Ph.D. Thesis
박사 학위논문

An Effective and Efficient Method for Tweaking Deep Neural Networks

Jinwook Kim (김 진 욱 金 珍 旭)

Department of
Information and Communication Engineering

DGIST

2021

Ph.D. Thesis
박사 학위논문

An Effective and Efficient Method for Tweaking Deep Neural Networks

Jinwook Kim (김 진 욱 金 珍 旭)

Department of
Information and Communication Engineering

DGIST

2021

An Effective and Efficient Method for Tweaking Deep Neural Networks

Advisor: Professor Daehoon Kim
Co-Advisor: Professor Min-Soo Kim

by

Jinwook Kim
Department of
Information and Communication Engineering
DGIST

A thesis submitted to the faculty of DGIST in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Information and Communication Engineering. The study was conducted in accordance with Code of Research Ethics[1].

05. 24. 2021

Approved by

Professor Daehoon Kim (Signature) [illegible]
(Advisor)

Professor Min-Soo Kim (Signature) [illegible]
(Co-Advisor)

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of DGIST, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

An Effective and Efficient Method for Tweaking Deep Neural Networks

Jinwook Kim

Accpeted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

05. 24. 2021

Head of Committee	김 민 수 Prof. Min-Soo Kim	[illegible]
Committee Member	김 대 훈 Prof. Daehoon Kim	[illegible]
Committee Member	장 재 은 Prof. Jae-Eun Jang	[illegible]
Committee Member	박 경 준 Prof. Kyung-Joon Park	[illegible]
Committee Member	최 지 환 Prof. Jihwan Choi	[illegible]

Ph.D/IC
201142005

김진욱. Jinwook Kim. An Effective and Efficient Method for Tweaking Deep Neural Networks. Department of Information and Communication Engineering . 2021. 77 pages. Advisor Prof. Dae-hoon Kim. Co-Advisor Prof. Min-Soo Kim.

Abstract

Today, training of deep learning models is conducted in the direction of learning using given training data and obtaining results with high overall accuracy for all classes for the validation data. That is, even if the accuracy of specific classes is very poor, the optimization proceeds in the direction of improving the overall accuracy by predicting correct answers for as many samples as possible regardless of class. Even if additional training is performed to improve a specific class having poor accuracy, the accuracy of each class fluctuates greatly as the learning progresses. That is, even if training is stopped at any point, classes that are significantly less accurate than average accuracy occur. This is an inevitable problem with current deep learning training methods. In particular, in applications where the accuracy of a specific class is important, such as medical artificial intelligence systems, this problem can be fatal. To solve this problem, we need a method to improve the accuracy of the specific target class, which is important in the given application, while maintaining the overall accuracy.

The first part of this dissertation, we propose the Synaptic Join method that precisely adjusts the accuracy of a specific class (target class) that the user wants to improve, rather than performing additional learning on an already learned deep learning model. The proposed synaptic join method finds active neurons that can improve the accuracy of the target class and minimize the damage on the accuracy of non-target classes. The active neurons are connected in the form of synapses to the output neurons to improve the target class. The proposed method can tweak the original model according to the user's request while maintaining the original model as it is. Also, we introduce a technique that can quickly process synaptic join operations on the limited memory of multiple GPUs. Experimental results compared to the retraining methods show that our method can better control and effectively improve the accuracy of target classes.

In the second part of this dissertation, we propose Network Augmentation with Active Neurons. The network augmentation method is one of the widely used techniques to extend a neural network model or perform transfer learning. The method adds a new hidden layer to the model that has already been trained and performs fine-tuning to obtain more accurate models or models that fit the purpose of given applications. However, adding layers to a large-scale deep neural network model can increase the learning time and the number of parameters required for training the model. We propose an augmented network with only a small number of active neurons as input values for efficient training. In addition, unlike general network augmentation, the proposed method learns only the augmented model while maintaining the original model. Compared to the depth augmented method, we show that our method can achieve similar or better accuracy with fewer parameters and shorter training time in all experiments.

In summary, this dissertation proposes the methods to improve the model to suit the user's purpose without changing the original deep neural network models. For improving the accuracy of the target class in the original model while minimizing the effect on the accuracy of the non-target class, we propose Synaptic Join method. To efficiently augment deep neural networks, we propose a network augmentation method using active neurons. The proposed methods are effective and efficient methods of adjusting neural networks according to the user's purpose and are useful in customized artificial intelligence service applications.

Keywords: Deep neural networks, synaptic join, network augmentation.

List of Contents

List of Tables

List of Figures

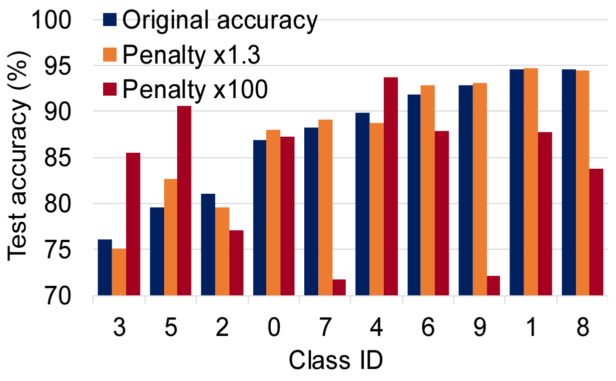
Chapter 1. Introduction

1.1 Introduction

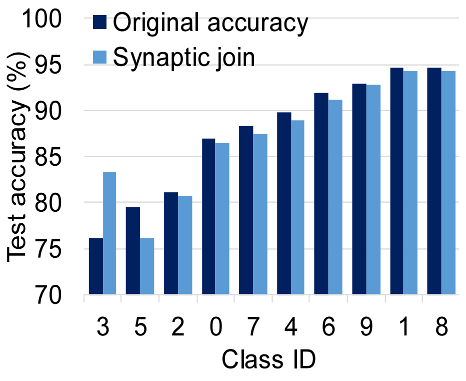
Deep neural networks have been widely used in many real applications in various areas such as computer vision [4, 48, 66, 68, 85, 87, 100], speech recognition [8, 21, 31, 36, 40, 64, 102] and natural language processing [18, 20, 41, 65, 88, 91, 96]. In general, they have a layer-by-layer architecture and are trained using tensor-based operations including convolution and matrix multiplication. In supervised learning, a neural network is trained to achieve the maximum overall accuracy through a learning process using given training data. In such cases, the accuracies of classes are usually different from each other. In particular, the accuracy of some classes might not be good enough, although these classes are more important than the other classes in certain applications (*e.g.*, in the medical area) or for certain users (*e.g.*, in customized AI service). This problem can occur not only for specific classes (*e.g.*, car class) but also for some kinds of data objects, *i.e.*, implicit sub-classes (*e.g.*, truck objects).

However, it is nontrivial to fix or adjust a neural network to further improve the accuracy for specific classes or objects after training because the whole network is already optimized through hundreds of thousands of iterations and there exist complex dependencies among the features and outputs.

A potential method to improve the accuracy of specific classes selectively is to perform additional training (*i.e.*, simple retraining) with a larger loss penalty than that of the other classes being assigned to the target classes. However, when the model has a complex structure (*e.g.*, deep neural networks), this simple retraining approach may not be effective to improve the accuracy of the target classes. Fig. 1.1a shows the result of the simple retraining of a deep neural network for CIFAR-10, performed to improve the accuracy of the class having the lowest accuracy, *i.e.*, class#3 by assigning a $\times 1.3$ loss penalty to class#3. A network in network (NIN) model of an overall accuracy of 87.57% [60] is used. The result shows that after retraining, the accuracy of class#3 is not improved and instead degraded, while the accuracy of the other classes (*e.g.*, class#5) is improved. Through numerous other experiments, the results of the weighted retraining method are unpredictable and uncontrollable when using relatively small loss penalties. Although using considerably large loss penalties could improve the accuracy of the target class, it might also adversely influence the results. The red bars in Fig. 1.1a show the results for the case when a loss penalty of $\times 100$ is assigned to the same class. It can be note that the accuracy of class#3 considerably improves from 76.1% to 85.5%; however, the overall accuracy is severely deteriorated from 87.57% to 77.59%.



(a) $\times 1.3$ & $\times 100$ loss penalties



(b) Synaptic join

Figure 1.1: Results of the simple retraining and synaptic join method (NIN for CIFAR-10, target: class#3).

To address this issue, we define a problem involving the adjustment of a given deep neural network to improve the accuracy of the specific classes, presented as Definition 1. The specific classes for which the accuracy is intended to be improved are referred to as *target* classes, and the other classes are termed as *off-target* classes. The target classes are user or application-specific. We consider a deep neural network as a weighted graph consisting of nodes and edges, where the nodes indicate neurons, and the edges indicate synapses (or weights). A given original neural network that is already optimized is denoted as nn. The overall accuracy of nn may be maintained by only slightly degrading the accuracy of the off-target classes while increasing the accuracy of the target classes. We can generate a neural network having synapses different from those of the original nn by applying one of three types of tweaking processes: adding new synapses across the layers to nn, deleting the existing synapses from nn, or changing the weights of the existing synapses. We denote the changes in synapses as Δnn.

Definition 1. *For a given deep neural network nn, the problem is to find Δnn such that $nn + \Delta nn$ can improve the accuracy of the user- or application-specific target classes while maintaining the original overall accuracy.*

In this study, we consider the approach of adding additional synapses to the original network nn, since the approach of deleting the existing synapses is usually used for compressing neural networks, as described in many existing studies including [16, 33, 34, 35, 38, 51, 51, 54, 56, 99], and the approach of changing the existing weights is similar to the simple retraining method described above. Δnn is user- or application-specific, and thus a neural network $nn+\Delta nn$ can be made to work differently for different users and applications by only changing Δnn.

In the first part of this dissertation, we propose the *synaptic join* method, which adds additional synapses from some nodes in the hidden layers to the target class nodes across the layers to improve the accuracy of the target classes almost without sacrificing the accuracies of the other classes. If a set of nodes $\{x\}$ in the hidden layers has a relatively stronger signal than that of the other nodes for a target class, we call the corresponding nodes $\{x\}$ as *active neurons* for the target class. The primary concept of our method is to strengthen the target class by adding a synapse with a suitable weight from the active neurons to the target class node.

Fig. 1.1b shows the result obtained using the proposed method for the target class#3. The accuracy of class#3 is considerably and selectively improved while sacrificing the accuracy of the off-target classes to a limited extent and retaining the same overall accuracy. These findings indicate that the result obtained using our method is predictable and controllable.

From a technological perspective, it is difficult to find the effective active neurons from a huge number of possible candidates and to determine the suitable weights of the synapses between the selected neurons and the target class node. In general, adding new arbitrary synapses to an already optimized network tends to degrade the accuracy. To overcome this issue, the proposed method finds the effective active neurons and determines the suitable weights of the synapses between the source and destination.

In the second of this dissertation, we present augmented deep neural networks with active neurons. Network augmentation is one of the widely used techniques for growing capacities of network models or transferring knowledge from already trained models [89]. By inserting new layers and fine-tuning the augmented networks, users can obtain models with better accuracy or get models that fit the purpose of their application. However, adding layers in a large-size deep neural network model can cause the model to grow too much in terms of the number of parameters, thus increase the training time. We propose the augmented networks, which feed only the active neurons to a small network model for efficient learning. The synaptic join method exploits only the linearity of every synapse to tweak a given neural network model; however, the proposed augmentation with active neurons can train models to learn non-linearity.

1.2 Main contributions

The main contributions of Synaptic Join are as follows:

- We propose a tweaking method for neural networks that can improve the accuracy of specific classes of interest without performing any retraining.
- We propose an algorithm θ that can evaluate the performance of all the possible candidate synapses in the training data.
- We propose the synaptic join and synaptic retraining methods based on θ.
- Through extensive experiments, we demonstrate that the proposed methods can control the test accuracy of the target and off-target classes in a more predictable and controllable manner than the other methods.

The main contributions of Augmentation with Active Neurons are as follows:

- We propose an augmentation method for neural network models that can efficiently improve the accuracy of all classes without modifying the original models.
- We expand the synaptic retraining method to learn non-linearity by configuring augmented models as small MLP models.
- Through extensive experiments, we demonstrate that the proposed methods can improve the accuracy with short training time and small number of parameters.

1.3 Structure of thesis

The structure of this dissertation is organized as follows. Chapter 2 introduces the background of this dissertation. In Section 2.1, we review commonly used deep neural networks models. In Section 2.2, we explain about network augmentation method. Chapter 3 describes and evaluates synaptic join method. Section 3.1 introduces the comparison simple retraining-based methods, Section 3.2 describes the proposed synaptic join method, and Section 3.3 presents the results of the experimental evaluation. Chapter 4 describes and evaluates our augmentation method using active neurons. Chapter 5 discusses the related work of this dissertation. Finally, conclusions are drawn in Chapter 6.

Chapter 2. Background

2.1 Deep Neural Network Models

2.1.1 Deep Neural Network

Deep neural network (DNN) is a sort of deep learning algorithm that attempts high-level abstraction through a combination of a number of nonlinear transducers [6, 14, 30]. The purpose of DNN is to teach computers how to think of humans by forming and learning artificial neural networks that resemble human brains. Today, DNN models show high levels of performance in computer vision [12, 32, 43, 57, 67, 98], speech recognition [3, 5, 58, 63, 69, 76, 77, 97], natural language processing [7, 19, 22, 29, 80, 95, 101], signal processing [9, 23, 24, 61, 71, 93, 103], and medical data analysis [10, 11, 15, 26, 28, 52, 55, 72, 75, 79, 82] applications. DNN has many variations, but in common, numerous hidden layers are stacked, and learning is performed by adjusting the weights connecting hidden neurons so that each hidden neuron can better express abstract features as the input data passes through each layer.

Learning of DNN is done with an error back-propagation [30] algorithm. Eq 2.1 represents the stochastic gradient descent that updates each weight w_{ij} as the model is trained to solve the multi-class classification problem. At this time, each weight w_{ij} is updated through the stochastic gradient descent method, such as eq 2.1.

$$\Delta w_{ij}(t+1) = \Delta w_{ij}(t) + \gamma \frac{\partial(-\sum_j d_j \log(p_j))}{\partial w_{ij}}. \tag{2.1}$$

Where, d represents the target probability for output unit j, p represents the probability output for j, and $-\sum_j d_j \log(p_j)$ stands for a cost function of cross entropy. γ is the learning rate and t is the iteration number of the current learning.

2.1.2 Convolutional Neural Networks

A convolutional neural network (CNN) model is one of the most commonly applied DNN to analyze visual images [86]. CNN is composed of several convolutional layers in charge of feature abstraction (representation) and artificial neural networks for output to serve user's purposes such as classification. With this structure, it is easy to use the input data of a two-dimensional structure; therefore, it is widely used for processing image, video, audio, and medical data. Representative algorithms of CNN include LeNet-5, VGG, GoogLeNet, ResNet, and DenseNet.

LeNet-5 [50] is the name of the CNN model developed in 1998 by Yann LeCun research team, which first developed CNN. The model has an input layer, three convolution layers, two subsampling (pooling) layers, one fully-connected layer, and an output layer. In the convolution layers, feature maps are obtained through a convolution operation using an input image as 5x5 kernels (filter). The subsampling layers reduce the size of each feature map by half. Fig 2.1 shows the LeNet-5 model. The calculated feature maps pass through the fully-connected layer and the output layer to predict the class label. At this time, the loss is calculated as much as the difference between the output and the actual correct answer, and the kernel and weight are modified by back-propagating it.

VGGs are relatively simple CNN models developed by Simonyan et al. (2014) [81]. It was developed to improve the accuracy of large-scale image recognition by increasing the depth of the CNN model. Fig 2.2 shows representative VGG models, VGG16 and VGG19. Instead of using a convolution layer with a large filter (*e.g.*, 7×7), VGG consists of a stack of several convolution layers having small filters of 3×3 size. This structure has two advantages:

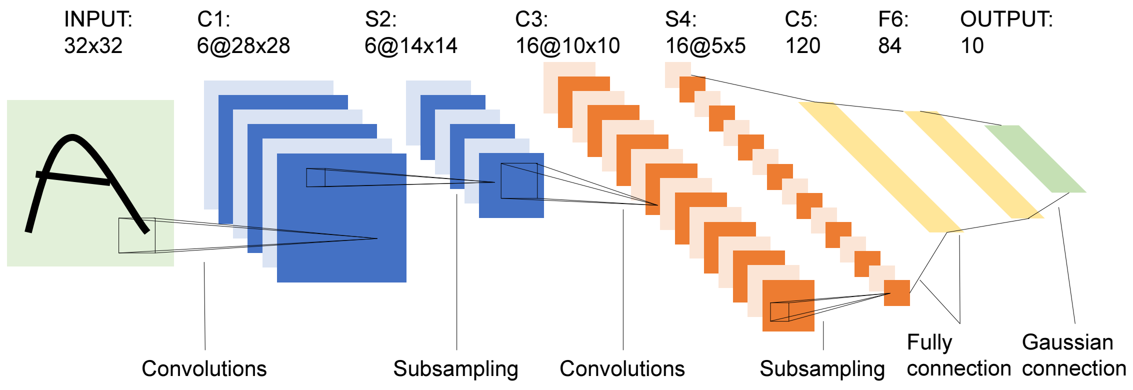


Figure 2.1: Architecture of LeNet-5 [50].

1) it is possible to learn more non-linearity from the input feature due to the many activation functions, and 2) it can learn faster because the number of weights to be learned is reduced. Their study showed that in image classification, the accuracy of VGG models improved with increasing their depth (up to 19 layers). However, later studies such as He et al. (2015) [37] pointed out that simply increasing the depth of the model too much has a problem in that the gradient vanishes in the error back-propagation learning stage, which in turn lowers the performance.

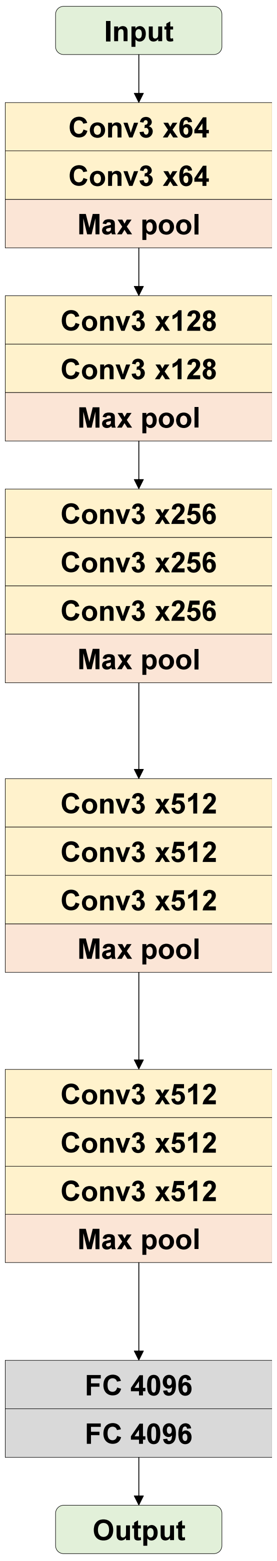
GoogLeNet was proposed by Szegedy et al. (2014) [83] and achieved the highest performance in the ImageNet Large Scale Visual Recognition Challenge 2014 (ILSVRC2014)[74]. As the depth of networks gets deeper, the number of parameters required for learning and the amount of computation required for learning increase, and as a result, the risk of falling into the vanishing gradient [42] problem increases. GoogleNet constructed a model by stacking inception modules to solve this problem. Fig 2.3a shows a naïve form of the inception module. Each module extracts features from the input image using a multi-scale Gabor filter [78]. The 1×1 convolution filter better preserves the spatial information of the input feature, while the 3×3 or 5×5 filter preserves more abstract information. Fig 2.3b shows the inception module used in GoogLeNet. In this version, a 1×1 convolution layer is added to each layer of the naïve module. The additional 1×1 convolution layers can reduce the dimension of feature maps, thereby reducing the amount of computation while maintaining feature-related information. Fig 2.3c shows the network structure of GoogLeNet. GoogLeNet has auxiliary classifiers in the middle of the stacked inception layers. The auxiliary classifier plays a role in mitigating gradient vanishing by generating an error in the middle of the model.

ResNet [37] is a model proposed to solve the phenomenon that the performance decreases as the layer of the model becomes too deep. As the layer of the model gets deeper, more differentiation needs to be done, so even if back-propagation is performed, the difference values become smaller as the layers in front are. As a result, the degree of weight affecting the output decreases, *i.e.*, a gradient vanishing problem occurs. ResNet overcomes this problem by introducing skip connections between layers. When input x, output $F(x)$ of block, output of block with skip connection is $F(x)+x$. Since the differential gradient is $F'(x)+1$, it is possible to guarantee a minimum gradient of 1 or more; thus, gradient vanishing can be solved. A block with a skip connection in which input x is directly connected to $F(x)$ is called an identity block, and its variation, the bottleneck block, is a method of adding x to $F(x)$ after 1×1 convolution operation. Usually, a residual block is composed of one identity block and multiple bottleneck blocks, and several residual blocks are stacked to construct a ResNet model. Fig 2.4 shows the components of ResNet and an example of the ResNet model. Fig 2.4a shows the identity block, Fig 2.4b shows the bottleneck block, and Fig 2.4c shows the ResNet-152 model with a total of 152 layers stacked with four residual blocks.

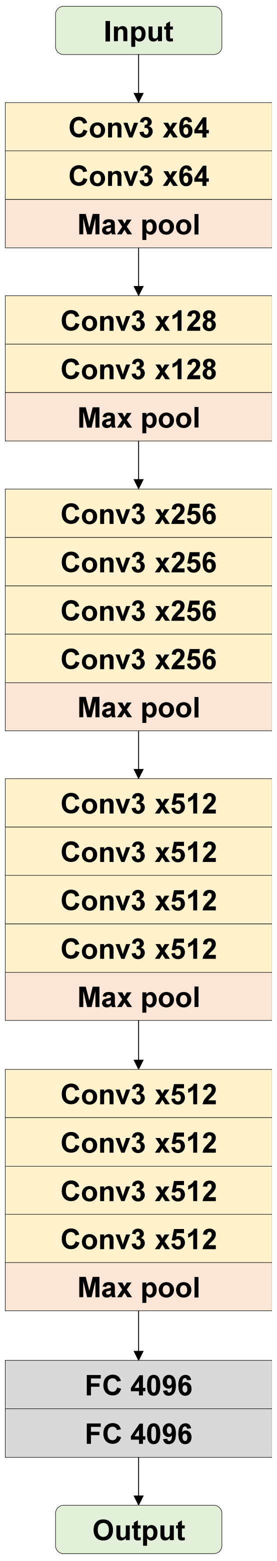
DenseNet [44] has a skip connection similar to ResNet; however, DenseNet connects each successive layer (*i.e.*, densely connected). That is, $L(L+1)/2$ direct connections exist for L consecutive convolution layers. Through this structure, DenseNet has the following advantages: alleviate the vanishing gradient, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. DenseNet also has a bottleneck structure similar to ResNet. One difference with ResNet is that it concatenates the input and the output of the bottleneck rather than adding them. Fig 2.5 shows each component of DenseNet and an example of the DenseNet model made from them. Fig 2.5a shows the bottleneck block. Fig 2.5b shows a dense block with six stacked bottlenecks and densely connected. Fig 2.5c shows the DenseNet-121 model consisting of four dense blocks and having a total of 121 layers.

2.2 Network Augmentation

The depth augmented (DA) and width augmented (WA) methods [89] are proposed for growing the topology of a neural network for fine-tuning. Given an already trained model, the DA method adds a new fully-connected layer after the last representation module, while the WA method adds a new fully-connected layer aside along with the last representation layer. For the both methods, parameters in the new layer are randomly initialized. Fig. 2.6 shows examples of DA and WA models [89] growing form AlexNet [48] structure. Most of deep neural networks consist of two parts, one is a representation module which extracts features of input example and the other is classifier module which predicts the label through the extracted features. In Fig. 2.6a, the original AlexNet has five convolutional layers (from C1 to C5) and two fully-connected layers (FC6 and FC7) for representation module; the model has one fully connected layer for classification module. Fig. 2.6b shows DA model which had additional FC7a layer after the representation module. Fig. 2.6c shows WA model which had additional FC7+ layer aside along the last representation layer (*i.e.*, FC7 layer).

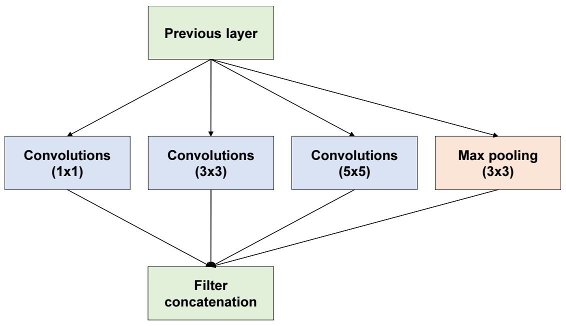


(a) VGG 16

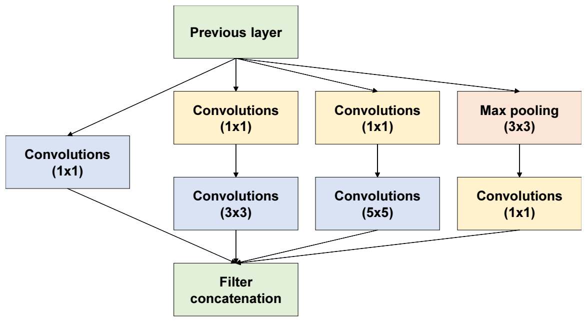


(b) VGG 19

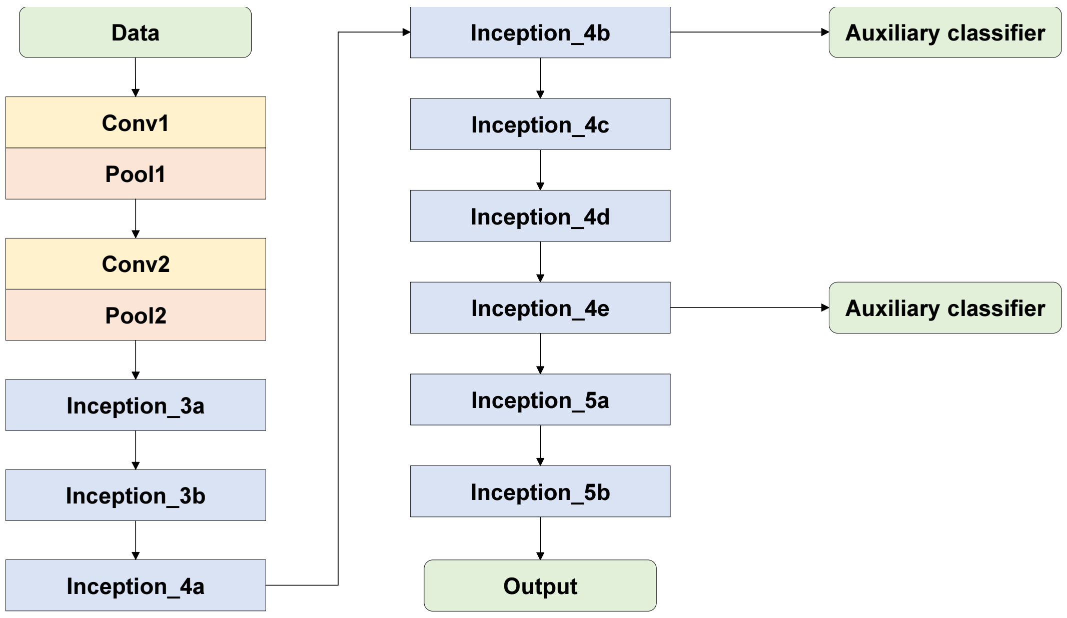
Figure 2.2: An example of VGG models.



(a) Inception module, naïve version

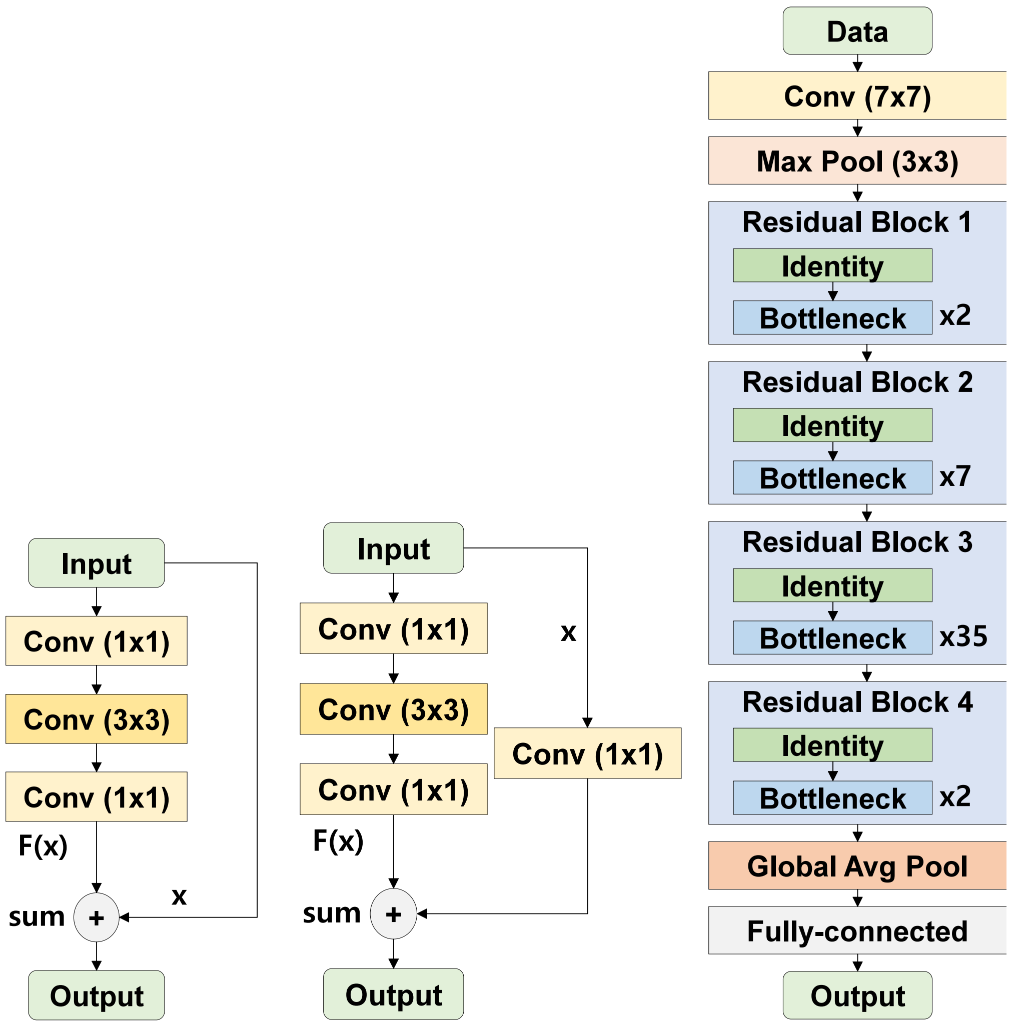


(b) Inception module with dimensionality reduction



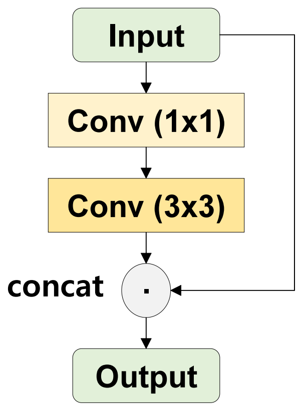
(c) GoogLeNet with auxiliary classifiers

Figure 2.3: Architectures of inception and GoogLeNet.

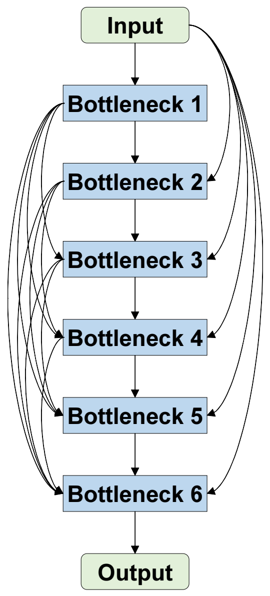


(a) Identity block (b) Bottleneck block (c) ResNet-152

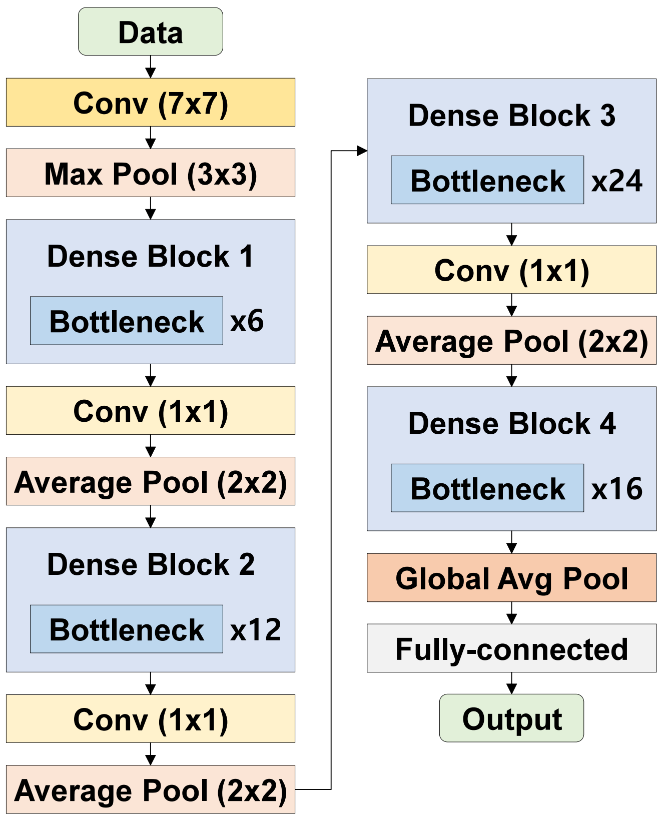
Figure 2.4: An example of the components of ResNet and ResNet-152 model.



(a) Bottleneck block

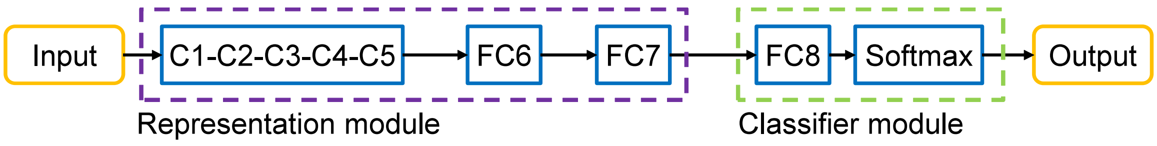


(b) Dense block

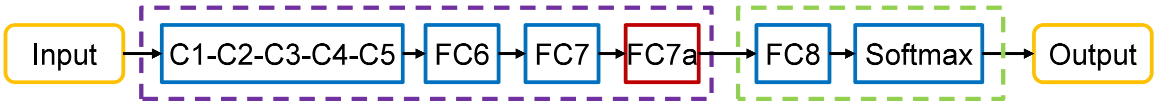


(c) DenseNet-121

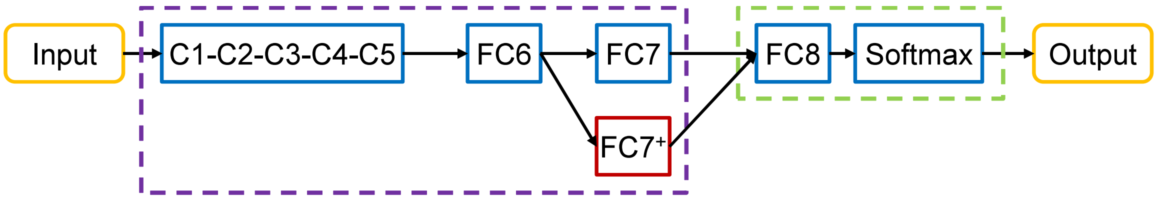
Figure 2.5: An example of the components of DenseNet and DenseNet-121 model.



(a) The original AlexNet [48]



(b) Depth augmented (DA) AlexNet [89]



(c) Width augmented (WA) AlexNet [89]

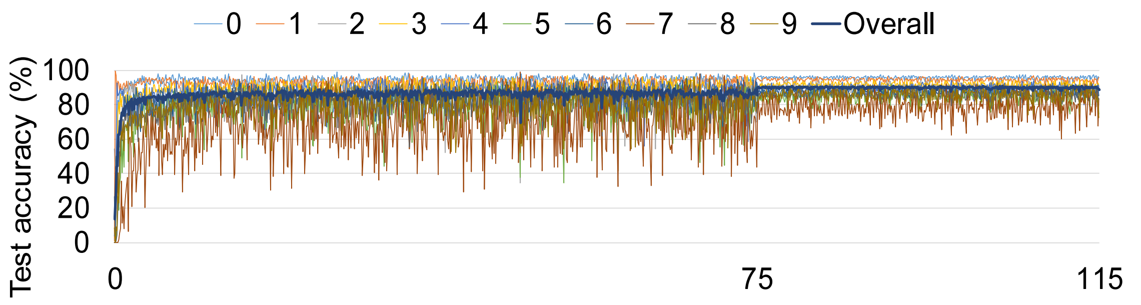
Figure 2.6: An example of DA and WA models on AlexNet.

Chapter 3. Tweaking Deep Neural Networks

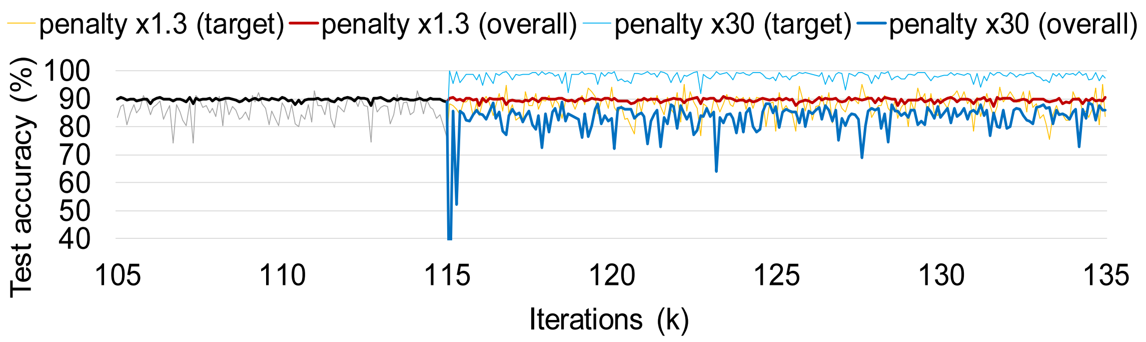
3.1 Simple Retraining Method

The reason that the result of the simple retraining method with a small loss penalty is unpredictable and uncontrollable in Fig. 1.1a is due to the fluctuation of the accuracy of each class during training. Fig. 3.1a shows the overall and per-class test accuracy curves of a 3-layer CNN for the SVHN during 115 k iterations. The learning rate starts with 0.001 and is reduced by 1/10 at 75 k iterations. In the figure, the per-class accuracy fluctuates widely, whereas the overall accuracy fluctuate narrowly. After reducing the learning rate, the absolute difference between the overall and per-class accuracy is reduced, but the relative difference is maintained. Wherever we stop training, some specific per-class accuracy would be much worse than some other per-class ones.

This result may be unavoidable under the current optimization of a neural network based on the overall accuracy. We consider the 3-layer CNN trained for the 115 k iterations in Fig. 3.1a as a given neural network. Class#9 has the lowest accuracy in the CNN, and so, we set it to the target class. Then, Fig. 3.1b shows the overall and target (class#9) test accuracy curves of the simple retraining method in the range between 115 k and 135 k iterations using $\times1.3$ and $\times30$ loss penalties. Retraining with a small loss penalty (*e.g.*, $\times1.3$) to the target class still shows the fluctuation of the per-class accuracy. Thus, it is difficult to improve the accuracy of the target class in a controllable manner. In general, stopping training at the point where the target accuracy becomes good, *e.g.*, similar to or larger than the overall accuracy, makes the accuracy of some of the other classes unpredictably worse. Retraining with a large loss penalty (*e.g.*, $\times30$) can improve the target accuracy markedly, but in that case, the overall accuracy is degraded severely, as in Fig. 3.1b.



(a) Overall and all per-class accuracy.



(b) Overall and class#9 accuracy during simple retraining

Figure 3.1: Overall and per-class test accuracy (3-layer CNN for SVHN, target: class#9)

Table 3.1 shows the target and overall validation accuracy while varying loss penalties between $\times 1.3$ and $\times 30$. We can extract a validation set from the training set and evaluate the target and the overall accuracy using the validation set for some candidate penalties. We note that the validation accuracy tends to be a little bit higher than the test accuracy in Section 3.3. As the loss penalty increases, the target accuracy increases, while the overall accuracy decreases. That is, there exists a trade-off between the target and overall accuracy. For the retraining method, we use a loss penalty that can achieve the best target accuracy while not degrading the overall accuracy too much (less than 1.0%) in this dissertation. In Table 3.1, $\times 2.0$ is such a penalty. Determination of proper penalty may be time-consuming since it needs to repeat a retraining process b times, where b is the number of candidate penalties.

Table 3.1: Validation accuracy between 126 k and 135 k iterations.

	base	x1.3	x1.6	x2.0	x5.0	x10	x20	x30
target	91.4	93.0	94.2	95.2	97.1	97.9	98.5	98.5
overall	90.6	90.1	89.9	89.7	88.3	87.2	83.0	81.2

3.2 Synaptic Join Method

In this section, we explain our synaptic join method that can determine the effective synapses from the active neurons to the target classes and their weights for tweaking neural networks. We present the tables required for synaptic join in Section 3.2.1 and the algorithm θ that can evaluate the performance of the synapses using the tables in Section 3.2.2. Then, we explain the synaptic join method based on θ in Section 3.2.3 and the retraining of the newly added synapses in Section 3.2.4.

3.2.1 Tables for Join

For determining the new synapses on a neural network, we perform a kind of theta join between two large-scale information tables: vertex table V and class table C. The tables V and C are extracted from the neural network nn and training data D during feed-forwarding D through nn once. Theta (θ) join[27] indicates a binary operator between given two tables that returns a set of tuple pairs satisfying a user-defined condition θ. Synaptic join is named after theta join for creating new synapses. We note that synaptic join is for tweaking the behavior of the neural network for new input (*i.e.*, test) data although V and C are built based on the training data.

We explain V and C by using an example of a simple neural network in Fig. 3.2a. The table V is extracted from the neurons of the hidden layers of nn and the training data D. Here, we can extract only interesting neurons rather than every neuron into the table V. For example, the table V in Fig. 3.2c is built using three neurons in the first hidden layer in Fig. 3.2a. The table C is extracted from the neurons of the output layer of nn and the training data D. For example, since there are two output neurons in Fig. 3.2c, the table C has two rows. The numbers of columns of both V and C are the same as $|D|$, *i.e.*, the number of training data objects.

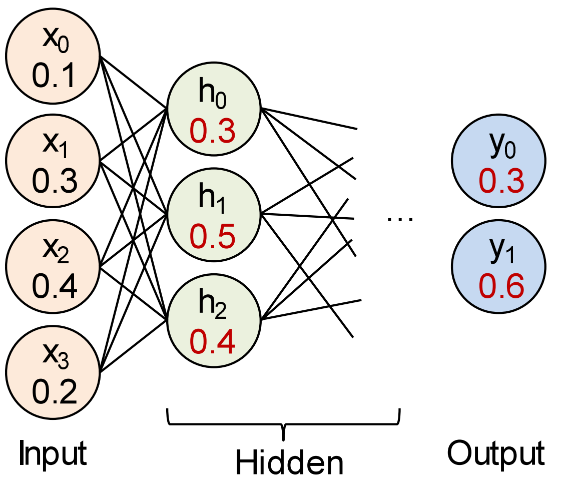
Synaptic join checks all the pairs in $V \times C$, where each tuple of V and C has $|D|$ dimensions. We will use the term "tuple" interchangeably with the term "node" (or "neuron") in our dissertation. We let a tuple of V be $V[i]$ (*i.e.*, $V[i] \in V$) and a tuple of C be $C[j]$ (*i.e.*, $C[j] \in C$). Then, we assign the value of node $V[i]$ for an input training data $D[k] \in D$ to the k-th column of the tuple $V[i]$, *i.e.*, $V[i][k]$. Likewise, we assign the value of node $C[j]$ for the data $D[k] \in D$ to the k-th column of the tuple $C[j]$, *i.e.*, $C[j][k]$. That is, the tables V and C record all the status of hidden and output neurons of the neural network during each data object in D passes through the neural network.

For example, we assume $|D| = 4$ in Fig. 3.2c. For the first data object $D[0] = \{0.1, 0.3, 0.4, 0.2\}$, we assume that the values of the hidden neurons $\{h_0, h_1, h_2\}$ are {0.3, 0.5, 0.4} and the values of the output neurons $\{y_0, y_1\}$ are {0.3, 0.6}. Then, the first column value of the first tuple of V, *i.e.*, $V[0][0]$ becomes 0.3, and the first column value of the second tuple of C, *i.e.*, $C[1][0]$ becomes 0.6.

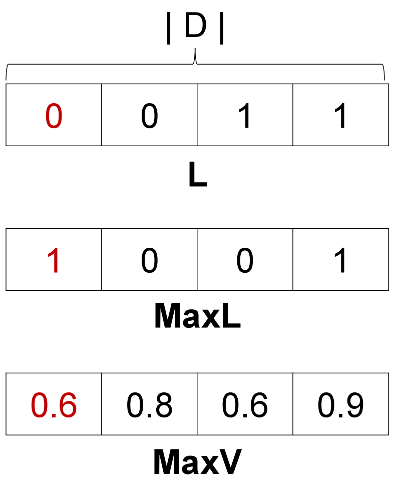
3.2.2 Algorithm θ

The purpose of synaptic join is to find the best pairs in $V \times C$, *i.e.*, the best synapses that can strengthen the target classes as much as possible without degrading the overall accuracy. In order to find the best synapses among a lot of possible candidate synapses, we need to evaluate the performance of a synapse $(V[i], C[j])$. We present the algorithm θ to evaluate the performance of a synapse $(V[i], C[j])$ for a single target class in Algorithm 1.

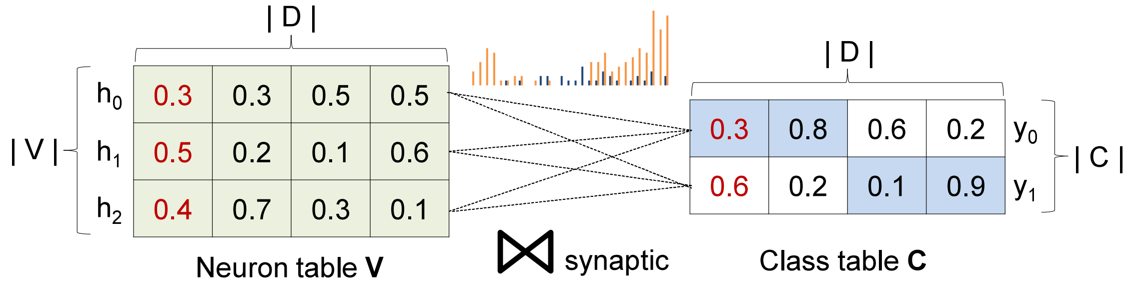
For the evaluation, we consider a set of auxiliary data structures L, $MaxL$, and $MaxV$, in addition to the V and C tables. L means the array of the ground truth for D, $MaxL$



(a) A simple neural network with propagating the first training data



(b) Auxiliary data structures



(c) Two tables V and C for four training data

Figure 3.2: An example of synaptic join ($|D| = 4$).

means the array of class labels when feed-forwarding D, and $MaxV$ means the array of the maximum values of the output neurons. All these arrays have a length of $|D|$. Fig. 3.2b shows examples of L, $MaxL$, and $MaxV$. In Fig. 3.2b, $L[0] = 0$ indicates that the class label of the first data object $D[0]$ is 0. $MaxL[0] = 1$ means that the resulting label of feed-forwarding $D[0]$ is 1, where the maximum value of the output neurons for $D[0]$, *i.e.*, $MaxV[0] = 0.6$.

For simplicity, we consider a single synapse that can make a target class $C[j]$ as correct as possible, while ensuring that the off-target classes are never incorrect, in terms of data set D. Next, the algorithm θ is used to evaluate the performance of a synapse based on the *distribution of all the possible proper weights* for the synapse.

The algorithm first finds the set of data objects D_w that correspond to class $C[j]$, but are incorrectly classified as belonging to the other classes (Lines 1-4). Similarly, the algorithm finds the set of data objects D_c that belonging to the off-target classes and are correctly classified as belonging to the corresponding class (Lines 5-8). If the $weight$ of synapse $(V[i], C[j])$ satisfies the condition of Eq. 3.1, the synapse will make the data object $k \in D_w$ correct, since the value of neuron $V[i]$ for k multiplied by the $weight$ makes the output value of the class $C[j]$ for k exceed the existing maximum output value $MaxV[k]$. We assign the infimum of the absolute values of the $weight$ satisfying Eq. 3.1 for k to E_w (Lines 9-11).

$$weight \cdot V[i][k] + C[j][k] > MaxV[k]. \tag{3.1}$$

Similarly, if the $weight$ of synapse $(V[i], C[j])$ satisfies the condition of Eq. 3.2, the synapse will make the data object $k \in D_c$ retain its correctness, since the value of neuron $V[i]$ for k multiplied by the $weight$ ensures that the output value of the class $C[j]$ for k does not exceed the existing maximum output value $MaxV[k]$. We assign the supremum of the absolute values of the $weight$ satisfying Eq. 3.2 for k to E_c (Lines 12-14).

$$weight \cdot V[i][k] + C[j][k] < MaxV[k]. \tag{3.2}$$

If the synapse has a weight satisfying both Eq. 3.1 and Eq. 3.2, it could correct some misclassifications in D_w while retaining correct classifications in D_c. In order to maximize the number of corrections, we try to find the absolute minimum value in E_c, which we define as the $threshold$. Any absolute value in E_w smaller than the $threshold$ satisfies both Eq. 3.1 for $k \in D_w$ and Eq. 3.2 for $k \in D_c$.

We can calculate the $threshold$ corresponding to the maximum range (Line 15) and

Algorithm 1 θ of Synaptic Join

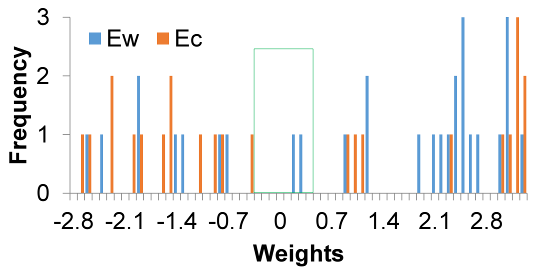
```
Input: $V[i]$; /* a tuple of vertex table */
       $C[j]$; /* a tuple of class table */
Variable: $L$; /* class label array (ground truth) */
       $MaxL$; /* class label array (result using a DNN) */
       $MaxV$; /* output value array for $MaxL$ */
Output: $\langle threshold, count \rangle$; /* the best edge weight and the number of corrections */
    /*Find a set of data objects wrongly classified for $C[j]$ class*/
 1: $D_w \leftarrow \emptyset$;
 2: for $0 \le k \le |D| - 1$ do
 3:     if $L[k] = C[j]$ and $L[k] \ne MaxL[k]$ then
 4:         $D_w \leftarrow D_w \cup \{k\}$;

    /*Find a set of data objects correctly classified for non-$C[j]$ class*/
 5: $D_c \leftarrow \emptyset$
 6: for $0 \le k \le |D| - 1$ do
 7:     if $L[k] \ne C[j]$ and $L[k] = MaxL[k]$ then
 8:         $D_c \leftarrow D_c \cup \{k\}$;

    /*Find a set of edge weights that can correct $D_w$*/
 9: $E_w \leftarrow \emptyset$;
10: for each $k \in D_w$ then
11:     $E_w \leftarrow E_w \cup \{\frac{MaxV[k] - C[j][k]}{V[i][k] + \varepsilon_1}\}$;
    /*$\varepsilon_1$ for preventing division by zero*/

    /*Find a set of edge weights that retain $D_c$ correct*/
12: $E_c \leftarrow \emptyset$;
13: for each $k \in D_c$ then
14:     $E_c \leftarrow E_c \cup \{\frac{MaxV[k] - C[j][k]}{V[i][k] + \varepsilon_1}\}$;

    /*Find the maximum edge weight that is harmless to off-target classes*/
15: $threshold \leftarrow min(abs(E_c)) - \varepsilon_2$;
16: $count \leftarrow 0$;
17: for each $w \in E_w$ then
18:     if $abs(w) \le threshold$ then
19:         $count \leftarrow count + 1$;
20: Return $\langle threshold, count \rangle$;
```

calculate the number of weight values within this range, which is considered as the $count$ (Lines 16-19). We subtract an extremely small value ε_2 from the $threshold$ for satisfying the condition in Eq. 3.2. The $threshold$ value indicates the proper weight of synapse $(V[i], C[j])$, and the $count$ value indicates the number of corrections for the training data when adding the synapse to the neural network.

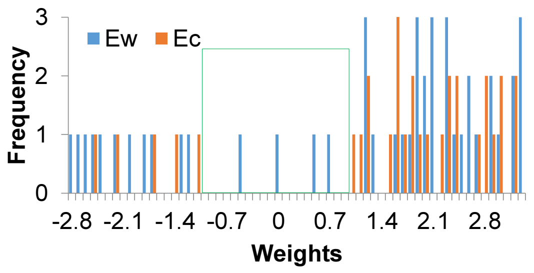
We present an example of Algorithm 1 using Fig. 3.2. We consider the synapse between $(V[0], C[0])$, where the target class is 0, and the off-target class is 1. $D_w = \{0\}$ since the set of data objects wrongly classified for the target class is $\{D[0]\}$. Likewise, $D_c = \{3\}$ since the set of data objects correctly classified for the off-target class is $\{D[3]\}$. For simplicity, we let $\varepsilon_1 = 0.0$ and $\varepsilon_2 = 0.01$. Then, $E_w = \{\frac{0.6-0.3}{0.3} = 1.0\}$ for $D_w = \{0\}$, $E_c = \{\frac{0.9-0.2}{0.5} = 1.4\}$ for $D_c = \{3\}$, and the $threshold$ becomes $1.4 - 0.01 = 1.39$. Under the value $threshold$, the number of corrections, $count$, becomes 1. Thus, Algorithm 1 returns $\langle 1.39, 1\rangle$ for the candidate synapse $(V[0], C[0])$.

Fig. 3.3 shows the distributions of E_w and E_c for four different synapses picked from the ResNet56 model for CIFAR-100 used in the experiments. The synapse in Fig. 3.3a has a $count = 2$, *i.e.*, the sum of frequencies of E_w (*i.e.*, blue bars) within the range of the $threshold$ (*i.e.*, green box), in its distribution. Likewise, the synapses in Fig. 3.3b, Fig. 3.3c, and Fig. 3.3d have $count = 4$, $count = 8$, and $count = 16$, respectively. Fig. 3.3c and Fig. 3.3d have a wider range of the $threshold$ and a higher $count$ than those shown in Fig. 3.3a and Fig. 3.3b. That means the former synapses are more target specific and so likely to be more effective than the latter synapses for improving the corresponding target class. In a distribution, negative weight values mean that the status of a neuron is also negative for the corresponding data objects.

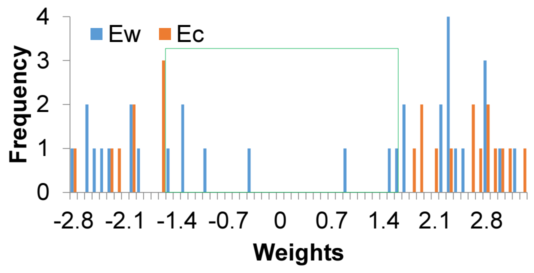
Fig. 3.4 shows the distribution of the $count$ for all synapses (a total of 8,192) in a hidden layer layers of the ResNet56 model for CIFAR-100 used in the experiments. As shown



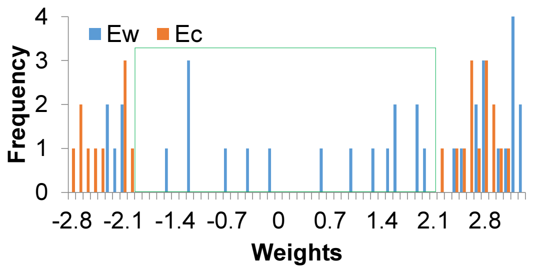
(a) $count = 2$, $threshold = 0.481$



(b) $count = 4$, $threshold = 0.92$



(c) $count = 8$, $threshold = 1.60$



(d) $count = 16$, $threshold = 2.09$

Figure 3.3: Distributions for E_w and E_c for four synapses picked from ResNet56 for CIFAR-100.

in the figure, the distribution follows a power law interestingly. Most of the synapses just have $count = 1$ or $count = 0$, *i.e.*, are non-target specific. The synapse of the highest rank has a $count = 16$, and the synapses having a $count \geq 8$ make up only 0.46% of all the synapses. We found this kind of power law distribution in all the data and models used in experiments. Table 3.2 shows the proportions of the high rank synapses having a $count \geq \frac{1}{2}max(count)$, where $max(count)$ means $count$ of the highest rank synapses. The proportions are between 0.46% and 6.19%. We consider the neurons having a $count \geq \frac{1}{2}max(count)$ as active neurons. Then, the last column of Table 3.2 shows the number of active neurons.

3.2.3 Synaptic Join Method

Synaptic join method performs Algorithm 1 for all the possible tuple pairs between V and C, and then, chooses the best synapses. So, the time complexity of synaptic join becomes $\mathcal{O}(|V||C||D|)$, where $|V|$ is the number of neurons of the base model, $|C|$ is the number of

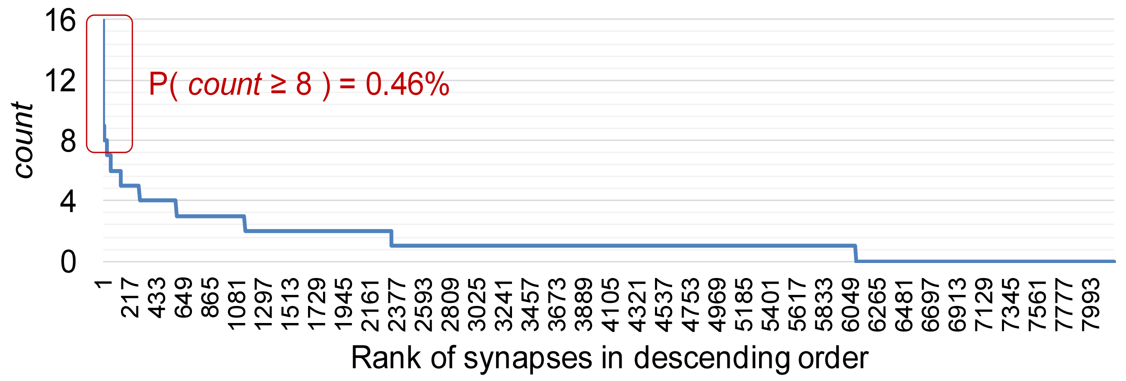


Figure 3.4: Distribution of the *count* of all the synapses in a hidden layer of the ResNet56 for CIFAR-100.

Table 3.2: Proportions of the high rank synapses.

Data	Model	# params (x1,000)	$P\left(\geq \frac{1}{2}max(count)\right)$	# active neurons
CIFAR-100	ResNet18	683	0.68%	8,057
CIFAR-100	ResNet56	869	0.46%	9,968
SVHN	3-layer CNN	89	6.19%	2,789
SVHN	DenseNet-40	1,019	0.85%	18,487
SUN-397	DenseNet-121	7,360	2.91%	346,626
ImageNet	GoogLeNet	6,990	0.67%	20,949

classes, and $|D|$ is the number of training data objects. We show the time cost in more detail in Section 3.3.9. A single synapse $(V[i], C[j])$ can improve the accuracy of a single target class $C[j]$ by $count$ corrections for the training data by amplifying the signal of $V[i]$ by $threshold$ times. If the training accuracy of the neural network nn is not 100%, we use the $count$ as a criterion for the performance of the synapses. We expect that the synapse having the largest the $count$ is the most effective for the test data as well. If the training accuracy of the neural network is 100%, then D_w and E_w are empty, and so, the $count$ becomes zero for all the synapses. In this case, we choose the synapse having the maximum $threshold$ under the assumption that the distribution for test data would be similar to that for the training data.

The synaptic join method adds the top-n synapses instead of a single best synapse for a target class to avoid overfitting. Here, by default, we divide the weight value of each synapse by n since a total of n synapses are connected to the target class. This is a heuristic approach for reducing the side effect that the overall accuracy is degraded. Adding n synapses together will have more chance to activate the target classes for test data. We note that the method would add a total of $n \times c$ synapses if the number of target classes is c.

For synaptic join, the value of n tends to affect the target and overall accuracy slightly. Fig. 3.5 shows the tendency of the target and overall accuracies while varying n from 1 to 150. Increasing n tends to continuously degrade the target accuracy slightly. On the contrary, the target accuracy is converged at around n=120, in particular when $scale$=40.

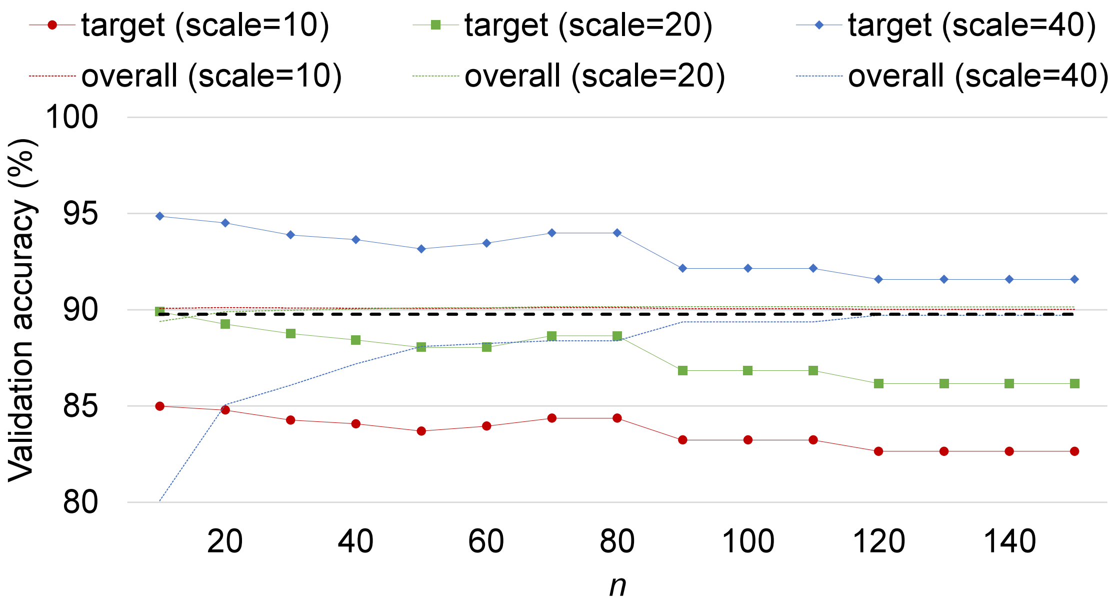


Figure 3.5: Target and overall accuracies while varying n (3-layer CNN for SVHN).

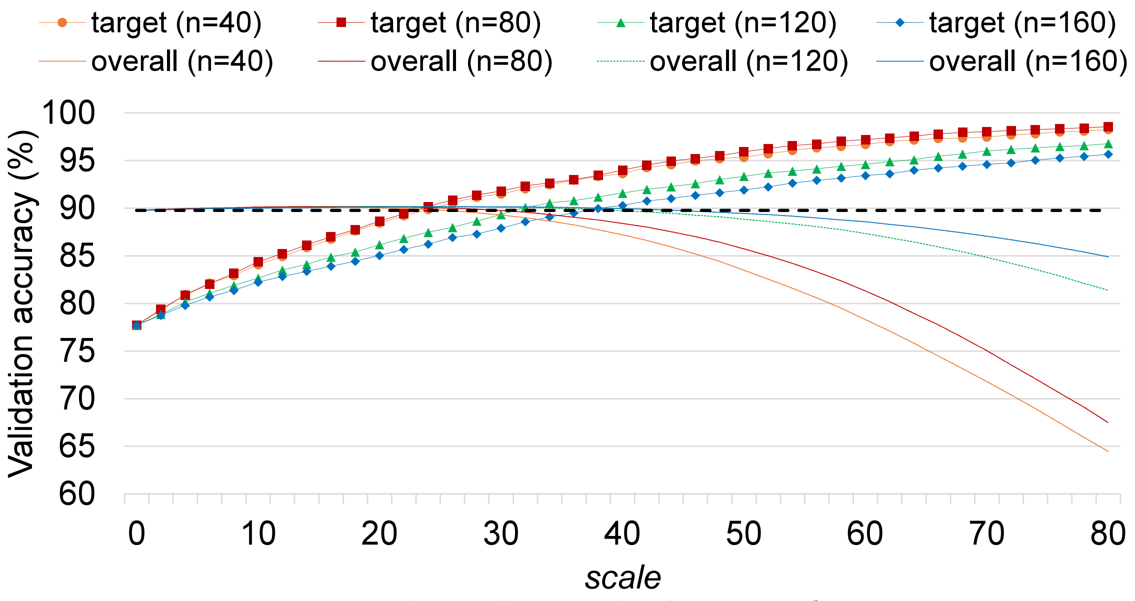
If we want to improve the target class largely and intentionally, we can achieve it by multiplying the weight of top-n synapses by a factor of $scale$. Fig. 3.6a shows the tendency of the target and overall training accuracies for $0 \leq scale \leq 80$. As the $scale$ increases, the target accuracy tends to increase, but the overall accuracy tends to increase slightly for a while and then decrease. Fig. 3.6b shows the tendency of the overall training accuracy for $scale \leq 60$ whose values are around the original overall accuracy. The reason why the overall accuracy increases for relatively small $scale$ values is that the target accuracy increases while the overall accuracy is maintained.

A heuristic approach for determining n and $scale$ is finding the n value where the target accuracy is converged and then finding the $scale$ value that can improve the target accuracy as much as possible, but does not degrade the overall accuracy, for the training data. For example, n=120 in Fig. 3.5 and $scale$=40 in Fig. 3.6b satisfy such conditions. The optimal n and $scale$ values can vary depending on the base model and the target class. Thus, we evaluate the target and overall accuracy for some candidate pairs of values using a validation set in an exhaustive manner and pick the best one. For example, we evaluate $[40, 80, 120]$ for n and $[20, 30, 40]$ for $scale$ and pick the one. Let α be the number of candidate n values, and β be the number of candidate $scale$ values. Then, there are a total of $\alpha \times \beta$ configurations. Since the size of n synapses is quite small, we can load the base model with α configurations together and evaluate their target and overall accuracies simultaneously while feed-forwarding the validation set once.

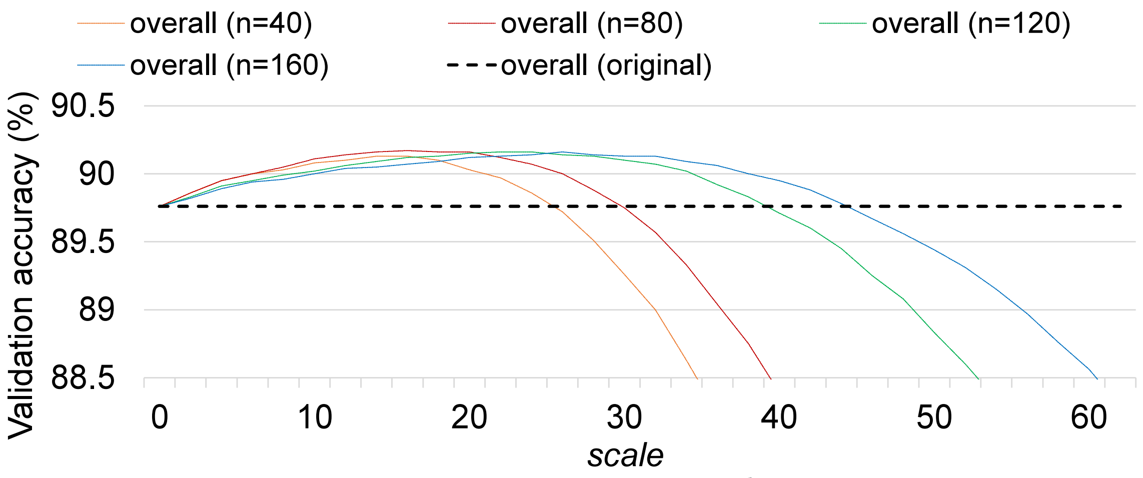
3.2.4 Synaptic Retraining Method

The synaptic join method is a kind of static method that determines a set of synapses and their weights based on the analysis of training data's passing through a neural network. Then, there may be a question: how the weights of the synapses newly added (Δnn) are changed if we retrain only those synapses with fixing all the weights of the original neural network (nn). We call this retraining method as *synaptic retraining*. It tries to maximize the overall accuracy as a normal training method does. As a result, the target accuracy tends to decreases a little bit, while the overall accuracy tends to increase a tiny bit, compared to the result of the synaptic join method.

In fact, the above tendency is an interesting effect of the synaptic join method. A model like GoogLeNet [83] is already highly optimized, and so, it is difficult to further improve



(a) Target and overall accuracies for $0 \leq scale \leq 80$



(b) Overall accuracies for $0 \leq scale \leq 60$

Figure 3.6: Target and overall accuracies while varying the $scale$ (3-layer CNN for SVHN).

its accuracy. The simple retraining method tends to rather degrade the overall accuracy as explained in Section 2. However, the synaptic retraining method can further improve the overall accuracy of the model by retraining only the weights of the new synapses for the target classes. Here, the number of iterations required for synaptic retraining is much smaller than that for simple retraining, and at the same time, the degree of fluctuation in synaptic retraining is much smaller than that in simple retraining (*e.g.*, Fig. 3.1), due to a small number of synapses to be trained.

3.3 Experimental evaluation

3.3.1 Environments

We use various neural networks for five datasets, SVHN (cropped digits) [62], CIFAR-10, CIFAR-100 [47], ImageNet (ILSVRC12, classification) [74], and SUN-397 [92]. SVHN is a real-world color image dataset of the digits obtained from house numbers in Google Street View images, which consists of 10 classes of digits and contains 73,257 training samples and 26,032 testing samples of 32×32 pixels. CIFAR-10 [47] is a collection of color images of 32×32 pixels that contains 10 classes, 5,000 training samples per class, and 1,000 testing samples per class. CIFAR-100 is similar to CIFAR-10, but contains 100 finer classes, 500 training samples per class, and 100 testing samples per class. ImageNet consists of 1,000 categorical real-world images. The numbers of training and testing samples are about 1.2 million and 50,000, respectively. SUN-397 is a skewed and long-tailed dataset of color images for scene understanding. The dataset consists of 108,754 images of 397 classes, and each class consists of 100 to 2,361 images. We randomly choose 50 images as test data for each class and use the remaining 50 to 2,311 images as training data. For both ImageNet and SUN-397, we resized the images to 256×256 pixels and properly cropped them to satisfy the requirement on the size of the input for each model.

In all the experiments about our method, we split training samples into training and validation samples. For SVHN, 2,000 samples per class are used for validation, and the remaining samples for training. For CIFAR-10 and CIFAR-100, 2,500 and 250 samples per class are used for validation, respectively, and the remaining samples for training. For SUN-397 and ImageNet, 25 and 500 samples per class are used for validation, respectively, and the remaining samples for training. We performed synaptic join using the training samples and determined hyperparameters (*i.e.*, n, $scale$) using the validation samples. Then, we measured the accuracy using the test samples.

We explain the neural network models used in the experiments. For SVHN, we have modified the 3-layer CNN model in [1] by adding a batch normalization layer to each convolution layer. After training the model for 115 k iterations with a start learning rate of 0.001 and reduced the learning rate by 1/10 at 75 k iterations, the model achieved 88.99% overall test accuracy. For CIFAR-10, basically, we used a 3-layer CNN model of the structure in [1], where each convolution layer is followed by pooling and ReLU layers. After 50 k iterations of training with a learning rate of 0.001, the model achieved 77.9% overall test accuracy. For CIFAR-100, we have modified ResNet-18 [2] and ResNet-56 [39] models. The ResNet-18 model has five residual blocks, and each block consists of three or four convolution layers. After 40 k iterations of training with an initial learning rate of 0.1, which was reduced by a factor of 10 for every 5 k iterations, the model achieved 54.8% overall accuracy. The ResNet-56 model has three residual groups, and each group has eight residual blocks, and each block consists of two or three convolution layers. After 64 k iterations of training with an initial learning rate of 0.1, which is reduced by a factor of 10 at 32 k and 48 k iterations, the model achieved 66.1% overall test accuracy. For ImageNet, we have used the GoogLeNet [83] binary model which has 68.98% overall accuracy (top-1). For more models used in a specific experiment, we explain them in the corresponding sections.

We conducted all the experiments on a workstation equipped with two Intel Xeon 2.2 GHz CPUs of ten cores (a total of 20 cores), 384 GB main memory, eight NVIDIA GTX 1080Ti GPUs of 11 GB memory, and 10 TB HDD. We trained and ran the models using CAFFE [45].

3.3.2 Comparison with Retraining and Relevant Methods

We first compared our method with the simple retraining method and its variants. We used the SVHN dataset and the 3-layer CNN model (overall accuracy: 88.99%). We retrained this base model using a learning rate of 0.0001 in the retraining methods. We consider class#9 as the target class, which had the lowest accuracy of 76.26% among all classes. We denote the simple retraining method with $\times 2.0$ loss penalty as *Retraining-1* and its variants as from *Retraining-2* to *Retraining-5*.

We also evaluate a method related to our method, in particular, the calibration method [17, 49]. The calibration method was originally proposed in the studies on zero-shot learning [17, 49]. It uses a hyperparameter μ $(0.0 \leq \mu \leq 1.0)$ for the target class and $(1-\mu)$ for the off-target classes to calibrate the prediction of a given neural network. The values μ and $(1-\mu)$ mean prior probabilities that a sample belongs to the target and off-target classes, respectively. The calibration method simply multiplies the softmax outputs of each target class and that of each off-target class by the prior probabilities μ and $(1-\mu)$, respectively. In general, as μ increases, the target accuracy increases, while the off-target accuracy decreases.

- *Retraining-1*: Retraining with $10\,\mathrm{k}$ iterations with a $\times 2.0$ loss penalty on the target class
- *Retraining-2*: Retraining with $10\,\mathrm{k}$ iterations with a $\times 0.9$ loss penalty on the off-target classes (*i.e.*, classes#0-8)
- *Retraining-3*: Training with $125\,\mathrm{k}$ iterations from the beginning with a $\times 2.0$ loss penalty on the target class
- *Retraining-4*: Retraining with $10\,\mathrm{k}$ iterations with a $\times 2.0$ loss penalty on the target class and using dropout (rate=0.5)
- *Retraining-5*: Retraining with $10\,\mathrm{k}$ iterations with $\times 2$ oversampling for the target class
- *Calibration*: Calibrating the softmax output of the original model by varying μ from 0.0 to 1.0
- *Synaptic join*: Adding new $n = 120$ synapses to the original model by varying the $scale$

from 5 to 45

- *Synaptic retraining-1*: Retraining the model modified by synaptic join for 10 k iterations
- *Synaptic retraining-2*: Retraining the model modified by synaptic join for 10 k iterations with a $\times 2.0$ loss penalty on the target class

Retraining-1 in Fig. 3.7a shows the box plot of the individual accuracy of each class from 116 K to 125 K retraining iterations. The accuracy of class#7 and class#9 fluctuates severely, while the overall accuracy is relatively stable. *Retraining-2* shows a similar tendency to *Retraining-1*, but a much worse target accuracy (we omit the figure). *Retraining-3* in Fig. 3.7b shows the best average target accuracy among the five retraining methods. *Retraining-4* shows a worse performance than *Retraining-1*, that is, dropout has no special effects on tweaking the model (we omit the figure). *Retraining-5* in Fig. 3.7c shows a slightly better overall accuracy than *Retaining-1* and *Retaining-3*, but a worse target accuracy than them. That is, using $\times 2$ loss penalty without dropout (*i.e.*, *Retaining-1* and *Retaining-3*) is more effective for target accuracy than $\times 2$ oversampling (*Retraining-5*). The detailed results of *Retraining* methods (including the omitted figures) are in Appendix A.

Table 3.3 shows the target and overall accuracy of the retraining methods. The first five rows show the results of the five retraining methods. In those methods, it is difficult to stop retraining exactly at the point where the target accuracy is improved, and at the same time, each non-target class accuracy is degraded only a little bit, not too much. It is also difficult to control the degree of improvement in the target accuracy. Where to stop retraining becomes more difficult if the target is not a single but multiple classes, since each per-class accuracy fluctuates almost independently. Furthermore, retraining itself is time consuming, and the changes to the model are permanent, *i.e.*, irreversible.

We present *Calibration* in Fig. 3.7d and *Synaptic join* in Fig. 3.7e in bar plots rather

Table 3.3: Results of the retraining methods (3-layer CNN for SVHN).

Method	Target accuracy	Overall accuracy
Retraining-1	91.44	89.50
Retraining-2	85.90	89.70
Retraining-3	91.53	89.51
Retraining-4	89.73	89.18
Retraining-5	90.94	89.68
Calibration	97.19	87.36
Synaptic join	91.97	89.02
Synaptic retraining-1	87.90	90.68
Synaptic retraining-2	91.74	90.59

than box plots, since their per class accuracies do not fluctuate randomly, but are controlled by parameters such as μ and $scale$. *Calibration* in Fig. 3.7d shows that the target accuracy increases from 0% to 100%, and each off-target accuracy decreases, as μ varies from 0.0 to 1.0. The trend of the overall accuracy is a bit more complex. It largely decreases compared to the original accuracy when $\mu = 0.0$ due to wrong answers for all the target class images. Then, it tends to increase due to the increased target accuracy and then decrease due to the decreased off-target accuracy. In general, we need to determine μ using the validation data. We can determine it using a simple method that finds the point where the target accuracy is maximized, and at the same time, the training overall accuracy is not degraded compared to the original overall accuracy, as we determine $scale$ in Section 3.2.3. Accordingly, we set μ=0.9. This method tends to achieve a high target accuracy more easily than the other methods, but its overall accuracy (*i.e.*, 87.36%) tends to be significantly degraded from the original one (*i.e.*, 88.99%).

Synaptic join in Fig. 3.7e shows the highest target accuracy except *Calibration* while maintaining overall accuracy. Compared to *Calibration*, *Synaptic join* achieves a better overall accuracy, and at the same time, its off-target accuracy is more predictable and controllable by the $scale$. For example, in Fig. 3.7d, the accuracies of class#5, class#7, and class#8 are suddenly decreased as μ increases, while the accuracy of some other classes including class#1 and class#4 is almost not changed. However, *Synaptic join* has no such a sudden decrease or constancy in terms of the off-target accuracy. As the $scale$ increases, the target accuracy increases gradually while each off-target class accuracy decreases a little bit.

In *Synaptic retraining-1* (we omit the figure) and *Synaptic retraining-2* (in Fig. 3.7f), all per-class accuracy converges quickly (only for 2-4 k iterations) due to a very small number of additional parameters ($n = 120$) and so the heights of box plots are very small. Compared to *Synaptic join*, *Synaptic retraining-1* improves the overall accuracy by 1.66% while sacrificing the target accuracy by 4.07%. In fact, *Synaptic retraining-1* is equal to normal training of a small number of new synapses between the target specific hidden neurons and the target output neurons. Since normal training tries to maximize the overall accuracy, and the number of new synapses is small, *Synaptic retraining-1* tends to increase the overall accuracy. However, it tends to decrease the target accuracy a little bit since it changes the weights of the synapses which are already almost optimized by *Synaptic join* to achieve the best target accuracy. *Synaptic retraining-2* is a synaptic retraining method to use a loss penalty like the retraining methods. It improves the target accuracy by 3.84% while sacrificing the overall accuracy by 0.09%, compared to *Synaptic retraining-1*.

In summary, for 3-layer CNN for SVHN, *Retraining-3* shows the best overall performance, except *Synaptic retraining-1* and *Synaptic retraining-2*. Although its target accuracy is slightly worse than *Synaptic join*, its overall accuracy is surely better than *Synaptic join*. However, *Synaptic retraining-2* outperforms *Retraining-3* in terms of both target and overall accuracy, which is done by retraining only a very small number of additional parameters chosen by synaptic join with a loss penalty.

3.3.3 Quantitative Analysis

In this section, we perform some quantitative analysis to measure the performance of the methods shown in Section 3.3.2. For the analysis, we propose Eq. 3.3, where o_i and t_i are the accuracies of the i-th class of an original model and the corresponding tweaked model, respectively. The intuition of Eq. 3.3 is that a method that degrades each per-class accuracy of a model evenly and as little as possible would be a better method for tweaking the model. To take account of only the case $(t_i - o_i) < 0$ as a penalty, we introduce an indicator function $\mathbb{1}_i$ that returns 1 if $t_k < o_k$ and 0 otherwise. A larger E value indicates a larger or more uneven degradation of the per-class accuracy in the tweaked model. Although Eq. 3.3 may not be the best measure, it would be able to quantify how good a tweaking method is in some aspects.

$$E = \sqrt{\frac{1}{|C|}\sum_{i=1}^{|C|}(t_i - o_i)^2 \mathbb{1}_i} \tag{3.3}$$

Table 3.4 shows the mean and standard deviation (SD) of E for the nine methods. Since the retraining methods and synaptic retraining methods have different E values at each 1 k iteration, they have both mean (E) and SD (E). In contrast, *Calibration* and *Synaptic join* have a single E value due to no retraining. *Synaptic retraining-1 and 2* have a very small SD (E) value due to the quick convergence and no fluctuation of per-class accuracy after the convergence. In Table 3.4, the relatively high mean (E) and SD (E) values of the retraining methods are due to the large and independent fluctuations of the per-class accuracy during the retraining. The high E value for *Calibration* is due to the sudden drop of the per-class accuracy in some classes. *Synaptic retraining-1 and 2* have the smallest E value since they improve the per-class accuracy of most classes and so achieve the best overall accuracy.

Table 3.4: Quantitative analysis of the methods.

Method	**mean** (E)	**SD** (E)
Retraining-1	3.01	1.90
Retraining-2	2.34	1.72
Retraining-3	3.86	1.75
Retraining-4	3.33	1.30
Retraining-5	2.47	1.25
Calibration	6.22	N/A
Synaptic Join	1.64	N/A
Synaptic Retraining-1	**0.62**	**0.10**
Synaptic Retraining-2	0.68	0.11

3.3.4 Evaluation of Different Models for the Same Data

In the experiments using 3-layer CNN for SVHN, *Retraining-1*, *Retraining-3*, and *Retraining-5* generally show the best performance among the existing methods. In this section, we compare the three methods with our methods for different network models on the same dataset (*i.e.*, SVHN). We use two additional network models, ResNet-20 [39] and DenseNet-40 [44, 104]. ResNet-20 has 272,464 weight parameters and 28,672 hidden neurons, and DenseNet-40 has 1,019,712 parameters and 270,336 neurons. ResNet-20 has achieved 95.0% after 64 k iterations, and DenseNet-40 has achieved 96.0% after 60 k iterations.

Table 3.5 shows the comparison results for ResNet-20 and DenseNet-40. We select the class of the lowest accuracy as the target class, which is class#7 for both ResNet-20 and DenseNet-40. For ResNet-20, *Synaptic join* clearly shows the best target accuracy among all methods compared. In terms of the overall accuracy, only *Retraining-5* is marginally better than others. For DenseNet-40, *Synaptic join* still shows the best target accuracy, but in terms of overall accuracy, *Synaptic retraining-1* shows the best accuracy. In terms of the overall accuracy, all methods except our method degrade the overall accuracy.

The reason why *Synaptic join* clearly shows a better performance than the existing

Table 3.5: Comparison of major retraining methods and synaptic join method (ResNet-20 and DenseNet-40 for SVHN).

Model [target#]	Methods	Avg. target	Avg. overall
ResNet-20 [7]	Original	91.9	95.0
	Retraining-1	92.1	95.0
	Retraining-3	93.0	95.0
	Retraining-5	92.1	**95.1**
	Synaptic join	**95.1**	95.0
	Synaptic retraining-1	93.0	95.0
	Synaptic retraining-2	93.9	95.0
DenseNet-40 [7]	Original	95.0	96.0
	Retraining-1	95.8	94.4
	Retraining-3	96.3	94.9
	Retraining-5	96.6	95.0
	Synaptic join	**97.1**	96.0
	Synaptic retraining-1	96.6	**96.3**
	Synaptic retraining-2	97.0	96.1

retraining methods in Table 3.5 is that both ResNet-20 and DenseNet-40 are larger than 3-layer CNN in terms of parameters and so can exploit much more number of active neurons for tweaking the networks. Fig. 3.8 shows the correlation between the number of parameters of the network models in Table 3.2 and the number of their active neurons. Since SV_RN (ResNet-20 for SVHN) has zero $count$ due to 100% training accuracy, we mark it as a vertical line in Fig. 3.8. In this case, we use $threshold$ instead of $count$ as explained in Section 3.2. In general, it is likely that *Synaptic join* can pick better (*i.e.*, target-specific) n hidden neurons and connect them to the target classes, as the number of active (or high $threshold$) neurons increase.

3.3.5 Distribution of Synapses

Fig. 3.9 shows the distribution of the active neurons that are selected by synaptic join across the layers. Each layer shows the number of active neurons and the percentage of those active neurons compared with the total neurons in that layer. For GoogLeNet, we set the worst ten classes to the target class with n=30 and show the distribution of the 300 active neurons selected. For 3-layer CNN, we set the worst class to the target class with n=120 and show the distribution of the 120 active neurons selected.

The result shows that both models tend to have more number of selected active neurons at lower-level layers. It makes sense because there are more number of neurons at lower-level layers, and at the same time, a lower-level layer tends to capture the relatively simple features specific to each class. However, in terms of percentage, 3-layer CNN has the largest percentage in the highest (*i.e.*, deepest)-level layer. GoogLeNet has 50,176 and 1,000 neurons at the highest-level and output layers, respectively (*i.e.*, about 50:1 ratio), while 3-layer CNN has 4,096 and 10 neurons (*i.e.*, about 400:1 ratio). Thus, it is likely that 3-layer CNN has more neurons specific to each class at the highest-level layer.

3.3.6 Characteristics of Synaptic Join

As explained in Section 3.3, we basically use the $count$ as a criterion for the performance of synapses, but should use the $threshold$ as the criterion if the $count$ is unavailable. Here, we compare the effectiveness of both the criteria using the ResNet18 model for CIFAR-100. The original overall accuracy of the model is 54.8%. Table 3.6 presents the test accuracy of synaptic join for three different target classes, *i.e.*, class#22 (chimpanzee), class#26 (crab), and class#51 (mushroom), when using the top-20 synapses sorted by the $count$ and the $threshold$. Here, we select the classes of the median accuracy rather than those of the lowest accuracy. For all the three classes, using the $count$ achieves better target accuracy than using the $threshold$. Both the criteria achieve the same overall accuracy.

Table 3.7 shows the results when we apply synaptic join to the three different target classes of the lowest (class#9), median (class#6), and highest (class#0) accuracies. The effect

Table 3.6: Results of synaptic join using the two different criteria, *count* and *threshold* (ResNet18 for CIFAR-100).

Target	Accuracy	Original	By *count*	By *threshold*
class#22	target	52.0	**59.0**	55.0
(chimpanzee)	overall	54.8	54.7	54.7
class#26	target	51.0	**56.0**	**56.0**
(crab)	overall	54.8	54.8	54.8
class#51	target	51.0	**56.0**	55.0
(mushroom)	overall	54.8	54.8	54.8

of synaptic join gets larger as the target class has lower accuracy. The target accuracy of class#9 (lowest) increases by 15.71%, while that of class#0 (highest) only by 0.17%.

Table 3.7: Results of synaptic join for three target classes of the lowest, median, and highest accuracies (3-layer CNN for SVHN).

Target class#	9 (lowest)	6 (median)	0 (highest)
Target accuracy	91.97 (+15.71)	90.34 (+2.92)	96.23 (+0.17)
Overall accuracy	89.02 (+0.03)	89.01 (+0.02)	88.99 (+0.00)

Table 3.8 shows the result when we apply synaptic join to multiple target classes. We select a total of c classes of the lowest accuracy and apply synaptic join to them. We vary c from $c = 2$ to $c = 10$ (*i.e.*, all the classes). The target accuracy in Table 3.8 means the average accuracy of the c classes after synaptic join. As c increases, the effect of the improvement of the target accuracy decreases gradually from 6.53% ($c = 2$) to 0.38% ($c = 10$). The overall accuracy is improved and converges to 89.37% by increasing c. We note that the overall accuracy when $c = 1$ (*i.e.*, 89.03%) is lower than that when $c = 2$ (*i.e.*, 89.27%). Consequently, the overall accuracy tends to be improved by adding the synapses to more target classes and converged to a certain accuracy.

Table 3.8: Results of synaptic join for multiple target classes (3-layer CNN for SVHN, $n = 40$).

Accuracy	c=2	c=4	c=6	c=8	c=10
Target	83.5 (+6.53)	81.52 (+2.64)	84.03 (+1.70)	86.53 (+1.04)	89.37 (+0.38)
Overall	89.27 (+0.28)	89.33 (+0.34)	89.38 (+0.39)	89.37 (+0.38)	89.37 (+0.38)

3.3.7 Characteristics of Synaptic Retraining

In this section, we investigate the effect of the combination of synaptic join and synaptic retraining, in particular, *Synaptic retraining-1*. Synaptic join can choose a small number of target specific synapses. Synaptic retraining can train the weights of the synapses quickly through a small number of iterations. Here is a possible new optimization strategy that exploits both. The strategy is repeating a pair of synaptic join (*i.e.*, choosing synapses) and synaptic retraining (*i.e.*, training the synapses) in a progressive manner. The motivation of the strategy is further optimization of a neural network already trained by further training only the worst classes. In detail, the strategy chooses the worst classes as target classes, adds new synapses for them, and performs synaptic retraining for the synapses. The strategy repeats the above process by narrowing the scope of optimization (*i.e.*, the number of target classes).

We evaluate the proposed strategy using GoogLeNet for the ImageNet (ILSVRC12) dataset described in Section 3.3.1 (top-1 accuracy: 68.98%). We denote synaptic join for 100 target classes as J_{100} and synaptic retraining for 100 target classes as R_{100}. The number of synapses per class is $n = 30$, and the number of iterations for retraining is 20 k with a learning rate of 10^{-6}. Table 3.9 shows the results of the proposed strategy while varying the steps of optimization. We consider ten classes having the lowest accuracy as the final target classes. In the table, Expr#1 shows that a single synaptic join J_{10} for the target classes improves their average target accuracy by 8.6%. Here, the overall accuracy is not changed due to being offset by the degradation of some off-target accuracies. In Expr#2, we perform a pair of synaptic join and retraining for the 100 target classes (*i.e.*, J_{100}+R_{100}), another pair of synaptic join and retraining for the ten target classes (*i.e.*, J_{10}+R_{10}) and the final synaptic join for the ten target classes (*i.e.*, J_{10}). Expr#2 achieves a better result than Expr#1 in terms of both target and overall accuracy. In Expr#3, we add one more pair J_{50}+R_{50} in the middle of the steps, and so, the scope of the target classes is narrowed down more gradually. As a result, Expr#3 achieves a better result than that of Expr#2.

We note that the above strategy is not the main method we propose for tweaking neural networks. A generalization of the strategy for various networks and datasets would be beyond the scope of this dissertation. We, however, believe that the strategy gives some hints about a post-optimization (not tweaking) method of neural networks.

Table 3.9: Results of repeating synaptic join and retraining.

Expr#	Steps	Target	Overall
1	J_{10}	23.40 (+8.60)	68.98 (+0.00)
2	J_{100}+R_{100}+J_{10}+R_{10}+J_{10}	25.40 (+10.60)	68.99 (+0.01)
3	J_{100}+R_{100}+J_{50}+R_{50}+J_{10}+R_{10}+J_{10}	**26.20 (+11.40)**	**69.02 (+0.04)**

3.3.8 Synaptic Join for Imbalanced Data

In this section, we evaluate synaptic join for imbalanced data, in particular, the SUN-397 dataset [92], described in Section 3.3.1. We prepare the base model by fine-tuning a pre-trained DenseNet-121 model for ImageNet as explained in [90], which has an overall accuracy of 49.8%. The oversampling method addressed in [90] oversamples the training data of each class to 2,311 images.

Due to data imbalance, the base model demonstrates quite low performance for the data-poor tail classes. It shows the average accuracy of 31.1% for the top 50 tail classes and the one of 35.2% for the top 100 tail classes. Fig. 3.10 shows the results of the oversampling method and our *Synaptic join* for the top 50 tail classes (c=50). In the figure, the x-axis means 397 classes, where the leftmost one indicates the top 1 head class, and the rightmost one the top 1 tail class. The zero value on the y-axis means the original accuracy of the base model for each class. The blue and orange bars in the figure indicate accuracy gains. *Synaptic join* improves the tail classes in a more controllable manner compared to the oversampling method. Table 3.10 shows the accuracies while varying the number of tail classes. The accuracies of *Synaptic join* are better than those of the oversampling method for c=50 and c=100, while the former becomes smaller than the latter for c=150. That is, *Synaptic join* methods tend to be effective for a relatively small number of tail classes, while the oversampling for a relatively large number of tail classes (more than about half).

Table 3.10 also shows the results of *Retraining-1, 3 and 5*. In the middle section of Table 3.10 (*i.e.*, no oversampling), *Synaptic join* shows the better performance than them since it can exploit the best active neurons among a lot of ones existing in DenseNet-121 for SUN-397, as in Table 3.2 and Fig. 3.8. Applying $\times 2.0$ loss penalty might be only appropriate for the data where each category has equal or similar number of examples, such as SVHN and ILSVRC12, but not for the highly imbalanced data such as SUN-397. Thus, we evaluate *Retraining-1 and 3* and other methods after making the data balanced via oversampling (*i.e.*, 2,311 images per class) and preparing the model trained using that data. The last section of Table 3.10 (*i.e.*, oversampling) shows the results. Although *Oversampling + Retraining-1 and 3* are better than *Synaptic join*, they are worse than *Oversampling + Synaptic join*. The improvement of *Retraining-5* after oversampling is not large since it oversamples the data that is

Table 3.10: Accuracy gains while varying the number of tail classes (DenseNet-121 for SUN-397).

Method	Target accuracy (%)		
	c=50	c=100	c=150
Original	31.12	35.20	37.79
Oversampling	49.42	50.80	51.59
Retraining-1	33.68	37.00	39.44
Retraining-3	41.68	44.58	41.11
Retraining-5	34.20	40.84	42.91
Synaptic join	50.08	51.62	51.48
Synaptic retraining-1	37.48	41.16	44.16
Synaptic retraining-2	42.28	45.92	48.32
Oversampling+Retraining-1	52.88	52.22	52.67
Oversampling+Retraining-3	53.44	51.24	52.76
Oversampling+Retraining-5	48.56	50.02	50.64
Oversampling+Synaptic join	**58.16**	56.42	55.47
Oversampling+Synaptic retraining-1	50.52	53.12	54.91
Oversampling+Synaptic retraining-2	57.00	**58.42**	**59.27**

already oversampled. *Synaptic retraining-2* tends to be more effective for oversampled data than imbalanced data, as *Retraining-1* does. Thus, *Oversampling + Synaptic retraining-2* becomes better than *Oversampling + Synaptic join*, in particular when $c = 100$ and $c = 150$.

3.3.9 Time and Space Cost of Synaptic Join

In this section, we evaluate the time and space cost of synaptic join. The cost is proportional to the three values, $|V|$, $|C|$, and $|D|$, as explained in Section 3.2.3. Since Algorithm 1 can be executed for each pair of a tuple of V and a tuple of C independently as in Fig. 3.2, synaptic join can be easily parallelized using GPUs, where Algorithm 1 is executed as a GPU kernel function. Using GPUs for synaptic join is a reasonable approach since the base model itself is usually trained using GPUs. In terms of implementation, we split the table V into chunks (smaller than the GPU memory), copy all the tables in Fig. 3.2 except V to GPUs, and then, copy each chunk of V to different GPUs while executing the kernel function.

Table 3.11 shows the elapsed times of (1) training a based model, (2) retraining the based model with a loss penalty, (3) performing synaptic join for all hidden neurons, in the same computer having four GPUs, and (4) performing synaptic retraining on the synapses selected by synaptic join (c=1 and n=120 for 3-layer CNN and c=10 and n=30 for GoogLeNet) until convergence. The time cost of synaptic join is much smaller than that of original training and retraining although we perform synaptic join for all hidden neurons. We note that the current implementation of synaptic join is not fully optimized, and so, can be further improved in terms of speed.

The time cost of synaptic retraining is also much smaller than that of the retraining methods. Since the number of synapses to be trained is small (*e.g.*, 120 for 3-layer CNN, 300 for GoogLeNet), the weights of the synapses are quickly converged (*e.g.*, 5k for 3-layer CNN, 20k for GoogLeNet). In addition, forward propagation of synaptic retraining does not need to pass through all synapses. The table V stores all the values of hidden neurons including active ones for every training data. Thus, forward propagation can be done quickly by regarding the active neurons as input neurons and skipping most of synapses. In terms of seconds per 1 k iterations, synaptic retraining improves the performance of the retraining methods by 7.25 times for 3-layer CNN and 8.14 times for GoogLeNet.

Fig. 3.11 shows the accuracies of *Retraining-1* and *Synaptic retraining-2* for 3-layer CNN at the correspondent iterations and time points. Synaptic retraining is more quickly converged than normal retraining, and at the same time, the time per iteration of synaptic retraining is shorter than that of normal retraining, as explained above. 5 k iterations of synaptic retraining are done about in 16 seconds, while only about 1.3 k iterations of normal retraining are done about in 31 seconds.

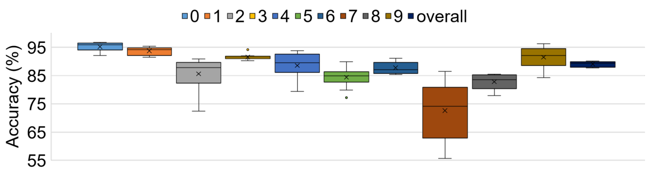
The time cost of determination of hyperparameters n and $scale$ in Section 3.2.3 is

Table 3.11: Time cost of training, retraining, synaptic join, and synaptic retraining.

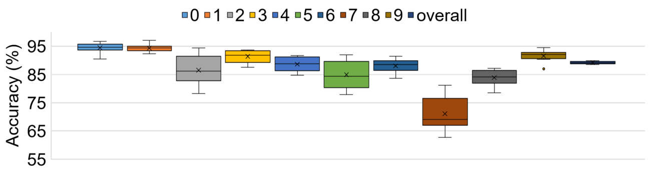
Experiment	SVHN (3-layer CNN)	ILSVRC12 (GoogLeNet)
# of neurons ($\|V\|$)	45,056	1,586,816
# of images ($\|D\|$)	73,257	1,281,167
# of classes ($\|C\|$)	10	1,000
Training	2,298 sec. (115k)	538,800 sec. (2,400k)
Retraining	232 sec. (10k)	54,090 sec. (240k)
Synaptic join	0.83 sec.	19,422 sec.
Synaptic retraining	16 sec. (5k)	554 sec. (20k)
Training / Retraining	23.2 sec. (1k)	225.4 sec. (1k)
Synaptic retraining	3.2 sec. (1k)	27.7 sec. (1k)

negligible compared to the above cost of Algorithm 1. We obtain a set of hidden neurons with their $\langle threshold, count\rangle$ as a result of Algorithm 1. If we consider the range between n_{min} and n_{max} for n, then we sort all hidden neurons by the $count$ and $threshold$ and select the top-n_{max} neurons. We determine both hyperparameters using a validation set, which is usually much smaller than the training set. The main cost for determination of both hyperparameters is just feed forwarding using the validation set while storing the following three kinds of values for each validation sample: (1) the values of output neurons, (2) the values of the top-n_{max} neurons, and (3) the weights of synapses between the top-n_{max} neurons and the target output neurons. We can calculate validation accuracy for each candidate pair of n and $scale$ only using simple operations (*e.g.*, matrix addition) on the stored values, and so, determination of n and $scale$ is done within few tens of seconds.

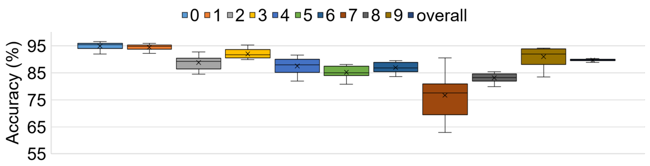
The space costs for storing the tables V and C in disks are $|V| \cdot |D|$ and $|C| \cdot |D|$, respectively, as shown in Fig. 3.2. Synaptic join has no memory problem since it is executed for each pair of a tuple of V and a tuple of C independently, as explained above. Synaptic retraining also can be done without a lack of memory, since it requires to load only small parts of the tables into main memory at once. We denote the set of active neurons by T. Then, as explained above, synaptic retraining regards T as input neurons and the T's rows of length $|D|$ in V as input examples. Thus, the amount of data that synaptic retraining needs to access is only $|T| \cdot |D| + |C| \cdot |D|$. In addition, synaptic retraining loads only one or a few batches of the data into main memory at once, as normal training or retraining does. We denote the batch size by $|B|$. In our experiments, we used the same batch sizes for normal training/retraining and synaptic retraining (e.g., $|B| = 150$ for SVHN and $|B| = 64$ for ILSVRC12). Thus, the amount of data that synaptic retraining loads into main memory is just $|T| \cdot |B| + |C| \cdot |B|$, which is quite small. For example, even if a 10% of all neurons of GoogLeNet are selected as active neurons, the size of data loaded is just $158,682 \cdot 64 \cdot 4 + 1,000 \cdot 64 \cdot 4 = 41\,MB$, when using 4-byte floating point numbers.



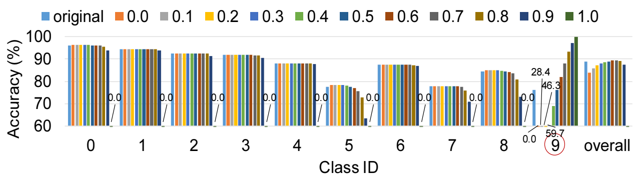
(a) *Retraining-1*: $\times 2.0$ loss penalty to the target class (average target accuracy: 91.44%, average overall accuracy: 89.50%)



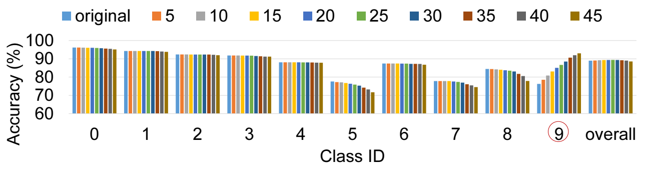
(b) *Retraining-3*: $\times 2.0$ loss penalty to the target class from the beginning (average target accuracy: 91.53%, average overall accuracy: 89.51%)



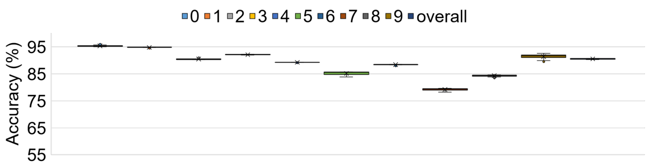
(c) *Retraining-5*: $\times 2$ oversampling for the target class (average target accuracy: 90.94%, average overall accuracy: 89.68%)



(d) *Calibration*: varying μ between 0.0 and 1.0 (when μ=0.9, the target accuracy is 97.19%, and overall accuracy is 87.36%)



(e) *Synaptic join*: n=120, varying $scale$ between 5 and 45 (when $scale$=40, the target accuracy is 91.97%, and overall accuracy is 89.02%)



(f) *Synaptic retraining-2*: $\times 2.0$ loss penalty to the target class and n=120 (target accuracy: 91.74%, overall accuracy: 90.59%).

Figure 3.7: Per-class and overall test accuracies of compared methods (3-layer CNN for SVHN).

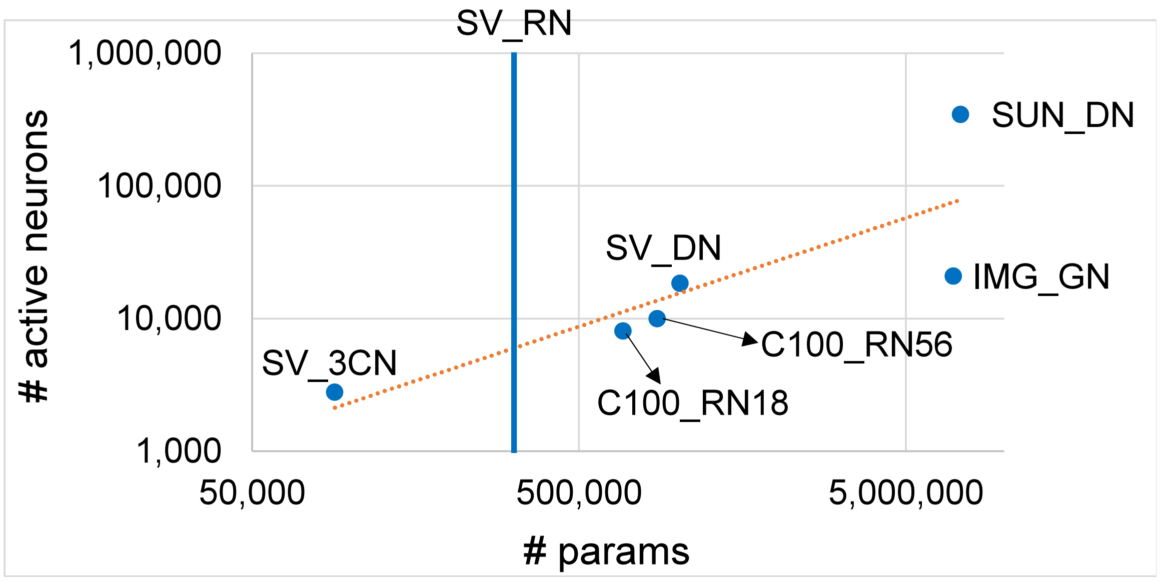
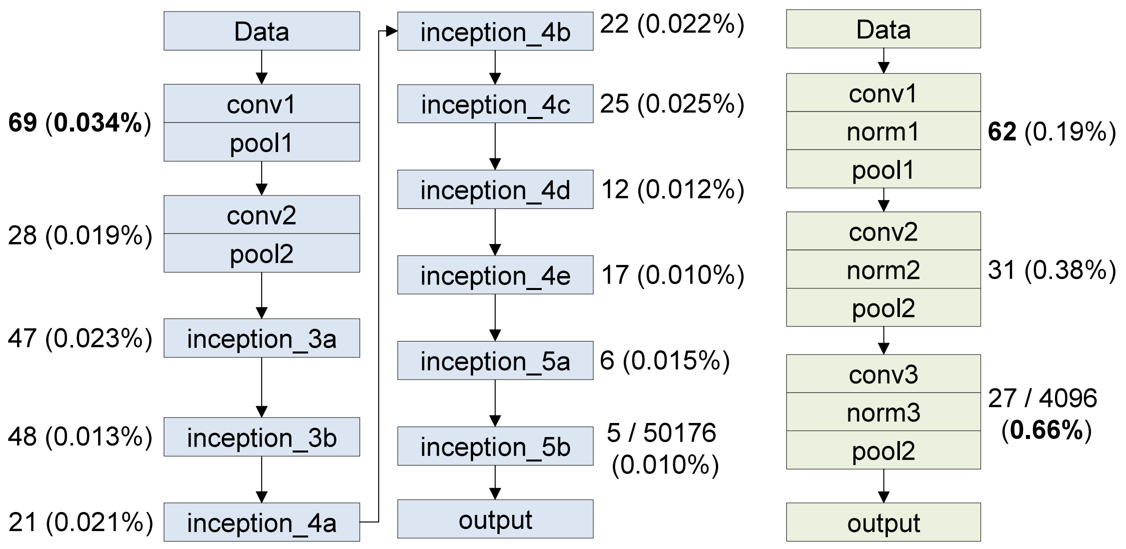


Figure 3.8: Correlation between the number of parameters and the number of active neurons (SV_3CN: 3-layer CNN for SVHN, SV_RN: ResNet-20 for SVHN, SV_DN: DenseNet-40 for SVHN, C100_RN18: ResNet18 for CIFAR-100, C100_RN56: ResNet18 for CIFAR-100, SUN_DN: DenseNet-121 for SUN-397, IMG_GN: GoogLeNet for ImageNet).



(a) GoogLeNet (b) 3-layer CNN

Figure 3.9: Distribution of active neurons.

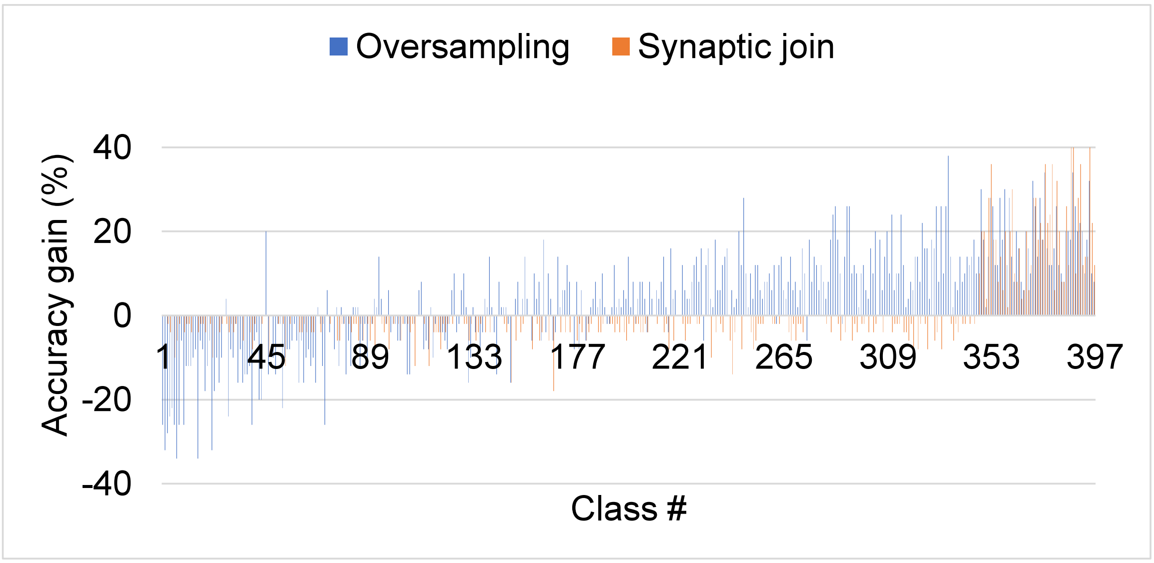
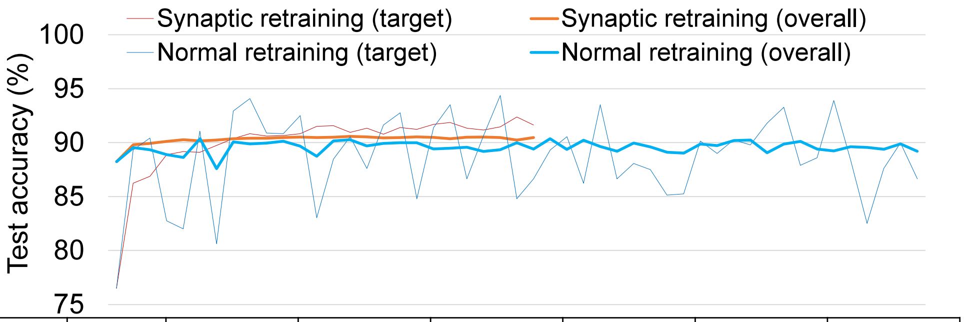


Figure 3.10: Oversampling method and synaptic join (DenseNet-121 for SUN-397, c=50, n=100, $scale$=6).



Retraining time (sec.)		0	5.12	10.24	15.36	20.48	25.6	30.72
Number of iterations	Synaptic	0	1600	3200	4800	–	–	–
	Normal	0	216	432	648	864	1080	1296

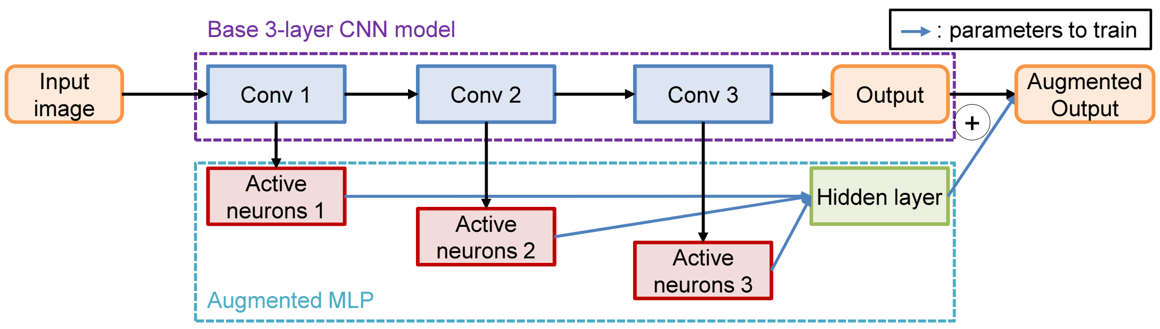
Figure 3.11: Accuracies of normal retraining and synaptic retraining at equal number of iterations.

Chapter 4. Augmenting Deep Neural Networks with Active Neurons

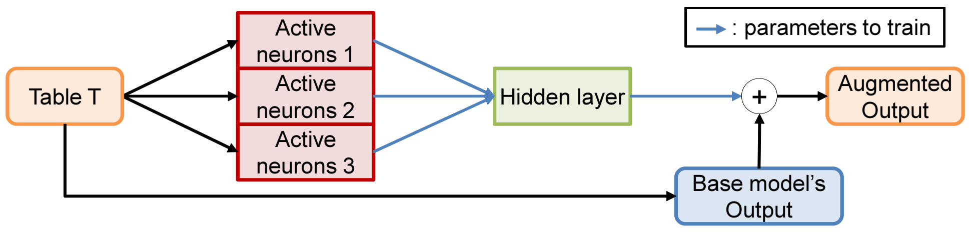
4.1 Augmentation with Active Neurons

In this chapter, we present augmented deep neural networks with active neurons. We call this augmentation with active neurons method as *AAN*. The proposed method is similar to but extended to synaptic retraining. Unlike the previous chapter, which improved only certain classes, the goal is to improve all classes. This method first selects active neurons to improve all classes, then constructs small-size networks and augments them to the original base model. Unlike the synaptic join and synaptic retraining introduced in the previous chapter, we designed the augmented networks in the form of a multi-layer perceptron (MLP) which has a single hidden layer in order to learn non-linearity in the training process. Also, the proposed augmentation method does not change the base models during the training phases; in other words, as in Chapter 3.3.9, by referring to the output values of active neurons and base model determined in tables V and C (which we call table T) and using them as inputs, fast training is possible.

Fig 4.1 shows an example of an augmented model in 3-layer CNN on SVHN. Fig 4.1a shows the concept that our method extracts active neurons from each layer of the base model, inputs them as hidden layers, and corrects the output. Fig 4.1b shows the AAN model actually used in our experiment. By using Table T in which active neurons and base model's output are already stored as input, the AAN model is efficiently learned without forward propagation of the base model.



(a) A concept design of augmented networks with a base model (3-layer CNN)



(b) Actual design of augmented networks

Figure 4.1: Augmented networks with active neurons.

4.2 Experimental Evaluation

4.2.1 Environments

We use various neural networks for four datasets, SVHN (cropped digits) [62], CIFAR-100 [47], ImageNet (ILSVRC12, classification) [74], and SUN-397 [92]. SVHN is a real-world color image dataset of the digits obtained from house numbers in Google Street View images, which consists of 10 classes of digits and contains 73,257 training samples and 26,032 testing samples of 32 $\times$ 32 pixels. CIFAR-100 [47] is a collection of color images of 32 $\times$ 32 pixels that contains 100 classes, 500 training samples per class, and 100 testing samples per class. ImageNet consists of 1,000 categorical real-world images. The numbers of training and testing samples are about 1.2 million and 50,000, respectively. SUN-397 is a skewed and long-tailed dataset of color images for scene understanding. The dataset consists of 108,754 images of 397 classes, and each class consists of 100 to 2,361 images. We randomly choose 50 images as test data for each class and use the remaining 50 to 2,311 images as training data. For both ImageNet and SUN-397, we resized the images to 256×256 pixels and properly cropped them to satisfy the requirement on the size of the input for each model.

In all the experiments, we split training samples into training and validation samples. For SVHN, 2,000 samples per class are used for validation, and the remaining samples for training. For SUN-397 and ImageNet, 25 and 500 samples per class are used for validation, respectively, and the remaining samples for training. We performed synaptic join using the training samples and determined hyperparameters using the validation samples. Then, we measured the accuracy using the test samples.

We explain the base neural network models used in the experiments. For SVHN, we have modified the 3-layer CNN model in [1] by adding a batch normalization layer to each convolution layer. After training the model for 115 k iterations with a start learning rate of 0.001 and reduced the learning rate by 1/10 at 75 k iterations, the model achieved 88.99% overall test accuracy. For CIFAR-100, we have modified ResNet-18 [2] and ResNet-56 [39] models. The ResNet-18 model has five residual blocks, and each block consists of three or four convolution layers. After 40 k iterations of training with an initial learning rate of 0.1, which was reduced by a factor of 10 for every 5 k iterations, the model achieved 54.8% overall accuracy. The ResNet-56 model has three residual groups, and each group has eight residual blocks, and each block consists of two or three convolution layers. After 64 k iterations of training with an initial learning rate of 0.1, which is reduced by a factor of 10 at 32 k and 48 k iterations, the model achieved 66.1% overall test accuracy. For ImageNet, we have used the GoogLeNet [83] binary model which has 68.98% overall accuracy (top-1). For more models used in a specific experiment, we explain them in the corresponding sections. For SUN-397, we prepare the base model by fine-tuning a pre-trained DenseNet-121 model for ImageNet as explained in [90], which has an overall accuracy of 49.8%.

We conducted all the experiments on a workstation equipped with two Intel Xeon 2.2 GHz CPUs of ten cores (a total of 20 cores), 384 GB main memory, eight NVIDIA GTX 1080Ti GPUs of 11 GB memory, and 10 TB HDD. We trained and ran the models using CAFFE [45].

4.2.2 Comparison with DA and WA Method

This chapter compares the depth augmented (DA) method and width augmented (WA) method with our augmentation with active neurons approach. The DA model is constructed by adding a fully-connected layer after the last representation module of each base model. The WA model is constructed by adding a fully-connected layer aside along with the last representation module of each base model. We add an augmented layer for each DA and WA model, with 512 hidden units for 3-layer CNN, ResNet-20, and DenseNet-40 models for SVHN data. Unlike the DA and WA methods, our method no longer uses the base model for training. Instead, our augmented models only require a relatively small amount of parameters because they consist only of MLPs with fully-connected active neurons and single hidden layers. In the case of 3-layer CNN for SVHN, a total of 1,000 active neurons are propagated to a hidden layer with 96 units. The augmented output is then obtained by adding the output of the base model to the output propagated from the augmented hidden layer. Since the entire outputs of the base model are simply referenced from the table T, which are already stored during synaptic join operation, our model does not require any forward-propagation of data to the base model. After that, the augmented networks are trained using the loss of the final augmented output. Table 4.1 shows the number of hidden units and active neurons used in the DA models, WA models, and our models. We selected the optimal value for the size of each unit using the validation sets.

Table 4.2 shows a comparison of the number of parameters between DA and WA

Table 4.1: Number of hidden units and active neurons used in DA, WA and our models.

Dataset	Model	DA	WA	AAN (ours)	
		# hidden units	# hidden units	# active neurons	# hidden units
SVHN	3-layer CNN	512	512	1,000	96
	ResNet-20	512	512	300	192
	DenseNet-40	512	512	1,600	192
CIFAR-100	ResNet-18	1,024	1,024	4,000	320
	ResNet-56	1,024	1,024	3,000	192
ILSVRC12	GoogLeNet	4,096	4,096	6,314	1,024
SUN-397	DenseNet-121	2,048	1,024	11,910	256

models. The numbers of additional parameters of the DA and WA methods for the base models are proportional to the number of neurons in the layer where augmentation starts, the number of hidden neurons in the DA and WA layers, and the number of classes. In Table 4.2, all the numbers of augmented parameters for DA models are smaller than those of the WA models. The WA model takes the feature maps of the layer before the last layer in the representation module as the input, whereas the DA model takes the feature maps of the last layer in the representation module as the input. Since the size of each layer (the size of feature maps) in deep neural networks usually gets decreased as the depth of the layer becomes deeper, the size of additional parameters on DA model also smaller than that of WA model.

Table 4.2: Comparison of the number of parameters between DA and WA models.

Dataset	Model	Original	DA		WA	
			# param	Ratio for original	# param	Ratio for original
SVHN	3-layer CNN	89.6 K	608 K	679%	1.14 M	1,275%
	ResNet-20	272 K	309 K	114%	10.9 M	1,077%
	DenseNet-40	1.02 M	1.25 M	123%	2.37 M	872%
CIFAR-100	ResNet-18	684 K	6.58 M	963%	30.2 M	4,429%
	ResNet-56	869 K	1.03 M	119%	5.17 M	594%
ILSVRC12	GoogLeNet	6.99 M	20.6 M	295%	184 M	2,639%
SUN-397	DenseNet-121	7.36 M	9.86 M	134%	41.3 M	561%

Table 4.3 shows a comparison of the number of parameters between DA and our models. The number of additional parameters of the DA method for the base models is proportional to the number of neurons in the last representation layer, the number of hidden neurons in the DA layer, and the number of classes. On the other hand, in our method, the number of whole parameters is only proportional to the number of active neurons, the number of hidden neurons in the added layer, and the number of classes. The last two columns of the table show the ratio of the number of parameters for the base models and the DA models, respectively. Our method can be configured with a small number of parameters less than about 22% of the original and DA even though the active neurons of a sufficiently large size are used to improve all classes.

Table 4.3: Comparison of the number of parameters between DA and our models.

Dataset	Model	Original	DA	Synaptic retraining		
				# param	Ratio for original	Ratio for DA
SVHN	3-layer CNN	89.6 K	608 K	35.2 K	39.2%	5.8%
	ResNet-20	272 K	309 K	21.7 K	8.0%	7.0%
	DenseNet-40	1.02 M	1.25 M	86.9 K	8.5%	7.0%
CIFAR-100	ResNet-18	684 K	6.58 M	300 K	43.9%	4.6%
	ResNet-56	869 K	1.03 M	217 K	25.0%	21.1%
ILSVRC12	GoogLeNet	6.99 M	20.6 M	2.71 M	38.9%	13.2%
SUN-397	DenseNet-121	7.36 M	9.86 M	840 K	11.4%	8.5%

Table 4.4 compares the accuracy of the base model, DA, WA, and our method. In the case of DA, the accuracy is improved over the original accuracy in most models; However, for already highly optimized models such as GoogLeNet and ResNet-56, the accuracy was rather reduced during the retraining process. On the other hand, our method obtained not only better accuracy in all models than the original even though a small number of parameters were used, but also mostly better or comparable results than the DA method. Even in GoogLeNet and ResNet-56 models that the DA method did not improve, our method was able to improve the

accuracy.

Table 4.4: Comparison of overall accuracy between DA, WA and our models.

Dataset	Model	Overall accuracy (%)			
		Original	DA	WA	AAN
SVHN	3-layer CNN	88.99	90.14	90.88	**91.00**
	ResNet-20	95.00	**95.09**	94.4	95.08
	DenseNet-40	96.00	**96.25**	94.53	96.19
CIFAR-100	ResNet-18	54.80	54.90	52.32	**54.92**
	ResNet-56	66.10	64.77	65.05	**66.21**
ILSVRC12	GoogLeNet	68.98	68.80	64.93	**69.03**
SUN-397	DenseNet-121	49.75	50.06	47.31	**50.36**

Table 4.5 compares the accuracy of our method to the synaptic join and synaptic retraining methods introduced in the previous chapter. For the synaptic join and synaptic retraining methods, the numbers of targets in synaptic join and synaptic retraining for SVHN and CIFAR-100 datasets are one; the number of targets for ILSVRC12 dataset is 10; the number of targets for SUN-397 dataset is 50. For the AAN method, the number of targets for each dataset is the same as the number of classes. Since the AAN method uses more active neurons and parameters than those of synaptic join and synaptic retraining methods, the overall accuracies of the AAN method are better for most models.

Table 4.5: Comparison of overall accuracy between synaptic join, synaptic retraining and our method.

Dataset	Model	Synaptic join	Synaptic retraining	AAN
SVHN	3-layer CNN	89.02 (+0.03)	90.68 (+1.69)	**91.00** (+2.01)
	ResNet-20	95.00 (+0.00)	95.00 (+0.00)	**95.08** (+0.08)
	DenseNet-40	96.00 (+0.00)	**96.30** (+0.30)	96.19 (+0.19)
CIFAR-100	ResNet-18	54.80 (+0.00)	54.83 (+0.03)	**54.92** (+0.12)
	ResNet-56	66.10 (+0.00)	66.11 (+0.01)	**66.21** (+0.11)
ILSVRC12	GoogLeNet	68.98 (+0.00)	68.99 (+0.01)	**69.03** (+0.05)
SUN-397	DenseNet-121	50.12 (+0.37)	50.19 (+0.44)	**50.36** (+0.61)

Table 4.6 compares the training times per 1 K iterations of original, DA, WA, and

our models. As the number of parameters increases, DA and WA models need a longer learning time for the same 1 K iterations. The model with the largest increase in training time is GoogLeNet, which takes about 514 seconds to train the DA model and 556 seconds to train the WA model by 1 K iterations. On the other hand, our method requires less than 20% training time for most models compared to the DA method. Exceptionally, in 3-layer CNN, both the base model and the DA model have a small model size and a short training time, so the training time of our method has a ratio of 33.6% for DA method. In particular, even in GoogLeNet, our method can learn 10 times faster than the DA method and obtain better accuracy.

Table 4.6: Comparison of training time of 1 K iterations between DA, WA and our models.

Dataset	Model	Training time (sec.)				Ratio for
		Original	DA	WA	AAN	DA
SVHN	3-layer CNN	23.2	24.1	34.2	**8.1**	33.6%
	ResNet-20	133.6	164.1	181.2	**7.8**	4.8%
	DenseNet-40	254.4	304.7	324.9	**14.1**	4.6%
CIFAR-100	ResNet-18	120.1	137.6	165.2	**22.4**	16.3%
	ResNet-56	364.1	395.5	426.0	**19.1**	4.8%
ILSVRC12	GoogLeNet	225.4	514.9	556.3	**48.9**	9.5%
SUN-397	DenseNet-121	172.5	379.6	401.3	**29.6**	7.8%

Chapter 5. Related Work

There are a number of methods that delete existing synapses from the original neural network. They are widely used for compressing the neural networks in order to save memory space and computational resources. In particular, they are useful in resource-limited environments such as mobile devices or embedded systems. Network pruning [34, 35] removes all the synapses of weights below a threshold from the original network, and so, converts a dense network into a sparse network. In particular, this pruning method can reduce much of the size and amount of computations for the networks having many fully connected layers since there exist much more number of synapses in fully connected layers than convolutional layers. We note that this method removes the existing synapses having weights around zero, while our method adds new synapses having weights around zero. Dynamic network surgery [33] has been proposed to avoid incorrect pruning of the synapses. This method evaluates the importance of every synapses at each iteration and decides each synapse should be pruned or spliced. Pruning filters [51] has been proposed to avoid sparse kernels (filters) in convolutional neural networks. This method prunes the whole sparse (*i.e.*, unimportant) filters with their connected feature maps to reduce the inference cost of the networks.

The network dissection method [13] tries to interpret a CNN by scoring the semantics of each hidden unit into six human-labeled visual concepts, which are the scene, object, part, material, texture, and color. The method evaluates every individual convolutional unit in a CNN as a solution to a binary segmentation task for every visual concept. In detail, it determines a threshold T_k for each convolutional unit k such that $P(a_k \geq T_k) = 0.005$, where a_k is the activation of unit k. Then, after selecting all the regions for which the activation exceeds the threshold T_k, the score of each unit k as a segmentation for concept c is calculated as a dataset-wide Intersection over Union (IoU). This method is similar to our method in terms of the concept of scoring each hidden unit k during the feed-forwarding, but quite different from our method in terms of the purpose of scoring and the method of scoring. The scoring of the network dissection method is for evaluating the quality of the segmentation of each unit. It also scores each unit for each visual concept individually. In contrast, our method scores each hidden neuron by considering both the target and off-target classes. Moreover, our method not only scores a neuron but also calculates a suitable weight from the neuron to the target class node.

Zero-shot learning [17, 49] aims to solve a task whose instances may not have been trained. It uses the side information of the classes, *i.e.*, attributes, to infer the label of one of the unseen classes. It first predicts the attributes of an input image whose class label is unseen during the training stage, and then, infers the class label of the image by searching the class which has the most similar set of attributes. Although zero-shot learning itself is not related to our method, the calibration method proposed in [17, 49] can be used for tweaking the neural networks, and thus, we compared it with our method. In detail, it calibrates the prediction for the target class with a prior probability μ and that for the off-target classes with a prior probability $(1-\mu)$ $(0.0 \leq \mu \leq 1.0)$. In our experiments, it tends to increase the target accuracy too drastically, and thus, some off-target class accuracies decrease suddenly and unexpectedly.

Recently, there is growing interest in automating designing good neural network architectures [25, 46, 53, 59, 70, 73, 84, 94, 105, 106]. There are two major types of network architecture search (NAS) methods: reinforcement learning-based (RL) methods and evolutionary algorithm-based (EV) methods. RL methods [70, 84, 105, 106] use a controller model that enumerates a bunch of candidate models and is updated using the validation accuracy of the candidate models. To reduce the search space, they usually assume a candidate model is composed of *cells* having the same architecture and focus on searching for the best cell architecture. A cell is composed of multiple blocks, and each block is composed of multiple operations, which are selected from a pool of various operations (*e.g.*, convolution, pooling). EV methods [46, 53, 59, 73, 94] search a good architecture based on evolutionary algorithm. The population is initialized with models with random architectures, and some models are sampled from that. The model with the highest validation fitness within the samples is selected as the parent (*i.e.*, exploitation), and a child having a mutation in terms of operations and skip connections is constructed from the parent (*i.e.*, exploration). The existing NAS methods usually search for a good architecture in terms of operations. In contrast, synaptic join can be considered as a greedy heuristic NAS method in terms of synapses.

Chapter 6. Conclusions

In this dissertation, we proposed Synaptic Join and Augmentation with Active Neurons methods that could improve the accuracy of a deep neural network model without modifying the original model.

In Chapter 3, the proposed method tweaks a neural network by adding a small number of new synapses from the active hidden neurons to the output neurons of the target classes. We proposed the algorithm θ that can evaluate the performance of all the possible candidate synapses in the training data. In addition, we proposed the synaptic join and synaptic retraining methods based on θ. Through experiments, we demonstrated that the methods could control the test accuracy of the target and off-target classes in a more predictable and controllable manner than the other methods. We note that the model tweaked by the proposed method is not permanent, *i.e.*, it can be recovered to the original model simply by removing the synapses; in contrast, the model tweaked by the conventional weight retraining methods is permanent. We expect that the proposed method can be used to operate a single original neural network differently by using different user-/application-specific sets of synapses, *i.e.*, this method can make customized AI service possible without retraining.

In Chapter 4, we proposed augmented deep neural networks with active neurons. The existing augmentation methods, which insert layers in the original model, usually generate too large-size models and require a long training time. The proposed method feeds only the active neurons to a small network model for efficient learning to overcome this problem. The synaptic join method in Chapter 3 exploits only the linearity of each synapse to tweak a given neural network model; however, the proposed augmentation with active neurons can train models to learn non-linearity. Through experiments, we demonstrated that the methods could improve the accuracy with a short training time and a small number of parameters.

In conclusion, the proposed methods can improve the model to suit the user's purpose without changing the original deep neural network. We believe that the proposed methods can be helpful in the application field of customized artificial intelligence services.

References

[1] 3-layer cnn for cifar10. `https://github.com/BVLC/caffe/blob/master/examples/cifar10`. Accessed on 31.10.2017.

[2] Resnet18 for cifar-100. `https://github.com/beniz/deepdetect/blob/master/templates/caffe/resnet_18/deploy.prototxt`. Accessed on 31.10.2017.

[3] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[4] C. Affonso, A. L. D. Rossi, F. H. A. Vieira, A. C. P. de Leon Ferreira, et al. Deep learning for biological image classification. *Expert Systems with Applications*, 85:114–122, 2017.

[5] T. Afouras, J. S. Chung, A. Senior, O. Vinyals, and A. Zisserman. Deep audio-visual speech recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[6] C. C. Aggarwal et al. Neural networks and deep learning. *Springer*, 10:978–3, 2018.

[7] F. Ahmad, A. Abbasi, J. Li, D. G. Dobolyi, R. G. Netemeyer, G. D. Clifford, and H. Chen. A deep learning architecture for psychometric natural language processing. *ACM Transactions on Information Systems (TOIS)*, 38(1):1–29, 2020.

[8] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, et al. Deep speech 2: End-to-end speech recognition in

english and mandarin. In *International conference on machine learning*, pages 173–182. PMLR, 2016.

[9] K. Antczak. Deep recurrent neural networks for ecg signal denoising. *arXiv preprint arXiv:1807.11551*, 2018.

[10] S. Bacchi, L. Oakden-Rayner, T. Zerner, T. Kleinig, S. Patel, and J. Jannes. Deep learning natural language processing successfully predicts the cerebrovascular cause of transient ischemic attack-like presentations. *Stroke*, 50(3):758–760, 2019.

[11] M. Bakator and D. Radosav. Deep learning and medical diagnosis: A review of literature. *Multimodal Technologies and Interaction*, 2(3):47, 2018.

[12] K. Bantupalli and Y. Xie. American sign language recognition using deep learning and computer vision. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4896–4899. IEEE, 2018.

[13] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 3319–3327. IEEE, 2017.

[14] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[15] M. Biswas, V. Kuppili, L. Saba, D. R. Edla, H. S. Suri, E. Cuadrado-Godia, J. R. Laird, R. T. Marinhoe, J. M. Sanches, A. Nicolaides, et al. State-of-the-art review on deep learning in medical imaging. *Frontiers in bioscience (Landmark edition)*, 24:392–426, 2019.

[16] M. A. Carreira-Perpinán and Y. Idelbayev. “learning-compression” algorithms for neural net pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8532–8541, 2018.

[17] W.-L. Chao, S. Changpinyo, B. Gong, and F. Sha. An empirical study and analysis of generalized zero-shot learning for object recognition in the wild. In *European Conference on Computer Vision*, pages 52–68. Springer, 2016.

[18] Q. Chen, X. Zhu, Z. Ling, S. Wei, H. Jiang, and D. Inkpen. Enhanced lstm for natural language inference. *arXiv preprint arXiv:1609.06038*, 2016.

[19] J.-T. Chien. Deep bayesian natural language processing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 25–30, 2019.

[20] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *JMLR*, 12(Aug):2493–2537, 2011.

[21] L. Deng, G. Hinton, and B. Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8599–8603. IEEE, 2013.

[22] L. Deng and Y. Liu. A joint introduction to natural language processing and to deep learning. In *Deep learning in natural language processing*, pages 1–22. Springer, 2018.

[23] L. Deng and D. Yu. Deep learning for signal and information processing. *Microsoft research monograph*, 2013.

[24] L. Deng and D. Yu. Deep learning: methods and applications. *Foundations and trends in signal processing*, 7(3–4):197–387, 2014.

[25] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[26] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019.

[27] H. Garcia-Molina. *Database systems: the complete book*. Pearson Education India, 2008.

[28] E. Goceri and N. Goceri. Deep learning in medical image analysis: recent advances and future trends. 2017.

[29] Y. Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

[30] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning (adaptive computation and machine learning series), 2016.

[31] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

[32] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, 2016.

[33] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *NIPS*, pages 1379–1387, 2016.

[34] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

[35] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural network. In *NIPS*, pages 1135–1143, 2015.

[36] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[37] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[38] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[39] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[40] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[41] J. Hirschberg and C. D. Manning. Advances in natural language processing. *Science*, 349(6245):261–266, 2015.

[42] S. Hochreiter, Y. Bengio, and P. Fransconi. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A Field Guide to Dynamical Recurrent Neural Networks*, 2011.

[43] L. Hongtao and Z. Qinchuan. Applications of deep convolutional neural network in computer vision. *Journal of Data Acquisition and Processing*, 31(1):1–17, 2016.

[44] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected con-

volutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[45] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, pages 675–678, 2014.

[46] R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015.

[47] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[49] C. H. Lampert, H. Nickisch, and S. Harmeling. Attribute-based classification for zero-shot visual object categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):453–465, 2014.

[50] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[51] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[52] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[53] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

[54] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[55] A. S. Lundervold and A. Lundervold. An overview of deep learning in medical imaging focusing on mri. *Zeitschrift für Medizinische Physik*, 29(2):102–127, 2019.

[56] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[57] M. Mehdipour Ghazi and H. Kemal Ekenel. A comprehensive analysis of deep learning based representation for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 34–41, 2016.

[58] H. Meng, T. Yan, F. Yuan, and H. Wei. Speech emotion recognition from 3d log-mel spectrograms with deep learning network. *IEEE access*, 7:125868–125881, 2019.

[59] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

[60] L. Min, Q. Chen, and S. Yan. Network in network. *arXiv:1312.4400*, 2013.

[61] A. Mousavi, A. B. Patel, and R. G. Baraniuk. A deep learning approach to structured signal recovery. In *2015 53rd annual allerton conference on communication, control, and computing (Allerton)*, pages 1336–1343. IEEE, 2015.

[62] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.

[63] H. Ninomiya, N. Kitaoka, S. Tamura, Y. Iribe, and K. Takeda. Integration of deep bottleneck features for audio-visual speech recognition. In *Sixteenth annual conference of the international speech communication association*, 2015.

[64] K. Noda, Y. Yamaguchi, K. Nakadai, H. G. Okuno, and T. Ogata. Audio-visual speech recognition using deep learning. *Applied Intelligence*, 42(4):722–737, 2015.

[65] D. W. Otter, J. R. Medina, and J. K. Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[66] W. Ouyang and X. Wang. Joint deep learning for pedestrian detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2056–2063, 2013.

[67] O. K. Oyedotun and A. Khashman. Deep learning in vision-based static hand gesture recognition. *Neural Computing and Applications*, 28(12):3941–3951, 2017.

[68] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh. Deep learning vs. traditional computer vision. In *Science and Information Conference*, pages 128–144. Springer, 2019.

[69] J. Padmanabhan and M. J. Johnson Premkumar. Machine learning in automatic speech recognition: A survey. *IETE Technical Review*, 32(4):240–251, 2015.

[70] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[71] H. Purwins, B. Li, T. Virtanen, J. Schlüter, S.-Y. Chang, and T. Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019.

[72] D. Ravì, C. Wong, F. Deligianni, M. Berthelot, J. Andreu-Perez, B. Lo, and G.-Z. Yang. Deep learning for health informatics. *IEEE journal of biomedical and health informatics*, 21(1):4–21, 2016.

[73] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[74] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[75] B. Sahiner, A. Pezeshk, L. M. Hadjiiski, X. Wang, K. Drukker, K. H. Cha, R. M. Summers, and M. L. Giger. Deep learning in medical imaging and radiation therapy. *Medical physics*, 46(1):e1–e36, 2019.

[76] M. E. Sánchez-Gutiérrez, E. M. Albornoz, F. Martinez-Licona, H. L. Rufiner, and J. Goddard. Deep learning for emotional speech recognition. In *Mexican conference on pattern recognition*, pages 311–320. Springer, 2014.

[77] A. Satt, S. Rozenberg, and R. Hoory. Efficient emotion recognition from speech using deep learning on spectrograms. In *Interspeech*, pages 1089–1093, 2017.

[78] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio. Robust object recogni-

tion with cortex-like mechanisms. *IEEE transactions on pattern analysis and machine intelligence*, 29(3):411–26, 2007.

[79] D. Shen, G. Wu, and H.-I. Suk. Deep learning in medical image analysis. *Annual review of biomedical engineering*, 19:221–248, 2017.

[80] A. Siddhant and Z. C. Lipton. Deep bayesian active learning for natural language processing: Results of a large-scale empirical study. *arXiv preprint arXiv:1808.05697*, 2018.

[81] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[82] S. P. Singh, L. Wang, S. Gupta, H. Goli, P. Padmanabhan, and B. Gulyás. 3d deep learning on medical images: a review. *Sensors*, 20(18):5097, 2020.

[83] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[84] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[85] Y. Tian, P. Luo, X. Wang, and X. Tang. Deep learning strong parts for pedestrian detection. In *Proceedings of the IEEE international conference on computer vision*, pages 1904–1912, 2015.

[86] M. V. Valueva, N. Nagornov, P. A. Lyakhov, G. V. Valuev, and N. I. Chervyakov. Application of the residue number system to reduce hardware costs of the convolutional

neural network implementation. *Mathematics and Computers in Simulation*, 177:232–243, 2020.

[87] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[88] S. Wang and J. Jiang. Learning natural language inference with lstm. *arXiv preprint arXiv:1512.08849*, 2015.

[89] Y.-X. Wang, D. Ramanan, and M. Hebert. Growing a brain: Fine-tuning by increasing model capacity. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[90] Y.-X. Wang, D. Ramanan, and M. Hebert. Learning to model the tail. In *Advances in Neural Information Processing Systems*, pages 7029–7039, 2017.

[91] S. Wu, K. Roberts, S. Datta, J. Du, Z. Ji, Y. Si, S. Soni, Q. Wang, Q. Wei, Y. Xiang, et al. Deep learning in clinical natural language processing: a methodical review. *Journal of the American Medical Informatics Association*, 27(3):457–470, 2020.

[92] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3485–3492. IEEE, 2010.

[93] D. Xie, L. Zhang, and L. Bai. Deep learning in visual computing and signal processing. *Applied Computational Intelligence and Soft Computing*, 2017, 2017.

[94] L. Xie and A. Yuille. Genetic cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1379–1388, 2017.

[95] H. Yang, L. Luo, L. P. Chueng, D. Ling, and F. Chin. Deep learning and its applications to natural language processing. In *Deep learning: Fundamentals, theory and applications*, pages 89–109. Springer, 2019.

[96] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[97] D. Yu, M. L. Seltzer, J. Li, J.-T. Huang, and F. Seide. Feature learning in deep neural networks-studies on speech recognition tasks. *arXiv preprint arXiv:1301.3605*, 2013.

[98] J. Yu, K. Weng, G. Liang, and G. Xie. A vision-based robotic grasping system using deep learning for 3d object recognition and pose estimation. In *2013 IEEE international conference on robotics and biomimetics (ROBIO)*, pages 1175–1180. IEEE, 2013.

[99] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[100] J. Zhang, Y. Xie, Q. Wu, and Y. Xia. Medical image classification using synergic deep learning. *Medical image analysis*, 54:10–19, 2019.

[101] W. E. Zhang, Q. Z. Sheng, A. Alhazmi, and C. Li. Adversarial attacks on deep-learning models in natural language processing: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(3):1–41, 2020.

[102] Z. Zhang, J. Geiger, J. Pohjalainen, A. E.-D. Mousa, W. Jin, and B. Schuller. Deep learning for environmentally robust speech recognition: An overview of recent developments. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 9(5):1–28, 2018.

[103] W. Zhu, S. M. Mousavi, and G. C. Beroza. Seismic signal denoising and decomposition using deep neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 57(11):9476–9488, 2019.

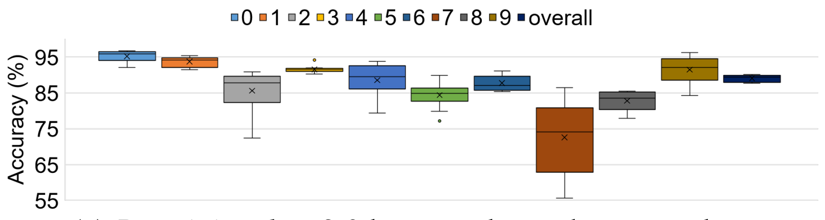
[104] L. Zhuang. Densenet-40 model reference for cifar dataset. `https://github.com/liuzhuang13/DenseNetCaffe`. Accessed on 31.10.2019.

[105] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

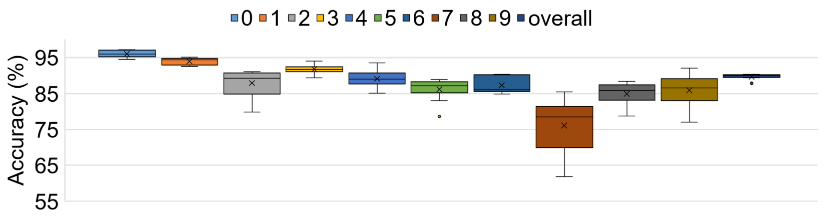
[106] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

APPENDIX

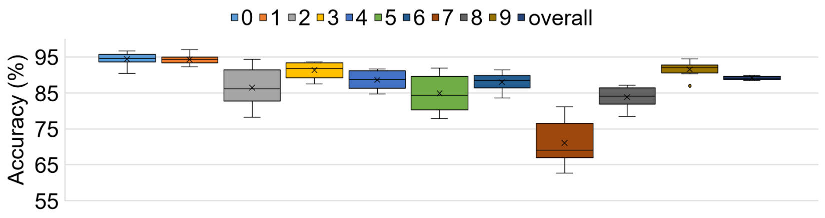
A Result of Simple Retraining Methods



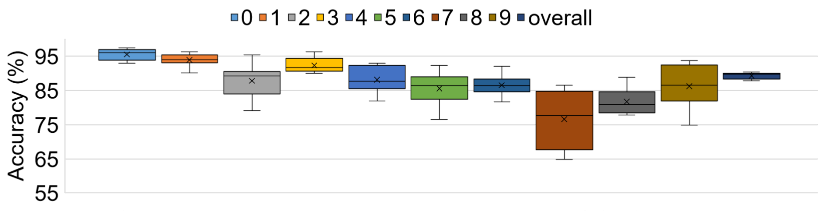
(a) *Retraining-1*: $\times 2.0$ loss penalty to the target class



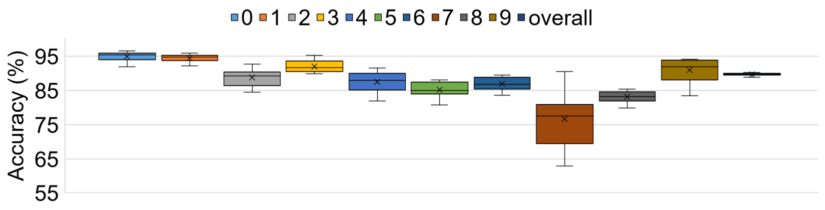
(b) *Retraining-2*: $\times 0.9$ loss penalty to the off-target classes



(c) *Retraining-3*: $\times 2.0$ loss penalty to the target class from the beginning

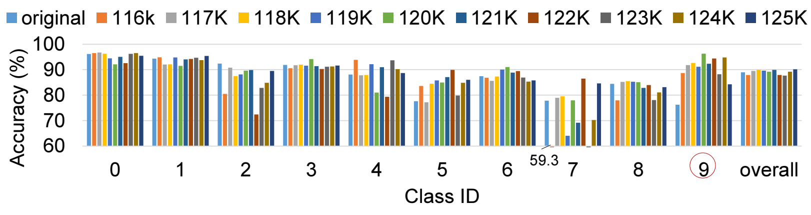


(d) *Retraining-4*: $\times 2.0$ loss penalty with dropout

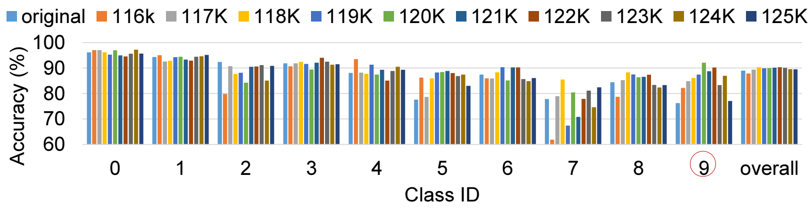


(e) *Retraining-5*: $\times 2.0$ oversampling for the target class

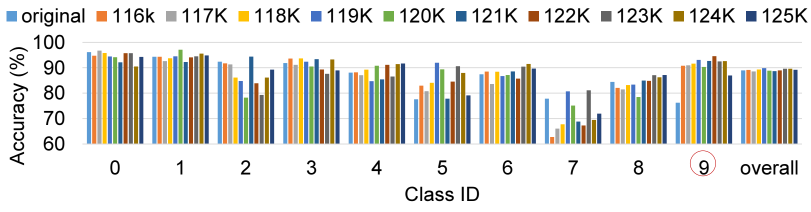
Figure 6.1: Boxplot comparison of class accuracy of five retraining methods (3-layer CNN for SVHN).



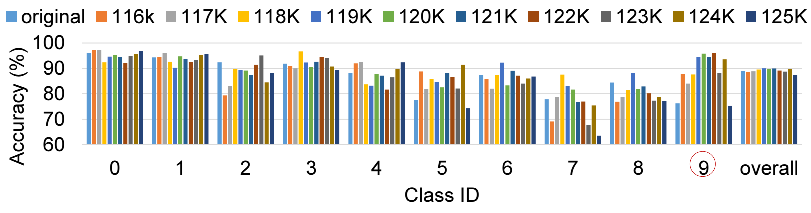
(a) *Retraining-1* (average target accuracy: 91.44%, average overall accuracy: 89.50%)



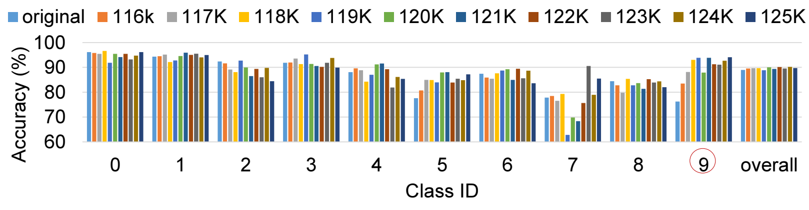
(b) *Retraining-2* (average target accuracy: 85.90%, average overall accuracy: 89.70%)



(c) *Retraining-3* (average target accuracy: 91.53%, average overall accuracy: 89.51%)



(d) *Retraining-4* (average target accuracy: 89.73%, average overall accuracy: 89.18%)



(e) *Retraining-5* (average target accuracy: 90.94%, average overall accuracy: 89.68%)

Figure 6.2: Barplot comparison of class accuracy of five retraining methods (3-layer CNN for SVHN).

요 약 문

심층 신경망을 조정하기 위한 효과적이고 효율적인 방법

오늘날 딥러닝 모델의 훈련은 주어진 훈련 데이터를 이용하여 학습하고 검증 데이터에 대해 모든 클래스들에 대한 평균(overall) 정확도가 높은 결과가 나오는 방향으로 이루어진다. 즉, 특정 클래스들의 정확도가 매우 나쁜 결과가 나오더라도, 클래스 상관없이 최대한 많은 검증 샘플들에 대해 정답을 맞힘으로써 평균 정확도를 향상시킬 수 있는 방향으로 최적화가 진행된다. 저조한 정확도를 가지는 특정 클래스를 개선시키기 위한 추가 훈련을 진행하더라도, 학습이 진행됨에 따라 각 클래스별 정확도의 변동이 매우 크다. 즉 어느 지점에서 훈련을 중단시키더라도 평균 정확도보다 현저히 정확도가 떨어지는 클래스들이 발생한다. 이는 현재의 딥 러닝 기술로서 피할 수 없는 문제이다. 특히 의료 인공지능 시스템 등과 같이 특정 클래스의 정확도가 중요한 응용에서는 이러한 문제가 치명적일 수 있다. 이를 위해서 평균 정확도를 유지하면서 응용에서 중요한 목표(target) 클래스의 정확도를 개선시킬 수 있는 방법이 필요하다.

본 학위논문의 첫 번째 부분에서는 이미 학습된 딥러닝 모델에 대해 추가 학습을 진행하는 것이 아닌 사용자가 개선을 원하는 특정 클래스(이하 목표 클래스)의 정확도를 정밀하게 조정하는 시냅틱 조인(synaptic join) 기술을 제안한다. 제안된 시냅틱 조인은 1) 학습이 완료된 원본 모델에서 목표 클래스의 정확도 개선에 사용될 수 있으면서 동시에 비목표 클래스의 정확도에 영향을 최소화할 수 있는 활성 뉴런(active neurons)들을 찾고, 2) 활성 뉴런들을 목표 클래스를 개선하는 간선(시냅스)의 형태로 연결하는 방식이다. 제안된 방식은 원본 모델을 그대로 유지하면서 시냅스들만 추가 그리고 삭제하면서 사용자의 요구에 따라 모델을 수정할 수 있다. 또한 우리는 시냅틱 조인 연산을 다중 GPU의 한정된 메모리 상에서 빠르게 처리할 수 있는 기술을 소개한다. 재학습 방식과 비교한 실험 결과는 우리의 방법이

특정 클래스의 정확도를 더 잘 컨트롤 할 수 있으며 또한 효과적으로 향상시킬 수 있음을 보여준다.

본 학위논문의 두 번째 부분에서는 시냅틱 조인을 통해 얻은 활성 뉴런을 활용하여 심층 신경망을 증강하는 연구를 제안하였다. 신경망 증강 기술은 신경망 모델을 확장하거나 전이학습을 하기 위해 널리 활용되는 기법 중 하나이다. 이 방식은 이미 학습이 완료된 모델에 새로운 은닉층을 추가하고 미세조정 학습을 하며, 그 결과 더 정확한 모델을 얻거나 응용 프로그램의 목적에 맞는 모델을 얻을 수 있다. 그러나 대규모 심층 신경망 모델에 레이어를 추가하면 모델의 학습에 필요한 매개 변수의 수가 많아져 학습 시간이 늘어날 수 있다. 우리는 효율적인 학습을 위해 적은 수의 활성 뉴런 만 입력 값으로 갖는 증강 네트워크를 제안한다. 또한 일반적인 신경망 증강과 달리 원본 모델의 학습 없이 증강된 모델만 효율적으로 학습하는 방식을 제안한다. 깊이 증강(depth augmented) 방식과 비교했을 때 우리 방식이 모든 실험에서 더 적은 수의 매개 변수와 빠른 학습 속도로 비슷하거나 더 나은 정확도를 얻을 수 있음을 보였다.

요약하여, 본 학위논문은 원본 심층 신경망의 변경없이 모델을 사용자의 목적에 맞게 개선할 수 있는 방법을 제안한다. 원본 모델에서 목표 클래스의 정확도 개선하면서 동시에 비 목표 클래스의 정확도에 영향을 최소화하는 시냅틱 조인 기술을 제안한다. 또한 심층 신경망 증강을 효율적으로 할 수 있는 활성 뉴런을 활용한 심층 신경망 증강 기술을 제안한다. 제안된 방법들은 사용자의 목적에 맞게 효과적이고 효율적으로 신경망의 조정하는 방법으로 맞춤형 인공지능 서비스 응용분야에 매우 유용하게 사용될 수 있을 것으로 사료된다.

핵 심 어 심층 신경망, 시냅틱 조인, 신경망 증강