Swap-Deep Neural Network: Incremental Inference and Learning for Embedded Systems

Taihei Asai and Koichiro Yamauchi[a]
Institute of Computer Science, Chubu University, Matsumoto-cho 1200, Kasugai, Aichi, Japan

Keywords: Hidden Neural Network, Incremental Learning, Continual Learning, Deep Neural Networks.

Abstract: We propose a new architecture called "swap-deep neural network" that enables the learning and inference of large-scale artificial neural networks on edge devices with low power consumption and computational complexity. The proposed method is based on finding and integrating subnetworks from randomly initialized networks for each incremental learning phase. We demonstrate that our method achieves a performance equivalent to that of conventional deep neural networks for a variety of various classification tasks.

1 INTRODUCTION

One reason for the rapid development of artificial intelligence in recent years is the establishment of learning methods for large-scale neural networks. In several cases, the neural network connections that are essential for realizing a complex task do not need to be considerably large. However, large-scale neural networks are typically required to achieve successful learning. This is because the probability of the initial state of the connections required for the task to be included in the randomly initialized neural network does not increase unless the scale is large. In other words, to successfully achieve both learning and recognition, it is desirable for the neural network to be as large as possible.

However, to practically realize such learning and inference, it is necessary to implement them on edge devices. Many existing edge devices are connected to cloud servers via the Internet and their heavy computational tasks are performed by the cloud servers. Particularly, they request the cloud server to perform the learning process and transfer back the resulting weight parameters. However, this system faces the risk of privacy leakage associated with network connections.

One way to solve this problem is to compress the large neural network to a small one. For example, some methods are the neural architecture selection (NAS) (e.g. (Sadel et al., 2023)), pruning useless connections by using the regularization term(Ishikawa, 1996) and distillation techniques(Hinton et al., 2015). These methods require all required samples to be prepared in advance. Moreover, in the case of the NAS, the system has to prepare several solution-candidates to find a better solution. These conditions are too hard to manage the system on-site, where the system has to incrementally learn new novel samples with a bounded memory space.

In this study, we propose a novel method to solve this problem. In particular, we propose a method for realizing both learning and inference on edge devices without connecting them to cloud servers. However, we assume a situation in which the size of the neural network required to learn and recognize a task that the user faces exceeds the capacity of the edge device. In this case, the proposed method divides the task into multiple subtasks and trains each subtask on a separate network on the edge device. The parameters realized in this manner are stored on a secondary storage device; during inference, the parameters are read in sequence and inferred, and the results are ensembled. However, this increases the power consumption owing to data transfer. Therefore, we infer subtasks in a manner similar to that of a hidden neural network (Ramanujan et al., 2020), using randomly initialized neural networks. In other words, we identify and retain only a few important weight connections that are necessary for inferring the subtask and store their position information on a secondary storage device. We expect that the power and time consumed during reading will be reduced.

In this paper, we present the theoretical analysis of power consumption and a simulation result. The re-

[a] https://orcid.org/0000-0003-1477-0391

Asai, T. and Yamauchi, K.
Swap-Deep Neural Network: Incremental Inference and Learning for Embedded Systems.
DOI: 10.5220/0012465500003654
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 13th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2024)*, pages 418-427
ISBN: 978-989-758-684-2; ISSN: 2184-4313
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

mainder of this paper is organized as follows. Section 2 describes related work. Section 3 presents a review of hidden neural networks. Section 4 describes the proposed incremental learning method. Section 5 analyze the power-consumption of the proposed learning method. Section 6 presents the experimental results and Section 7 discusses the computational complexity and suitable learning strategy of the proposed system. Finally Section 8 concludes the paper.

2 RELATED WORKS

The learning of dividing a task into multiple subtasks and learning them individually can be considered incremental learning (continual learning). At the same time, this method of learning a task as multiple subtasks and integrating them is also known as ensemble learning, particularly boosting method. Furthermore, there are several techniques for reducing power consumption and increasing processing efficiency when implemented on edge devices. In this section, we compare the differences between existing methods and the proposed method.

2.1 Comparison with Boosting Methods

A classical boosting method, Adaboost (Freund and Schapire, 1997), fixes weights by learning with weak learners. A weak learner increases the weight of the incorrect learning sample, decreases the weight of the correct learning sample, and then learns with the next weak learner. Using the set of weak learners created in this manner, unknown inputs are inferred in parallel. The average of the answers obtained in this manner is weighted and the final output is obtained, resulting in a higher recognition rate than that of individual weak learners. This method is not suitable for learning in a completely divided data form as done in the proposed method, because it uses the same dataset for all subnetworks (weak learners). In contrast, Foster (Feature boosting) (Wang et al., 2022) assumes that different domain learning samples are added to learn instead of the same domain learning samples as Adaboost. Therefore, this method is similar to the proposed method and we incorporate Foster's concept into the proposed method. However, Foster adopted a form of parallel computation that maintained the parameters of each subnetwork, which was not suitable for low-power computation on edge devices, as assumed in the proposed method.

In our method, we adopt a method of apply the connection position information used in each subnetwork to a network initialized randomly in each subnetwork by incorporating a hidden neural network (Ramanujan et al., 2020) method, which is suitable for cases in which all subnetworks share and use small edge-device resources.

2.2 Comparison with Existing Incremental Learning Methods

When new domain data are learned using an existing neural network, Catastrophic Forgetting (French, 1999) occurs. There are two main methods to prevent this problem. One is to learn new domain samples together with old domain samples (e.g. (French, 1997), (Yamauchi et al., 1999), (Hsu et al., 2018), (Hayes et al., 2019)). These methods have the advantage that once learned, past memories are not only prevented from catastrophic forgetting but also reset by relearning. Our proposed method adopts a similar approach in that it uses past data to set the parameters. However, networks that have been learned in the past by fixing their parameters are not relearned. The advantage of this method is that it reduces the number of computations required. The second method is to fix few weight parameters ((Rusu et al., 2016), (Kirkpatrick et al., 2017) (Zenke et al., 2017) (Mallya and Lazebnik, 2018) (Wang et al., 2022)) to ensure that what has been learned in the past is preserved. Our proposed method is similar to these methods in that it fixes the parameters learned in the past. However, in principle, only the model proposed by Wang et al. (Wang et al., 2022) can sequentially load and calculate these fixed parameters in order and calculate the final output. However, as mentioned previously, no model among them considers the implementation on edge devices.

2.3 Comparison with Neuromorphic Computing

Neuromorphic computing is a general term for models that can efficiently compute neural networks by providing hardware specialized for neural network computation (Zheng and Mazumder, 2020). In contrast, the proposed method does not use a neuromorphic model. This is merely a proposal on operating hardware with limited computing resources. However, more efficient operations can be achieved by executing individual neural networks in the neuromorphic models. Hirose et al. (Hirose et al., 2022) proposed a novel hardware neural network that reduced power consumption by introducing a hidden neural network architecture (Ramanujan et al., 2020). We also introduce the same aspects as (Hirose et al., 2022) to reduce power consumption. The difference between the proposed method and the model proposed by (Hirose et al., 2022) is that the proposed method supports incremental learning in large- scale tasks.

2.4 Contribution of Our Proposed Method

Our method virtually realizes a large-scale neural network on hardware with limited computing resources, while inheriting the ideas of the abovementioned existing methods. The differences between our method and other methods are summarized in Table 1.

3 REVIEW OF HIDDEN NEURAL NETWORK

Before describing the proposed method, this section reviews the hidden neural network used in the proposed method, as explained in Section . Ramanujan et al. (Ramanujan et al., 2020) found that randomly initialized neural networks include a subnetwork suitable for solving a specified task. They also presented an effective method for finding a subnetwork using a gradient descent method. Although this network uses the gradient descent method, it does not modify the weight parameters but finds the subnetwork.

3.1 Subnetwork

The subnetwork of a fully connected neural network is described as follows (Ramanujan et al., 2020). The l-th layer of a fully connected neuron consists of n_l nodes $\mathcal{V}^{(l)} = \{v_1^{(l)}, v_2^{(l)}, \cdots, v_{n_l}^{(l)}\}$. The subnetwork found via gradient descent is described by $\mathcal{G}(\mathcal{V}, \mathcal{E})$. The node $v \in \mathcal{V}$ output is denoted as

$$Z_v = \sigma(I_v) \tag{1}$$

where $\sigma()$ is the *ReLU* (Krizhevsky et al., 2012) and I_v is given as

$$I_v = \sum_{(u,v)\in\mathcal{E}} w_{uv} Z_u \tag{2}$$

To find a good sub-network, a score s_{uv} for each weight is introduced. The top k-percent of the absolute values of the scores are selected as the subnetwork. Therefore, (2) can be written as

$$I_v = \sum_{u\in\mathcal{V}^{(l-1)}} w_{uv} Z_u h(s_{uv}), \tag{3}$$

where $h(s_{uv}) = 1$ if $|s_{uv}|$ is included in the top k scores. In the case of the convolution layer, the scoring method is almost the same as that of the fully connected layer, except that the nodes are replaced with channels.

3.2 Find a Good Sub-Network (EDGE-POPUP)

Although the weight parameter values are fixed at random values, the score values s_{uv} are optimized using a gradient descent algorithm as follows:

$$s_{uv} = s_{uv} - \eta \frac{\partial \mathcal{L}}{\partial s_{uv}}, \tag{4}$$

where $\mathcal{L}$ denotes the loss function. Note that we have to rewrite (3) to

$$I_v = \sum_{u\in\mathcal{V}^{(l-1)}} w_{uv} Z_u |s_{uv}| \tag{5}$$

to calculate the gradient in (4) [1] .

The above equation omits the momentum and weight decay terms. Moreover, the change in the learning rate η is also ignored. These omitted terms should be added according to the optimization method used.

4 SWAP-DEEP NEURAL NETWORK

The proposed learning system:Swap-deep neural network (Swap-NN) consists of two parts: a convectional neural network (CNN) and a secondary-storage (see Fig 1). The CNN parameters are randomly initialized and fixed. Instead, CNN adaptation is achieved by identifying the important connections found using (2) and (4). The indices of the top k-scored connections are stored in the secondary-storage.

We assume that the proposed learning system is suitable for classification tasks. Learning samples are divided into several groups based on their labels. For example, consider the case of dividing the MNIST dataset into two groups. To realize this, the samples are divided for every five labels: $\{0, 1, 2, 3, 4\}$, $\{5, 6, 7, 8, 9\}$. In this case, the learning system generates two subnetworks to learn the datasets. Therefore,

[1]The 'simple minist example' presented by the authors https://github.com/allenai/hidden-networks/ , $\mathcal{L}$ is calculated by using (3). However, the backward calculation for the $\mathcal{L}$ is executed by (5). This means that the learning method is slightly different from (4). However, we found that the convergence speed of the simple sample code is faster than the code which derives $\mathcal{L}$ by (5). So, we have developed our code based on the 'simple mnist example'.

Table 1: Comparison with the other similar methods.

	SwapNN (ours)	Foster	Feature boost	Hidden NN	Naïve rehearsal	PackNet
Learning of large-scale dataset	○	○	○	○	○	○
Incremental Learning	○	○	×	×	○	○
Execution on a small device	○	×	×	○	×	×
Low-power consumption	○	×	×	○	×	×

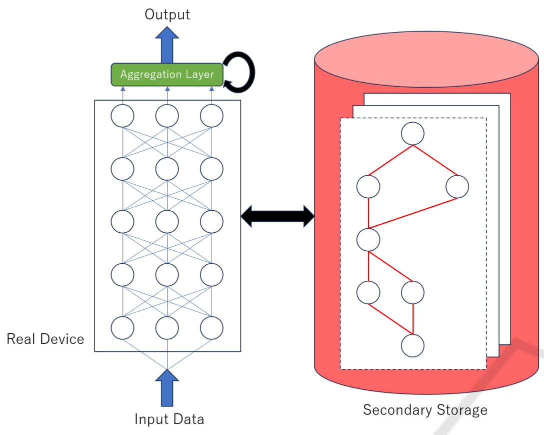

Figure 1: System structure: CNN with aggregation layer(left), and secondary storage(right).

the first subetwork learns the dataset whose labels are $\{0,1,2,3,4\}$ using (4). After learning, the most important k-connection indices and corressponding full-connection(FC) layer parameters are stored in the secondary storage. Similarly, the second subnetwork learns the datasets which have labels of $\{5,6,7,8,9\}$ together with a part of the first dataset, in the same manner as the first subnetwork, and stores a k important connection index and FC layer parameter. During the inference phase, an unknown input is assigned to the input layer of the CNN. The system loads the first subnetwork connection indices and FC-layer parameters from the secondary storage and calculates them. The output is sent to the aggregation layer, which consists of the FC layers used by Foster (Wang et al., 2022). The FC layers are also adjusted for optimal mixing of all subnetwork outputs. This algorithm is explained in Section 4.2. The second subnetwork connection information is also loaded from the secondary storage, and CNN output is calculated. The output is sent to the FC layers and the two sets of outputs are aggregated (see Fig 2). The aggregated output is the final output of the system. Therefore, the proposed network executes inferences sequentially by swapping the current connection with next subnetwork connection one by one (see Algorithm 2).

4.1 Problem Formulation

We assume that incremental learning is performed when a novel sample set is presented. Let $\chi_t \equiv \{(x_p^{(t)}, y_p^{(t)})\}_{p=1}^{n_t}$ be the novel sample set at t, where $t = 1,2,\cdots,T$. The learning system learns χ_t at the t-th incremental learning phase by referring old sample sets stored in χ_{old}, where $|\chi_{old}| \leq B$. The old samples are needed for the learner to obtain better class-boundaries. To store current data into $|\chi_{old}|$, an element in χ_{old} is randomly selected to be disposed such that the condition $|\chi_{old}| \leq B$ holds (see Algorithm 1).

After each incremental learning phase, the learning system is evaluated by using test datasets, which include all classes included in χ_t, where $t = 1,2,\cdots,T$. To prevent catastrophic forgetting, the past scores s_{uv} at $t = 1,\cdots,t-1$ are not modified during the later learning phases.

4.2 Incremental Learning by Adding Sub-Networks

During the t-th incremental learning phase, the t-th sub-network $S^{(t)}$ learns $(x_p^{(t)}, y_p^{(t)}) \in \chi_t \cup \chi_{old}$ from scratch. Let $\Delta s_{uv}^{(t)}$ be the change in the score for w_{uv}. $\Delta s_{uv}^{(t)}$ derived by the steepest gradient descent with a momentum term. Therefore,

$$\Delta s_{uv}^{(t)} = 0.9\Delta s_{uv}^{(t-1)} - \eta \frac{\partial \mathcal{L}}{\partial s_{uv}^{(t-1)}}, \tag{6}$$

where $\mathcal{L}$ denotes the cross-entropy loss, $s_{uv}^{(t)}$ denotes the score of the w_{uv} at t, η denotes the learning speed and $\eta < 1$. $s_{uv}^{(t)}$ is modified as

$$s_{uv}^{(t)} = s_{uv}^{(t-1)} + \Delta s_{uv}^{(t)}. \tag{7}$$

Note that the learning of samples in $\chi_t \cup \chi_{old}$ makes the sub-network acquire correct classification boundaries between the target class and the other classes. After learning, the fully-connected aggregation layer in the subnetwork is stored as the connection set for the k maximum scores.

$$S^t = \{(u,v)|h(s_{uv}^{(t)}) = 1\}, \tag{8}$$

where the function $h(\cdot)$ is the same function used in (3). The summarized algorithm is shown in 1. After the learning S^t is stored in the 2nd storage.

```
Data: $\chi_t, \chi_{old}$
Result: $S^t, \chi_{old}$
Initialize $s_{uv}^{(0)}$
for $i = 1$ to $NumEpoch$ do
    for $(x,y)$ in $\chi_t \cup \chi_{old}$ do
        Calculate $\mathcal{L}$ by using $S^t$.
        Score Update by (6), (7).
    end
end
$S^t = \{(u,v) | h(s_{uv}^{(t)}) = 1\}$
for $i = 1$ to $NumEpoch$ do
    for $(x,y)$ in $\chi_t \cup \chi_{old}$ do
        Calculate $\mathcal{L}_{aggregation}$ by using all
          subnetworks.
        modify aggregation layer by (11)(12).
    end
end
for $(x,y)$ in $\chi_t$ do
    if $|\chi_{old}| < B$ then
        $\chi_{old} = \chi_{old} \cup \{(x,y)\}$
    else
        $p \sim |\chi_{old}| U(0,1)$
        $\chi_{old} = \chi_{old} \setminus \{(x_p, y_p)\}$
        $\chi_{old} = \chi_{old} \cup \{(x,y)\}$
    end
end
return $S^t, \chi_{old}$
```

Algorithm 1: Pseudo code for subnetwork learning.

The activation function of the last layer of the subnetwork is a linear function to suit the cross-entropy loss. However, after the learning, the activation function is changed to Rectified Linear Function (ReLU), which was first proposed by (Fukushima, 1975), to prevent negative output values from giving adverse effect.

4.3 Adjusting the Aggregation Layer

After learning the subnetwork, the aggregation layer is adjusted to fit the final output. The aggregation layer consists of a fully-connected linear layer, and is used to re-adjust the old sub-network output without changing the old sub-network parameters. Let Y_i be the i-th aggregation layer outputs, then

$$Y_i \equiv ReLU\left[{W_1}^T y_i + {W_2}^T Y_{i-1}\right], \tag{9}$$

where y_i and Y_{i-1} denote the i-th sub-network output vector and the $i-1$-th final output vectors, respectively. W_1 and W_2 are the weight vectors connected to the i-th sub-network output and the $i-1$-th final output vectors, respectively (see Fig 2). Then, the final classification output is

$$c = \arg\max_j \{Y_{ij}\}, \tag{10}$$

where $Y_i = [Y_{i1}, \cdots Y_{in}]^T$.

This layer is adjusted as follows

$$W_1 = W_1 - \eta(t)\nabla_{W_1}\mathcal{L}_{aggregation} \tag{11}$$

$$W_2 = W_2 - \eta(t)\nabla_{W_2}\mathcal{L}_{aggregation} \tag{12}$$

where $\eta(t)$ is an adaptive learning rate controlled by Adagrad (Luo et al., 2019), $\mathcal{L}_{aggregation}$ denotes mean square error loss given as

$$\mathcal{L}_{aggregation} \equiv \sum_{t \in \chi_t \cup \chi_{old}} \|Y_i - y_t\|^2 \tag{13}$$

```
Data: $x$
Result: $Y_{N_{subnet}}$
for $i = 1$ to $N_{subnet}$ do
    $S = load(S_i)$ //load $S_i$ from the
      secondary storage
    $W_1 = load(W_{i1})$
    $W_2 = load(W_{i2})$
    $y_i \leftarrow \phi(x, S)$ // calculate the $i$-th
      subnetwork.
    $Y_i \equiv ReLU\left[{W_1}^T y_i + {W_2}^T Y_{i-1}\right]$
end
return $Y_{N_{subnet}}$
```

Algorithm 2: Pseudo code for inference.

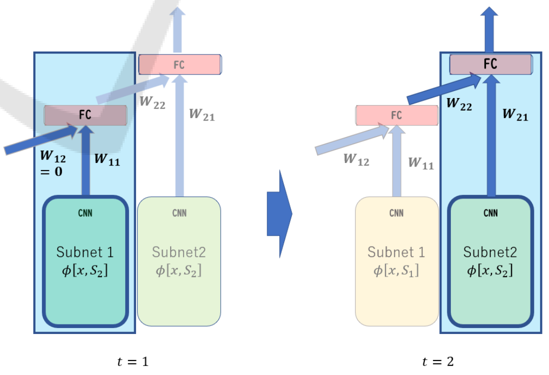

Figure 2: Example of inference phase behavior.

5 POWER-CONSUMPTION ANALYSIS

We assume the power-consumption P is proportional to the computational complexity. So, we estimate P from the complexity. Each sub-network uses only some parameters with the top k scores, reducing P by:

1. Less multiplication of weights and inputs.
2. Less setup for weights. The system only activates some weight connections, not sending parameters.

We consider the demands in table2.

Table 2: Notations for analysing power consumption.

item	notation
Select the top k score connections.	P_{select}
Calculation of the t-th sub-network for a single input vector.	$P_{Csub}(t)$
Calculation of the t-th aggregation-layer for a single input vector.	$P_{Cagg}(t)$
Learning of the t-th sub-network for getting the top k scores.	$P_{Lsub}(t)$
Learning of the t-th aggregation-layer.	$P_{LAgg}(t)$

5.1 Forward Calculation

The electricity of forward calculation for the t-th sub-network and corresponding aggregation layer $P_{forward}(t)$ can be represented recursively as follows.

$$P_{forward}(t) = P_{Cagg}(t) + P_{Csub}(t) + P_{forward}(t-1) \tag{14}$$

where $P_{Cagg}(t)$ is the calculation cost for t-th aggregation layer. $P_{Cagg}(t)$ depends on the number of output connections of $t = 1 \cdots T$ sub-networks. In the t-th round, the aggregation layer's input size is $t \times m$, where m is the output size of each sub-network. Therefore, if the number of classes is N_c (output size of the final layer), the electricity for calculating the aggregation layer is

$$P_{Cagg}(t) = \begin{cases} m \cdot N_c & t = 1 \\ 2 \cdot m \cdot N_c & t \geq 2 \end{cases} \tag{15}$$

Therefore, we obtain

$$P_{forward}(t) = m \cdot N_c(2t-1) + \sum_{i=1}^{t} P_{Csub}(t), \tag{16}$$

where $P_{Csub}(t)$ depends on the number of the active connections, Each sub-network is formed by selecting the top k connections. The selected connections are set to active and the others are set to inactive. This process wastes a lot of electrical power by sending the information of the k connection indexes to the device. This process wastes $P_{select} = k\nu_{send}$ where ν_{send} denotes a positive coefficient for sending data. From this, the sub-network wastes

$$\begin{aligned} P_{Csub}(t) &= P_{select} + k\nu_c \\ &= k(\nu_{send} + \nu_c), \end{aligned} \tag{17}$$

where ν_c denotes a positive coefficient, which depends on the network structure. Therefore,

$$P_{forward}(t) = m \cdot N_c(2t-1) + tk(\nu_{send} + \nu_c), \tag{18}$$

5.2 Learning

The electricity for the learning of the t-th sub-network and the t-th aggregation layer $P_{learning}(t)$ is represented by

$$P_{learning}(t) = P_{Lsub}(t) + P_{Lagg}(t). \tag{19}$$

The power consumption for the learning sub-network $P_{Lsub}(t)$ depends on the number of parameters in the device N_{device} and the number of samples for the t-th round $N_{samples}(t)$.

$$P_{Lsub}(t) = N_{samples}(t) N_{epoch} N_{device} \nu_l \tag{20}$$

where ν_l denotes a positive coefficient for the learning that is related to the network structure, N_{epoch} is the number of epochs. Note that the learning process also includes the calculation process. We assume that ν_l also includes the calculation cost, which is executed during the learning.

The power consumption for the learning aggregation layer $P_{Lagg}(t)$ is

$$P_{Lagg}(t) = N_{samples}(t) \cdot N_{epoch}(t \cdot m \cdot N_c + P_{forward}(t-1)), \tag{21}$$

Therefore,

$$\begin{aligned} P_{learning}(t) &= N_{samples}(t) N_{epoch} \left(N_{device} \nu_l \right. \\ &+ \left. t \cdot m \cdot N_c + P_{forward}(t-1) \right), \end{aligned} \tag{22}$$

5.3 Power-Consumption vs # of Sub-Networks

The power consumption for the forward calculation 5.1 and the learning 5.2 are $O(t)$ for the t-th round. In the t-th round, the total number of sub-networks is t. This means that there is a trade-off between the power-consumption and the ability of the system, which increases by growing the number of sub-networks. This also means that the system should limit the number of sub-networks to a certain number to reduce the power consumption.

6 EXPERIMENTS

Two experiments were conducted in this study. The first experiment involved checking the proposed network Swap-NN for incremental learning tasks using the MNIST dataset. The MNIST dataset was divided into two or five sub-datasets.

The second experiment involved checking the performance of the Swap-NN for the large-scale learning tasks. The CIFEAR-10 dataset was used to achieve this. The CIFEAR-10 was divided into 2 subdatasets, and Swap-NN repeated the learning 2 times.

6.1 Experiment for MNIST Dataset

We developed our program-code based on a sample.

`https://github.com/allenai/hidden-networks/simple_mnist_example.py`

The size of each layer is the same as that of the original program as listed in Table 3.

Table 3: Network size for MNIST: 'Conv', 'fc', 'C', 'RF' and '$f[\cdot]$ denote convolution layer, full-connection layer, channel, receptive field and activation function respectively. 'fc2:L' and 'fc2:R' denote the fc2 layer for the learning and recognition, respectively.

Layer	Input size	C	RF	$f[\cdot]$
Conv1	28×28	32	3×3	ReLU
Conv2	$28 \times 28 \times 32$	64	3×3	ReLU
MaxPool	28×28	64	2×2	–
fc1	1×9216	128	–	ReLU
fc2:L	1×128	10	–	Linear
fc2:R				ReLU

It should be noted that non of the convolution or fully-connecting layers have bias terms. The ReLU was used as the activation function(Krizhevsky et al., 2012). The initialization of each weight strength w_{uv} for the l-th layer was set using the Kaiming-normal distribution(He et al., 2015) $D_l = \mathcal{N}(0, \sqrt{2/n_{l-1}})$, where n_{l-1} denotes the number of inputs from the previous layer. In contrast, the initial scores s_{uv} were sampled uniformly from the set $\{-\sqrt{5}, \sqrt{5}\}$.

6.1.1 Set Up of the Learning Samples

To evaluate the incremental learning ability of the proposed method, its accuracy was evaluated for the test samples after each incremental learning phase.

To this end, we used split MNIST, in which the MNIST was divided into several groups. For example, if MNIST was divided into two groups, the first group was the set of samples for labels $\{0,1,2,3,4\}$, and the second group was the set of samples for labels $\{5,6,7,8,9\}$. During each incremental learning phase, samples from a specific group were presented.

6.1.2 Evaluation

The learning task used in this paper corresponds to 'incremental-class learning' in the literature (Hsu et al., 2018). We compared SwapNN's performances with those of the other methods on 'incremental-class learning' listed in (Hsu et al., 2018). After the last incremental learning phase, the accuracy was evaluated using all test samples. If the learning machine causes the catastrophic forgetting due to the incremental learning, the accuracy after the last incremental learning will be low. If the learning method yielded an accuracy of approximately 100$\times$ index of the incremental learning phase/total number of incremental learning, the learning method had achieved the desired results.

6.1.3 Results

Preliminary results are shown in Table 4, This table shows the accuracy of our proposed system, Swap-NN, and other learning methods after learning the MNIST dataset, which is divided into five sub-groups. The table also shows the accuracy of other learning methods: EWC (Kirkpatrick et al., 2017), SI (Zenke et al., 2017) , Naive-rehearsal (Hsu et al., 2018) , MAS (Aljundi et al., 2018) and LwF (Li and Hoiem, 2016) , which are the same as the data reported in (Hsu et al., 2018). Additionally, we list the Swap-NN with two subnetworks, which learns MNIST divided into two groups, at the bottom of the table.

The hyper parameter k of Swap-NN, which is the number of the most important connections for yielding the output, was set to $k = \kappa \times N_{device}$. Note that $\kappa(<1)$ denote the ratio and $\kappa = 0.7$. The replay-buffer size B of Swap-NN was set to 12000. The Swap-NN that learned five classes par each incremental learning phase showed the highest accuracy (see the bottom of Table 4).

We evaluated the memory consumption and computational complexity of the models. The network size of the competing models except Swap-NN was almost the same, and the number of variable weight parameters was 1708446. The total number of weight connections in Swap-NN was 1199648. However, Swap-NN used $\kappa\times$ 1199648 connections to compute a single subnetwork output for the current input. Therefore, in the case of the Swap-NN having two subnetworks, it uses 2 $\times\kappa\times$ 1199648 = 1679927 connections to obtain the final output. This number of connections is not much different from the competing models.

6.2 Experiment for CIFAR-10

Swap-NN was also tested for a more challenging learning task by using CIFAR-10. CIFAR-10 dataset was divided into two sub-datasets, thus each of the two sub-datasets includes five classes dataset. The network size is shown in Table 5, whose total number

Table 4: Accuracy for MNIST dataset. The results reported in (Hsu et al., 2018) are listed except those for Swap-NNs.

Method	Incremental class learning
EWC	19.80 ± 0.05
SI	19.67 ± 0.09
MAS	19.52 ± 0.29
LwF	24.17 ± 0.33
Naive rehearsal	90.78 ± 0.85
Swap-NN (five sub-networks)	90.92 ± 0.97
Swap-NN (two sub-networks)	94.29 ± 0.32

of parameters are the same as that of the network used in the previous experiment. Note that this network size is relatively smaller than the normal network size for the learning of CIFAR-10.

Table 5: Network size for CIFEAR-10: 'Conv', 'fc','RF', 'C' and 'BN' denote convolution layer, full-connection layer , receptive field, channel and batch-normalization layer respectively. 'fc2:L' and 'fc2:R' denote the fc2 layer for the learning and recognition, respectively.

Layer	Input size	C	RF	$f[\cdot]$
Conv1	$32 \times 32 \times 3$	64	3×3	ReLU
Conv2	$32 \times 32 \times 64$	64	3×3	ReLU
BN1	$32 \times 32 \times 64$	64	–	–
MaxPool1	32×32	64	2×2	–
Conv3	$32 \times 32 \times 64$	128	3×3	ReLU
Conv4	$32 \times 32 \times 128$	128	3×3	ReLU
BN2	$32 \times 32 \times 128$	128	–	–
MaxPool2	32×32	128	2×2	ReLU
Conv5	$32 \times 32 \times 128$	256	3×3	ReLU
Conv6	$32 \times 32 \times 256$	256	3×3	ReLU
BN3	$32 \times 32 \times 256$	256	–	–
MaxPool3	32×32	256	2×2	ReLU
fc1	1×4096	256	–	ReLU
fc2:L	1×256	10	–	Linear
fc2:R				ReLU

We compare the Swap-NN performance with that of the same sized neural network, which has learned all training samples of the CIFAR-10. The performances of the both networks after finishing the learning were evaluated by using the validation set in the CIFAR-10. The test was repeated 10 times by changing the initial weight connections randomly. The results were averaged over the 10 trials and 95 percent confidence intervals were also estimated.

6.2.1 Results

The buffer size B was set to 20000.

We compared the accuracy rate of SwapNN, which learned five classes at a time, with that of a network that learned all classes simultaneously, under the same conditions with those of SwapNN (the same network size and 5 epochs). Table 6 shows the accuracy after the initial and incremental learning.

This result suggests that the accuracy of our proposed SwapNN is about 5.3 points less than the network, which learns all classes at a time. However, after the initial learning, the accuracy for the 1st five classes was 79.8 %. Each subnetwork has no opportunity to learn the class boundaries between the future presented classes. Therefore, each subnetwork output will conflict to the future unknown classes. To overcome this problem, the last aggregation layer has to adjust to this kind of mismatch. However, there are possibility that the last aggregation layer could not adjust this conflict completely in the case of CIFAR10.

Table 6: 95% confidence interval of SwappNN accuracy after each incremental learning on CIFAR-10 (The 'Competitor' is the network that learned all classes at once.).

After initial learning	After the incremental learning	Competitor
39.42 ± 0.20 % (79.8% for learned 5 classes)	65.09 ± 1.17 %	70.95 ± 0.53 %

7 DISCUSSION

7.1 Computational Complexity vs Accuracy

Obviously, the computational complexity is low if k is small. Instead of the light weight computation of the sub-network, the proposed method has to repeat the setup of the sub-network and calculation of the sub-network. To reduce the computation time, k should be small enough to realize a quick computation. However, too small k will yield the degradation of the performance.

7.2 Suitable Learning Strategy for Each Sub-Network

In the classification tasks, each sub-network should learn the class-boundaries. Without the information for the boundaries, the learning machine will fail to acquire a high performances. So, the number of classes, of which the learning samples presented during an incremental learning phase, should be large.

The connection set found in each learning phase are stored in the second storage device and fixed so far. However, the connection set should be updated when environment is changed. The distillation strategy employed in Foster (Wang et al., 2022) is also useful for reducing and re-adjusting the subnetworks. The distillation operation will give a clear opportunity to make each subnetwork and corresponding aggregation layer learn class boundaries.

8 CONCLUSION

In this paper, a hidden neural network based incremental learning method: Swap deep neural network is proposed. This model has been followed two advantages:

1. The hidden neural network can reuse its neurons in several different tasks by reconfiguration of the neural network circuit. So, this architecture is suitable for a small embedded systems, where the storage capacity is limited.
2. The sub-network is not modified after the creation. This means that the system does not cause the catastrophic forgetting.

The simulation results suggest that Swap-NN realizes an effective execution of large-scale neural networks with a small amount of resources.

REFERENCES

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. (2018). Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV 2018)*.

French, R. M. (1997). Pseudo-recurrent connectionist networks: An approach to the "sensitivity stability" dilemma. *Connection Science*, 9(4):353–379.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *TRENDS in Cognitive Sciences*, 3(4):128–135.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Fukushima, K. (1975). Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20(3):121–136.

Hayes, T. L., Kafle, K., Shrestha, R., Acharya, M., and Kanan, C. (2019). Remind your neural network to prevent catastrophic forgetting. *CoRR*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv.org*.

Hirose, K., Yu, J., Ando, K., Okoshi, Y., García-Arias, Á. L., Suzuki, J., Chu, T. V., Kawamura, K., and Motomura, M. (2022). Hiddenite: 4k-pe hidden network inference 4d-tensor engine exploiting on-chip model construction achieving 34.8-to-16.0tops/w for cifar-100 and imagenet. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3.

Hsu, Y.-C., Liu, Y.-C., Ramasamy, A., and Kira, Z. (2018). Re-evaluating continual learning scenarios: A categorization and case for strong baselines. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc.

Ishikawa, M. (1996). Structural learning with forgetting. *Neural Networks*, 9(3):509–521.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., and Ramalho, T. (2017). Overcoming catastrophic forgetting in neural networks. *Proceeding of the National Acacemy of United States of America*, 114(13):3521–3526.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

Li, Z. and Hoiem, D. (2016). Learning without forgetting. In *European Conference on Computer Vision 2016*, pages 614–629. SCITEPRESS ? Science and Technology Publications, Lda.

Luo, L., Xiong, Y., Liu, Y., and Sun, X. (2019). Adaptive gradient methods with dynamic bound of learning rate. In *International Conference on Learning Representations ICLR2019*.

Mallya, A. and Lazebnik, S. (2018). Packnet: Adding multiple tasks to a single network by iterative pruning. In *CVPR, Conference on computer vision and pattern recognition 2018*, pages 7765–7773.

Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., and Rastegari, M. (2020). What's hidden in a randomly weighted neural network? *arXiv.org*.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *CoRR*.

Sadel, J., Kawulok, M., Przeliorz, M., Nalepa, J., and Kostrzewa, D. (2023). Genetic structural nas: A neural network architecture search with flexible slot connections. In *GECCO'23 Companion: Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pages 79–80. Association for Computing Machinery.

Wang, F.-Y., Zhou, D.-W., Ye, H.-J., and Zhan, D.-C. (2022). Foster: Feature boosting and compression for class-incremental learning. In *ECCV 2022: 17th*

European Conference, Proceedings, Part XXV, pages 398–414.

Yamauchi, K., Yamaguchi, N., and Ishii, N. (1999). Incremental learning methods with retrieving interfered patterns. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 10(6):1351–1365.

Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.*

Zheng, N. and Mazumder, P. (2020). *Learning in Energy-Efficient Neuromorphic Computing: Algorithm and Architecture Co-Design*. John Wiley & Sons Inc.