{R}^{f^2\times 1} \quad (7)$$

Similarly, we have

$$z_{12} = \boldsymbol{x}_{12}^f \boldsymbol{v} \quad (8)$$

$$\vdots$$

$$z_{mm} = \boldsymbol{x}_{mm}^f \boldsymbol{v} \quad (9)$$

Generally, for $i, j = 1..m$,

$$z_{ij} = \boldsymbol{x}_{ij}^f \boldsymbol{v} \quad (10)$$

where $\boldsymbol{x}_{ij}^f = [x_{ij}, x_{i(j+1)}, \cdots, x_{i(j+f-1)}, x_{(i+1)j}, \cdots, x_{(i+f-1)(j+f-1)}]$. Noting that $\boldsymbol{x}_{ij}^f$ is simply a row vector (with f^2 elements) containing all the pixels of the input image that correspond to the output neuron at the i^{th} row and j^{th} column. In other words, $\boldsymbol{x}_{ij}^f$ is the vector form of the neuron's receptive field.

We can combine all z_{ij} and the equations for the output in a compact form can be given as follows

$$\boldsymbol{z} = \mathbf{X}^f \boldsymbol{v} \quad (11)$$

where

$$\boldsymbol{z} = [z_{11}, z_{12}, \cdots, z_{1m}, z_{21}, \cdots, z_{mm}]^T \in \mathbb{R}^{m^2\times 1} \quad (12)$$

is a vector which represents the convolution output, and

$$\mathbf{X}^f = [\boldsymbol{x}_{11}^f, \boldsymbol{x}_{12}^f, \cdots, \boldsymbol{x}_{1m}^f, \cdots, \boldsymbol{x}_{mm}^f]^T \in \mathbb{R}^{m^2\times f^2} \quad (13)$$

$$= \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1f} & x_{21} & \cdots & x_{ff} \\ x_{12} & x_{13} & \cdots & x_{1(f+1)} & x_{22} & \cdots & x_{f(f+1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_{1m} & x_{1(m+1)} & \cdots & x_{1n} & x_{2m} & \cdots & x_{fn} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_{mm} & x_{m(m+1)} & \cdots & x_{mn} & x_{(m+1)m} & \cdots & x_{nn} \end{bmatrix} \quad (14)$$

is defined as the *filter-dependent input matrix*, which is constructed from the input image based on the filter design, and $\boldsymbol{v} \in \mathbb{R}^{f^2\times 1}$ defined in (7) represents the unknown weights of the filter.

When there are n_f filters to exact n_f features from the input images as shown in Fig. 2, equation (11) can be extended as follows

$$\mathbf{Z} = \mathbf{X}^f \mathbf{V} \quad (15)$$

where $\mathbf{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_{n_f}] \in \mathbb{R}^{f^2\times n_f}$ with $\boldsymbol{v}_i$ representing the weights of the i^{th} filter, and $\mathbf{Z} = [\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_{n_f}] \in \mathbb{R}^{m^2\times n_f}$ with $\boldsymbol{z}_i$ being the resulting output feature when convoluting the input with the i^{th} filter.

2) CONVOLUTION WITH 3D INPUTS

We now consider an input of size $n \times n \times n_c$ with n being the width (and also the height) and n_c being the number of channels (or the depth). For the input layer of a CNN, n_c should be 1 for gray images (like previous subsection) and 3 for RGB color images. To do the convolution, each filter should have the dimension of $f \times f \times n_c$. The convolution operation with 3D inputs is illustrated in Fig. 4.

Similar to the 2D inputs, it is possible to represent the convolution operations in a similar form as in (10) by defining $\boldsymbol{x}_{ij}^f$ and $\boldsymbol{v}$ as follows

$$\boldsymbol{x}_{ij}^f = [x_{ij1}, x_{ij2}, \cdots, x_{ijn_c}, x_{i(j+1)1}, \cdots, x_{i(j+f-1)n_c}, x_{(i+1)j1}, \cdots, x_{(i+f-1)(j+f-1)n_c}] \in \mathbb{R}^{1\times f^2 n_c} \quad (16)$$

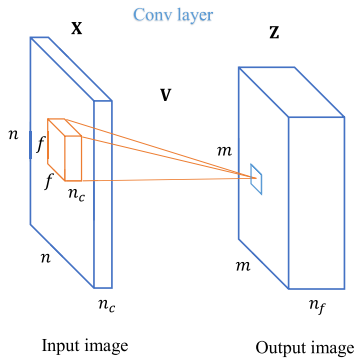


FIGURE 4. Convolution with 3D inputs.

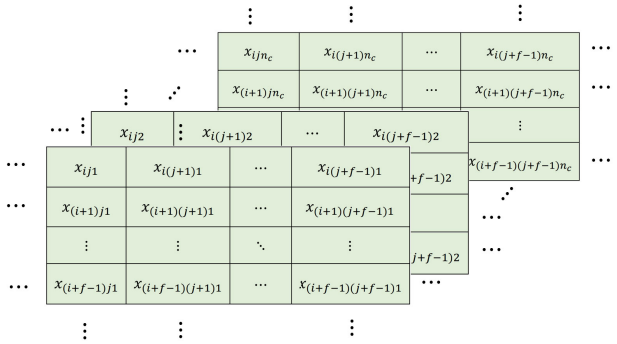


FIGURE 5. The part of the 3D input image (receptive field) that corresponds to the neuron at the i^{th} row and j^{th} column of the output.

and

$$\boldsymbol{v} = [v_{111}, v_{112}, \cdots, v_{11n_c}, v_{121}, \cdots, v_{1fn_c}, v_{211}, \cdots, v_{ffn_c}]^T \in \mathbb{R}^{f^2 n_c \times 1} \quad (17)$$

where x_{ijk} and v_{ijk} are respectively the pixel values of the input as seen in Fig. 5 and the weights of the filter as seen in Fig. 6. Again, noting that $\boldsymbol{x}_{ij}^f$ is simply a row vector (with $f^2 \cdot n_c$ elements) containing all the pixels in the receptive field of the i^{th} row, j^{th} column output neuron, and $\boldsymbol{v}$ is a column vector (with $f^2 \cdot n_c$ elements) containing all the weights of the filter. Hence, the output can be calculated as

$$z_{ij} = \boldsymbol{x}_{ij}^f \boldsymbol{v} \quad (18)$$

Similarly, denoting $\boldsymbol{z} \in \mathbb{R}^{m^2 \times 1}$ (a column vector of m^2 elements) and $\mathbf{X}^f \in \mathbb{R}^{m^2 \times f^2 n_c}$ (a matrix of m^2 rows and $f^2 \cdot n_c$ columns) as in (12) and (13) respectively, we also have

$$\boldsymbol{z} = \mathbf{X}^f \boldsymbol{v} \quad (19)$$

When there are n_f filters to exact n_f features from the input images, equation (19) can be extended as follows

$$\mathbf{Z} = \mathbf{X}^f \mathbf{V} \quad (20)$$

where $\mathbf{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_{n_f}] \in \mathbb{R}^{f^2 n_c \times n_f}$ with $\boldsymbol{v}_i$ representing the weights of the i^{th} filter, and $\mathbf{Z} = [\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_{n_f}] \in \mathbb{R}^{m^2 \times n_f}$ with $\boldsymbol{z}_i$ being the resulting output feature when convoluting the input with the i^{th} filter.

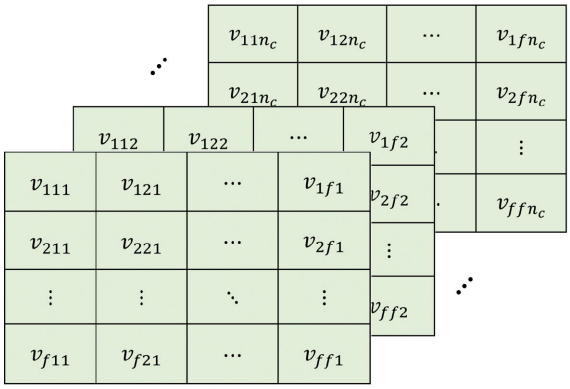


FIGURE 6. A 3D filter.

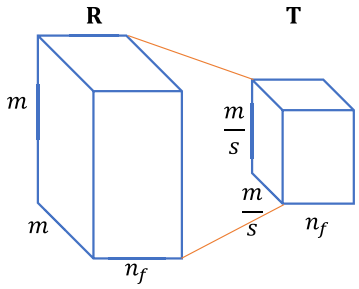


FIGURE 7. Pooling with 3D inputs.

3) ACTIVATION FUNCTION

After the convolution operation, an activation function is applied for each element of the resulting matrix $\mathbf{Z}$ in (20) to produce the matrix $\mathbf{R}$ which has the same dimension as $\mathbf{Z}$ (m^2 rows and n_f columns).

$$\mathbf{R} = \boldsymbol{\Phi}(\mathbf{Z}) = \boldsymbol{\Phi}(\mathbf{X}^f \mathbf{V}) \quad (21)$$

A commonly used activation function is ReLU [26].

4) POOLING OPERATIONS

The function of the pooling layer is to down-sample the feature maps in each channel. A kernel of size $k_p \times k_p$ is used to summary the key feature of the region it covers in each channel of the feature maps. The kernel is moved with a stride of $s \times s$. We consider the case where the kernel size and the stride of the pooling layer are equal, or $k_p = s$. In this case, with an input of size $m \times m \times n_f$ as in Fig. 7, the output of the pooling operation has a size of $\frac{m}{s} \times \frac{m}{s} \times n_f$. With $\mathbf{R}$ being the matrix representation of the pooling input (output of equation (21) above), $\mathbf{T}$ being the matrix representation of the pooling output, the pooling operation can be represented as follows:

$$\begin{aligned} \mathcal{P} &: \mathbf{R} \rightarrow \mathbf{T} \\ \mathbb{R}^{m^2 \times n_f} &\rightarrow \mathbb{R}^{\frac{m^2}{s^2} \times n_f} \end{aligned} \quad (22)$$

As the pooling operation is done independently for each channel, it can be represented as follows for the i^{th} channel

of the pooling input $\mathbf{R}$ (in total n_f channels),

$$\begin{aligned} \mathcal{P}: \boldsymbol{r}_i &\to \boldsymbol{t}_i \\ \mathbb{R}^{m^2\times 1} &\to \mathbb{R}^{\frac{m^2}{s^2}\times 1} \end{aligned} \tag{23}$$

where $\boldsymbol{r}_i$ is the i^{th} column of $\mathbf{R}$, and $\boldsymbol{t}_i$ is the i^{th} column of $\mathbf{T}$.

For example, an image with the size of 4×4, after pooling with kernel size 2×2:

$$\begin{array}{|cc|cc|}\hline r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ \hline r_{31} & r_{32} & r_{33} & r_{34} \\ r_{41} & r_{42} & r_{43} & r_{44} \\ \hline \end{array} \to \begin{array}{|cc|}\hline t_{11} & t_{12} \\ t_{21} & t_{22} \\ \hline \end{array}$$

using the vector representation gives: $\mathcal{P}: \ \mathbb{R}^{16\times 1} \to \mathbb{R}^{4\times 1}$

$$\begin{array}{|c|}\hline r_{11} \\ r_{12} \\ r_{13} \\ r_{14} \\ r_{21} \\ r_{22} \\ r_{23} \\ r_{24} \\ r_{31} \\ r_{32} \\ r_{33} \\ r_{34} \\ r_{41} \\ r_{42} \\ r_{43} \\ r_{44} \\ \hline \end{array} \to \begin{array}{|c|}\hline t_{11} \\ t_{12} \\ t_{21} \\ t_{22} \\ \hline \end{array} = \begin{array}{|c|}\hline f(r_{11}, r_{12}, r_{21}, r_{22}) \\ f(r_{13}, r_{14}, r_{23}, r_{24}) \\ f(r_{31}, r_{32}, r_{41}, r_{42}) \\ f(r_{33}, r_{34}, r_{43}, r_{44}) \\ \hline \end{array}$$

Two common types of pooling layers are: average pooling and max pooling.

For average pooling, we have

$$\begin{array}{|c|}\hline t_{11} \\ t_{12} \\ t_{21} \\ t_{22} \\ \hline \end{array} = \begin{array}{|l|}\hline \frac{1}{4}(r_{11}+r_{12}+r_{21}+r_{22}) \\ d\,\frac{1}{4}(r_{13}+r_{14}+r_{23}+r_{24}) \\ d\,\frac{1}{4}(r_{31}+r_{32}+r_{41}+r_{42}) \\ \frac{1}{4}(r_{33}+r_{34}+r_{43}+r_{44}) \\ \hline \end{array}$$

For max pooling, we have

$$\begin{array}{|c|}\hline t_{11} \\ t_{12} \\ t_{21} \\ t_{22} \\ \hline \end{array} = \begin{array}{|l|}\hline \max(r_{11}, r_{12}, r_{21}, r_{22}) \\ \max(r_{13}, r_{14}, r_{23}, r_{24}) \\ \max(r_{31}, r_{32}, r_{41}, r_{42}) \\ \max(r_{33}, r_{34}, r_{43}, r_{44}) \\ \hline \end{array}$$

a, Average pooling $\mathcal{P}_{avg}$

It can be shown that

$$\mathcal{P}_{avg}(\boldsymbol{r}) = \mathbf{\Gamma}_{avg}\boldsymbol{r} \tag{24}$$

where $\mathbf{\Gamma}_{avg} \in \mathbb{R}^{(m^2/s^2)\times m^2}$ is a matrix that does not depend on the input vector $\boldsymbol{r}$. Indeed, with kernel size of $s\times s$ we have

$$\begin{aligned} \mathcal{P}_{avg}(\boldsymbol{r}) &= \mathcal{P}_{avg}\left(\begin{bmatrix} r_{11} \\ r_{12} \\ \vdots \\ r_{mm} \end{bmatrix}\right) \\ &= \begin{bmatrix} \frac{1}{s^2}(r_{11}+r_{12}+\cdots+r_{ss}) \\ \vdots \\ \frac{1}{s^2}(r_{(m-s+1)(m-s+1)}+\cdots+r_{mm}) \end{bmatrix} \\ &= \mathbf{\Gamma}_{avg}\boldsymbol{r} \end{aligned} \tag{25}$$

where

$$\begin{aligned} &\mathbf{\Gamma}_{avg} \\ &= \begin{bmatrix} \frac{1}{s^2} & \cdots & \frac{1}{s^2} & 0 & \cdots & 0 & \frac{1}{s^2} & \cdots & \frac{1}{s^2} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & \frac{1}{s^2} \end{bmatrix} \end{aligned} \tag{26}$$

and

$$\begin{aligned} \boldsymbol{r} = [&r_{11}, \cdots, r_{1s}, r_{1(s+1)}, \cdots, r_{(s-1)m}, r_{s1}, \cdots, r_{ss}, \\ &r_{s(s+1)}, \cdots, r_{mm}]^T \end{aligned} \tag{27}$$

b, Max pooling $\mathcal{P}_{max}$

It can be shown that

$$\mathcal{P}_{max}(\boldsymbol{r}) = \mathbf{\Gamma}_{max}(\boldsymbol{r})\boldsymbol{r} \tag{28}$$

where $\mathbf{\Gamma}_{max}(\boldsymbol{r}) \in \mathbb{R}^{(m^2/s^2)\times m^2}$.

$$\begin{aligned} \mathcal{P}_{max}(\boldsymbol{r}) &= \mathcal{P}_{max}\left(\begin{bmatrix} r_{11} \\ r_{12} \\ \vdots \\ r_{mm} \end{bmatrix}\right) \\ &= \begin{bmatrix} \max(r_{11}, \cdots, r_{1s}, \cdots, r_{s1}, \cdots, r_{ss}, \cdots) \\ \vdots \\ \max(r_{(m-1)(m-1)}, r_{(m-1)m}, r_{m(m-1)}, r_{mm}) \end{bmatrix} \\ &= \mathbf{\Gamma}_{max}(\boldsymbol{r})\boldsymbol{r} \end{aligned} \tag{29}$$

where

$$\begin{aligned} &\mathbf{\Gamma}_{max}(\boldsymbol{r}) \\ &= \begin{bmatrix} \tilde{1}_{11} & \cdots & \tilde{1}_{1s} & 0 & \cdots & 0 & \tilde{1}_{s1} & \cdots & \tilde{1}_{ss} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & \tilde{1}_{mm} \end{bmatrix} \end{aligned} \tag{30}$$

where $\tilde{1}_{ij}$ is either 0 or 1. $\tilde{1}_{ij} = 1$ if $\max(\cdots, r_{ij}, \cdots)$ is r_{ij} and $\tilde{1}_{ij} = 0$ otherwise; and

$$\begin{aligned} \boldsymbol{r} = [&r_{11}, \cdots, r_{1s}, r_{1(s+1)}, \cdots, r_{(s-1)m}, r_{s1}, \cdots, r_{ss}, \\ &r_{s(s+1)}, \cdots, r_{mm}]^T \end{aligned} \tag{31}$$

c, General representation of pooling $\mathcal{P}$: In general, both (24) and (28) can be written as

$$\boldsymbol{t} = \mathcal{P}(\boldsymbol{r}) = \mathbf{\Gamma}(\boldsymbol{r})\boldsymbol{r} \tag{32}$$

or simply

$$\boldsymbol{t} = \mathcal{P}(\boldsymbol{r}) = \mathbf{\Gamma}\boldsymbol{r} \tag{33}$$

For n_f channels, we have

$$\mathbf{T} = \mathcal{P}(\mathbf{R}) = \mathbf{\Gamma}\mathbf{R} \tag{34}$$

Combining with (21), we have

$$\mathbf{T} = \mathcal{P}(\mathbf{\Phi}(\mathbf{X}^f\mathbf{V})) = \mathbf{\Gamma}\mathbf{\Phi}(\mathbf{X}^f\mathbf{V}) \tag{35}$$

Noting that equation (35) can also be used for the convolutional layer that does not have the pooling operation by assuming $s = 1$. In this case, the matrix $\mathbf{\Gamma}$ is an identity matrix.

5) FLATTENING OPERATION

In the end of the convolutional part of a CNN, a flattening operation is required to convert the 3D output feature map into a list of elements before being fed to the fully connected layers. For the matrix representation, the flattening operation is just a rearrangement of the elements in the matrix $\mathbf{T} \in \mathbb{R}^{(m^2/s^2)\times n_f}$ into a vector. We can write the matrix $\mathbf{T}$ as

$$\begin{aligned}\mathbf{T} &= \begin{bmatrix}\boldsymbol{t}_1 & \boldsymbol{t}_2 & \cdots & \boldsymbol{t}_{n_f}\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{\Gamma}_1\boldsymbol{r}_1 & \mathbf{\Gamma}_2\boldsymbol{r}_2 & \cdots & \mathbf{\Gamma}_{n_f}\boldsymbol{r}_{n_f}\end{bmatrix}\end{aligned} \tag{36}$$

where $\boldsymbol{t}_i$ is the i^{th} channel of the output of pooling layer. The flattening operation can be expressed as

$$\boldsymbol{\varphi} = F[\mathbf{T}] = \begin{bmatrix}\boldsymbol{t}_1 \\ \boldsymbol{t}_2 \\ \vdots \\ \boldsymbol{t}_{n_f}\end{bmatrix} = \begin{bmatrix}\mathbf{\Gamma}_1\boldsymbol{r}_1 \\ \mathbf{\Gamma}_2\boldsymbol{r}_2 \\ \vdots \\ \mathbf{\Gamma}_{n_f}\boldsymbol{r}_{n_f}\end{bmatrix} \tag{37}$$

B. EQUATION OF DEEP CONVOLUTIONAL NEURAL NETWORKS

We consider a deep CNN with n_{conv} convolutional layers and n_{FC} fully connected layers as illustrated in Fig. 8. At the j^{th} conv layer, denoting n_j as the width and the height, n_{cj} as the number of channels (or the depth) of the input volume, n_{f_j} as the number of filters, m_j as the width and the height of the output volume after the convolution operation, and s_j as the stride of the pooling operation. The output volume of the j^{th} conv layer can be computed by

$$\mathbf{X}_{j+1} = \mathbf{T}_j = \mathcal{P}\left(\mathbf{\Phi}_j(\mathbf{X}_j^{f_j}\mathbf{V}_j)\right) = \mathbf{\Gamma}_j\mathbf{\Phi}_j(\mathbf{X}_j^{f_j}\mathbf{V}_j) \quad \text{for } j = 1..n_{conv}. \tag{38}$$

where $\mathbf{X}_{j+1} \in \mathbb{R}^{\frac{m_j^2}{s_j^2}\times n_{f_j}}$ or $\mathbb{R}^{n_{j+1}^2\times n_{c(j+1)}}$ (at the $(j+1)^{\text{th}}$ conv layer, the width of the input is $n_{j+1} = \frac{m_j^2}{s_j^2}$ and the depth is $n_{c(j+1)} = n_{f_j}$). Noting that the input volume $\mathbf{X}_j \in \mathbb{R}^{n_j^2\times n_{cj}}$ is different from the filter-dependent matrix $\mathbf{X}_j^{f_j} \in \mathbb{R}^{m_j^2\times f_j^2 n_{cj}}$. For the convolutional layer that does not have the pooling operation, $\mathbf{\Gamma}_j$ is an identity matrix. At the last conv layer, we have

$$\begin{aligned}\mathbf{T}_{n_{conv}} &= \mathcal{P}\left(\mathbf{\Phi}_{n_{conv}}(\mathbf{X}_{n_{conv}}^{f_{n_{conv}}}\mathbf{V}_{n_{conv}})\right) \\ &= \mathbf{\Gamma}_{n_{conv}}\mathbf{\Phi}_{n_{conv}}(\mathbf{X}_{n_{conv}}^{f_{n_{conv}}}\mathbf{V}_{n_{conv}})\end{aligned} \tag{39}$$

The flattening operation occurs at this layer, the input of the FC part is hence computed as follows

$$\begin{aligned}\boldsymbol{\varphi} &= F\left\{\mathbf{T}_{n_{conv}}\right\} \\ &= F\left\{\mathbf{\Gamma}_{n_{conv}}\mathbf{\Phi}_{n_{conv}}(\mathbf{X}_{n_{conv}}^{f_{n_{conv}}}\mathbf{V}_{n_{conv}})\right\}\end{aligned} \tag{40}$$

The output of the CNN can thus be computed as

$$\boldsymbol{y}_{CNN} = \boldsymbol{\varphi}_{n_{FC}}\Big(\mathbf{W}_{n_{FC}}\cdots\boldsymbol{\varphi}_2(\mathbf{W}_2\boldsymbol{\varphi}_1(\mathbf{W}_1\boldsymbol{\varphi}))\cdots\Big) \tag{41}$$

where $\mathbf{W}_j$ and $\boldsymbol{\varphi}_j$ are the weight matrix and the vector of activation funtions at the j^{th} FC layer respectively. The output activation function vector can also be denoted as $\boldsymbol{\sigma} \triangleq \boldsymbol{\varphi}_{n_{FC}}$.

Therefore, the overall equation is given as

$$\begin{aligned}&\boldsymbol{y}_{CNN} \\ &= \boldsymbol{\sigma}\Big(\mathbf{W}_{n_{FC}}\cdots\boldsymbol{\varphi}_1\Big(\mathbf{W}_1 F\left\{\mathbf{\Gamma}_{n_{conv}}\mathbf{\Phi}_{n_{conv}}\right. \\ &\quad\left.\times([\cdots[\mathbf{\Gamma}_2\mathbf{\Phi}_2([\mathbf{\Gamma}_1\mathbf{\Phi}_1(\mathbf{X}_1^{f_1}\mathbf{V}_1)]^{f_2}\mathbf{V}_2)]\cdots]^{f_{n_{conv}}}\mathbf{V}_{n_{conv}})\right\}\Big)\cdots\Big)\end{aligned} \tag{42}$$

III. FORWARD PROGRESSIVE LEARNING OF CONVOLUTIONAL NEURAL NETWORKS

A. FORMULATION

In forward progressive learning (FPL) of CNNs, the entire CNN is trained part by part in sequence. That is, the convolutional part is trained first and then the fully connected part is trained afterwards. The convolutional part of the CNN is learned in a layer-wise manner as illustrated in Fig. 9. Each time, a convolutional subnet which contains 1 convolutional layer and 1 FC layer is trained. The subnet, which can be referred to as *two-layer training CNN* in this work, is detailed in Fig. 10. The presence of the pooling action in the subnet depends on whether the convolutional layer in the original structure of the entire CNN has the pooling action or not. When learning the subnet of the j^{th} convolutional layer with filter weight matrix $\mathbf{V}_j$, an FC layer with pseudo weight matrix $\mathbf{W}_j^{\triangleright}$ is used. After the subnet is trained, the FC layer is discarded. $\mathbf{V}_j$ is then frozen and the computed output $\mathbf{T}_j$ of the j^{th} convolutional layer is used as the input $\mathbf{X}_{j+1}$ of the subnet of the next convolutional layer. The learning process continues until reaching the last convolutional layer.

The subsequent subsection will introduce in details the subnets (*two-layer training CNNs*) in the FPL of CNNs.

B. TWO-LAYER TRAINING CNNs

Fig. 10 shows a two-layer training CNN as the subnet of the j^{th} convolutional layer of the entire CNN, with one convolutional layer and one FC layer. The output of the two-layer training CNN that contains one convolutional layer and one fully connected layer can be expressed as follows

$$\begin{aligned}\boldsymbol{y}_{trCNN} &= \boldsymbol{\sigma}\left(\mathbf{W}_j^{\triangleright}\boldsymbol{\varphi}_j\right) \\ &= \boldsymbol{\sigma}\left(\mathbf{W}_j^{\triangleright}F(\mathbf{\Gamma}_j\mathbf{\Phi}_j(\mathbf{X}_j^{f_j}\mathbf{V}_j))\right)\end{aligned} \tag{43}$$

where $\boldsymbol{\varphi}_j = F(\mathbf{\Gamma}_j\mathbf{\Phi}_j(\mathbf{X}_j^{f_j}\mathbf{V}_j))$.

Denoting n_j as the width and the height, n_{cj} as the number of channels (or the depth) of the input volume, f_j as the kernel size of the filters, n_{f_j} as the number of filters, m_j as the width and the height of the output volume after the convolution operation, and s_j as the stride of the pooling operation, n_{flj} as the number of neurons after the flattening

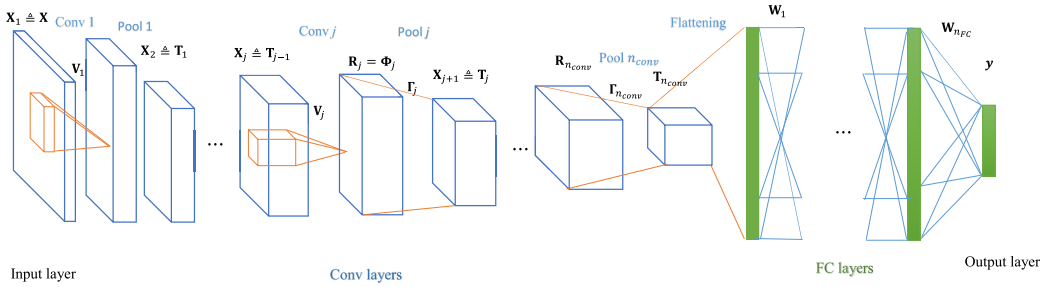


FIGURE 8. A deep CNN with n_{conv} convolutional layers and n_{FC} fully connected layers.

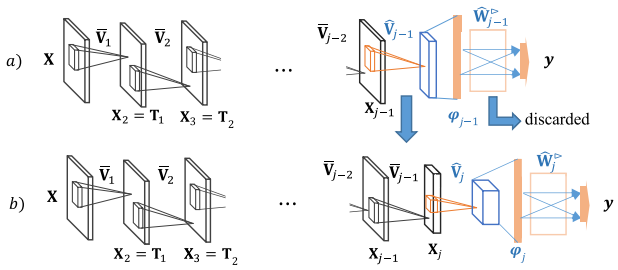


FIGURE 9. The forward progressive learning of CNNs.

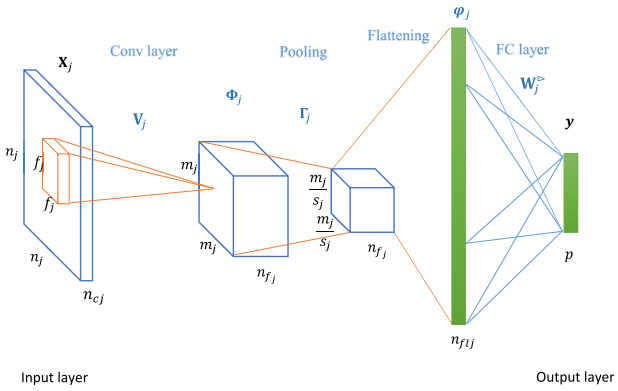


FIGURE 10. A *two-layer training CNN*: The subnet of the j^{th} conv layer.

operation. Hence, $\mathbf{X}_j^{f_j} \in \mathbb{R}^{m_j^2 \times f_j^2 n_{cj}}$ is the filter-dependent input matrix, $\mathbf{V}_j \in \mathbb{R}^{f_j^2 n_{cj} \times n_{fj}}$ is the filter matrix, F is the denotation for the flattening operation in which the matrix $\mathbb{R}^{(m_j^2/s_j^2) \times n_{fj}}$ becomes $\mathbb{R}^{n_{flj} \times 1}$ ($n_{flj} = \frac{m_j^2}{s_j^2} n_{fj}$), $\boldsymbol{\varphi}_j \in \mathbb{R}^{n_{flj} \times 1}$ is the vector after flattening, $\mathbf{W}_j^{\triangleright} \in \mathbb{R}^{p \times n_{flj}}$ is the FC pseudo weight matrix, and $\boldsymbol{y} \in \mathbb{R}^{p \times 1}$ is the output vector of the network.

The flattening operation can be expressed as

$$\boldsymbol{\varphi}_j = F[\mathbf{T}_j] = \begin{bmatrix} \boldsymbol{t}_{j,1} \\ \boldsymbol{t}_{j,2} \\ \vdots \\ \boldsymbol{t}_{j,n_{fj}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Gamma}_{j,1} \boldsymbol{\phi}_{j,1}(\mathbf{X}_j^{f_j} \boldsymbol{v}_{j,1}) \\ \boldsymbol{\Gamma}_{j,2} \boldsymbol{\phi}_{j,2}(\mathbf{X}_j^{f_j} \boldsymbol{v}_{j,2}) \\ \vdots \\ \boldsymbol{\Gamma}_{j,n_{fj}} \boldsymbol{\phi}_{j,n_{fj}}(\mathbf{X}_j^{f_j} \boldsymbol{v}_{j,n_{fj}}) \end{bmatrix} \tag{44}$$

Equation (43) can be rewritten as follows

$$\boldsymbol{y}_{trCNN} = \boldsymbol{\sigma}\left(\begin{bmatrix} \mathbf{W}_{j,1}^{\triangleright} & \mathbf{W}_{j,2}^{\triangleright} & \cdots & \mathbf{W}_{j,n_{fj}}^{\triangleright} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Gamma}_{j,1} \boldsymbol{\phi}_{j,1}(\mathbf{X}_j^{f_j} \boldsymbol{v}_{j,1}) \\ \boldsymbol{\Gamma}_{j,2} \boldsymbol{\phi}_{j,2}(\mathbf{X}_j^{f_j} \boldsymbol{v}_{j,2}) \\ \vdots \\ \boldsymbol{\Gamma}_{j,n_{fj}} \boldsymbol{\phi}_{j,n_{fj}}(\mathbf{X}_j^{f_j} \boldsymbol{v}_{j,n_{fj}}) \end{bmatrix} \right)$$

or

$$\boldsymbol{y}_{trCNN} = \boldsymbol{\sigma}\left(\sum_{i=1}^{n_{fj}} \mathbf{W}_{j,i}^{\triangleright} \boldsymbol{\Gamma}_{j,i} \boldsymbol{\phi}_{j,i}(\mathbf{X}_j^{f_j} \boldsymbol{v}_{j,i}) \right) \tag{45}$$

where $\mathbf{W}_{j,i}^{\triangleright} \in \mathbb{R}^{p \times (m_j^2/s_j^2)}$ denotes a sub-matrix of $\mathbf{W}_j^{\triangleright}$, $\boldsymbol{v}_{j,i}$ is the i^{th} column vector of matrix $\mathbf{V}_j$, and $\boldsymbol{\phi}_{j,i}$ is the i^{th} column vector of matrix $\boldsymbol{\Phi}_j$. It can be seen that $\boldsymbol{v}_{j,i}$ and $\boldsymbol{\phi}_{j,i}$ correspond to the weight of the i^{th} filter and the respective output ($i = 1..n_{fj}$).

Each *two-layer training CNN* is trained through 2 phases: Pre-training and fine-tuning. For the pre-training phase, the FC layer ($\mathbf{W}_j^{\triangleright}$) can be trained by using any standard learning algorithm for the output weights such as least square methods [27], [28] or one-layer update algorithm for FCNs [21]. The fine-tuning phase for the subnet is presented in the subsequent subsection.

C. FINE-TUNING ALGORITHM

In this section, we develop update laws to fine-tune concurrently the filter weights $\mathbf{V}_j$ and the pseudo output weights $\mathbf{W}_j^{\triangleright}$ of the *two-layer training CNN*. With sufficient neurons in the hidden layer, there exist optimal weight matrices $\mathbf{V}_j$ and $\mathbf{W}_j^{\triangleright}$ such that the output of the training CNN given in (43) can approximate the target $\boldsymbol{y}$ in (1). At the k^{th} learning step, we have

$$\boldsymbol{y}(k) = \boldsymbol{\sigma}\left(\mathbf{W}_j^{\triangleright} F[\boldsymbol{\Gamma}_j \boldsymbol{\Phi}_j(\mathbf{X}_j^{f_j}(k) \mathbf{V}_j)]\right) \tag{46}$$

The weight matrices $\mathbf{V}_j$ and $\mathbf{W}_j^{\triangleright}$ are updated incrementally by 2 update laws. Their estimated values at the k^{th} step of learning are denoted as $\hat{\mathbf{V}}_j(k)$ and $\hat{\mathbf{W}}_j^{\triangleright}(k)$ respectively. The estimated output $\hat{\boldsymbol{y}}(k)$ at the k^{th} step is defined as

$$\hat{\boldsymbol{y}}(k) = \boldsymbol{\sigma}\left(\hat{\mathbf{W}}_j^{\triangleright}(k) F[\hat{\boldsymbol{\Gamma}}_j(k) \boldsymbol{\Phi}_j(\mathbf{X}_j^{f_j}(k) \hat{\mathbf{V}}_j(k))]\right) \tag{47}$$

The output estimation error at the k^{th} step can be calculated by $\boldsymbol{e}(k) = \boldsymbol{y}(k) - \hat{\boldsymbol{y}}(k)$. Hence,

$$\boldsymbol{e}(k) = \boldsymbol{\sigma}\left(\mathbf{W}_j^{\triangleright} F[\boldsymbol{\Gamma}_j \boldsymbol{\Phi}_j(\mathbf{X}_j^{f_j}(k)\mathbf{V}_j)]\right) - \boldsymbol{\sigma}\left(\hat{\mathbf{W}}_j^{\triangleright}(k) F[\hat{\boldsymbol{\Gamma}}_j(k)\boldsymbol{\Phi}_j(\mathbf{X}_j^{f_j}(k)\hat{\mathbf{V}}_j(k))]\right) \quad (48)$$

Let

$$\boldsymbol{\delta}(k) \triangleq \mathbf{W}_j^{\triangleright} F[\boldsymbol{\Gamma}_j \boldsymbol{\Phi}_j(\mathbf{X}_j^{f_j}(k)\mathbf{V}_j)] - \hat{\mathbf{W}}_j^{\triangleright}(k) F[\hat{\boldsymbol{\Gamma}}_j(k)\boldsymbol{\Phi}_j(\mathbf{X}_j^{f_j}(k)\hat{\mathbf{V}}_j(k))] \quad (49)$$

which can be written as

$$\boldsymbol{\delta}(k) = \hat{\mathbf{W}}_j^{\triangleright}(k)\Delta\boldsymbol{\varphi}_j(k) + \Delta\mathbf{W}_j^{\triangleright}(k)\hat{\boldsymbol{\varphi}}_j(k) + \Delta\mathbf{W}_j^{\triangleright}(k)\Delta\boldsymbol{\varphi}_j(k) \quad (50)$$

where $\Delta\boldsymbol{\varphi}_j(k) = F[\boldsymbol{\Gamma}_j\boldsymbol{\Phi}_j(\mathbf{X}_j^{f_j}(k)\mathbf{V}_j)] - F[\hat{\boldsymbol{\Gamma}}_j(k)\boldsymbol{\Phi}_j(\mathbf{X}_j^{f_j}(k)\hat{\mathbf{V}}_j(k))]$, $\hat{\boldsymbol{\varphi}}_j(k) = F[\hat{\boldsymbol{\Gamma}}_j(k)\boldsymbol{\Phi}_j(\mathbf{X}_j^{f_j}(k)\hat{\mathbf{V}}_j(k))]$, and $\Delta\mathbf{W}_j^{\triangleright}(k) = \mathbf{W}_j^{\triangleright} - \hat{\mathbf{W}}_j^{\triangleright}(k)$.

Properties of $\Delta\boldsymbol{\varphi}_j(k)$. Similar to (45), we have

$$\hat{\mathbf{W}}_j^{\triangleright}(k)\Delta\boldsymbol{\varphi}_j(k) = \sum_{i=1}^{n_{f_j}} \hat{\mathbf{W}}_{j,i}^{\triangleright}(k)\Delta\boldsymbol{\varphi}_{j,i}(k) \quad (51)$$

where

$$\Delta\boldsymbol{\varphi}_{j,i}(k) = \boldsymbol{\Gamma}_{j,i}\boldsymbol{\phi}_{j,i}(\mathbf{X}_j^{f_j}(k)\boldsymbol{v}_{j,i}) - \hat{\boldsymbol{\Gamma}}_{j,i}(k)\boldsymbol{\phi}_{j,i}(\mathbf{X}_j^{f_j}(k)\hat{\boldsymbol{v}}_{j,i}(k)) \quad (52)$$

For average pooling, $\boldsymbol{\Gamma}_{j,i} = \hat{\boldsymbol{\Gamma}}_{j,i}(k)$. For max pooling, $\boldsymbol{\Gamma}_{j,i} \approx \hat{\boldsymbol{\Gamma}}_{j,i}(k)$ in the fine-tuning phase. In this phase, it is also possible to use the following property

$$\boldsymbol{\Gamma}_{j,i}\boldsymbol{\phi}_{j,i}(\mathbf{X}_j^{f_j}(k)\boldsymbol{v}_{j,i}) - \hat{\boldsymbol{\Gamma}}_{j,i}(k)\boldsymbol{\phi}_{j,i}(\mathbf{X}_j^{f_j}(k)\hat{\boldsymbol{v}}_{j,i}(k)) \approx \hat{\boldsymbol{\Gamma}}_{j,i}(k)\boldsymbol{\Phi}'_{j,i}(k)\mathbf{X}_j^{f_j}(k)\Delta\boldsymbol{v}_{j,i}(k) \quad (53)$$

where $\boldsymbol{\Phi}'_{j,i}(k) \in \mathbb{R}^{m_j^2 \times m_j^2}$ is a diagonal matrix whose diagonal entries are defined as

$$\hat{\phi}'_{j,i,l}(k) = \left.\frac{d\phi_{j,i,l}(x(k))}{dx(k)}\right|_{x(k)=\boldsymbol{x}_{j,rl}(k)\hat{\boldsymbol{v}}_{j,i}(k)} \quad (54)$$

with $\hat{\phi}'_{j,i,l}(k)$ being the l^{th}-row, l^{th}-column entry of $\boldsymbol{\Phi}'_{j,i}(k)$, $\boldsymbol{x}_{j,rl}(k)$ being the l^{th} row of $\mathbf{X}_j^{f_j}(k)$. Therefore,

$$\hat{\mathbf{W}}_j^{\triangleright}(k)\Delta\boldsymbol{\varphi}_j(k) = \sum_{i=1}^{n_{f_j}} \hat{\mathbf{W}}_{j,i}^{\triangleright}(k)\Delta\boldsymbol{\varphi}_{j,i}(k) \approx \sum_{i=1}^{n_{f_j}} \hat{\mathbf{W}}_{j,i}^{\triangleright}(k)\hat{\boldsymbol{\Gamma}}_{j,i}(k)\boldsymbol{\Phi}'_{j,i}(k)\mathbf{X}_j^{f_j}(k)\Delta\boldsymbol{v}_{j,i}(k) \quad (55)$$

Properties of $\boldsymbol{\delta}(k)$. Let the activation functions $\boldsymbol{\sigma}$ be monotonically increasing and their derivatives be bounded above by f_σ, the following properties will hold:

i, The corresponding elements of $\boldsymbol{e}(k)$ in (48) and $\boldsymbol{\delta}(k)$ in (49) have the same sign, i.e.

$$e_i(k)\delta_i(k) \geq 0, \quad \forall i = 1..p \quad (56)$$

ii, The absolute value of each element of $\boldsymbol{e}(k)$ in (48) is less than or equal to f_σ times the absolute value of the corresponding element of $\boldsymbol{\delta}(k)$ in (49), i.e.

$$|e_i(k)| \leq f_\sigma |\delta_i(k)|, \quad \forall i = 1..p \quad (57)$$

Based on the output estimation error $\boldsymbol{e}(k)$, the learning law to update the estimated weight $\hat{\mathbf{W}}_j^{\triangleright}(k)$ is proposed as follows

$$\hat{\mathbf{W}}_j^{\triangleright}(k+1) = \hat{\mathbf{W}}_j^{\triangleright}(k) + \alpha_2 \mathbf{L}(k)\boldsymbol{e}(k)\hat{\boldsymbol{\varphi}}_j^T(k) \quad (58)$$

where α_2 is a positive scalar, $\mathbf{L}(k) \in \mathbb{R}^{p\times p}$ is a positive diagonal matrix.

In (58), let $\boldsymbol{w}_{j,h}^{\triangleright}$ denote the h^{th} column vector of matrix $\mathbf{W}_j^{\triangleright}$, $\hat{\boldsymbol{w}}_{j,h}^{\triangleright}(k)$ the h^{th} column vector of $\hat{\mathbf{W}}_j^{\triangleright}(k)$ and $\hat{\varphi}_{j,h}(k)$ the h^{th} element of vector $\hat{\boldsymbol{\varphi}}_j(k)$. The update law (58) can be rewritten in vector form as

$$\hat{\boldsymbol{w}}_{j,h}^{\triangleright}(k+1) = \hat{\boldsymbol{w}}_{j,h}^{\triangleright}(k) + \alpha_2\hat{\varphi}_{j,h}(k)\mathbf{L}(k)\boldsymbol{e}(k) \quad (59)$$

The learning law to update the estimated filter weight matrix $\hat{\mathbf{V}}_j(k)$ based on the output estimation error $\boldsymbol{e}(k)$ is proposed as

$$\hat{\boldsymbol{v}}_{j,i}(k+1) = \hat{\boldsymbol{v}}_{j,i}(k) + \alpha_1 \mathbf{X}_j^{f_j T}(k)\mathbf{P}_i(k)\boldsymbol{e}(k) \quad (60)$$

where $\hat{\boldsymbol{v}}_{j,i}(k)$ denotes the the i^{th} column vector of matrix $\hat{\mathbf{V}}_j(k)$, α_1 is a positive scalar. It can be seen that $\boldsymbol{v}_{j,i}$ corresponds to the i^{th} filter in totally n_{f_j} filters of the convolution operation. $\mathbf{P}_i(k)$ is chosen as

$$\mathbf{P}_i(k) = \boldsymbol{\Phi}'^T_{j,i}(k)\hat{\boldsymbol{\Gamma}}^T_{ji}(k)\hat{\mathbf{W}}^{\triangleright T}_{j,i}(k)\mathbf{L}(k) \quad (61)$$

To prove the convergence, an objective function is defined as

$$V(k) = \frac{1}{\alpha_2}\sum_{h=1}^{n_{flj}} \Delta\boldsymbol{w}_{j,h}^{\triangleright T}(k)\Delta\boldsymbol{w}_{j,h}^{\triangleright}(k) + \frac{1}{\alpha_1}\sum_{i=1}^{n_{f_j}} \Delta\boldsymbol{v}_{j,i}^T(k)\Delta\boldsymbol{v}_{j,i}(k) \quad (62)$$

where $\Delta\boldsymbol{w}_{j,h}^{\triangleright}(k) = \boldsymbol{w}_{j,h}^{\triangleright} - \hat{\boldsymbol{w}}_{j,h}^{\triangleright}(k)$ and $\Delta\boldsymbol{v}_{j,i}(k) = \boldsymbol{v}_{j,i} - \hat{\boldsymbol{v}}_{j,i}(k)$. Using (59) and (60), the objective function at the next step of learning can be expressed as

$$V(k+1) = \frac{1}{\alpha_2}\sum_{h=1}^{n_{flj}} \Delta\boldsymbol{w}_{j,h}^{\triangleright T}(k+1)\Delta\boldsymbol{w}_{j,h}^{\triangleright}(k+1) + \frac{1}{\alpha_1}\sum_{i=1}^{n_{f_j}} \Delta\boldsymbol{v}_{j,i}^T(k+1)\Delta\boldsymbol{v}_{j,i}(k+1)$$

$$= \frac{1}{\alpha_2}\sum_{h=1}^{n_{flj}}\left(\Delta \boldsymbol{w}_{j,h}^{\triangleright}(k) - \alpha_2\hat{\varphi}_{j,h}(k)\mathbf{L}(k)\boldsymbol{e}(k)\right)^T \times \left(\Delta \boldsymbol{w}_{j,h}^{\triangleright}(k) - \alpha_2\hat{\varphi}_{j,h}(k)\mathbf{L}(k)\boldsymbol{e}(k)\right) + \frac{1}{\alpha_1}\sum_{i=1}^{n_{fj}}\left\{\left(\Delta \boldsymbol{v}_{j,i}(k) - \alpha_1 \mathbf{X}_j^{f_j T}(k)\mathbf{P}_i(k)\boldsymbol{e}(k)\right)^T \times \left(\Delta \boldsymbol{v}_{j,i}(k) - \alpha_1 \mathbf{X}_j^{f_j T}(k)\mathbf{P}_i(k)\boldsymbol{e}(k)\right)\right\} \quad (63)$$

A change of the objective function from current learning step k^{th} to the next learning step $(k+1)^{\text{th}}$ can therefore be calculated as

$$\begin{aligned}
\Delta V(k) &= V(k+1) - V(k)\\
\Delta V(k) &= \frac{1}{\alpha_2}\sum_{h=1}^{n_{flj}}\Big(-\alpha_2\hat{\varphi}_{j,h}(k)\Delta \boldsymbol{w}_{j,h}^{\triangleright T}(k)\mathbf{L}(k)\boldsymbol{e}(k) - \alpha_2\hat{\varphi}_{j,h}(k)\boldsymbol{e}^T(k)\mathbf{L}^T(k)\Delta \boldsymbol{w}_{j,h}^{\triangleright}(k) + \alpha_2^2\hat{\varphi}_{j,h}^2(k)\boldsymbol{e}^T(k)\mathbf{L}^T(k)\mathbf{L}(k)\boldsymbol{e}(k)\Big)\\
&\quad + \frac{1}{\alpha_1}\sum_{i=1}^{n_{fj}}\Big(-\alpha_1\Delta \boldsymbol{v}_{j,i}^T(k)\mathbf{X}_j^{f_j T}(k)\mathbf{P}_i(k)\boldsymbol{e}(k) - \alpha_1\boldsymbol{e}^T(k)\mathbf{P}_i^T(k)\mathbf{X}_j^{f_j}(k)\Delta \boldsymbol{v}_{j,i}(k) + \alpha_1^2\boldsymbol{e}^T(k)\mathbf{P}_i^T(k)\mathbf{X}_j^{f_j}(k)\mathbf{X}_j^{f_j T}(k)\mathbf{P}_i(k)\boldsymbol{e}(k)\Big)\\
&= -\hat{\boldsymbol{\varphi}}_j^T(k)\Delta \mathbf{W}_j^{\triangleright T}(k)\mathbf{L}(k)\boldsymbol{e}(k) - \sum_{i=1}^{n_{fj}}\Delta \boldsymbol{v}_{j,i}^T(k)\mathbf{X}_j^{f_j T}(k)\mathbf{P}_i(k)\boldsymbol{e}(k) - \boldsymbol{e}^T(k)\mathbf{L}^T(k)\Delta \mathbf{W}_j^{\triangleright}(k)\hat{\boldsymbol{\varphi}}_j(k) - \sum_{i=1}^{n_{fj}}\boldsymbol{e}^T(k)\mathbf{P}_i^T(k)\mathbf{X}_j^{f_j}(k)\Delta \boldsymbol{v}_{j,i}(k)\\
&\quad + \boldsymbol{e}^T(k)\Big(\alpha_2\sum_{h=1}^{n_{flj}}\hat{\varphi}_{j,h}^2(k)\mathbf{L}^T(k)\mathbf{L}(k) + \alpha_1\sum_{i=1}^{n_{fj}}\mathbf{P}_i^T(k)\mathbf{X}_j^{f_j}(k)\mathbf{X}_j^{f_j T}(k)\mathbf{P}_i(k)\Big)\boldsymbol{e}(k)
\end{aligned} \quad (64)$$

Denoting

$$\mathbf{\Lambda}_j(k) \triangleq \alpha_2\sum_{h=1}^{n_{flj}}\hat{\varphi}_{j,h}^2(k)\mathbf{L}^T(k)\mathbf{L}(k) + \alpha_1\sum_{i=1}^{n_{fj}}\mathbf{P}_i^T(k)\mathbf{X}_j^{f_j}(k)\mathbf{X}_j^{f_j T}(k)\mathbf{P}_i(k) \quad (65)$$

From (50), we have

$$\Delta \mathbf{W}_j^{\triangleright}(k)\hat{\boldsymbol{\varphi}}_j(k) = \boldsymbol{\delta}(k) - \hat{\mathbf{W}}_j^{\triangleright}(k)\Delta \boldsymbol{\varphi}_j(k) - \Delta \mathbf{W}_j^{\triangleright}(k)\Delta \boldsymbol{\varphi}_j(k) \quad (66)$$

Next, substituting into (64) gives

$$\begin{aligned}
\Delta V(k) &= -\Big[\boldsymbol{\delta}^T(k) - \Delta \boldsymbol{\varphi}_j^T(k)\hat{\mathbf{W}}_j^{\triangleright T}(k) - \Delta \boldsymbol{\varphi}_j^T(k)\Delta \mathbf{W}_j^{\triangleright T}(k)\Big] \times \mathbf{L}(k)\boldsymbol{e}(k) - \sum_{i=1}^{n_{fj}}\Delta \boldsymbol{v}_{j,i}^T(k)\mathbf{X}_j^{f_j T}(k)\mathbf{P}_i(k)\boldsymbol{e}(k)\\
&\quad - \boldsymbol{e}^T(k)\mathbf{L}^T(k)\Big[\boldsymbol{\delta}(k) - \hat{\mathbf{W}}_j^{\triangleright}(k)\Delta \boldsymbol{\varphi}_j(k) - \Delta \mathbf{W}_j^{\triangleright}(k)\Delta \boldsymbol{\varphi}_j(k)\Big]\\
&\quad - \sum_{i=1}^{n_{fj}}\boldsymbol{e}^T(k)\mathbf{P}_i^T(k)\mathbf{X}_j^{f_j}(k)\Delta \boldsymbol{v}_{j,i}(k) + \boldsymbol{e}^T(k)\mathbf{\Lambda}_j(k)\boldsymbol{e}(k)\\
&= -\boldsymbol{\delta}^T(k)\mathbf{L}(k)\boldsymbol{e}(k) - \boldsymbol{e}^T(k)\mathbf{L}^T(k)\boldsymbol{\delta}(k) + \boldsymbol{\xi}^T(k)\mathbf{L}(k)\boldsymbol{e}(k) + \boldsymbol{e}^T(k)\mathbf{L}^T(k)\boldsymbol{\xi}(k) + \boldsymbol{e}^T(k)\mathbf{\Lambda}_j(k)\boldsymbol{e}(k)\\
&\quad + \Delta \boldsymbol{\varphi}_j^T(k)\Delta \mathbf{W}_j^{\triangleright T}(k)\mathbf{L}(k)\boldsymbol{e}(k) + \boldsymbol{e}^T(k)\mathbf{L}^T(k)\Delta \mathbf{W}_j^{\triangleright}(k)\Delta \boldsymbol{\varphi}_j(k)
\end{aligned} \quad (67)$$

where

$$\boldsymbol{\xi}(k) \triangleq \hat{\mathbf{W}}_j^{\triangleright}(k)\Delta \boldsymbol{\varphi}_j(k) - \sum_{i=1}^{n_{fj}}\mathbf{L}^{-T}(k)\mathbf{P}_i^T(k)\mathbf{X}_j^{f_j}(k)\Delta \boldsymbol{v}_{j,i}(k) \quad (68)$$

Replacing (61) and (55) into $\boldsymbol{\xi}(k)$ leads to $\boldsymbol{\xi}(k) \approx 0$. Thus, (67) becomes

$$\begin{aligned}
\Delta V(k) &= -\boldsymbol{\delta}^T(k)\mathbf{L}(k)\boldsymbol{e}(k) - \boldsymbol{e}^T(k)\mathbf{L}^T(k)\boldsymbol{\delta}(k) + \boldsymbol{e}^T(k)\mathbf{\Lambda}_j(k)\boldsymbol{e}(k)\\
&\quad + \Delta \boldsymbol{\varphi}_j^T(k)\Delta \mathbf{W}_j^{\triangleright T}(k)\mathbf{L}(k)\boldsymbol{e}(k) + \boldsymbol{e}^T(k)\mathbf{L}^T(k)\Delta \mathbf{W}_j^{\triangleright}(k)\Delta \boldsymbol{\varphi}_j(k)
\end{aligned} \quad (69)$$

At the fine-tuning phase where the errors are adequately small, the last 2 terms in (69) can be negligible as they are of O^3 while the other terms are of O^2. The equation (69) becomes

$$\Delta V(k) = -\boldsymbol{\delta}^T(k)\mathbf{L}(k)\boldsymbol{e}(k) - \boldsymbol{e}^T(k)\mathbf{L}^T(k)\boldsymbol{\delta}(k) + \boldsymbol{e}^T(k)\mathbf{\Lambda}_j(k)\boldsymbol{e}(k) \quad (70)$$

Using the properties (56), (57) yields

$$\Delta V(k) \leq -\frac{2}{f_\sigma}\boldsymbol{e}^T(k)\mathbf{L}(k)\boldsymbol{e}(k) + \boldsymbol{e}^T(k)\mathbf{\Lambda}_j(k)\boldsymbol{e}(k) \quad (71)$$

When $\mathbf{L}(k)$ is selected such that

$$\frac{2}{f_\sigma}\mathbf{L}(k) - \mathbf{\Lambda}_j(k) > 0 \quad (72)$$

or

$$\frac{2}{f_\sigma}\mathbf{L}(k) - \Big(\alpha_2\sum_{h=1}^{n_{flj}}\hat{\varphi}_{j,h}^2(k)\mathbf{L}^T(k)\mathbf{L}(k) + \alpha_1\sum_{i=1}^{n_{fj}}\mathbf{P}_i^T(k)\mathbf{X}_j^{f_j}(k)\mathbf{X}_j^{f_j T}(k)\mathbf{P}_i(k)\Big) > 0 \quad (73)$$

then $\Delta V(k) \leq 0$ for any $\boldsymbol{e}(k)$. Hence, the value of the objective function satisfies $V(k+1) \leq V(k)$. Moreover, the function $V(k)$ is bounded from below as it is non-negative, we thus have $\Delta V(k)$ converges. So, from (71) we have $\boldsymbol{e}(k)$ converges when k increases.

Remark 1: For networks with insufficient number of hidden neurons, it can be guaranteed that $\Delta V(k) \leq 0$ if

$$\|\boldsymbol{e}(k)\| \geq \frac{b}{2\left(\frac{2L_m}{f_{\sigma M}} - d_M L_M^2\right)}\left[\frac{4L_m}{f_{\sigma M}} + \frac{2L_M}{f_{\sigma m}} + \sqrt{\frac{8L_m}{f_{\sigma M}} d_M L_M^2 + \frac{8L_M}{f_{\sigma m}}\left(\frac{4L_m}{f_{\sigma M}} - d_M L_M^2\right) + \left(\frac{2L_M}{f_{\sigma m}}\right)^2}\right] \tag{74}$$

$$\text{and} \quad \frac{2L_m}{f_{\sigma M}} - d_M L_M^2 > 0 \tag{75}$$

where b denotes the upper bound of the NN approximation error, L_m and L_M respectively denote the minimum and maximum eigenvalues of the matrix $\mathbf{L}(k)$ for $\forall k$, $f_{\sigma m}$ and $f_{\sigma M}$ respectively denote the lower and the upper bounds of the derivative of $\boldsymbol{\sigma}$, and d_M is a positive constant such that

$$\boldsymbol{e}^T(k)\Big(\alpha_2 \sum_{h=1}^{n_{fij}} \hat{\varphi}_{j,h}^2(k)\mathbf{L}^T(k)\mathbf{L}(k) + \alpha_1 \sum_{i=1}^{n_{fj}} \mathbf{P}_i^T(k)\mathbf{X}_j^{fj}(k)\mathbf{X}_j^{fj\,T}(k)\mathbf{P}_i(k)\Big)\boldsymbol{e}(k) \leq d_M L_M^2 \|\boldsymbol{e}(k)\|^2 \tag{76}$$

Therefore, there exists an ultimate bound such that the error always stay within the bound after reaching it. Noting from (74) that this ultimate bound tends to zero when the bound of the NN approximation error tends to zero.

Remark 2: While the convergence issue in deep learning has only received more attention from the ML community recently due to the interests of explainable AI (XAI), convergence and stability analyses have always been important for neural network-based learning control since the early days of its research [13]–[15]. For robotic applications, the neural network-based control has also been an active research topic [29]–[35]. These problems are mostly formulated as regression problems for dynamic control and most of the studies has focused on shallow networks only. In [21], multilayer fully connected networks were employed for both regression and classification tasks in robotic systems where Lyapunov-like method was developed for convergence analysis of deep learning networks. It has thus bridged the gaps between the fields of deep learning, control and robotics, so that deep dense networks can be used reliably in robotic applications. These formulations are all based on dense or fully connected networks, but for image classification tasks, deep convolutional neural networks have shown to be more effective. The convolutional neural networks can be treated as more general networks for image classification tasks as they consist of convolutional layers, polling layers and fully connected layers.

IV. CASE STUDIES

In this section, the proposed FPL method for CNNs is evaluated based on different network architectures and datasets. Comparisons are made with the stochastic gradient descent (SGD) method.

A. SVHN

The first classification task is based on SVHN dataset [36]. It is a dataset that contains real-world images of house numbers in 10 classes, each class for each digit. It has 73,257 digits for training, 26032 digits for testing. Each of these images has the size of 32×32 pixels. SVHN dataset is similar to the classical MNIST dataset [24], but it contains natural images of house numbers instead of images of handwritten digits.

The VGG11 [3] was used for the classification task of SVHN. The architecture of the VGG11 is shown in Fig. 15. The VGG11 has 8 convolutional layers with pooling operations at conv 1, 2, 4, 6 and 8, and 3 fully connected layers. The activation functions of the convolutional layers and inner fully connected layers are ReLU, and the activation function of the output layer is sigmoid.

1) CONVERGENCE ISSUE IN SGD

The SGD optimizer was first used to train the VGG11. For consistency with the FPL method, the batch size for SGD was chosen as 1. The number of epochs was set at 300. The learning rate was initially set as 0.01, and scheduled to be halved after the 150$^{\text{th}}$ epoch. The learning converged at first, but then failed to converge after the 100$^{\text{th}}$ epoch, as can be seen in Fig. 11 and Fig. 12.

2) NETWORK PERFORMANCE

To illustrate the performance of the proposed method, the convolutional part of the VGG11 was trained by FPL through training 8 conv subnets sequentially, each of which was a two-layer training CNN, and the fully connected part of the network was trained by FPL through training 2 FC subnets. There were 2 phases in learning of each subnet: pre-training and fine-tuning. In pre-training phase, the output layer, i.e. the fully connected part, of the subnet was trained in 2 loops by using the one-layer update algorithm of fully connected networks [21]. In this phase, the learning gain was automatically calculated. In the fine-tuning phase, the update laws (58) and (60) were used where the convolutional part were updated. There were about 200-600 loops for fine-tuning of each conv subnet and 100 loops for each FC subnet. For the conv subnets, the training stopped at the 200$^{\text{th}}$, 500$^{\text{th}}$ or 600$^{\text{th}}$ loop when the overfitting was likely to start happening. The initial gain for the fine-tuning of all subnet was set at 0.01. To make sure that the initial gain was not too large for convergence, in the first several loops, the gain was automatically reduced by checking the condition (73). After the gain was adjusted

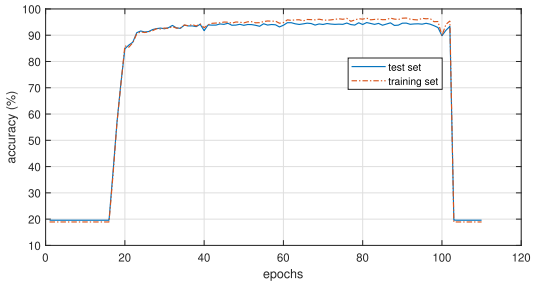


FIGURE 11. SVHN with VGG11: Accuracies of the traning set and test set in SGD with learning rate 0.01. The convergence failed after 100 epochs.

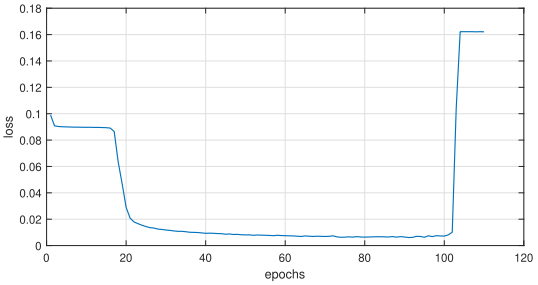


FIGURE 12. SVHN with VGG11: The loss in SGD with learning rate 0.01. The convergence failed after 100 epochs.

TABLE 1. SVHN with VGG11: Training & test accuracies (%) by FPL and SGD.

FPL				SGD	
subnets[a]		full net[b]		(full net[b])	
training	test	training	test	training	test
96.39	94.06	97.29	93.91	99.16	95.70

[a] At layer conv4, no. of parameters: 1.00E+06,
[b] no. of parameters: 9.75E+06

to a suitable value, the new value of the gain matrix was kept for the successive loops of fine-tuning.

Fig. 13 shows the best test accuracy for each subnet. It can be seen that the test accuracy increases when adding a new convolutional layer for the first 4 convolutional layers and then maintains similar accuracies when adding the remaining convolutional and fully connected layers. The accuracy for the last hidden fully connected layer is 93.91%.

For comparison, the SGD was used again but with a smaller learning rate (0.005) so that the convergence did not fail as in subsection IV-A1. There were again 300 epochs. The obtained results are shown in Table 1. It can be seen from the table that there is a trade-off in the performance where the proposed FPL can guarantee convergence but the test accuracy (full net) is slightly lower than that of SGD.

3) THE POSSIBILITY OF PRUNING TOP LAYERS

Besides ensuring convergence, another advantage of using the proposed layer-wise learning is the possibility of constructing the optimal number of layers of the convolutional filters. As seen in Fig. 13, the test accuracy peaks after adding the first few convolutional layers and then does not improve further with more layers added. To see if the trend happens for a new test set with unseen data as well, we have also tested the trained subnets with the MNIST dataset where the classification task is similar. The tests were done for both training set and test set of MNIST. To avoid confusion, we shall call the training set and test set of MNIST in this case as test set A and test set B respectively. To do the tests, we extended the number of channels of the input images from 1 to 3, to match the number of channels of color images of SVHN. It can be seen from Fig. 14 that the trend in the test set of SVHN is also present for both test set A and test set B obtained from MNIST. This indicates the possibility of pruning the convolutional layers based on the proposed layer wise method. The layer-wise learning can also be terminated when the accuracies do not further improve by adding one or two more layers.

TABLE 2. SVHN with VGG11: The trained networks were tested with MNIST dataset. Test set A indicates the training set of MNIST, test set B indicates the test set of MNIST.

FPL				SGD	
subnets[a]		full net[b]		(full net[b])	
Test set A	Test set B	Test set A	Test set B	Test set A	Test set B
60.42	61.95	56.35	57.48	52.87	53.74

[a] At layer conv4, no. of parameters: 1.00E+06,
[b] no. of parameters: 9.75E+06

For comparison, the entire VGG11 network trained by SGD was also tested with the MNIST dataset. The results for FPL (at the conv4 subnet and full net) and SGD (full net) are shown at Table 2. It can be seen that the FPL generalizes better for this specific problem even with the use of less layers and parameters.

B. FASHION MNIST

Fashion MNIST [25] is a dataset of fashion products in 10 classes: t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot. The dataset contains 60,000 training examples and 10,000 test examples. Each example is a 28×28 grayscale image. The dataset was intended to serve as a replacement for the classical MNIST database.

1) AlexNet-LIKE CNN

We also conducted the experiments on a CNN with no hidden fully connected layers in the full net, aiming to test the FPL algorithm for convolutional layers which has been developed in this paper. An AlexNet-like CNN whose architecture is shown in Fig. 16 was chosen to achieve that aim. The design of the network was inspired by the original structure of the AlexNet first introduced in [2]. All of the input images were resized to 32×32 before being fed into the network. The AlexNet-like CNN has 5 convolutional layers with pooling operations at conv 1, 2 and 5. The activation functions of the convolutional layers are ReLU, and the activation

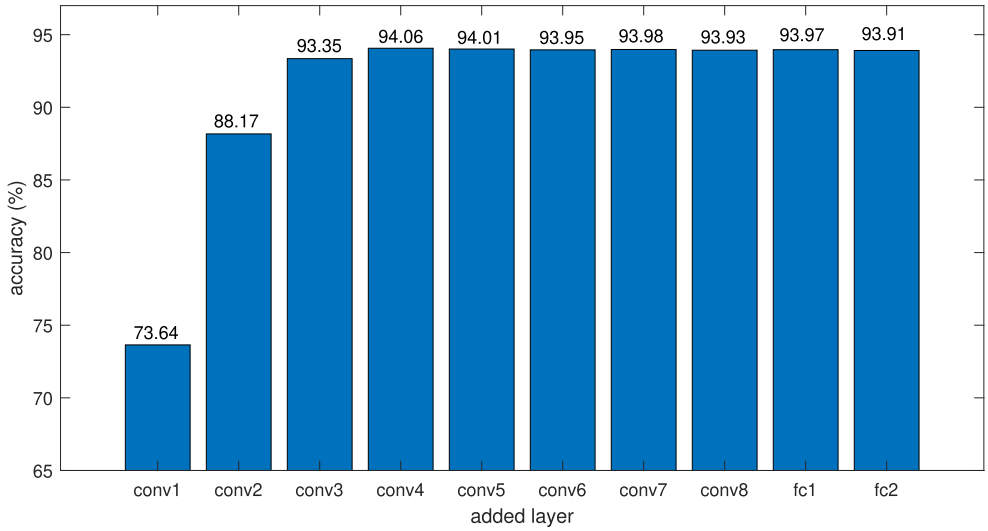


FIGURE 13. SVHN with VGG11: The test accuracies of the 10 subnets.

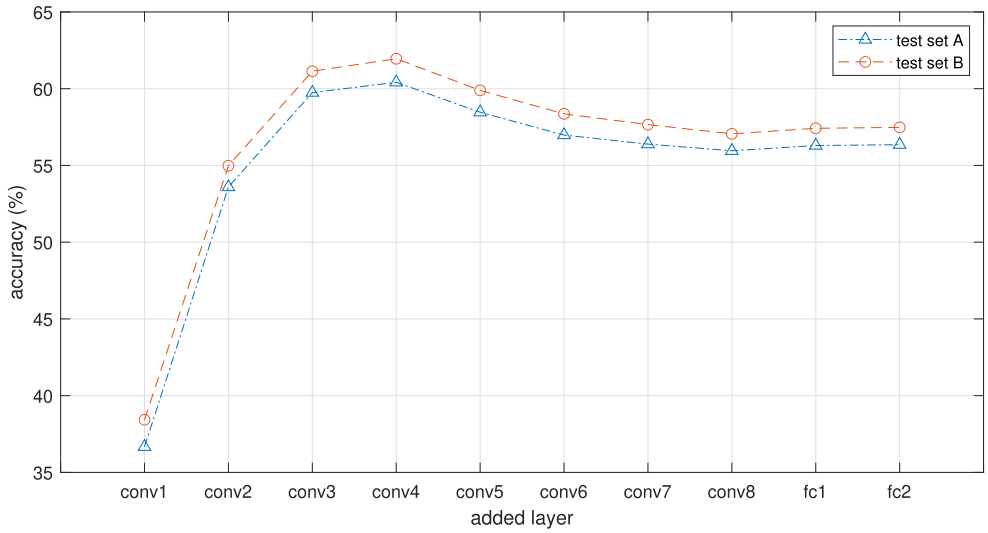


FIGURE 14. SVHN with VGG11: The subnets which have been trained with SVHN are tested with MNIST dataset. Test set A indicates the training set of MNIST, test set B indicates the test set of MNIST.

TABLE 3. Number of parameters and neurons in the VGG11 whose structure is given in Fig. 15.

	conv1	conv2	conv3	conv4	conv5	conv6	conv7	conv8	fc1	fc2
paras	1.66E+05	1.57E+05	5.34E+05	1.00E+06	2.22E+06	4.52E+06	6.88E+06	9.22E+06	9.48E+06	9.75E+06
neurons	8.19E+04	1.23E+05	1.39E+05	1.60E+05	1.68E+05	1.78E+05	1.80E+05	1.83E+05	1.83E+05	1.84E+05

function of the output layer (the only fully connected layer) is sigmoid.

The network was trained by FPL through training 5 subnets sequentially, each of which was a two-layer training CNN. There were 2 phases in learning of each subnet: pre-training and fine-tuning. In pre-training phase, the output layer, i.e. the fully connected part, of the subnet was trained in 2 loops using the one-layer update algorithm which has been developed in [21]. In this phase, the learning gain was automatically calculated. In the fine-tuning phase, the update laws (58) and (60) were used. There were 200-600 loops for training of each subnet. The training stopped at the 200^{th}, 400^{th} or 600^{th} loop when the overfitting was likely to start happening. The initial gain for the fine-tuning of all subnet was set at 0.001. To make sure that the initial gain was not too large for convergence, in the first several loops, the gain was automatically reduced

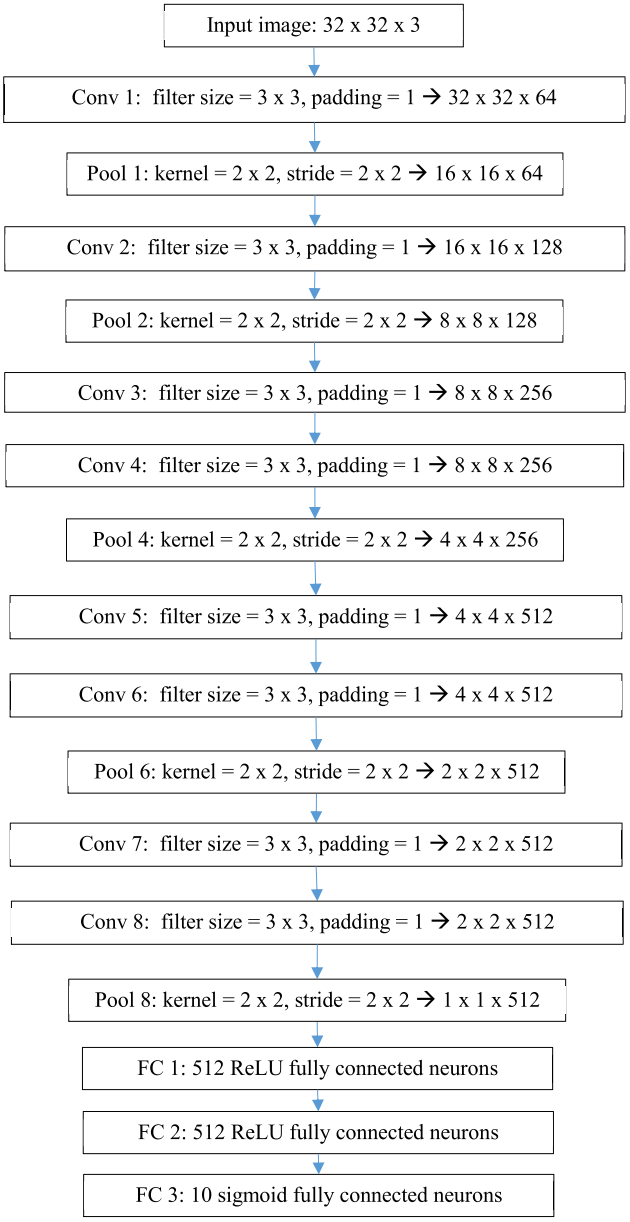


FIGURE 15. The architecture of the VGG11.

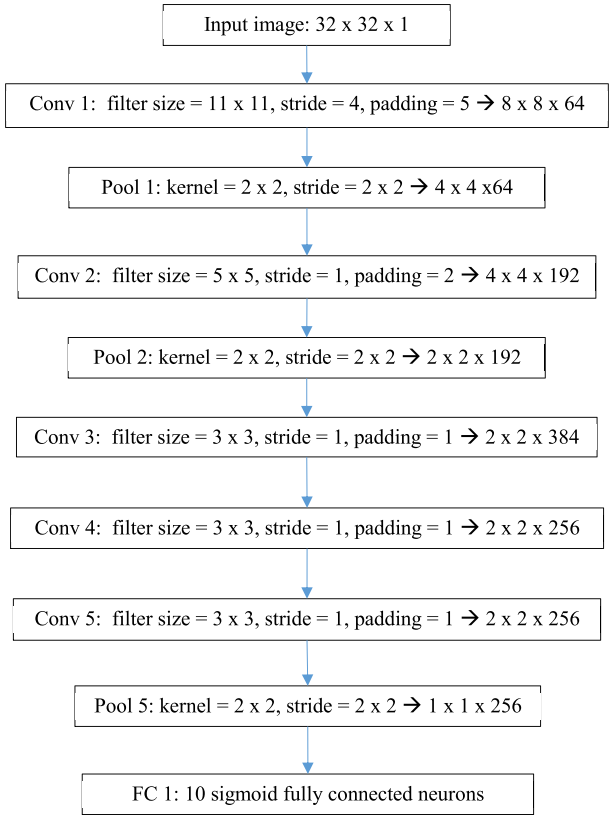


FIGURE 16. The architecture of the AlexNet-like CNN.

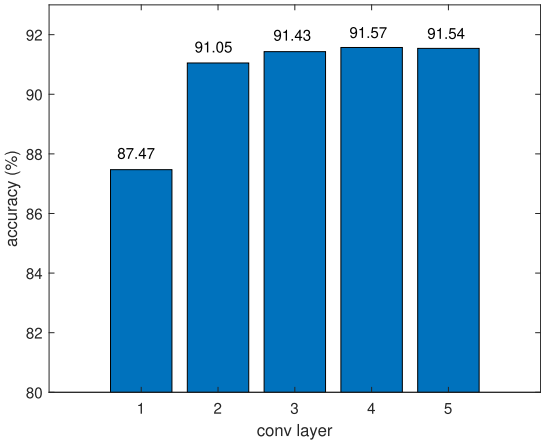


FIGURE 17. Fashion MNIST with AlexNet-like CNN: The test accuracies of the 5 subnets.

by checking the condition (73). After the gain was adjusted to a suitable value, the new value of the gain matrix was kept for the successive loops of fine-tuning.

Fig. 17 shows the best test accuracy for each subnet. It can be seen that the test accuracy increases when adding a new convolutional layer for the first 4 convolutional layers and then maintains a similar value when adding the last convolutional layer. The accuracies for the 3^{th}, 4^{th}, 5^{th} layer are in fact quite similar.

The SGD optimizer was also used to train the AlexNet-like CNN. There were 500 epochs. The learning rate was initially set as 0.001, the momentum was 0.9. The highest accuracy for the test set was recorded during training and is shown in Table 4. The accuracy for the training set in the table was selected at the epoch where the test accuracy peaked.

From Table 4, it is noted that the test and training accuracies of FPL are comparable with those of SGD. Since the network is not deep as compared to VGG11, the reduction in parameters by implementing the subnet is not significant but a similar test accuracy can be maintained.

2) VGG11

We then continued the experiments on Fashion MNIST with VGG11. All of the input images were also resized to 32×32

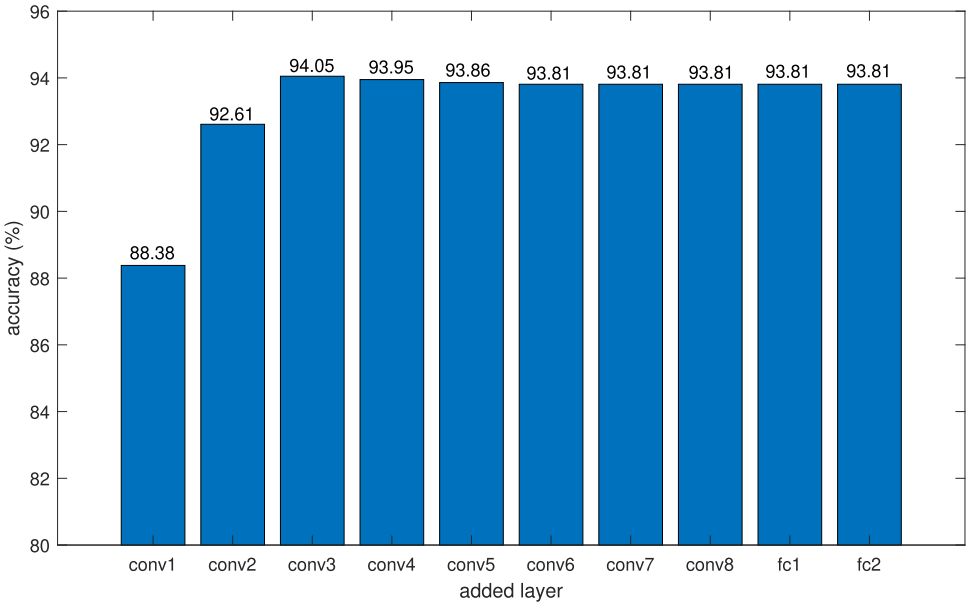


FIGURE 18. Fashion MNIST with VGG11: The test accuracies of the 10 subnets.

TABLE 4. Fashion MNIST with AlexNet-like CNN: Training & test accuracies (%) by FPL and SGD.

FPL				SGD	
subnets[a]		full net[b]		(full net[b])	
training	test	training	test	training	test
93.62	91.57	93.92	91.54	94.81	91.81

[a] At layer conv4, no. of parameters: 1.87E+06,
[b] no. of parameters: 2.46E+06

before being fed into the network. The network was trained similarly as the training of SVHN.

Fig. 18 shows the best test accuracy for each subnet. It can be seen that the test accuracy increases when adding a new convolutional layer for the first 3 convolutional layers and then stays at similar values when adding the 4^{th}, 5^{th}, 6^{th} convolutional layer. The accuracies for the remaining successive layers are the same (93.81%).

The SGD was also used to train the entire VGG11. It can be seen from Table 5 that the test and training accuracies of FPL (full net and subnet) are comparable with those of SGD. In addition, the proposed training method also indicates that a subnet can be implemented with lesser parameters but similar accuracy.

C. CIFAR-10

The next classification task is based on CIFAR-10 database [37]. It is a database that contains color images of objects in 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. It has 50,000 examples in the training set and 10,000 examples in the test set. Each of these images has the size of 32×32 pixels.

The VGG11 was used for the classification task of CIFAR-10. The network was trained similarly as the training for Fashion MNIST.

TABLE 5. Fashion MNIST with VGG11: Training & test accuracies (%) by FPL and SGD.

FPL				SGD	
subnets[a]		full net[b]		(full net[b])	
training	test	training	test	training	test
97.16	94.05	98.64	93.81	99.57	93.78

[a] At layer conv3, no. of parameters: 5.33E+05,
[b] no. of parameters: 9.75E+06

TABLE 6. CIFAR-10 with VGG11: Training & test accuracies (%) by FPL and SGD.

FPL				SGD	
subnets[a]		full net[b]		(full net[b])	
training	test	training	test	training	test
98.25	87.61	98.95	87.30	99.89	89.24

[a] At layer conv4, no. of parameters: 1.00E+06,
[b] no. of parameters: 9.75E+06

Fig. 19 shows the best test accuracy for each subnet. It can be seen that the test accuracy increases when adding a new convolutional layer for the first 4 convolutional layers and then stays at similar values when adding the remaining convolutional and fully connected layers. The accuracy for the last hidden fully connected layer is 87.30%.

The SGD was also used to train the entire VGG11. It can be seen from Table 6 that the test accuracy of FPL (full net) is lower than that of SGD (about 2% less for full net and 1.6% less for subnet). Though there is a trade off in performance, the convergence can now be ensured in training and the model can be implemented with lesser parameters while achieving a similar accuracy as the full net trained by FPL.

D. KMNIST

The next classification task is based on KMNIST database [38]. It is a database that contains 10 classes of

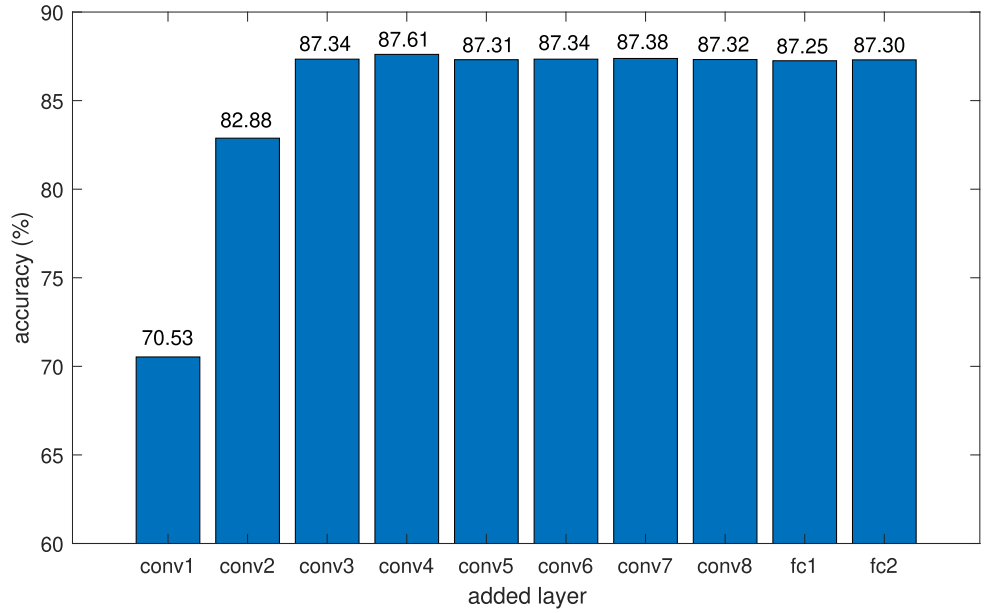


FIGURE 19. CIFAR-10 with VGG11: The test accuracies of the 10 subnets.

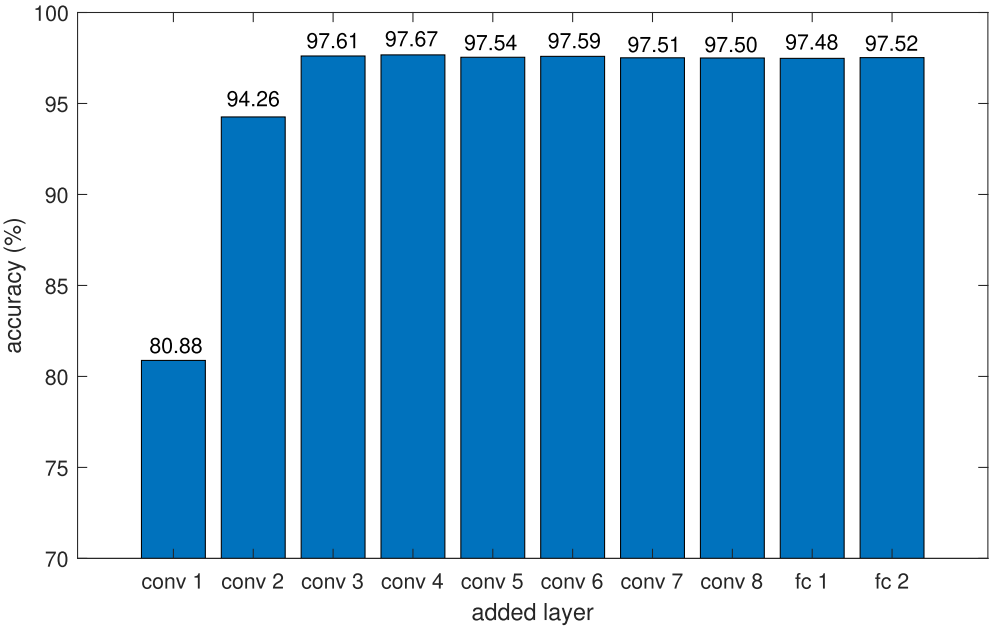


FIGURE 20. KMNIST with VGG11: The test accuracies of the 10 subnets.

Japanese characters. It has 60,000 images for training, 10,000 images for testing. Each of these images has the size of 28×28 pixels.

The VGG11 was used for the classification task of KMNIST. The images were resized to 32×32 before being fed into the networks. The network was trained similarly as the training for Fashion MNIST.

Fig. 20 shows the best test accuracy for each subnet. It can be seen that the test accuracy increases when adding a new convolutional layer for the first 4 convolutional layers and then stays at similar values when adding the remaining convolutional and fully connected layers. The accuracy for the last hidden fully connected layer is 93.63%.

The SGD was also used to train the entire VGG11. It can be seen from Table 7 that the training and test accuracies of FPL (full net and subnet) are similar to those of SGD.

E. DISCUSSION

It can be seen from the experimental results that there is a trade-off between test accuracy and analytic learning. The test

TABLE 7. KMNIST with VGG11: Training & test accuracies (%) by FPL and SGD.

FPL				SGD	
subnets[a]		full net[b]		(full net[b])	
training	test	training	test	training	test
99.99	97.67	99.99	97.52	99.95	98.68

[a] At layer conv4, no. of parameters: 1.00E+06,
[b] no. of parameters: 9.75E+06

accuracies of some tasks are comparable with SGD and of some other tasks are slightly less than SGD. The convergence is observed through all case studies of the proposed method while the convergence cannot be assured in the process of training in SGD if the learning rate is not chosen carefully via trial and error, as reported in subsection IV-A1.

With FPL method, the convergence can be guaranteed and the learning gain can be automatically tuned using condition (73). Another benefit of FPL is the possibility of cutting down the number of layers: As we can see from Fig. 13, Fig. 14, and Fig. 18 to Fig. 20 and as discussed in section IV-A, the test accuracies peak after the first few convolutional layers and then maintain similar values when more layers are added. Therefore, with the FPL method, one could consider pruning the top layers of the full network or terminate the layer-wise training earlier to save computational resources and training time. Table 3 shows the number of parameters and neurons of VGG11 so that one knows how many could be saved by cutting down the top layers of the full network. By using the results of every subnet, it can be seen that the FPL can also perform better than SGD for some of the cases.

V. CONCLUSION

In this paper, deep convolutional neural networks have been analyzed and trained by the forward progressive learning framework. The convergence of the proposed framework can be guaranteed by theoretical analysis. The proposed method has been validated in several classification tasks with popular benchmarking datasets. It can be drawn from the experimental results that the proposed method can yield comparable accuracies as the gradient descent method for most cases. In some classification tasks, there is a trade-off between performance and guarantee of convergence in which the end-to-end gradient descent method performs slightly better than the layer-wise approach in terms of accuracy while the proposed layer-wise approach is able to guarantee the convergence of the learning algorithm and also to be used as an indicator to determine the appropriate number of layers in final implementations of the models. It has been shown in the case studies that the final implementations of some widely used CNNs do not need as many convolutional layers as in their original structure to achieve reasonable accuracies. Additionally, while gradient descent method performs well for each specific dataset, the comparison of the generalization property by using the MNIST dataset as test sets for the models trained on SVHN dataset has shown that the gradient descent method cannot generalize well for similar tasks. In contrast, our proposed method has performed better in terms of the generalization property for similar tasks in this case study.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[4] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE Access*, vol. 7, pp. 53040–53065, 2019.
[5] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (XAI)," *IEEE Access*, vol. 6, pp. 52138–52160, 2018.
[6] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G.-Z. Yang, "XAI—Explainable artificial intelligence," *Sci. Robot.*, vol. 4, no. 37, 2019. [Online]. Available: https://sites.cc.gatech.edu/~alanwags/DLAI2016/(Gunning)%20IJCAI-16%20DLAI%20WS.pdf
[7] M. Jaderberg, W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, and K. Kavukcuoglu, "Decoupled neural interfaces using synthetic gradients," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1627–1635.
[8] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," in *Proc. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2211–2221.
[9] A. Nøkland and L. H. Eidnes, "Training neural networks with local error signals," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 4839–4850.
[10] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2007, pp. 153–160.
[11] C. Hettinger, T. Christensen, B. Ehlert, J. Humpherys, T. Jarvis, and S. Wade, "Forward thinking: Building and training neural networks one layer at a time," 2017, *arXiv:1706.02480*.
[12] S. Yoa, M. Jeon, Y. Oh, and H. J. Kim, "Learning to balance local losses via meta-learning," *IEEE Access*, vol. 9, pp. 130834–130844, 2021.
[13] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Netw.*, vol. 1, no. 1, pp. 4–27, Mar. 1990.
[14] W. T. Miller, P. J. Werbos, and R. S. Sutton, *Neural Networks for Control*. Cambridge, MA, USA: MIT Press, 1995.
[15] R. Fierro and F. L. Lewis, "Control of a nonholonomic mobile robot using neural networks," *IEEE Trans. Neural Netw.*, vol. 9, no. 4, pp. 589–600, Jul. 1998.
[16] Y. Li and Y. Yuan, "Convergence analysis of two-layer neural networks with ReLU activation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1–11.
[17] C. Fang, H. Dong, and T. Zhang, "Mathematical models of overparameterized neural networks," *Proc. IEEE*, vol. 109, no. 5, pp. 683–703, May 2021.
[18] H. T. Pham and P.-M. Nguyen, "Global convergence of three-layer neural networks in the mean field regime," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–35.
[19] S. Oymak and M. Soltanolkotabi, "Toward moderate overparameterization: Global convergence guarantees for training shallow neural networks," *IEEE J. Sel. Areas Inf. Theory*, vol. 1, no. 1, pp. 84–105, May 2020.
[20] C. Song, A. Ramezani-Kebrya, T. Pethick, A. Eftekhari, and V. Cevher, "Subquadratic overparameterization for shallow neural networks," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2021, pp. 1–13.
[21] H.-T. Nguyen, C. C. Cheah, and K.-A. Toh, "An analytic layer-wise deep learning framework with applications to robotics," *Automatica*, vol. 135, Jan. 2022, Art. no. 110007.
[22] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 242–252.
[23] S. Du, J. Lee, H. Li, L. Wang, and X. Zhai, "Gradient descent finds global minima of deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 1675–1685.
[24] Y. LeCun. (1998). *The MNIST Database of Handwritten Digits*. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[25] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.
[26] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 315–323.
[27] K.-A. Toh, "Learning from the kernel and the range space," in *Proc. IEEE/ACIS 17th Int. Conf. Comput. Inf. Sci. (ICIS)*, Jun. 2018, pp. 1–6.
[28] G. Strang, *Introduction to Linear Algebra*, vol. 3. Wellesley, MA, USA: Wellesley-Cambridge Press, 1993.
[29] L. Cheng, Z.-G. Hou, and M. Tan, "Adaptive neural network tracking control for manipulators with uncertain kinematics, dynamics and actuator model," *Automatica*, vol. 45, no. 10, pp. 2312–2318, 2009.
[30] Z. Chu, D. Zhu, and S. X. Yang, "Observer-based adaptive neural network trajectory tracking control for remotely operated vehicle," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 7, pp. 1633–1645, Jul. 2017.
[31] J. Qiu, K. Sun, I. J. Rudas, and H. Gao, "Command filter-based adaptive NN control for MIMO nonlinear systems with full-state constraints and actuator hysteresis," *IEEE Trans. Cybern.*, vol. 50, no. 7, pp. 2905–2915, Jul. 2020.
[32] C. Mu and Y. Zhang, "Learning-based robust tracking control of quadrotor with time-varying and coupling uncertainties," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 1, pp. 259–273, Jan. 2020.
[33] C. Yang, G. Peng, L. Cheng, J. Na, and Z. Li, "Force sensorless admittance control for teleoperation of uncertain robot manipulator using neural networks," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 51, no. 5, pp. 3282–3292, May 2021.
[34] S. Cremer, S. K. Das, I. B. Wijayasinghe, D. O. Popa, and F. L. Lewis, "Model-free online neuroadaptive controller with intent estimation for physical human–robot interaction," *IEEE Trans. Robot.*, vol. 36, no. 1, pp. 240–253, Feb. 2020.
[35] S. Lyu and C. C. Cheah, "Data-driven learning for robot control with unknown Jacobian," *Automatica*, vol. 120, Oct. 2020, Art. no. 109120.
[36] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," Tech. Rep., 2011. [Online]. Available: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf
[37] A. Krizhevsky, V. Nair, and G. Hinton. (2014). *The CIFAR-10 Dataset*. [Online]. Available: http://www.cs.toronto.edu/kriz/cifar.html
[38] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical Japanese literature," 2018, *arXiv:1812.01718*.

HUU-THIET NGUYEN received the B.E. degree in control and automation engineering from the Hanoi University of Science and Technology, Hanoi, Vietnam, in 2015. He is currently pursuing the Ph.D. degree in electrical and electronic engineering with Nanyang Technological University, Singapore. His research interests include robot control, robot learning, and deep learning in robotics.

SITAN LI received the B.S. degree in automation from Xi'an Jiaotong University, and the M.Sc. degree in computer control and automation from NTU. She is currently pursuing the Ph.D. degree with the School of Electrical and Electronic Engineering, Nanyang Technological University. Her research interests include deep learning and robotic control.

CHIEN CHERN CHEAH (Senior Member, IEEE) was born in Singapore. He received the B.Eng. degree in electrical engineering from the National University of Singapore, in 1990, and the M.Eng. and Ph.D. degrees in electrical engineering from Nanyang Technological University, Singapore, in 1993 and 1996, respectively. From 1990 to 1991, he worked as a Design Engineer with Chartered Electronics Industries, Singapore. He was a Research Fellow with the Department of Robotics, Ritsumeikan University, Japan, from 1996 to 1998. He is currently an Associate Professor with Nanyang Technological University. He serves as an Associate Editor for *Automatica*. He has served as an Associate Editor for IEEE TRANSACTIONS ON ROBOTICS, from 2010 to 2013, the Program Co-Chair for IEEE International Conference on Robotics and Automation, in 2017, the Award Chair for IEEE/RSJ International Conference on Intelligent Robots and Systems, in 2019, and the Lead Guest Editor for IEEE TRANSACTIONS ON MECHATRONICS, in 2021.