Optimal Architecture for Deep Neural Networks with Heterogeneous Sensitivity

Hyunjoong Cho, Jinhyeok Jang, Chanhyeok Lee, and Seungjoon Yang

Abstract—This work presents a neural network that consists of nodes with heterogeneous sensitivity. Each node in a network is assigned a variable that determines the sensitivity with which it learns to perform a given task. The network is trained by a constrained optimization that maximizes the sparsity of the sensitivity variables while ensuring the network’s performance. As a result, the network learns to perform a given task using only a small number of sensitive nodes. The L-curve is used to find a regularization parameter for the constrained optimization. To validate our approach, we design networks with optimal architectures for autoregression, object recognition, facial expression recognition, and object detection. In our experiments, the optimal networks designed by the proposed method provide the same or higher performance but with far less computational complexity.

Index Terms—Deep neural networks, optimal architecture, heterogeneous sensitivity, constrained optimization.

I. INTRODUCTION

NEURAL networks often consist of a large number of nodes arranged in deep layers. As researchers have addressed an increasingly wide variety of problems using neural networks, these network architectures have become increasingly complicated. Currently famous and high-performance networks often contain thousands of nodes [1], [2], [3], [4], [5]. One property of neural networks is that nodes share the workload [6]. A network is trained to perform a given task utilizing all the available nodes. Thus, all the nodes provided in a network architecture contribute towards solving a given problem.

However, as neural networks are implemented on platforms with less computational power, and as neural networks are asked to perform more tasks on platforms with massive but still finite computational power, designing a network with less computational complexity has become an important issue [7], [8]. Numerous studies have been conducted to determine how to prune network nodes and obtain a computationally efficient network [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. In these approaches, the networks are first trained with a large amount of nodes; then, the importance of each node is evaluated by analyzing the node weight using various measures. Finally, the nodes with less importance are removed from the trained network. However, because all the nodes in trained network share the workload, removing a node from a trained network—even if its node weight indicates that is has lesser importance, will degrade the network’s performance. Consequently, pruning approaches are usually followed by retraining the pruned network to recover the performance losses from pruning.

In this paper, we propose a network that consists of nodes with heterogeneous sensitivity. Each node in a network is assigned a variable that determines its sensitivity to a given task. Then, the network learns to perform the task by relying more on the sensitive nodes and less on the insensitive nodes. In extreme cases where the sensitivity of a node is zero, the network does not utilize the node at all; it is essentially disconnected. Node sensitivity is learned during training through a constrained optimization. The sparsity of the sensitivity is maximized while the network performance is constrained within a certain range. By using the ℓ1 norm as a sparsity measure, many of the sensitivity variables obtained by the optimization have zero values. Hence, the network learns to perform a given task using only a small number of sensitive nodes. Finally, by simply removing the nodes with zero sensitivity, the optimal architecture of a deep network for a given task is achieved. The constrained optimization to simultaneously design the optimal architecture during training requires a regularization parameter used to weight the network’s performance and adjust the sparsity of the sensitivity. We find this parameter using the L-curve, which has previously been used for restoration problems [19], [20].

We assign sensitivity to nodes in a network by introducing a layer we call a sensitivity layer. The sensitivity layer can be implemented as a special type of dense or convolutional layer. Then, a network that includes these new sensitivity layers can be implemented and trained using functions available in standard deep learning packages [21], [22], [23]. Our approach does not require any special optimization routine to solve the constrained optimization problem that we designed to enforce the sparsity of the sensitivity variables while training.

We applied these networks with heterogeneous sensitivity to design a simple autoencoder and a deep convolutional neural network (CNN) for analysis. For autoregression with an autoencoder, the optimal network architecture is found using the MNIST dataset [24]. For object recognition with a CNN, we used the CIFAR-10 dataset [25]. The effects of the regularization parameters on the network’s performance and architecture are analyzed through the L-curve. Next, we applied the sensitivity layers to famous networks transferred to solve simpler problems. Because the original networks are
designed and trained to solve more complicated problems, the network architecture is usually excessive for the simpler problems to which they are transferred. The networks considered here are VGG \cite{simonyan14c} and ResNet \cite{he15} for facial expression recognition using the CK+ dataset \cite{koestinger11} and YOLO \cite{redmon16} for object detection using the VOC dataset \cite{everingham10}. The optimal architectures found by the proposed method are compared to those found by pruning approaches reported by previous studies. By introducing nodes with heterogeneous sensitivity to the networks and enforcing the sparsity of the sensitivity, we were able to design networks that consist of notably fewer nodes but that exhibit the same or even better performance. The proposed method can be used to design an efficient network containing the optimal number of nodes.

The rest of this paper is organized as follows. We introduce networks consisting of nodes with heterogeneous sensitivity in Section II-A and address their implementation and training in Section II-B. Network operation in an autoregressive setting is analyzed in Section II-C. Comparisons with other approaches are given in Section II-D. We present the experimental results and discussions for the autoencoder in Section III-A, for the deep networks in Section III-B and for the transferred networks in Section III-C. Section IV concludes the paper.

II. Optimal Architecture for Deep Neural Networks

A. Neural Networks with Heterogeneous Sensitivity

Consider a network whose lth layer consists of the following operations. The intermediate output $u^l_i$ is computed by either a dense layer,

$$ u^l_i = \sum_{j=1}^{n_{l-1}} W^l_{ij} x^{l-1}_j, $$

or by a convolutional layer,

$$ u^l_i = \text{conv}(W^l_{ij}, x^{l-1}), $$

where $u^l_i$ is the i
th intermediate output node, $x^{l-1}_j$ and $x^{l-1}$ are the jth nodes and the node volume in the (l − 1)th layers, respectively. The network parameters $W^l_{ij}$ and $W^l_i$ denote a weight in the dense layer and a filter in the convolutional layer, respectively. The number of nodes in the lth layer are denoted as $n_l$. The intermediate output is activated by an activation function:

$$ u^l_i = f^l(u^l_i). $$

The activated output $v^l_i$ is weighted by a newly introduced layer:

$$ x^l_i = s^l_i v^l_i, $$

where $s^l_i \geq 0$ is a variable that determines the sensitivity of the ith node in the lth layer $x^l_i$. We denote this new layer as a sensitivity layer.

The sensitivity variable $s^l_i$ in the sensitivity layer allows us to apply heterogeneous sensitivity to the nodes in a network. Consider a network trained to minimize a cost function $E$ using backpropagation \cite{rumelhart86, lecun06}. The weight matrix $W_l$, whose element is the weight $W^l_{ij}$ of the lth layer, is updated by

$$ W_l \leftarrow W_l - \eta \delta_i x^l_{i-1}, \quad (5) $$

where $\eta$ is the step size and $x^l_{i-1}$ is a vector whose element is $x^{l-1}_i$. In the lth layer, each element of the sensitivity vector $\delta_i$ is

$$ \delta^l_i = s^l_i \sum_{k=1}^{n_{l+1}} \delta^{l+1}_k W^{l+1}_{kj} \frac{f^{l}(u^l)}{\partial u^l_i}. $$

The variable $s^l_i$ is a weight that reflects the sensitivity of a node. After the weights are updated, nodes with larger values of $s^l_i$ will respond more sensitively than those with smaller $s^l_i$ values. In the extreme case, when $s^l_i = 0$, the node is completely insensitive. These zero-sensitivity nodes can be regarded as disconnected nodes.

Node sensitivity (the sensitivity variable $s^l_i$) can be adjusted so that a network learns to perform a given task using only a small number of sensitive nodes. To accomplish this, we designed an optimization problem to train the network:

$$ \text{minimize} \sum_{l=1}^{L} \|s_l\|_1 $$

subject to \quad $E < \epsilon,$

where $s_l$ is a vector whose element is $s^l_i$ and $E$ is a deviation penalty that measures the deviations of network outputs from the ground truth values. The cost function, which is the sum of the $\ell_1$ norm of $s_l$, makes the vector $s_l$ sparse, so that the network uses only a few sensitive nodes and includes as many disconnected nodes as possible. The constraint on $E$ guarantees the performance of the network within the
threshold $\epsilon$. The deviation penalty $E$ is a typical cost function used to train a network. For example, $E$ can be

$$ E = \sum_{m} \| y^m - \sum_{l=1}^{L} s_l \odot f^l(W_l x^m_l) \|_2, \quad (8) $$

where $(x^m_0, y^m)$ are the input and ground truth pairs in a training set, and $\odot$ is the Hadamard (element by element) product. The last layer of a network is usually determined by a specific task. We set the sensitivity variable for the last layer $s^L$ to one.

The optimization problem can be rewritten as follows:

$$ \min_{s_1, s_2, \ldots, s_L, W_1, W_2, \ldots, W_L} E + \lambda \sum_{l=1}^{L} \| s^l \|_1, \quad (9) $$

where the regularization parameter $\lambda$ weighs the deviation penalty $E$ and the sparsity penalty $\sum \| s^l \|_1$. When $\lambda$ is large, the sparsity penalty dominates the cost function in (9). The trained network will consist of a small number of sensitive nodes with as many insensitive (i.e., disconnected) nodes as possible. However, because the deviation penalty $E$ is neglected during the training, the network will fail to provide accurate outputs. In contrast, when $\lambda$ is small, the deviation penalty dominates the cost function, and the network will be trained to provide accurate outputs but will utilize most of the available nodes. Thus few insensitive (i.e., disconnected) nodes will exist. Ideally, the goal is to find a $\lambda$ that balances the deviation and sparsity penalties.

An L-curve is a plot of the two penalties for various values of $\lambda$. The L-curve has previously been used in restoration problems [19], [20]. In our problem, the L-curve shows the deviation penalty vs. the sparsity penalty for various values of $\lambda$. One region of $\lambda$ values provides solutions that are dominated by the deviation penalty, and another region of $\lambda$ values provides solutions that are dominated by the sparsity penalty. The plot generally has an L-shape, the corner of which provides a $\lambda$ value that balances the two penalties. We use the L-curve to determine the best value of the regularization parameter $\lambda$ for the optimization problem in (9).

### B. Sensitivity Layer Implementation

The sensitivity layer can be regarded as a special type of dense or a convolutional layer. A sensitivity layer added after a dense layer with $n_1$ nodes can be regarded as a set of $n_1$ (dense) layers with one input node, one output node, and one weight. A sensitivity layer added after a convolutional layer with $n_1$ nodes can be regarded as a set of $n_1$ convolutional layers with one input node, one output node, and a single one-by-one filter. Hence, the sensitivity layer can be implemented using layer definitions and functions already available in deep learning packages [21], [22], [23].

Moreover, training a network that includes sensitivity layers can be accomplished with various training methods already available in deep learning packages. The weights in the dense or convolutional layers are updated by

$$ W^l_{ij} \leftarrow W^l_{ij} - \eta \frac{\partial E}{\partial W^l_{ij}}, \quad (10) $$

and the parameters in the sensitivity layers are updated by

$$ s^l_i \leftarrow s^l_i - \eta \frac{\partial E}{\partial s^l_i} - \eta \lambda \frac{s^l_i}{|s^l_i|}, \quad (11) $$

These approaches can be implemented as the training for either dense or convolutional layers under $\ell_1$ regularization. Hence, by regarding the sensitivity layers as simply special cases of dense or convolutional layers, we can implement and train a network with the sensitivity layers using standard deep learning packages. Our approach does not require any special optimization routines to solve the constrained optimization problem.

### C. Analysis of Neural Networks with Sensitivity Layers

To understand how a network with sparse sensitivity variables is trained to perform a given task, consider a simple network in an autoregression setting. As an example, we use a network with two dense layers and linear activation. The sensitivity layer is implemented in the first layer. Then, network operation can be written as follows:

$$ y = Ax \quad (12) $$

$$ = W_2 S_1 W_1 x \quad (13) $$

$$ = \sum_{k=1}^{n_1} s^1_k w^2_k (w^1_k)^T x, \quad (14) $$

where $W_1$ and $W_2$ are the weight matrices. The matrix $S_1$ is a diagonal matrix whose diagonal elements are the sensitivity variables in $s^1$. The vectors $w^2_k$ and $(w^1_k)^T$ are the $k$th column and row of $W_2$ and $W_1$, respectively. The network operation is written as a linear combination of $w^2_k (w^1_k)^T$. The weights for the linear combination are given by the sensitivity variable $s^1_k$. The contribution of $w^2_k (w^1_k)^T$ with small $s^1_k$ values to the operation of the network is small, and vice versa.

We find the columns of the matrices $W_1$, $W_2$ and the diagonal matrix $S_1$ through the optimization problem in (7). The sensitivity variable $s^1$ obtained by the optimization will have many zero elements because sparsity is enforced. Without loss of generality, let the elements of $s^1$ be sorted such that the $(K+1)$th to $n_1$th elements are zero. The contribution of $w^2_k (w^1_k)^T$ with $s^1_k = 0$ is zero. Hence, we can remove those terms from the linear combination in (14). Then, we approximate the network’s operation by

$$ A \approx \sum_{k=1}^{K} s^1_k w^2_k (w^1_k)^T. \quad (15) $$

Principal component analysis (PCA) is a method that represents the input using only a small number of principal components [30]. The proposed network with heterogeneous sensitivity trained via the optimization problem in (7) operates similarly to PCA—it represents the inputs through the small number of sensitive, or principal, nodes.

### D. Comparison with Pruning Methods

Previous studies have investigated how to design compact and efficient networks to allow deep networks to be deployed...
on devices with restricted computational capabilities. A survey of efficient deep network design methods can be found in [7], [8]. Many approaches can prune nodes with high computational complexity and memory requirements to obtain a more efficient network. In these pruning approaches, a network is first trained, and then the importance of each node is evaluated with various measures. Finally, the less nodes are pruned from the trained network. The issue of how the cost function used for the training changes with a small weight perturbations was analyzed in [9], [10], [11], [12]. For example, the Hessian of a cost function provides information on how small weight changes affect the cost. Connections in a trained network that induce insignificant changes in the cost function were removed from the network. In [13], [14], [15], [16], [17], the importance of weights in a trained network was evaluated using the $l_2$, $l_1$, and $l_0$ norms of the node weights. Then, the less important connections between nodes were removed from the trained network based on the evaluated measures. To encourage a network to have node weights that result in smaller measures, regularization by the $l_2$, $l_1$, and $l_0$ norms of the node weights is used during training.

Many pruning approaches use regularization as a function of node weights. As a result some connections in dense layers and some filter coefficients in convolutional layers have small values. Removing a node from a network entirely is not straightforward for dense layers and is difficult for convolutional layers. Examples of pruning with removed connections and filter coefficients are shown in Fig. 2 (a). The network is trained to perform a task utilizing all the available nodes. Because nodes in a network share the workload [6], removing a node—even one with a smaller measure of importance—from a trained network will degrade the network’s performance. Consequently, pruning approaches typically retrain the pruned network to recover the performance loss from pruning.

The proposed method uses regularization as a function of the sensitivity. As a result, removing zero-sensitivity nodes is straightforward because they are already disconnected by the end of the training. Examples of disconnected nodes for dense and convolutional layers are shown in Fig. 2 (b). The network is trained to perform a task utilizing only the sensitive nodes; therefore, removing zero-sensitivity nodes has no effect on the network’s performance because the network has already been trained to perform the task without them.

In [13], networks with activation functions with nodewise variant slopes were introduced. Using this approach, the nodes with steeper slopes learn more important features, and vice versa. After training, the nodes with lower slopes are pruned from the trained network. The assignment of nodewise variant slopes to activation functions plays a role similar to that of the sensitivity layers presented in this paper; however, that study used a predefined set of values for the slopes regardless of the data. Because a predefined set of slopes does not reflect the actual data statistics, workload sharing still exists; hence, a performance loss occurs after pruning. Our proposed method can be viewed as an improvement of the work in [13] in which the slopes of the activation functions are learned from the data statistics by solving an optimization problem during the training.

In [31], a layer similar to the sensitivity layer in our proposed network was introduced to a trained network. The variables in the added layer were then used for pruning. An optimization problem was constructed to determine which nodes could be removed from the trained network while still ensuring the network’s performance. Again, however, because nodes in a trained network share the workload, node pruning—even when done through an optimization approach, degrades the network’s performance. In contrast, the variables in the sensitivity layers in our approach are found during network training, thus avoiding the need for further pruning.

In [32], a group lasso of node weights is used as a regularization factor. The group lasso enforces groupwise sparsity. By defining all the coefficients in a filter as a group, a node can be effectively removed from a convolutional layer. By defining only a part of the filter coefficients as a group, filters with different support levels can be used in a convolutional layer. Consequently, different network architectures can be designed by defining different groups. Such a design requires multiple regularization parameters; however, the study did not address how to choose the regularization parameters to obtain the optimal architecture. In contrast, our approach uses regularization as a function of node sensitivity, which allows us to simply disconnect a node in both dense and convolutional layers. The regularization parameter is chosen using an L-curve to help find the optimal architecture.

III. EXPERIMENTS AND DISCUSSIONS

A. Autoencoder

Here, we analyze a simple network with heterogeneous sensitivity. We prepared a network with 784 hidden nodes in an auto-associative setting to reconstruct the inputs. The ReLU function is used as the activation function for all the nodes. The sensitivity layer is added after the activation functions and implemented as a collection of 784 individual dense layers, each of which has one input, one output, and one weight. As explained in Section II-B, by implementing the sensitivity layer as a special type of dense layer, we can
implement and train the network using standard functions in deep learning packages. Here, we used the Keras Python deep learning library for implementation and training. The sensitivity variables are initialized to one. We adopted the $\ell_1$ regularization in the sensitivity layer as a training option and trained the network using the MNIST dataset [24].

Optimizing the proposed networks requires the regularization parameter $\lambda$ that weights the deviation and sparsity penalties. We found the appropriate $\lambda$ value using the L-curve. Fig. 3 shows the L-curve for the network using the MNIST dataset, plotted using the deviation penalty $E$ vs. the sparsity $\sum_i \|s_i\|_1$ at different $\lambda$ values. For small values of $\lambda$, for example $\lambda = 1.0 \times 10^{-5}$, the deviation penalty dominates the cost function of the unconstrained optimization problem in (9). Then, the solution to the optimization problem provides only a small deviation penalty but a large sparsity penalty. In contrast, for large values of $\lambda$, for example $\lambda = 1.0 \times 10^{-1}$, the sparsity penalty dominates the cost function, providing only a small sparsity penalty but a large deviation penalty. The balance of the two penalties can be achieved using the value from the corner of the L-curve. We used a $\lambda$ value of $1.0 \times 10^{-3}$ for the optimization, which corresponds to the corner of the L-curve.

Fig. 3. L-curve for autoencoder using MNIST dataset, showing $E$ vs. $\sum_i \|s_i\|_1$ for different values of $\lambda$; red: $\lambda = 1.0 \times 10^{-5}$; blue: $\lambda = 1.0 \times 10^{-3}$; green: $\lambda = 1.0 \times 10^{-1}$.

Fig. 4 shows examples of node sensitivity in networks trained with different values of the regularization parameter $\lambda$. The sensitivity variable $s_i$ are shown, in the decreasing order, for $\lambda = 1.0 \times 10^{-5}, 1.0 \times 10^{-4}, 1.0 \times 10^{-3},$ and $1.0 \times 10^{-2}$ in Fig. 4 (a), (b), (c), and (d), respectively. As the $\lambda$ value increases, there are more nodes with small sensitivity. When $\lambda = 1.0 \times 10^{-2}$, many nodes can be removed from the network. However, the trained network fails to provide acceptable performance at this $\lambda$ value. In contrast, a network trained with $\lambda = 1.0 \times 10^{-3}$ shown in (c) corresponds to the $\lambda$ value at the corner of the L-curve. At this setting, the network provides an acceptable deviation penalty yet has as many zero sensitivity nodes as possible. In effect, the nodes with zero sensitivity are disconnected. The optimal network architecture of the autoencoder with one hidden layer for the MNIST dataset is to have 75 hidden nodes.

Fig. 4. Node sensitivity for an autoencoder trained with different $\lambda$ values using the MNIST dataset: (a) $\lambda = 1.0 \times 10^{-5}$; (b) $\lambda = 1.0 \times 10^{-4}$; (c) $\lambda = 1.0 \times 10^{-3}$; and (d) $\lambda = 1.0 \times 10^{-2}$.

with $\lambda = 1.0 \times 10^{-3}$. This network has 75 nodes. When all the 75 nodes are used to reconstruct the images, the MSE is lower than one would achieve by reconstructing the images using the 75 principal components. However, when fewer than 75 nodes are used, the MSE values degrade faster than the PCA results. The average MSE values between the inputs and their reconstructions for the network trained with $\lambda = 6.0 \times 10^{-3}$ are shown in green line. This network has 46 nodes. When all the 46 nodes are used to reconstruct the images, the MSE is lower than one would achieve by reconstructing the images using the 46 principal components. This observation suggests that a network with a fewer number of nodes should be designed by training a network using a higher regularization parameter value rather than removing nodes from a trained network.

Fig. 5 shows the result of PCA on the images in the MNIST dataset. The average mean square error (MSE) values between the inputs and their reconstructions using the $k$ principal components are shown in blue line. The average MSE values between the inputs and their reconstructions using the $k$ most sensitive nodes are shown in red line for the network trained with $\lambda = 1.0 \times 10^{-3}$. This network has 75 nodes. When all the 75 nodes are used to reconstruct the images, the MSE is lower than one would achieve by reconstructing the images using the 75 principal components. However, when fewer than 75 nodes are used, the MSE values degrade faster than the PCA results. The average MSE values between the inputs and their reconstructions for the network trained with $\lambda = 6.0 \times 10^{-3}$ are shown in green line. This network has 46 nodes. When all the 46 nodes are used to reconstruct the images, the MSE is lower than one would achieve by reconstructing the images using the 46 principal components. This observation suggests that a network with a fewer number of nodes should be designed by training a network using a higher regularization parameter value rather than removing nodes from a trained network.

Fig. 5. PCA of images in MNIST dataset, average MSE of reconstructed images; red: when the first $k$ principal components are used; blue: when the first $k$ nodes in the network with heterogeneous sensitivity are used ($\lambda = 1.0 \times 10^{-3}$); green: when the first $k$ nodes in the network with heterogeneous sensitivity are used ($\lambda = 6.0 \times 10^{-3}$).
B. Deep Networks

1) Deep CNN: In this experiment, we consider a deep CNN with heterogeneous sensitivity for object recognition using the CIFAR-10 dataset [25]. We prepared a network with four convolutional layers and two dense layers, adding sensitivity layers to all the convolutional layers and the first dense layer. The second dense layer is designed to perform object recognition. The sensitivity layers for the convolutional layers are implemented as a collection of convolutional layers, each of which has one input node and one output node with a one-by-one filter. The sensitivity layers for the dense layers are implemented as a collection of dense layers, each of which has one input node, one output node, and one weight. We applied $\ell_1$ regularization in the sensitivity layers for training. For comparison, we trained a baseline CNN with the same number of nodes and layers and with the ReLU activation function using the same training set.

Fig. 6 shows the L-curve for the CNN. The corner of the L-curve corresponds to the $\lambda$ value of $6.0 \times 10^{-4}$. Fig. 7 shows the sensitivity of the nodes, in decreasing order, in each layer. The sensitivity variables $s_i$ are sparse in all the layers and have many zero elements. Only the nodes with non-zero sensitivity need to be included in the optimal architecture. We used a thresholding approach to remove the nodes with sensitivity values numerically close to zero.

The number of nodes included in each layer in the optimal architecture is listed in Table I. The sensitivity approach reduced the total number of nodes from 1,290 to 611. In other words, compared to the baseline CNN, the network with heterogeneous sensitivity learned to perform its task using only 47.36% of the original nodes. Table I shows the computational complexity of the networks in terms of the number of weights. The optimal CNN uses only 25.95% of the total weights in the baseline CNN.

The accuracies of the baseline and optimal CNNs are listed in Table III. By using nodes with heterogeneous sensitivity, the optimized CNN achieves the same performance object recognition performance as the baseline CNN using only 47.36% of the nodes, or 25.95% of the weights. Thus, the optimal CNN is trained to perform using an optimal number of nodes. Note that the design of the optimal network architecture is acquired simultaneously with network training.

2) Comparisons with other methods: We compared the complexity and performance of an optimal network found by the proposed method to the pruning results reported in [13], [17], [33], [34], [35], [18]. The LeNet-5 [11] network with heterogeneous sensitivity was prepared and trained using the MNIST dataset. The ratios of the weights remaining after the optimization and the classification errors are reported in Table II.

![Fig. 6. L-curve for CNN with CIFAR-10 data, showing $E$ vs. $\sum_{i=1}^{L} ||s_i||_1$ at different values of $\lambda$.](image)

![Fig. 7. Sensitivity variables for the optimal CNN with CFAR10 data: (a) 1st conv layer; (b) 2nd conv layer; (c) 3rd conv layer; (d) 4th conv layer; and (e) 1st dense layer.](image)

### TABLE I

| Layer | # of nodes | # of weights |
|-------|------------|--------------|
| 1 conv | 128        | 42           |
| 2 conv | 128        | 41           |
| 3 conv | 256        | 121          |
| 4 conv | 256        | 141          |
| 5 dense | 512        | 256          |
| 6 dense | 10         | 10           |
| overall | 1290       | 611          |

| Layer | # of weights |
|-------|--------------|
| 1 conv | 3584        |
| 2 conv | 147584      |
| 3 conv | 295168      |
| 4 conv | 590080      |
| 5 dense | 3276800     |
| 6 dense | 5120        |
| overall | 4318336     |

![Fig. 6. L-curve for CNN with CIFAR-10 data, showing $E$ vs. $\sum_{i=1}^{L} ||s_i||_1$ at different values of $\lambda$.](image)

![Fig. 7. Sensitivity variables for the optimal CNN with CFAR10 data: (a) 1st conv layer; (b) 2nd conv layer; (c) 3rd conv layer; (d) 4th conv layer; and (e) 1st dense layer.](image)

### TABLE II

| Layer     | # of weights |
|-----------|--------------|
| 1 conv    | 2322         |
| 2 conv    | 15498        |
| 3 conv    | 44649        |
| 4 conv    | 135349       |
| 5 dense   | 902259       |
| 6 dense   | 2555         |
| overall   | 1120832      |

| Layer | # of weights |
|-------|--------------|
| 1 conv | 3584        |
| 2 conv | 147584      |
| 3 conv | 295168      |
| 4 conv | 590080      |
| 5 dense | 3276800     |
| 6 dense | 5120        |
| overall | 4318336     |

![Fig. 6. L-curve for CNN with CIFAR-10 data, showing $E$ vs. $\sum_{i=1}^{L} ||s_i||_1$ at different values of $\lambda$.](image)

![Fig. 7. Sensitivity variables for the optimal CNN with CFAR10 data: (a) 1st conv layer; (b) 2nd conv layer; (c) 3rd conv layer; (d) 4th conv layer; and (e) 1st dense layer.](image)
TABLE III
PERFORMANCES OF OPTIMAL CNN WITH CIFAR-10 DATA

| ratio   | accuracy |
|---------|----------|
| baseline| 82.95%   |
| optimal | 25.95%   |

The optimal network designed by the proposed method provides the highest performance with the least computational complexity.

TABLE IV
COMPARISON TO PRUNING METHODS REPORTED WITH MNIST DATASET

| Method | Network | cl   | Ratio  | Error |
|--------|---------|------|--------|-------|
| [13]   | LeNet-5 | clf  | 8.24%  | 0.80% |
| [17]   | LeNet-5 | (modified) | 10.25% | 0.87% |
| [33]   | LeNet-5 | 9.01% | 0.71%  |
| [34]   | LeNet-5 | conv layers only | 8.33% | N/A |
| [35]   | LeNet-5 | dense layer only | 16.00% | 0.94% |
| [35]   | LeNet-5 | dense layer only | 12.00% | 0.94% |
| [17]   | LeNet-5 | 6.73% | 0.81%  |
| proposed| LeNet-5 | 6.40% | 0.69%  |

C. Transferred Deep Network

Famous networks can be transferred to form the basis for the design of a network intended for another task. A network that shares its architecture with a famous network is first prepared, initialized with the available weights, and then fine tuned with a new training set prepared for the given task. Because famous networks are usually prepared for more complicated problems, their network architecture is usually excessive for the simpler problems to which they are transferred. Although weight transfer is a well-established design method to achieve good network performance, the end result can be an unwieldy large network architecture that is computationally too expensive for a given task.

In this section, we first prepared transferred networks and then added the sensitivity layers to find the optimal architectures for smaller tasks. We tested both VGG [1] and ResNet [4] for facial expression recognition using the CK+ dataset [20] and tested YOLO [5] for object detection with smaller object classes using the VOC dataset [27]. We provide the results of comparisons to pruning methods previously applied to transferred networks using VGG and ResNet with the CIFAR-10 dataset.

1) VGG and ResNet: The VGG-16 and ResNet-50 networks with heterogeneous sensitivity were prepared by adding the sensitivity layers after all the convolutional and dense layers. The networks were trained using the CK+ facial recognition dataset. We determined the regularization parameter $\lambda$ using the L-curve as described earlier.

Figs. 8 and 9 show the node sensitivities in each layer of VGG and ResNet, respectively. We included only the nodes with non-zero sensitivity in the optimal architecture. The number of nodes in each layer in the optimal VGG and ResNet networks are listed in Tables V and VI respectively. The optimized networks reduced the number of nodes to only 40.94% and 13.00% of the baseline VGG and ResNet models, respectively. The computational complexities in terms of the numbers of weights are reported in Table VII and VIII for the VGG and ResNet networks, respectively. The optimized networks reduced the numbers of weights to only 9.90% and 2.31% of the baseline models for VGG and ResNet, respectively.

The accuracies of the baseline and optimal VGG and ResNet models are listed in Table IX. By using nodes with heterogeneous sensitivity, VGG and ResNet learn to recognize facial expressions at the same accuracy but use only an optimal number of nodes.

2) YOLO: We prepared a YOLO network with heterogeneous sensitivity by adding sensitivity layers after all the convolutional and dense layers. We simplified the object detection task by considering only three object classes: pedestrian, biker, and people. The network was trained using the VOC dataset. The regularization parameter $\lambda$ was found using the L-curve as described previously.

Fig. 10 shows the node sensitivity in each layer of the YOLO. Only the nodes with non-zero sensitivity are included in the optimal architecture. The number of nodes in each layer

Fig. 8. Sensitivity variables for optimal VGG16 with CK+ Dataset

TABLE V
OPTIMAL NETWORK ARCHITECTURE OF VGG-16 WITH CK+ DATASET

| Layer | Baseline | Optimal | Ratio |
|-------|----------|---------|-------|
| 1 conv | 64 | 41 | 64.06% |
| 2 conv | 64 | 48 | 75.00% |
| 3 conv | 128 | 88 | 68.75% |
| 4 conv | 128 | 100 | 78.12% |
| 5 conv | 256 | 174 | 67.96% |
| 6 conv | 256 | 141 | 55.07% |
| 7 conv | 256 | 147 | 57.42% |
| 8 conv | 512 | 168 | 32.81% |
| 9 conv | 512 | 138 | 26.95% |
| 10 conv | 512 | 89 | 17.38% |
| 11 conv | 512 | 110 | 21.48% |
| 12 conv | 512 | 99 | 19.33% |
| 13 conv | 512 | 99 | 19.33% |
| 14 dense | 512 | 195 | 38.08% |
| 15 dense | 512 | 512 | 100.00% |
| overall | 5248 | 2149 | 40.94% |
in the optimal YOLO is given in Table \textbf{X}. The total number of nodes was reduced to 56.69\% of the baseline network. The computational complexity in terms of the number of weights is reported in Table \textbf{XI}. The number of weights were reduced to 26.18\% of the baseline.

The accuracies of the baseline and optimal VGG and ResNet networks with regard to mAP scores are listed in Table \textbf{XII}. By using nodes with heterogeneous sensitivity, the YOLO learns to detect objects with the same accuracy as the baseline but uses only an optimal number of nodes.

3) \textbf{Comparisons with Other Methods:} We also compared the complexity and performance of the optimal networks with the pruning results reported in [16], [23], [18]. The VGG-16, ResNet-56, ResNet-20 networks with heterogeneous sensitivity were prepared using the CIFAR-10 dataset. The ratios of the weights remaining after the optimization and the classification errors are reported in Table \textbf{XIII}. The optimal networks designed by the proposed method provide the same or higher performance but with less computational complexity.

\section*{IV. Conclusion}

In this study, we trained networks consisting of nodes with heterogeneous sensitivity to perform a given task using only a small number of sensitive nodes. The training is formulated as a constrained optimization problem whose parameter is found using the L-curve. By introducing sensitivity layers that assign sensitivity variables to nodes, we were able to
TABLE VII
Complexity of Optimal VGG-16 with CK+ Dataset.

| Layer | # of weights Baseline | Optimal | Ratio |
|-------|----------------------|---------|-------|
| 1 conv | 1792                | 1107    | 61.77%|
| 2 conv | 36928               | 17112   | 47.96%|
| 3 conv | 73856               | 38016   | 51.47%|
| 4 conv | 147584              | 79200   | 53.66%|
| 5 conv | 295168              | 156600  | 53.05%|
| 6 conv | 590080              | 220806  | 37.41%|
| 7 conv | 590080              | 186543  | 31.61%|
| 8 conv | 1180160             | 222264  | 18.83%|
| 9 conv | 2359808             | 208656  | 8.84% |
| 10 conv | 2359808            | 110538  | 4.68% |
| 11 conv | 2359808            | 88110   | 3.73% |
| 12 conv | 2359808            | 98010   | 4.15% |
| 13 conv | 2359808            | 88209   | 3.73% |
| dense | 2359808             | 173745  | 7.36% |
| dense | 3591                | 1365    | 38.01%|

overall | 17078087 | 1690881 | 9.90% |

Fig. 10. Sensitivity variables for optimal YOLO with the VOC dataset.

Implement and train a network without using a complicated optimization tool. The networks trained in this manner possess an optimal network architecture and simultaneously meet the performance criteria. In our experiments, the optimal networks designed by the proposed method provide the same or higher performance but with far less computational complexity. The proposed method can be used to determine the optimal network architectures of deep networks.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, pp. 1097–1105.

[2] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” arXiv preprint arXiv:1312.6199, 2013.

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

[4] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in Proceedings of the IEEE

TABLE VIII
Complexity of optimal ResNet-50 with the CK+ dataset.

| Layer | # of weights Baseline | Optimal | Ratio |
|-------|----------------------|---------|-------|
| 1 conv | 93472               | 7044    | 80.70%|
| 2 conv | 4160                | 3224    | 77.50%|
| 3 conv | 36928               | 31248   | 84.61%|
| 4 conv | 16640               | 7504    | 45.09%|
| 5 conv | 16640               | 6968    | 41.87%|
| 6 conv | 16448               | 8576    | 52.14%|
| 7 conv | 36928               | 36288   | 98.26%|
| 8 conv | 16640               | 8190    | 49.21%|
| 9 conv | 16448               | 8190    | 49.79%|
| 10 conv | 36928              | 36288   | 98.26%|
| 11 conv | 16640              | 6976    | 41.92%|
| 12 conv | 32896              | 9592    | 29.15%|
| 13 conv | 147584             | 70488   | 47.76%|
| 14 conv | 66048              | 10324   | 15.63%|
| 15 conv | 131584             | 12644   | 9.60% |
| 16 conv | 65664              | 12180   | 18.54%|
| 17 conv | 147584             | 92610   | 62.75%|
| 18 conv | 66048              | 7350    | 11.12%|
| 19 conv | 65664              | 4800    | 7.30% |
| 20 conv | 147584             | 46656   | 31.61%|
| 21 conv | 66048              | 6399    | 9.68% |
| 22 conv | 65664              | 7663    | 11.67%|
| 23 conv | 147584             | 70713   | 47.91%|
| 24 conv | 66048              | 5508    | 8.33% |
| 25 conv | 131328             | 4284    | 3.26% |
| 26 conv | 590080             | 35154   | 5.95% |
| 27 conv | 263168             | 4154    | 1.57% |
| 28 conv | 525312             | 4556    | 0.86% |
| 29 conv | 262400             | 4958    | 1.88% |
| 30 conv | 590080             | 48618   | 8.23% |
| 31 conv | 263168             | 3066    | 1.16% |
| 32 conv | 262400             | 2226    | 0.84% |
| 33 conv | 590080             | 23850   | 4.04% |
| 34 conv | 263168             | 1100    | 0.41% |
| 35 conv | 262400             | 792     | 0.30% |
| 36 conv | 590080             | 6804    | 1.15% |
| 37 conv | 263168             | 315     | 0.11% |
| 38 conv | 262400             | 375     | 0.14% |
| 39 conv | 590080             | 4950    | 0.83% |
| 40 conv | 263168             | 220     | 0.08% |
| 41 conv | 262400             | 210     | 0.08% |
| 42 conv | 590080             | 3969    | 0.67% |
| 43 conv | 263168             | 231     | 0.08% |
| 44 conv | 524800             | 165     | 0.03% |
| 45 conv | 2359808            | 1485    | 0.06% |
| 46 conv | 1050624            | 220     | 0.02% |
| 47 conv | 2009200            | 220     | 0.01% |
| 48 conv | 1049088            | 560     | 0.05% |
| 49 conv | 2359808            | 5544    | 0.23% |
| 50 conv | 1050624            | 418     | 0.03% |
| 51 conv | 1049088            | 380     | 0.03% |
| 52 conv | 2359808            | 6120    | 0.25% |
| 53 conv | 1050624            | 1972    | 0.18% |
| 54 conv | 8388864            | 52896   | 0.63% |
| 55 conv | 1799               | 399     | 22.17%|

overall | 31872135 | 738234 | 2.31% |

TABLE IX
Performances of optimal VGG-16 and ResNet50 with the CK+ dataset.

| Network | baseline | optimal |
|---------|----------|---------|
| VGG-16  | 96.65%   | 97.39%  |
| ResNet-50 | 96.77%  | 96.91%  |
Hyunjoong Cho received a B. S. degree in 2015 and an M.S. degree in 2017 from the Ulsan National Institute of Science and Technology, Ulsan, South Korea, where he is currently pursuing a doctor’s degree in electrical and computer engineering. His research interests include image processing, human perception, and human recognition.

Jinhyeok Jang received a B.S. degree in 2014 and an M.S. degree in 2016 from the School of Electrical and Computer Engineering of the Ulsan National Institute of Science and Technology, Ulsan, South Korea. He currently works at the Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea. His research interests include image processing, blur estimation, human facial recognition, and human action recognition.

Chanhyeok Lee received a B.S. degree from the Ulsan National Institute of Science and Technology, Ulsan, South Korea, where he is currently pursuing a master’s degree in electrical and computer engineering. His research interests include image processing, image classification, and object detection.

Seungjoon Yang (S’99-M’00) received a B.S. degree from Seoul National University, Seoul, Korea in 1990 and M.S. and Ph.D. degrees from the University of Wisconsin-Madison in 1993 and 2000, respectively, all in electrical engineering. He worked at the Digital Media R&D Center at Samsung Electronics Co., Ltd. from September 2000 to August 2008 and is currently with the School of Electrical and Computer Engineering at the Ulsan National Institute of Science and Technology in Ulsan, Korea. His research interests include image processing, estimation theory, and multi-rate systems. Professor Yang received the Samsung Award for the Best Technology Achievement of the Year in 2008 for his work on the premium digital television platform project.