

Received 1 March 2024, accepted 3 March 2024, date of publication 4 March 2024, date of current version 13 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3373692

RESEARCH ARTICLE

Efficient Ensemble via Rotation-Based Self-Supervised Learning Technique and Multi-Input Multi-Output Network

JAEHOON PARK , (Student Member, IEEE)

Department of Artificial Intelligence, Korea University, Anam-dong, Seoul 02841, South Korea

e-mail: vmtmxmf@naver.com


ABSTRACT Multi-input multi-output structures have been developed to boost performance by learning multiple ensemble members at a small additional cost to a single network. There were several attempts to further develop multi-input multi-output structures; however, integrating the benefits of self-supervised learning into a multi-input multi-output structure has not yet been studied. In this work, we develop a multi-input multi-output structure designed to jointly learn original and self-supervised tasks, thereby leveraging the benefits of self-supervised learning. Specifically, in terms of multiple inputs, we improve the mixing strategy and minibatch structure for rotation-based self-supervised learning technique, and in terms of multiple outputs, we extend the label space of multiple classifiers to predict both the original class and true rotation degree. We observe that our method with wider networks on CIFAR-10, CIFAR-100, and Tiny ImageNet datasets shows better performance compared to previous works, even with nearly half the number of parameters, e.g., using only about 45.8% of the number of parameters compared to the best-performing multi-input multi-output method, MixMo, in the Tiny ImageNet dataset, while still achieving a 2.01% improvement.

INDEX TERMS Convolution neural networks (CNNs), deep ensemble, multi-input multi-output network.

I. INTRODUCTION

Researchers have studied effective ways to improve the performance of convolutional neural networks (CNNs) in image classification tasks. A line of works investigates network architectures. For example, it has been empirically shown that a wider CNN is typically considered an effective way to enhance performance [1], [2]. Another line of works focuses on ensemble methods [3], [4], [5]. Deep ensembles [5] train multiple neural networks of the same architecture with different random initializations and average the predictions of these networks. However, these approaches require more parameters to have better performance, which incurs more computational and memory costs.

Recent research has focused on how to *efficiently* boost performance with mild overhead. One such example is

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

implicit deep ensembles [6], [7], [8], [9], [10], which share the parameters of a single neural network among several ensemble members. Here, a small number of additional parameters for each member is introduced to make those ensemble members non-identical. This method has empirically shown performance comparable to traditional ensemble methods while requiring a smaller number of additional parameters. A representative example is the multi-input multi-output (MIMO) method [9] (see Section III-B for details), which *efficiently* reduces the inference cost by learning multiple ensemble members in a single network architecture. Another important work is the self-supervised label augmentation (SLA) method [11], which *effectively* integrates the benefits of self-supervised learning into the implicit deep ensemble method by extending the label space to predict the labels of the true class and pretext task at once. This method not only reduces the number of parameters compared to traditional deep ensembles but also the performance of SLA

exceeds that of CNN and nearly matches that of deep ensembles.

Building on these foundations, we focus on two key advantages: (i) MIMO improves performance over a single network while *maintaining similar computational costs*, and (ii) SLA shows effective performance, nearly matching that of deep ensembles *by leveraging the benefits of self-supervised learning*. Our key motivation is to integrate the advantages of both structures into one, thereby enhancing cost-effective performance: firstly, to design a structure that utilizes the benefits of self-supervised learning for effective performance; and secondly, to incorporate the advantages of a multi-input multi-output structure into our structure, for maintaining computational costs similar to SLA while simultaneously achieving better performance. For that purpose, we develop the multi-input multi-output structure to integrate the benefits of rotation-based self-supervised learning as follows: First, in the aspect of multiple inputs, we modify the training minibatch structure and the mixing strategy, which are key to combining multiple inputs in a network [9], to incorporate rotation-based self-supervised learning techniques; second, in the aspect of multiple outputs, we expand the label space of classifiers, enabling the joint learning of both original and pretext tasks.

Our contributions are summarized as follows:

- Our method improves performance compared to multi-input multi-output structures, thereby showing the effectiveness of integrating benefits of self-supervised learning into a multi-input multi-output structure.
- While our method has similar computational costs to SLA, our method shows better performance in wide networks. Furthermore, the performance gap between our method and SLA increases as the width of a network gets larger.
- Our method achieves higher performance compared to previous works, even with almost half the number of parameters.

II. RELATED WORKS

A. SELF-SUPERVISED LEARNING AND VARIANTS

The research in self-supervised learning has been driven by the goal of improving downstream-task performance through the use of cost-effective unlabeled data [12], [13], [14], [15], [16]. In [17], a transformation-based self-supervised learning method is developed, where rotations of 0, 90, 180, and 270 degrees are applied to input images, and these rotation degrees are used as artificial labels to train a neural network. This surrogate (pretext) task helps the neural network to learn useful representations in the pretraining phase.

Recently, there have been research efforts to incorporate self-supervised learning techniques into (semi-)supervised learning. For example, Hendrycks et al. [18] utilize self-supervised learning techniques for semi-supervised learning. To effectively use limited labeled data and massive unlabeled data, Hendrycks et al. [18] combine semi-supervised loss with auxiliary rotation loss, derived

from a transformation-based self-supervised learning technique. In addition, to improve the robustness in supervised learning, Zhai et al. [19] also modify the training loss by using a transformation-based self-supervised learning technique for the image classification task. Furthermore, in [11], the SLA method aims to improve the performance of a fully-supervised classification task by simultaneously learning both the original and self-supervised tasks.

B. IMPLICIT DEEP ENSEMBLES

Traditional deep ensemble methods train multiple neural networks with different random initializations and average their predictions for a single prediction [5]. Although this method improves classification accuracy, using multiple neural networks incurs additional memory and time costs proportional to the number of neural networks used. To reduce these costs, implicit deep ensemble methods have been suggested, sharing the parameters of a single neural network and yielding the same number of predictions as in traditional deep ensembles [20]. These implicit deep ensemble methods maintain accuracy comparable to the traditional deep ensemble method while their number of parameters and inference cost are approximately that of a single neural network.

There have been extensive research efforts to reduce both time and memory costs via implicit deep ensembles. One such method is BatchEnsemble [6] in which the weights of each ensemble member are derived by multiplying a member-specific low-rank matrix by a weight matrix of a single shared network. After creating these ensemble members, their predictions are averaged for the ensemble result. BatchEnsemble can achieve performance similar to the traditional deep ensemble method with only a small number of additional parameters from the parameterized vectors. Pruning-based approaches decreasing the number of floating-point operations (FLOPs) relative to the traditional deep ensemble method have been also developed [7], [8].

Multi-Input Multi-Output (MIMO) architecture [9] can make multiple predictions in a single forward pass by passing multiple inputs at once; as a result, the time and memory costs have been simultaneously reduced to levels approaching those of a single network (see Section III-B for more details). A key factor to consider here is a *mixing strategy* that decides how to combine multiple inputs for a single forward pass, which has been further developed in MixMo [10].

III. PRELIMINARIES

A. NOTATIONS

We denote scalar values by lower-case alphabets a, b, \dots, z , vector values by arrows over lower-case alphabets $\vec{a}, \vec{b}, \dots, \vec{z}$, and 3-dimensional tensor values by bold lower-case $\mathbf{a}, \mathbf{b}, \dots, \mathbf{z}$. The subscript in a vector, such as \vec{a}_c where $c \in [n] := \{1, \dots, n\}$ for $n \in \mathbb{N}$, indicates the vector's c -th element. Note that a subscript in a vector is not limited to natural numbers and varies with the index set. In addition,

we use a tuple to simultaneously represent n numbers of tensors and scalars. For example, we express this as $(\mathbf{a}^i, b^i)_{i \in [n]} = (\mathbf{a}^1, b^1, \dots, \mathbf{a}^n, b^n)$. Furthermore, we define the symbol \oplus to concatenate tuples. For instance, if $\mathcal{V} = (\mathbf{a}^1, b^1)$ and $\mathcal{W} = (\mathbf{a}^3, b^3)$, then $\mathcal{V} \oplus \mathcal{W} = (\mathbf{a}^1, b^1, \mathbf{a}^3, b^3)$.

Consider h, w , and d as representing the height, width, and dimension of an image, respectively, with $h = w$ in this study. In this context, $\mathbf{x} \in \mathbb{R}^{h \times w \times d}$ is referred to as the input for an image classification task, where $y \in \mathcal{Y} := \{1, 2, \dots, C\}$ denotes the corresponding class label. The dataset, denoted as $\mathcal{D} := (\mathbf{x}^i, y^i)_{i \in [n]}$, is used for forming a minibatch by sampling a subset, represented as $\mathcal{B} := (\mathbf{x}^i, y^i)_{i \in \mathcal{I}}$ where $\mathcal{I} \subset [n]$. The rotation transformation of an image is defined as $R^{(r)}(\mathbf{x})$ that means rotating the image counterclockwise by r degrees.

To understand the methods in this paper, we first define a convolutional layer as $\kappa : \mathbb{R}^{h \times w \times d} \rightarrow \mathbb{R}^{h \times w \times l}$ and a convolutional neural network (CNN) as $f : \mathbb{R}^{h \times w \times l} \rightarrow \mathbb{R}^p$. The feature extracted by passing through κ and f is then transformed by an affine map $\psi : \mathbb{R}^p \rightarrow \mathbb{R}^C$ into the logit vector $\vec{s} \in \mathbb{R}^C$. Following this, $\tilde{\delta} : \mathbb{R}^C \rightarrow \mathcal{Y}$ derives the classified result: $\tilde{\delta}(\vec{s}) := \arg \max_{c \in \mathcal{Y}} \delta(\vec{s})_c$, where $\delta(\vec{s})_c := \exp(\vec{s}_c) / \sum_{k=1}^C \exp(\vec{s}_k)$. We define the composition of δ and ψ as a softmax classifier, $\sigma := (\delta \circ \psi)$, and its output as the prediction. The cross-entropy loss function is $\mathcal{L}_{CE}((\sigma \circ f \circ \kappa)(\mathbf{x}), y) := -\log((\sigma \circ f \circ \kappa)(\mathbf{x})_y)$.

B. MULTI-INPUT MULTI-OUTPUT

1) OVERVIEW

The multi-input multi-output (MIMO) method processes M inputs to generate M predictions in a single forward pass of a network f [9] (see Fig. 1 for its illustration). The important aspect of this method is that MIMO uses only one CNN f for M multiple predictions, unlike deep ensembles, which require M CNNs, i.e., MIMO reduces both computational and memory costs compared to deep ensembles.

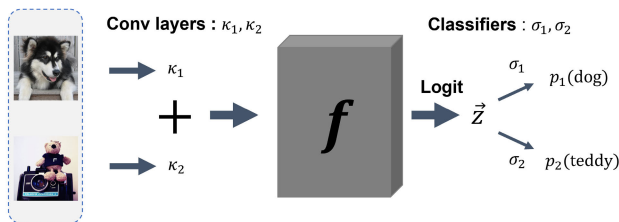


FIGURE 1. For $M = 2$, the process of MIMO [9] is divided into three steps: (i) MIMO encodes the data \mathbf{x}^j and $\mathbf{x}^{j'}$ through the first convolution layers κ_1 and κ_2 , obtaining $\kappa_1(\mathbf{x}^j)$ and $\kappa_2(\mathbf{x}^{j'})$. (ii) It passes $\kappa_1(\mathbf{x}^j) + \kappa_2(\mathbf{x}^{j'})$ to the CNN f to obtain the output feature map $\vec{z} = f(\kappa_1(\mathbf{x}^j) + \kappa_2(\mathbf{x}^{j'}))$. (iii) MIMO uses different softmax classifiers σ_1 and σ_2 to classify \vec{z} , yielding two predictions $\sigma_1(\vec{z})$ and $\sigma_2(\vec{z})$.

2) MULTIPLE INPUTS AND OUTPUTS

a: MULTIPLE INPUTS

MIMO modifies the training minibatch structure to handle multiple inputs. Let \mathcal{A}_t represent the set formed by uniformly

randomly selecting a proportion of q in the minibatch index set \mathcal{I} (i.e., $\mathcal{A}_t \subseteq \mathcal{I}$ and $|\mathcal{A}_t| = q \times |\mathcal{I}|$). The training minibatch of MIMO is constructed as follows:

$$\mathcal{B}_{\text{MIMO}} := \bigoplus_{t=1}^b \left(\left(\mathbf{x}^{\hat{\phi}_1^t(i)}, \dots, \mathbf{x}^{\hat{\phi}_M^t(i)} \right), \left(y^{\hat{\phi}_1^t(i)}, \dots, y^{\hat{\phi}_M^t(i)} \right) \right)_{i \in \mathcal{I}} \tag{1}$$

where

$$\hat{\phi}_m^t(i) = \begin{cases} \text{id}(i) & \text{if } i \in \mathcal{A}_t \\ \phi_m^t(i) & \text{otherwise} \end{cases}$$

is a function that randomly shuffles the indices in \mathcal{A}_t^c for each ensemble member $m \in [M]$.

Here, two performance-enhancing techniques are used in $\mathcal{B}_{\text{MIMO}}$. The first technique is the batch repetition rate b , which indicates the number of times that training examples in the minibatch are duplicated. This leads to the minibatch $\mathcal{B}_{\text{MIMO}}$ containing $|\mathcal{I}| \times M \times b$ instances. The second technique is the input repetition rate $q \in [0, 1]$, specifying the rate that the identity function is used instead of ϕ_m^t . This means that as q increases, $\kappa_1, \dots, \kappa_M$ more frequently encounter a pair of the same instances. Empirical evidence in [9] shows that higher batch repetition rates slightly improve accuracy, peaking at $b = 4$, and that the most effective input repetition rate varies with the scenarios. When the capacity of CNN is sufficiently large for a task, such as WideResNet-10 for the CIFAR-10 dataset, a small input repetition rate improves performance. Conversely, when the capacity of CNN is limited, as in the case with ResNet-50 for the ImageNet dataset, a high input repetition rate is shown to be beneficial for accuracy.

The next step is to forward a pair of multiple inputs in $\mathcal{B}_{\text{MIMO}}$ to a single f . First, after drawing M multiple inputs $(\mathbf{x}^1, \dots, \mathbf{x}^M)$ from $\mathcal{B}_{\text{MIMO}}$, this method passes each of these M multiple inputs to their corresponding first convolution layers (i.e., $\kappa_1, \dots, \kappa_M$) and then sums their values as follows:

$$\tau(\mathbf{x}^1, \dots, \mathbf{x}^M) := \sum_{m=1}^M \kappa_m(\mathbf{x}^m) \tag{2}$$

Then, MIMO passes $\tau(\mathbf{x}^1, \dots, \mathbf{x}^M)$ into f for a forward pass, resulting in a single feature map $\vec{z} = (f \circ \tau)(\mathbf{x}^1, \dots, \mathbf{x}^M)$.

b: MULTIPLE OUTPUTS

Section III-B This feature map \vec{z} is then used to generate multiple outputs via M softmax classifiers, resulting in $(\sigma_1(\vec{z}), \dots, \sigma_M(\vec{z}))$. In this scenario, each classifier learns to ignore other inputs, and a single f assists these classifiers to ensure that the output feature maps are distinguishable [9]. This approach can efficiently generate multiple outputs with a single f , unlike traditional deep ensembles that require M separate CNNs for multiple outputs.

3) TRAINING AND INFERENCE

Given multiple inputs and their corresponding labels, the training loss for MIMO can be expressed as follows:

$$\begin{aligned} \mathcal{L}_{\text{MIMO}} & \left((\mathbf{x}^1, \dots, \mathbf{x}^M), (y^1, \dots, y^M) \right) \\ & = \sum_{m=1}^M \mathcal{L}_{\text{CE}} \left((\sigma_m \circ f \circ \tau) (\mathbf{x}^1, \dots, \mathbf{x}^M), y^m \right). \end{aligned} \quad (3)$$

At last, for each minibatch, the parameters of the model are updated using SGD based on the gradient of

$$\sum_{\substack{((\mathbf{x}^1, \dots, \mathbf{x}^M), \\ (y^1, \dots, y^M)) \in \mathcal{B}_{\text{MIMO}}}} \frac{\mathcal{L}_{\text{MIMO}} ((\mathbf{x}^1, \dots, \mathbf{x}^M), (y^1, \dots, y^M))}{b \times |\mathcal{I}|}. \quad (4)$$

In the inference phase, a single image \mathbf{x} is duplicated M times to generate multiple inputs $(\mathbf{x}, \dots, \mathbf{x})$. Corresponding to these M inputs, MIMO yields M predictions, and then averages them for an ensemble prediction:

$$\frac{1}{M} \sum_{m=1}^M (\sigma_m \circ f \circ \tau) (\mathbf{x}, \dots, \mathbf{x}). \quad (5)$$

4) VARIANT OF MIMO

MixMo is a subsequent study of MIMO that has led to performance improvements by changing how multiple inputs are mixed, specifically focusing on $M = 2$ [10]. Unlike MIMO’s straightforward combination of inputs as $\kappa_1(\mathbf{x}^1) + \kappa_2(\mathbf{x}^2)$, MixMo uses a hybrid strategy for training. This method probabilistically switches two mixing strategies between linear interpolation and the CutMix technique [21] for each training minibatch. In the case of linear interpolation, MixMo uses $2[\alpha\kappa_1(\mathbf{x}^1) + (1 - \alpha)\kappa_2(\mathbf{x}^2)]$, where $\alpha \sim \text{Beta}(2, 2)$. In the case of CutMix, the mixing strategy is represented as $2[\mathbf{1}_\alpha \odot \kappa_1(\mathbf{x}^1) + (\mathbf{1} - \mathbf{1}_\alpha) \odot \kappa_2(\mathbf{x}^2)]$, where \odot is element-wise multiplication, $\mathbf{1} \in \{1\}^{h \times w \times l}$ is a tensor filled entirely with ones, and $\mathbf{1}_\alpha \in \{0, 1\}^{h \times w \times l}$ is a binary mask consisting of a rectangular cuboid region with value 1 and its complement with value 0; here, given $\alpha \sim \text{Beta}(2, 2)$, the rectangular cuboid in a binary mask is represented as $\{1\}^{h_\alpha \times w_\alpha \times l}$, where any values of h_α and w_α satisfy that $h_\alpha, w_\alpha < h$ and $(h_\alpha \times w_\alpha)/hw = \alpha$. Then, the probability of switching between linear interpolation and the CutMix technique is determined as follows: Up to 11/12 of the total number of training epochs, these two techniques are randomly selected with equal probability 0.5, and beyond 11/12 of the total number of training epochs, the probability of the CutMix gradually decreases with the probability of $p_e := (\text{the \# of epochs} - \text{current epoch}) / (\frac{1}{12} \times \text{the \# of epochs})$.

C. SELF-SUPERVISED LABEL AUGMENTATION

1) OVERVIEW

The self-supervised label augmentation (SLA) method not only uses the class label of an image but also an artificial label for training, which is the transformation applied to the

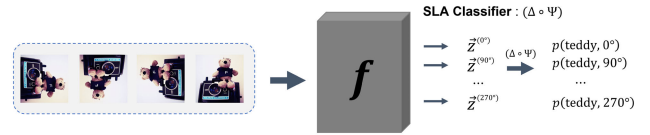


FIGURE 2. The process of SLA [11] is divided into three steps: (i) rotating an image by all degrees in \mathcal{R} , (ii) CNN f extracting feature maps from each of these rotations, and (iii) a joint classifier using these feature maps to predict the original class and the specific rotation degree of the image (see Section III-C2 for more details about a joint classifier).

image [11]. To describe SLA in more detail, we outline its input-output structure in three steps, as shown in Fig. 2; here, we focus on rotation transformation using the set of rotation degrees $\mathcal{R} := \{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$ and set the input dimension of CNN f as $\mathbb{R}^{h \times w \times d}$ (i.e., $d = l$). Since this method also uses a single f , it reduces memory cost compared to deep ensembles.

2) SLA INPUTS AND OUTPUTS

a: SLA INPUTS

In SLA, the minibatch is expanded by rotating each image \mathbf{x}^i by every degree $r \in \mathcal{R}$, and each rotated image is paired with its corresponding class label y^i and rotation degree r :

$$\mathcal{B}_{\text{SLA}} := \bigoplus_{r \in \mathcal{R}} \left(R^{(r)}(\mathbf{x}^i), (y^i, r) \right)_{i \in \mathcal{I}}. \quad (6)$$

SLA uses \mathcal{B}_{SLA} instead of \mathcal{B} , resulting in the total number of image instances in the minibatch increasing from $|\mathcal{I}|$ to $|\mathcal{R}| \times |\mathcal{I}|$. In addition, since (y^i, r) is used for the label of \mathcal{B}_{SLA} , the label space is expanded from \mathcal{Y} to $\hat{\mathcal{Y}} = \{(1, 0^\circ), (1, 90^\circ), (1, 180^\circ), (1, 270^\circ), (2, 0^\circ), \dots, (C, 270^\circ)\}$, and the size of the SLA label space is $|\hat{\mathcal{Y}}| = 4C$.

The next step is forwarding all rotated images to a network f for output feature maps, each denoted as $\vec{z}^{(r)} = f(R^{(r)}(\mathbf{x})) \in \mathbb{R}^p$ for $r \in \mathcal{R}$. Since a single network f performs a forward pass for each rotated image $R^{(r)}(\mathbf{x})$, there are a total of $|\mathcal{R}|$ output feature maps per image.

b: SLA OUTPUTS

After obtaining output feature maps $(\vec{z}^{(r)})_{r \in \mathcal{R}}$, SLA uses an affine map defined as $\Psi : \mathbb{R}^p \rightarrow \mathbb{R}^{C \times |\mathcal{R}|}$ to produce the logit $\vec{s}^{(r)} := \Psi(\vec{z}^{(r)})$. The affine map enables the joint prediction of both the original class and the rotation degree. Afterward, SLA’s next step is to apply softmax to all logits $\vec{s}^{(r)}$ for $r \in \mathcal{R}$ as follows:

$$\Delta(\vec{s}^{(r)})_{(c, r')} := \frac{\exp(\vec{s}_{(c, r')}^{(r)})}{\sum_{(c'', r'') \in \hat{\mathcal{Y}}} \exp(\vec{s}_{(c'', r'')}^{(r)})}. \quad (7)$$

Note that $\Delta(\vec{s}^{(r)})$ is a vector within the range $(0, 1)^{C \times |\mathcal{R}|}$, and (c, r') represents an index within this vector’s index set, which is denoted as $\hat{\mathcal{Y}}$. In addition, we refer to $(\Delta \circ \Psi)$ as a joint classifier.

3) TRAINING AND INFERENCE

SLA's training loss is expressed as follows:

$$\begin{aligned} \mathcal{L}_{\text{SLA}}((\mathbf{x}), (y, r)) \\ = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \mathcal{L}_{\text{CE}} \left((\Delta \circ \Psi \circ f \circ R^{(r)}) (\mathbf{x}), (y, r) \right). \end{aligned} \quad (8)$$

Therefore, the training loss of SLA is the average of $|\mathcal{R}|$ different cross-entropy losses, computed using predictions corresponding to the true label and degree (y, r) .

During inference with SLA, predicting both the original class and rotation degree simultaneously becomes unnecessary because we already have information about the transformations. Given the true degree r , we can focus on a label space of size C instead of $C \times |\mathcal{R}|$. In such cases, the index set changes from $\hat{\mathcal{Y}}$ to $\{(1, r), (2, r), \dots, (C, r)\}$. Here, we express the logit as $\Psi^r(\vec{z}^{(r)}) \in \mathbb{R}^C$, which is a vector composed of the positions corresponding to the modified index set.

SLA then aggregates these logits for inference in a specific manner:

$$\begin{aligned} \bar{\Psi}(\mathbf{x}, \mathcal{R}) &= \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \left(\Psi^r \circ f \circ R^{(r)} \right) (\mathbf{x}) \\ &= \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \Psi^r(\vec{z}^{(r)}) \in \mathbb{R}^C \end{aligned} \quad (9)$$

$$p_{\text{agg}}(c|\mathbf{x}, \mathcal{R}) = \frac{\exp(\bar{\Psi}(\mathbf{x}, \mathcal{R})_c)}{\sum_{k=1}^C \exp(\bar{\Psi}(\mathbf{x}, \mathcal{R})_k)} \quad (10)$$

$$p_{\text{si}}(c|\mathbf{x}, 0^\circ) = \frac{\exp(\Psi^{0^\circ}(\mathbf{x})_c)}{\sum_{k=1}^C \exp(\Psi^{0^\circ}(\mathbf{x})_k)}. \quad (11)$$

Finally, there are two types of inference methods: either using $\arg \max_{c \in \mathcal{Y}} p_{\text{agg}}(c|\mathbf{x}, \mathcal{R})$ or $\arg \max_{c \in \mathcal{Y}} p_{\text{si}}(c|\mathbf{x}, 0^\circ)$ to obtain the classified result. When using p_{agg} , similar to the training process, a single f executes $|\mathcal{R}|$ forward passes during inference. In contrast, selecting p_{si} performs only a single forward pass. Note that a key advantage of using p_{agg} is that SLA effectively achieves performance nearly close to that of deep ensembles.

IV. METHODOLOGY

1) OVERVIEW

For both the *efficiency* of MIMO and the *effectiveness* of SLA, we use a multi-input multi-output structure while jointly learning original and rotation-based self-supervised tasks. To briefly describe the input-output structure of our method for training, as illustrated in Fig. 3 for the $M = 2$ case, the process is as follows: (i) a pair of multiple inputs is rotated by the same degree, then each input is passed through its corresponding convolution layer (κ_1 or κ_2), and the outputs are summed up; (ii) this value is then forwarded to f to obtain the output feature map $\vec{z}^{(r)}$ for $r \in \mathcal{R}$; (iii) it is classified by different multiple joint classifiers.

Here, by using a multi-input multi-output structure, we *efficiently* generate more ensemble predictions than SLA, yet at a similar cost. In addition, by changing from multiple classifiers to multiple joint classifiers, we enhance *effectiveness* compared to the MIMO-based model, incorporating the advantages of rotation-based self-supervised learning into supervised learning tasks. Furthermore, we aim to not only integrate these techniques but also to improve them, with a focus on the training minibatch and mixing strategy, which we regard as key elements in combining the MIMO and SLA methods (see Section IV-3 for details).

2) ROTATION DEGREES

Our method uses a rotation-based self-supervised learning technique, based on the set of rotation degrees $\mathcal{R} = \{0, 90, 180, 270\}$. Rotating images by 0, 90, 180, and 270 degrees is a widely adopted practice in research using a rotation-based self-supervised learning technique for two primary reasons [11], [17], [18], [19], [22]. First, images captured by humans typically show objects in an up-standing position, thereby reducing ambiguity in rotation transformations [17]. Second, rotations of 0, 90, 180, and 270 degrees are simple to implement as they can be achieved through flip or transpose operations [11], [17]. We also base our rotation transformation on 0, 90, 180, and 270 degrees due to well defined up-standing position of objects in images and simplicity in implementation. In addition, we experiment with a different set of rotation degrees, which is discussed in Table 5.

3) MIXING MULTIPLE INPUTS

To combine the self-supervised learning technique with the method of mixing M inputs, our minibatch is constructed as follows:

$$\begin{aligned} \mathcal{B}_{\text{ours}} \\ := \bigoplus_{r \in \mathcal{R}} \left(\left(R^{(r)}(\mathbf{x}^{\hat{\phi}_1^t(i)}) \right), \dots, R^{(r)}(\mathbf{x}^{\hat{\phi}_M^t(i)}) \right), \\ \times \left(\left(y^{\hat{\phi}_1^t(i)}, r \right), \dots, \left(y^{\hat{\phi}_M^t(i)}, r \right) \right)_{i \in \mathcal{I}} \end{aligned} \quad (12)$$

where

$$\hat{\phi}_m^t(i) = \begin{cases} \text{id}(i) & \text{if } i \in \mathcal{A} \\ \phi_m^t(i) & \text{otherwise} \end{cases}$$

and $m \in [M]$. We note that \mathcal{A} is the set formed by uniformly randomly selecting a proportion of q in the minibatch index set \mathcal{I} , and ϕ_m^t is a function that randomly shuffles the indices in \mathcal{A}^c . Here, unlike MIMO, we design our minibatch structure to share \mathcal{A} . This approach is an extension of the philosophy of SLA. SLA rotates each image by the rotation degrees in \mathcal{R} to construct a training minibatch. Similarly, for $i \in \mathcal{A}$, our method rotates a pair of the same images by each rotation degree $r \in \mathcal{R}$, i.e., $\left(\left(R^{(r)}(\mathbf{x}^i), \dots, R^{(r)}(\mathbf{x}^i) \right) \right)_{r \in \mathcal{R}}$ to form a training minibatch (see the dashed green rectangle in Fig. 3).

For our mixing strategy, we draw M rotated multiple inputs from $\mathcal{B}_{\text{ours}}$, i.e., $\left(R^{(r)}(\mathbf{x}^1), \dots, R^{(r)}(\mathbf{x}^M) \right)$ for $r \in \mathcal{R} = \{0^\circ,$

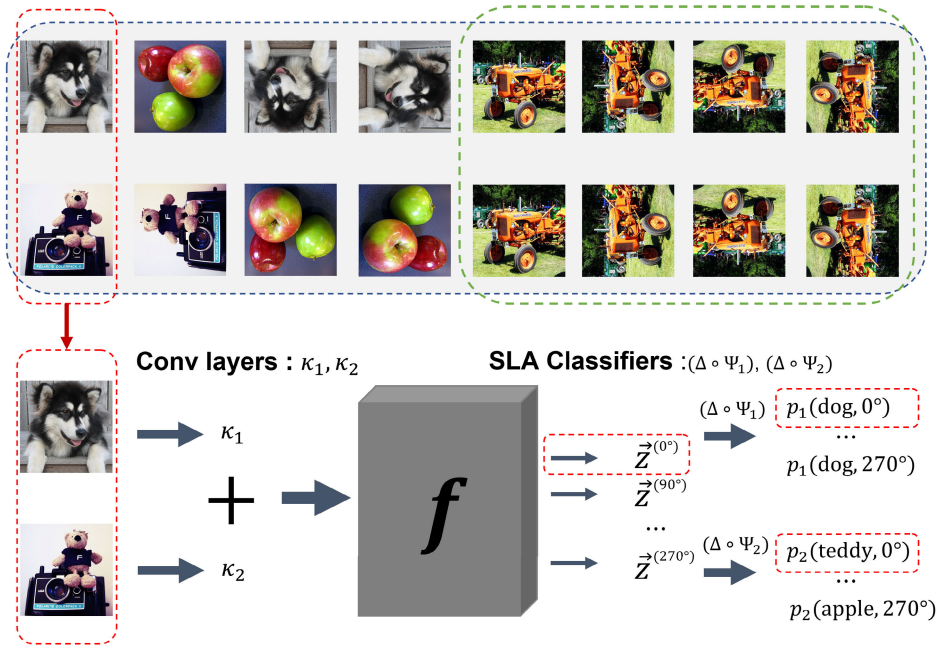


FIGURE 3. (a) Training.

$90^\circ, 180^\circ, 270^\circ$. Subsequently, each of convolution layers $\kappa_1, \dots, \kappa_M$ processes its corresponding rotated image. Then, we sum the features of these rotated images:

$$\tau \left(R^{(r)}(\mathbf{x}^1), \dots, R^{(r)}(\mathbf{x}^M) \right) := \sum_{m=1}^M \kappa_m \left(R^{(r)}(\mathbf{x}^m) \right). \quad (13)$$

This summed output is then fed into f to produce the output feature map. That is,

$$\vec{z}^{(r)} = (f \circ \tau) \left(R^{(r)}(\mathbf{x}^1), \dots, R^{(r)}(\mathbf{x}^M) \right) \in \mathbb{R}^p. \quad (14)$$

We note that for $i \in \mathcal{A}$, since there are $|\mathcal{R}|$ instances of a pair of multiple images $((R^{(r)}(\mathbf{x}^i), \dots, R^{(r)}(\mathbf{x}^i)))_{r \in \mathcal{R}}$ in $\mathcal{B}_{\text{ours}}$, there also exist $|\mathcal{R}|$ corresponding output feature maps (i.e., $(\vec{z}^{(r)})_{r \in \mathcal{R}}$).

In our mixing strategy, a key aspect is that we rotate each pair of multiple inputs by the same degree, even though rotating them by different degrees is possible; this approach is empirically examined in Section V-E.

4) MULTIPLE OUTPUTS WITH SELF-SUPERVISED LABEL AUGMENTATION

Just as a reminder, the joint linear classifier $(\Delta \circ \Psi)$, which enables prediction of both classes and degrees, consists of $\Psi : \mathbb{R}^p \rightarrow \mathbb{R}^{C \times |\mathcal{R}|}$ and $\Delta : \mathbb{R}^{C \times |\mathcal{R}|} \rightarrow (0, 1)^{C \times |\mathcal{R}|}$, the latter representing the softmax layer following the affine map. In this context, we use M joint classifiers $(\Delta_1 \circ \Psi_1), \dots, (\Delta_M \circ \Psi_M)$, with each m -th affine map denoted as Ψ_m .

When our method performs a single forward pass, the m -th prediction is as follows:

$$g_m \left(R^{(r)}(\mathbf{x}^1), \dots, R^{(r)}(\mathbf{x}^M) \right)$$

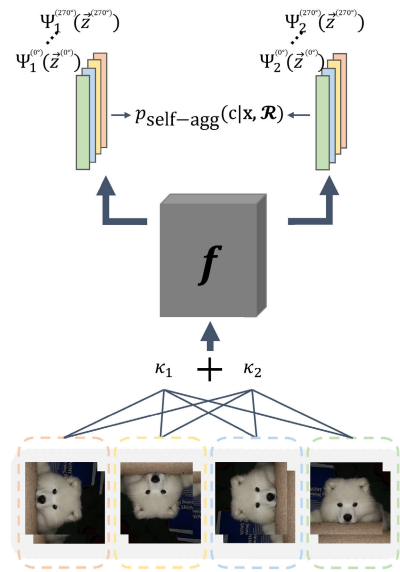


FIGURE 4. (b) Inference.

$$:= (\Delta \circ \Psi_m \circ f \circ \tau) \left(R^{(r)}(\mathbf{x}^1), \dots, R^{(r)}(\mathbf{x}^M) \right) \quad (15)$$

for $m \in [M]$ and $r \in \mathcal{R}$. In a single forward pass, our method yields M such predictions, and considering all pairs of multiple inputs in $\mathcal{B}_{\text{ours}}$, our method generates a total of $M \times |\mathcal{R}| \times |\mathcal{I}|$ predictions.

5) TRAINING LOSS AND INFERENCE

Given a pair of M multiple inputs and its corresponding pair of labels, we compute the training loss for our method as follows:

$$\mathcal{L}_{\text{ours}} \left(\left(R^{(r)}(\mathbf{x}^1), \dots, R^{(r)}(\mathbf{x}^M) \right), \left((y^1, r), \dots, (y^M, r) \right) \right)$$

$$= \sum_{m=1}^M \mathcal{L}_{\text{CE}} \left(g_m \left(R^{(r)}(\mathbf{x}^1), \dots, R^{(r)}(\mathbf{x}^M) \right), (y^m, r) \right). \quad (16)$$

Then, for each pair of training examples in the minibatch $\mathcal{B}_{\text{ours}}$, we compute the gradient of $\mathcal{L}_{\text{ours}}$. Using the average of these gradients, the parameters of the model are updated via SGD.

When inferring from a single image \mathbf{x} , we generate the following pairs of multiple inputs:

$$\left(\left(R^{(r)}(\mathbf{x}), \dots, R^{(r)}(\mathbf{x}) \right) \right)_{r \in \mathcal{R}} \quad (17)$$

where each pair of multiple inputs consists of M instances (see Fig. 4 for an illustration). Next, we use $(f \circ \tau)$ for forward passes across all pairs of multiple inputs to obtain $(\tilde{z}^{(r)})_{r \in \mathcal{R}}$. The output feature maps are then classified using a different approach from training. Since we already have information about the true degree r , it is unnecessary to predict both the class and rotation degree simultaneously. We only need to consider $\Psi_m^r(\tilde{z}^{(r)}) \in \mathbb{R}^C$, a logit computed based on the reduced label space, without using Ψ_m . We classify the output feature maps using an extended version of the SLA aggregation method [11] as follows:

$$\begin{aligned} & \tilde{\Psi}_{\text{self}}((\mathbf{x}, \dots, \mathbf{x}), \mathcal{R}) \\ &= \sum_{r \in \mathcal{R}} \sum_{m=1}^M \frac{(\Psi_m^r \circ f \circ \tau^{(r)})(\mathbf{x}, \dots, \mathbf{x})}{M \times |\mathcal{R}|} \end{aligned} \quad (18)$$

$$\begin{aligned} & p_{\text{self-agg}}(c | (\mathbf{x}, \dots, \mathbf{x}), \mathcal{R}) \\ &= \frac{\exp(\tilde{\Psi}_{\text{self}}((\mathbf{x}, \dots, \mathbf{x}), \mathcal{R})_c)}{\sum_{k=1}^C \exp(\tilde{\Psi}_{\text{self}}((\mathbf{x}, \dots, \mathbf{x}), \mathcal{R})_k)} \end{aligned} \quad (19)$$

$$\begin{aligned} & p_{\text{self-si}}(c | (\mathbf{x}, \dots, \mathbf{x}), 0^\circ) \\ &= \frac{\exp\left(\frac{1}{M} \sum_{m=1}^M (\Psi_m^{0^\circ} \circ f \circ \tau^{(0^\circ)})(\mathbf{x}, \dots, \mathbf{x})_c\right)}{\sum_{k=1}^C \exp\left(\frac{1}{M} \sum_{m=1}^M (\Psi_m^{0^\circ} \circ f \circ \tau^{(0^\circ)})(\mathbf{x}, \dots, \mathbf{x})_k\right)}. \end{aligned} \quad (20)$$

We note that the classified result can be obtained using either $\arg \max_{c \in \mathcal{Y}} p_{\text{self-agg}}(c | (\mathbf{x}, \dots, \mathbf{x}), \mathcal{R})$ or $\arg \max_{c \in \mathcal{Y}} p_{\text{self-si}}(c | (\mathbf{x}, \dots, \mathbf{x}), 0^\circ)$. There are two methods for inference, similar to SLA.

V. EXPERIMENTS

In this section, we empirically verify our method in image classification by comparing it with SLA and the MIMO-based model on benchmark datasets (Section V-B). Then, we investigate the effects of hyperparameters in our algorithm. We first consider the input repetition rate q in our method, which is a key factor affecting its performance (Section V-C). We then examine the performance of our method by varying several hyperparameters, such as the number of multiple inputs and outputs M and the range of

rotation degrees \mathcal{R} (Section V-D). Lastly, we study different alternatives to our minibatch constructions for training and mixing strategies for multiple inputs (Section V-E).

A. EXPERIMENTAL SETUP

a: DATASETS

For our image classification task, we use benchmark datasets such as CIFAR- $\{10, 100\}$ [23] and Tiny ImageNet [24]. The CIFAR- $\{10, 100\}$ datasets consist of 32×32 3-channel color images, with 50,000 images for training and 10,000 images for testing. The CIFAR-10 dataset contains 6,000 images per class across 10 classes, and the CIFAR-100 dataset has 600 images per class across 100 classes. The Tiny ImageNet dataset, a downsized version of ImageNet [25], is composed of 64×64 3-channel color images, with a total 100,000 images for training (500 images per class across 200 classes). As a public test dataset is not provided in the Tiny ImageNet dataset, we use the validation set consisting of 10,000 images as our test dataset.

b: MODELS AND HYPERPARAMETERS

When training on the CIFAR- $\{10, 100\}$ datasets, we primarily use WideResNet28-W (WRN-W) [2]; the network width increases in proportion to the width parameter W and the network depth is set at 28. For the CIFAR-10 dataset, WRN- $\{6, 10, 14\}$ consist of 13.16M, 36.50M, and 71.50M parameters respectively.

For both the CIFAR- $\{10, 100\}$ datasets, we use the SGD optimizer with a learning rate of $(0.1 \times \text{minibatch size}) / (128 \times b')$. Here, b' represents the batch repetition rate $b = 4$ in the MIMO-based model [9], [10], and it denotes the number of rotation degrees $|\mathcal{R}| = 4$ in both SLA and our method. We use Nesterov momentum updates with the momentum parameter 0.9, a warm-up epoch of 1, a minibatch size of 64, and L2 regularization with the regularization parameter $3e - 4$. We also use learning rate decay with a decay ratio of 0.2: 300 epochs with learning rate decays at epochs $\{120, 240, 270\}$ for the CIFAR-10 dataset and 350 epochs with decays at $\{120, 240, 270, 300, 330\}$ for the CIFAR-100 dataset.

For the Tiny ImageNet dataset, we use PreActResNet18-W (PRN-W), where the network depth is set at 18 and the network width scales proportionally with the width parameter W , ranging from $W = 1$ to $W = 3$ [26]. The number of parameters for $W = 1$, $W = 2$, and $W = 3$ are 11.28M, 44.80M, and 100.56M, respectively. When training on the Tiny ImageNet dataset, we use the same hyperparameters as in the CIFAR-100 case, except for a minibatch size and warm-up epochs. The minibatch size is set to 128, and the number of warm-up epochs varies with the width parameter W in PRN-W. Specifically, for $W = 1$, all method use 2 warm-up epochs. For $W = 2$, both SLA and our method still use 2 warm-up epochs, MixMo uses 3, and the original PRN-2 uses 5 due to instability in the early stages of training. Likewise, for $W = 3$, SLA and our method continue

TABLE 1. Accuracy measured using the WRN- W architecture on the CIFAR-10 dataset.

Method	Forward Passes	WRN-6	WRN-10	WRN-14
Our SI	1	96.49 \pm 0.08	96.96 \pm 0.01	97.10 \pm 0.08
Our AGG	4	97.32 \pm 0.01	97.59 \pm 0.10	97.72 \pm 0.03
Baseline	1	95.72 \pm 0.10	95.77 \pm 0.19	95.92 \pm 0.10
SLA SI	1	96.50 \pm 0.18	96.56 \pm 0.09	96.78 \pm 0.12
SLA AGG	4	97.33 \pm 0.10	97.42 \pm 0.04	97.49 \pm 0.09
MixMo	1	97.22 \pm 0.05	97.48 \pm 0.10	97.34 \pm 0.28
MixMo 5C	5	97.24 \pm 0.05	97.53 \pm 0.09	97.37 \pm 0.21

TABLE 2. Accuracy measured using the WRN- W architecture on the CIFAR-100 dataset.

Method	Forward Passes	WRN-6	WRN-10	WRN-14
Our SI	1	81.95 \pm 0.14	83.14 \pm 0.14	83.79 \pm 0.20
Our AGG	4	84.33 \pm 0.12	85.59 \pm 0.02	85.77 \pm 0.10
Baseline	1	78.73 \pm 0.07	79.62 \pm 0.25	79.73 \pm 0.22
SLA SI	1	80.84 \pm 0.18	82.01 \pm 0.17	82.27 \pm 0.18
SLA AGG	4	84.63 \pm 0.24	85.40 \pm 0.11	85.53 \pm 0.10
MixMo	1	82.75 \pm 0.10	84.16 \pm 0.20	84.48 \pm 0.10
MixMo 5C	5	83.34 \pm 0.06	84.64 \pm 0.12	85.11 \pm 0.12

TABLE 3. Accuracy measured using the PRN- W architecture on the Tiny ImageNet dataset.

Method	Forward Passes	PRN-1	PRN-2	PRN-3
Our SI	1	66.58 \pm 0.28	69.32 \pm 0.09	70.69 \pm 0.20
Our AGG	4	69.44 \pm 0.16	72.74 \pm 0.27	73.73 \pm 0.07
Baseline	1	63.37 \pm 0.21	66.31 \pm 0.05	67.81 \pm 0.26
SLA SI	1	64.05 \pm 0.11	68.17 \pm 0.16	69.48 \pm 0.19
SLA AGG	4	69.05 \pm 0.07	71.79 \pm 0.29	72.59 \pm 0.13
MixMo	1	66.81 \pm 0.16	69.81 \pm 0.14	70.73 \pm 0.16
MixMo 5C	5	67.14 \pm 0.14	70.24 \pm 0.07	71.11 \pm 0.22

with 2 warm-up epochs, but MixMo uses 6 and the original PRN-3 uses 8 for the same reason. When more than 2 warm-up epochs are used for MixMo and/or original PRN, we extend the total training epochs to maintain an equal number of (total training epochs – warm-up epochs).

For all datasets, our methods mainly use the number of multiple inputs and outputs $M = 2$, and the set of all rotation degrees $\mathcal{R} = \{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$ (for detailed explanations, see Section V-D).

c: OTHER SETUPS

For the CIFAR- $\{10, 100\}$ datasets and the Tiny ImageNet dataset, all networks were trained three times using the TensorFlow framework. All the results of these image classification experiments were reported with the test accuracy at the last iteration, rather than selecting the best test accuracy without a validation dataset as seen in approaches like [10].

B. MAIN EXPERIMENTS

a: PREPARATION

In the main results, we compare the performance of our method with that of SLA and MixMo, which is an improved version of MIMO [10]. Here, since the MixMo paper reports performance improvements with the application of CutMix

augmentation to the input images [21], we follow this setup. In addition, because our method and SLA can use four forward passes during inference, we also evaluated the performance of MixMo using multiple forward passes via test-time augmentation (TTA) [27], [28], [29]. In particular, we use 5 crop TTA for MixMo (MixMo 5C), a widely used TTA technique involving center cropping and cropping each of the four corners of an image, thus requiring five forward passes [27], [28], [29], [30]. In all of our experiments, SLA SI accuracy and SLA AGG accuracy refer to accuracies measured using $\arg \max_{c \in \mathcal{Y}} p_{\text{si}}(c | \mathbf{x}, 0^\circ)$ and $\arg \max_{c \in \mathcal{Y}} p_{\text{agg}}(c | \mathbf{x}, \mathcal{R})$, respectively. Similarly, our SI accuracy and our AGG accuracy denote accuracies measured using $\arg \max_{c \in \mathcal{Y}} p_{\text{self-si}}(c | (\mathbf{x}, \dots, \mathbf{x}), 0^\circ)$ and $\arg \max_{c \in \mathcal{Y}} p_{\text{self-agg}}(c | (\mathbf{x}, \dots, \mathbf{x}), \mathcal{R})$. The term baseline represents the performance of a vanilla network without any ensemble and augmentation methods.

b: MAIN RESULTS

We observed that our method performs better than SLA and MixMo with wider CNNs, especially in the CIFAR- $\{10, 100\}$ results when using the WRN- $\{10, 14\}$ architectures (Table 1, 2), and in the Tiny ImageNet results for all PRN- $\{1, 2, 3\}$ cases (Table 3). Furthermore, in Table 1, 2, 3,

we observed that as the width of a network gets larger, our method can achieve better performance than SLA and MixMo even with fewer parameters.

c: MAIN RESULTS: COMPARING OUR METHOD WITH SLA

Looking at the CIFAR-10 and the Tiny ImageNet experimental results (Table 1, 2, 3), we observed that the difference (our AGG accuracy – SLA AGG accuracy) increased as the width parameter W of the network increased. For the CIFAR-10 results obtained with the WRN-6 architecture (Table 1), our AGG accuracy and SLA AGG accuracy showed similar results. However, in the CIFAR-10 results with the WRN-10 architecture (Table 1), our AGG accuracy exceeded SLA AGG accuracy by 0.17%, and this difference even increased to 0.23% with the WRN-14 architecture. In addition, in the CIFAR-100 results using the WRN-6 architecture (Table 2), our AGG accuracy was 0.3% lower than SLA AGG accuracy. However, with wider architectures, our AGG accuracy outperformed SLA AGG accuracy; when using the WRN-10, our AGG accuracy showed a 0.19% improvement, and with the WRN-14, this increase was even more pronounced at 0.24%. Moving to the Tiny ImageNet dataset (Table 3), our AGG accuracy consistently outperformed SLA AGG accuracy across all PRN-1, 2, 3 architectures. Specifically, with the PRN-1 architecture, our AGG accuracy achieved a 0.39% higher performance than SLA AGG accuracy. This gap widened with the PRN-2 architecture, where our AGG accuracy exceeded SLA AGG accuracy by more than 0.95%. Moreover, with the PRN-3 architecture, the difference further increased to 1.14%. In addition, our SI accuracy consistently outperformed SLA SI accuracy in all results (Table 1, 2, 3), except when using the WRN-6 architecture on the CIFAR-10 dataset, where our SI accuracy was almost similar to SLA SI accuracy.

d: MAIN RESULTS: COMPARING OUR METHOD WITH MIXMO

Our AGG accuracy outperformed MixMo 5C accuracy in all experimental results (Table 1, 2, 3). Especially for the Tiny ImageNet results with all PRN-1, 2, 3 architectures (Table 3), the performance gap between our AGG accuracy and MixMo 5C accuracy was $> 2.3\%$. As the width parameter W increased, the performance of our method improved over that of MixMo, with (our AGG accuracy – MixMo 5C accuracy) being 2.3% for PRN-1 architecture, 2.5% for PRN-2 architecture, and 2.62% for PRN-3 architecture.

e: MAIN RESULTS: PERFORMANCE EFFICIENCY

In Table 1, 2, 3, we observed that as the width of a network gets larger, our AGG method outperforms SLA AGG and MixMo 5C even with fewer parameters. For example in Table 1, our method, based on the WRN-10 architecture, uses 36.54 million parameters, whereas SLA and MixMo, based on the WRN-14 architecture, use 71.52 million

and 71.51 million parameters, respectively. Yet, our AGG accuracy was $> 0.1\%$ compared to SLA AGG and MixMo 5C accuracy. Similarly, in Table 2, we compare our method, which uses the WRN-10 architecture with 37.01 million parameters, to SLA and MixMo. Both of these are based on the WRN-14 architecture and use 71.85 million and 71.67 million parameters, respectively. Our AGG accuracy was $> 0.06\%$. Furthermore, in Table 3, although our method with the PRN-2 architecture uses 46.24 million parameters, our AGG accuracy was $> 0.15\%$ compared to SLA AGG and MixMo 5C accuracy, both with the PRN-3 architecture, which use more than double at 101.48 million and 100.87 million parameters, respectively.

f: SUMMARIZATION

To summarize our findings (Tables 1, 2, 3), our method with wider networks can achieve better performance compared to other methods. In particular, although our method incurs similar computational costs to the SLA method, the AGG accuracy gap between our method and the SLA method becomes more pronounced as the width of the networks increases. Furthermore, our method can achieve higher accuracy compared to the previous works with similar computational costs, even with almost half the number of parameters.

C. IMPACT OF INPUT REPETITION RATE

1) PERFORMANCE

a: PREPARATION

In this section, we examined the influence of the input repetition rate q on the performance of our method. To better understand the role of q , we also compared the performance of our method with MIMO when both methods used the same q value.

In Fig. 5, the impact of different input repetition rates q on performance is demonstrated. We separately trained the WRN-10 architecture with our method for each specific $q \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$ on the CIFAR-10 dataset. For each q value, we conducted three independent trials and then computed the average accuracy. In each trial, accuracy was measured using our method's $\arg \max_{c \in \mathcal{Y}} p_{\text{self-agg}}(c|\mathbf{x}, \mathcal{R})$. We followed a similar procedure for MIMO, using the same range of q values and the same number of trials.

b: EXPERIMENTAL RESULTS

There are two notable observations in our experiment illustrated in Fig. 5. First, with our method, the input repetition rate $q = 0.5$ showed the highest accuracy. From this observation, we used $q = 0.5$ as the default value for our method. Second, MIMO showed higher accuracy with lower q values, but our method exhibited a different pattern: performance was better at $q \in \{0.75, 1.0\}$ compared to lower values of $q \in \{0.0, 0.25\}$. This indicates that the pattern of accuracy variations with different q values in our method differs from that observed in MIMO.

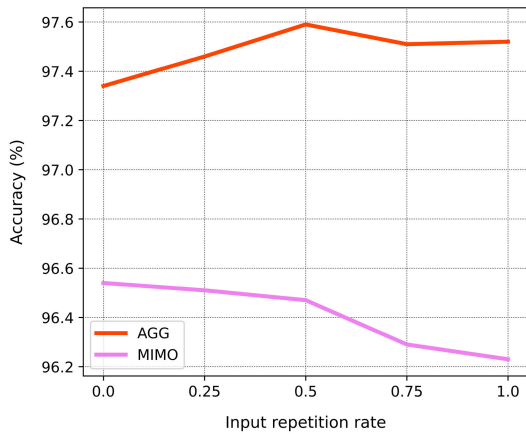


FIGURE 5. We conducted three independent trials for each q value and computed the average of these trials' accuracies for our method, referred to as "AGG". Similarly, the average accuracy obtained from MIMO ($M = 2$) across these trials is denoted as "MIMO".

c: DISCUSSION

In Fig. 5, Our method achieves over 1% higher performance in its optimal case ($q = 0.5$) compared to the best-performing case of MIMO ($q = 0.0$), specifically when using WRN28-10 on the CIFAR-10 dataset. In other words, integrating the benefits of self-supervised learning into a multi-input multi-output structure results in a performance improvement of over 1% compared to scenarios where these benefits are not applied.

2) DIVERSITY BETWEEN MEMBERS

We also measured changes in diversity among individual predictions when varying q , since this diversity in ensemble learning is one of the key factors associated with performance [8], [10], [31], [32].

a: PREPARATION

KL-divergence is a popular metric for measuring the diversity between ensemble members [8], [9], [33], [34]. Note that for two C -dimensional probability vectors, \vec{p} and \vec{q} , KL-divergence can be computed as $D_{KL}(\vec{p}||\vec{q}) = \sum_{c \in \mathcal{Y}} \vec{p}_c (\log \vec{p}_c - \log \vec{q}_c)$. This metric is employed in our analysis to investigate the diversity between predictions of ensemble members in our method.

Before delving into the diversity analysis, we define the following notation: for $r \in \mathcal{R} = \{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$ and $m \in [M] = \{1, 2\}$, the $\{m, r\}$ -individual prediction $\hat{\sigma}_m^{(r)}(\mathbf{x}, \mathbf{x}) \in \mathbb{R}^C$ is defined as

$$\hat{\sigma}_m^{(r)}(\mathbf{x}, \mathbf{x})_c := \frac{\exp((\Psi_m^r \circ f \circ \tau^{(r)})(\mathbf{x}, \mathbf{x})_c)}{\sum_{k=1}^C \exp((\Psi_m^r \circ f \circ \tau^{(r)})(\mathbf{x}, \mathbf{x})_k)} \quad (21)$$

for all $c \in \mathcal{Y}$. Here, performing a single forward pass for a rotation degree $r \in \mathcal{R}$ results in a total of $M = 2$ individual predictions, denoted as $(\hat{\sigma}_m^{(r)}(\mathbf{x}, \mathbf{x}))_{m \in [M]}$.

These individual predictions together are referred to as r -predictions. Furthermore, performing forward passes for all degrees $\{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$ results in a total of $M \times |\mathcal{R}| = 2 \times 4 = 8$ individual predictions.

b: DIVERSITY METRICS

To better understand the diversity of predictions generated by our methods, we measure two quantities: classifier-wise KL-divergence and rotation-wise KL-divergence. The classifier-wise KL-divergence measures the diversity between r -predictions in our method given the same rotation degree, which is defined as

Classifier-wise KL-divergence

$$:= \frac{1}{2 \times |\mathcal{R}|} \sum_{r \in \mathcal{R}} \sum_{m_1 \neq m_2} D_{KL}(\hat{\sigma}_{m_1}^{(r)}(\mathbf{x}, \mathbf{x}) || \hat{\sigma}_{m_2}^{(r)}(\mathbf{x}, \mathbf{x})). \quad (22)$$

where $m_1, m_2 \in \{1, 2\}$. We note that this diversity measure is similar to the diversity measure used in the original MIMO paper [9]: $\sum_{m_1 \neq m_2} D_{KL}(\sigma_{m_1}(\mathbf{x}, \mathbf{x}) || \sigma_{m_2}(\mathbf{x}, \mathbf{x}))/2$ where $m_1, m_2 \in \{1, 2\}$.

The rotation-wise KL-divergence is defined as follows: for each $r \in \mathcal{R}$ and $\vec{p}^{(r)} \in \mathbb{R}^C$ defined as $\vec{p}_c^{(r)} = p_{\text{self-si}}(c|\mathbf{x}, \mathbf{x}, r)$,

Rotation-wise KL-divergence

$$:= \frac{1}{|\mathcal{R}|(|\mathcal{R}| - 1)} \sum_{r_1 \neq r_2} D_{KL}(\vec{p}^{(r_1)} || \vec{p}^{(r_2)}) \quad (23)$$

where $r_1, r_2 \in \mathcal{R}$. This measure does not compare the diversity between individual predictions; instead, it measures the diversity between $(\vec{p}^{(0^\circ)}, \vec{p}^{(90^\circ)}, \vec{p}^{(180^\circ)}, \vec{p}^{(270^\circ)})$. We note that the rotation-wise KL-divergence corresponds to the diversity between multiple forward passes in SLA if we replace $p_{\text{self-si}}$ by p_{si} .

c: EXPERIMENTAL RESULTS

We measured the classifier-wise KL-divergence and rotation-wise KL-divergence of our method by varying the input repetition rate q (Table 4) and compared them with the diversity in MIMO and SLA. We conducted three independent trials for each method—our method, SLA, and MIMO—on the CIFAR-10 dataset, each employing the WRN-10 architecture, and then measured the average of diversity metrics. We note that MIMO uses the input repetition rate q , whereas SLA does not employ q in its approach.

In Table 4, we observed that both the classifier-wise KL-divergence in our method and MIMO KL-divergence decreased with lower q values, such as $q = 0$ or $q = 0.25$. However, the rotation-wise KL-divergence in our method showed a different trend, where it tended to increase as q values increased, particularly peaking at $q = 0.75$. This suggests that a relatively high $q \in \{0.5, 0.75, 1.0\}$, may reduce the diversity between r -predictions in a single forward pass, but it can enhance the diversity between predictions from rotation-wise multiple forward passes. When varying q ,

TABLE 4. Diversity analysis for varying input repetition rates q .

Input repetition rate	0.0	0.25	0.5	0.75	1.0
Classifier-wise KL-Divergence	0.0727	0.0106	0.0026	0.0015	0.0000
MIMO KL-Divergence	0.0770	0.0137	0.0040	0.0005	0.0000
Rotation-wise KL-Divergence	0.0530	0.0606	0.0748	0.1033	0.0880
SLA KL-Divergence	0.0922	N/A	N/A	N/A	N/A

we expect that the influence of rotation-wise KL-divergence in our method contributes to the different performance patterns compared to MIMO.

D. OTHER NUMBER OF MULTIPLE INPUTS/OUTPUTS AND SET OF ROTATION DEGREES

a: PREPARATION

In Section V-A, we initially introduced the hyperparameters of our method with the number of multiple inputs and outputs $M = 2$, and the set of rotation degrees $\mathcal{R} = \{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$. In this section, we explore different values of $M \in \{1, 3, 4\}$ and the alternative set of rotation degrees $\mathcal{R}' = \{0^\circ, 180^\circ\}$.

We evaluated the performance of our method by varying the number of multiple inputs and outputs $M \in \{1, 2, 3, 4\}$ and the set of rotation degrees from $\{\mathcal{R}, \mathcal{R}'\}$. When we use \mathcal{R}' , to maintain a consistent learning rate with $b' = 4$, we construct the training minibatch as follows:

$$\bigoplus_{r \in \{0^\circ, 180^\circ, 0^\circ, 180^\circ\}} \left(\left(R^{(r)}(\mathbf{x}^{\hat{\phi}_1^i(i)}), \dots, R^{(r)}(\mathbf{x}^{\hat{\phi}_M^i(i)}) \right), \right. \\ \left. \times \left(\left(y^{\hat{\phi}_1^i(i)}, r \right), \dots, \left(y^{\hat{\phi}_M^i(i)}, r \right) \right) \right)_{i \in \mathcal{I}}. \quad (24)$$

Note that our method requires two forward passes with \mathcal{R}' for a single inference, unlike \mathcal{R} with four passes.

b: EXPERIMENTAL RESULTS

In Table 5, we conducted three independent trials and averaged the accuracy for each setup on the CIFAR-10 dataset using the WRN-10 architecture for our method. Within this table, we observed that our method achieved better performance using the set of rotation degrees \mathcal{R} compared to the set \mathcal{R}' , and the higher performance was observed when using $M = 2$ across $\{1, 2, 3, 4\}$; hence, we use $M = 2$ and \mathcal{R} as default hyperparameters for our method in Section V-B. We also used $M = 2$ and \mathcal{R} for other datasets and network architectures.

E. OTHER MINIBATCH CONSTRUCTIONS AND MIXING STRATEGIES

Our method combines two input images $\mathbf{x}^1, \mathbf{x}^2$ for training using the mixing strategy $\kappa_1(R^{(r)}(\mathbf{x}^1)) + \kappa_2(R^{(r)}(\mathbf{x}^2))$ for $r \in \mathcal{R}$. This mixed result is then passed into a network. Instead of using the same rotation degree r for both images (i.e., $(R^{(r)}(\mathbf{x}^1), R^{(r)}(\mathbf{x}^2))$), we can try other mixing strategies and training minibatch structures by applying other pairs of rotation degrees to the images (i.e., $(R^{(r_1)}(\mathbf{x}^1), R^{(r_2)}(\mathbf{x}^2))$ for

TABLE 5. Accuracy results from varying number of multiple inputs/outputs and sets of rotation degrees.

M	Degrees	Accuracy of $p_{\text{self-agg}}$
1	$\{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$	97.44
2	$\{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$	97.59
3	$\{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$	97.32
4	$\{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$	97.17
1	$\{0^\circ, 180^\circ\}$	96.86
2	$\{0^\circ, 180^\circ\}$	96.93
3	$\{0^\circ, 180^\circ\}$	96.61
4	$\{0^\circ, 180^\circ\}$	96.43

$r_1, r_2 \in \mathcal{R}$). In this section, we introduce two alternatives and compare their performances with that of our method.

1) OTHER PAIRS OF ROTATION DEGREES

Here, we introduce mixing strategies and training minibatch structures using $(R^{(r_1)}(\mathbf{x}^1), R^{(r_2)}(\mathbf{x}^2))$ for $r_1, r_2 \in \mathcal{R}$.

a: ALTERNATIVE 1

In the first alternative, we use the following mixing strategy:

$$\kappa_1(R^{(\gamma_1^i)}(\mathbf{x}^1)) + \kappa_2(R^{(\gamma_2^i)}(\mathbf{x}^2)) \quad \forall i \in \mathcal{I} \quad (25)$$

where random variables γ_1^i, γ_2^i follows the uniform distribution over $\{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$. Note that this method can create a total of 16 pairs of degree combinations. The training minibatch of Alternative 1 is composed as

$$\bigoplus_{i=1}^b \left(\left(R^{(\gamma_1^i)}(\mathbf{x}^{\hat{\phi}_1^i(i)}), R^{(\gamma_2^i)}(\mathbf{x}^{\hat{\phi}_2^i(i)}) \right), \right. \\ \left. \left(\left(y^{\hat{\phi}_1^i(i)}, \gamma_1^i \right), \left(y^{\hat{\phi}_2^i(i)}, \gamma_2^i \right) \right) \right)_{i \in \mathcal{I}} \quad (26)$$

where

$$\hat{\phi}_m^i(i) = \begin{cases} \text{id}(i) & \text{if } i \in \mathcal{A} \\ \phi_m^t(i) & \text{otherwise} \end{cases}$$

for $m \in \{1, 2\}$ and $\gamma_1^i, \gamma_2^i \sim \text{unif}(\{0^\circ, 90^\circ, 180^\circ, 270^\circ\})$. Recalling that \mathcal{A} is the set formed by uniformly randomly selecting a proportion of q in the minibatch index set \mathcal{I} , and ϕ_m^t is a function that randomly shuffles the indices in \mathcal{A}^c .

b: ALTERNATIVE 2

We similarly construct the training minibatch of Alternative 2. Here, the difference with Alternative 1 is that Alternative 2 applies the same rotation degree for $i \in \mathcal{A}$

as follows:

$$\bigoplus_{t=1}^b \left(\left(R(\gamma_1^i)(\mathbf{x}^{\hat{\phi}_1^t(i)}) \right), R(\hat{\gamma}^i)(\mathbf{x}^{\hat{\phi}_2^t(i)}) \right) \left(\left(y^{\hat{\phi}_1^t(i)}, \gamma_1^i \right), \left(y^{\hat{\phi}_2^t(i)}, \hat{\gamma}^i \right) \right)_{i \in \mathcal{I}} \quad (27)$$

where

$$\hat{\gamma}^i = \begin{cases} \gamma_1^i & \text{if } i \in \mathcal{A} \\ \gamma_2^i & \text{otherwise} \end{cases}$$

and

$$\hat{\phi}_m^t(i) = \begin{cases} \text{id}(i) & \text{if } i \in \mathcal{A} \\ \phi_m^t(i) & \text{otherwise} \end{cases}$$

for $m \in \{1, 2\}$ and $\gamma_1^i, \gamma_2^i \sim \text{unif}(\{0^\circ, 90^\circ, 180^\circ, 270^\circ\})$. The mixing strategy of Alternative 2 can be described as

$$\kappa_1 \left(R(\gamma_1^i)(\mathbf{x}^1) \right) + \kappa_2 \left(R(\hat{\gamma}^i)(\mathbf{x}^2) \right). \quad (28)$$

2) COMPARISON AMONG MIXING STRATEGIES

a: PREPARATION

In this section, we compare the performances of our method with different mixing strategies and training minibatch structures.

In Fig. 6, SI accuracy and AGG accuracy are measured in the same manner as in Section V-B (Table 1, 2, 3), based on the CIFAR-10 dataset using the WRN-10 architecture. Furthermore, for the cases of Alternative 1 and Alternative 2, we denote the accuracy obtained by $\arg \max_{c \in \mathcal{Y}} p_{\text{self-ext}}(c | (\mathbf{x}, \dots, \mathbf{x}), \mathcal{R})$ as AGG16, an extended version of the $p_{\text{self-agg}}$ aggregation method, detailed as follows:

$$\bar{\Psi}_{\text{self-ext}}((\mathbf{x}, \dots, \mathbf{x}), \mathcal{R}) = \sum_{r_1 \in \mathcal{R}} \sum_{r_2 \in \mathcal{R}} \sum_{m=1}^M \frac{(\Psi_m^r \circ f \circ \tau^{(r)})(\mathbf{x}, \dots, \mathbf{x})}{M \times |\mathcal{R}| \times |\mathcal{R}|} \quad (29)$$

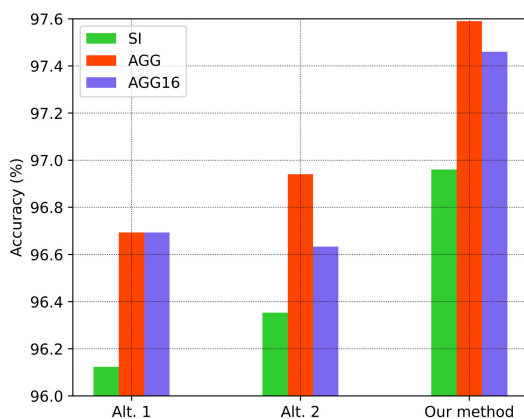


FIGURE 6. Accuracy variations based on the mixing strategies for multiple inputs and training minibatch structures.

$$p_{\text{self-ext}}(c | (\mathbf{x}, \dots, \mathbf{x}), \mathcal{R}) = \frac{\exp(\bar{\Psi}_{\text{ext}}((\mathbf{x}, \dots, \mathbf{x}), \mathcal{R})_c)}{\sum_{k=1}^C \exp(\bar{\Psi}_{\text{ext}}((\mathbf{x}, \dots, \mathbf{x}), \mathcal{R})_k)} \quad (30)$$

for $M = 2$ and $\mathcal{R} = \{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$.

b: EXPERIMENTAL RESULTS

In Fig. 6, we observed that for both Alternative 1 and Alternative 2, the AGG accuracy is at least the AGG16 accuracy, even though AGG16 requires 12 more forward passes than AGG. Similarly, our method also showed higher AGG accuracy compared to AGG16 accuracy. Furthermore, in Fig. 6, we compared AGG accuracies of our method, Alternative 1, and Alternative 2. The AGG accuracy of our method was higher by 0.896% than that of Alternative 1 and by 0.649% higher than Alternative 2.

VI. CONCLUSION

The key concept of our method is the integration of the benefits of self-supervised learning into a multi-input multi-output structure, enabling both rotation-wise and classifier-wise predictions for inference to boost the ensemble effect. This structure shows that a multi-input multi-output structure can synergize with self-supervised learning techniques. Specifically, our method not only improves performance in wider networks compared to previous works but also achieves higher performance while using an architecture with nearly half the number of parameters.

Currently, our objective is centered on integrating benefits from rotation-based self-supervised learning into a multi-input multi-output structure. However, in our future works, we plan to develop an implicit deep ensemble method capable of robust inference across all degrees of rotation transformation.

REFERENCES

- [1] S. I. Mirzadeh, A. Chaudhry, D. Yin, H. Hu, R. Pascanu, D. Gorur, and M. Farajtabar, "Wide neural networks forget less catastrophically," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 15699–15717.
- [2] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proc. Brit. Mach. Vis. Conf.*, 2016.
- [3] D. Barber and C. M. Bishop, "Ensemble learning in Bayesian neural networks," *Nato ASI F Comput. Syst. Sci.*, vol. 168, pp. 215–238, Jan. 1998.
- [4] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1050–1059.
- [5] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.
- [6] Y. Wen, D. Tran, and J. Ba, "BatchEnsemble: An alternative approach to efficient ensemble and lifelong learning," in *Proc. Int. Conf. Learn. Represent.*, 2019.
- [7] T. Whitaker and D. Whitley, "Prune and tune ensembles: Low-cost ensemble learning with sparse independent subnetworks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 36, 2022, pp. 8638–8646.
- [8] S. Liu, T. Chen, Z. Atashgahi, X. Chen, G. Sokar, E. Mocuano, M. Pechenizkiy, Z. Wang, and D. Constantin Mocuano, "Deep ensembling with no overhead for either training or testing: The all-round blessings of dynamic sparsity," 2021, *arXiv:2106.14568*.

- [9] M. Havasi, R. Jenatton, S. Fort, J. Zhe Liu, J. Snoek, B. Lakshminarayanan, A. M. Dai, and D. Tran, "Training independent subnetworks for robust prediction," 2020, *arXiv:2010.06610*.
- [10] A. Ramé, R. Sun, and M. Cord, "MixMo: Mixing multiple inputs for multiple outputs via deep subnetworks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 803–813.
- [11] H. Lee, S. J. Hwang, and J. Shin, "Self-supervised label augmentation via input transformations," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5714–5724.
- [12] J. D. M.-W. C. Kenton and L. K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT*, vol. 1, 2019, p. 2.
- [13] J. Gui, T. Chen, J. Zhang, Q. Cao, Z. Sun, H. Luo, and D. Tao, "A survey on self-supervised learning: Algorithms, applications, and future trends," 2023, *arXiv:2301.05712*.
- [14] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1422–1430.
- [15] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Cham, Switzerland: Springer, 2016, pp. 69–84.
- [16] G. Larsson, M. Maire, and G. Shakhnarovich, "Colorization as a proxy task for visual understanding," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 840–849.
- [17] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," in *Proc. ICLR*, 2018.
- [18] D. Hendrycks, M. Mazeika, S. Kadavath, and D. Song, "Using self-supervised learning can improve model robustness and uncertainty," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.
- [19] X. Zhai, A. Oliver, A. Kolesnikov, and L. Beyer, "S4L: Self-supervised semi-supervised learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1476–1485.
- [20] M. A. Ganaie, M. Hu, A. Malik, M. Tanveer, and P. Suganthan, "Ensemble deep learning: A review," *Eng. Appl. Artif. Intell.*, vol. 115, Jan. 2022, Art. no. 105151.
- [21] S. Yun, D. Han, S. Chun, S. J. Oh, Y. Yoo, and J. Choe, "CutMix: Regularization strategy to train strong classifiers with localizable features," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 6022–6031.
- [22] A. Kolesnikov, X. Zhai, and L. Beyer, "Revisiting self-supervised visual representation learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 1920–1929.
- [23] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009. [Online]. Available: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>
- [24] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of ImageNet as an alternative to the CIFAR datasets," 2017, *arXiv:1707.08819*.
- [25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.
- [26] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.*, Amsterdam, The Netherlands. Cham, Switzerland: Springer, 2016, pp. 630–645.
- [27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012.
- [28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
- [29] D. Shanmugam, D. Blalock, G. Balakrishnan, and J. Guttag, "Better aggregation in test-time augmentation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 1194–1203.
- [30] I. Kim, Y. Kim, and S. Kim, "Learning loss for test-time augmentation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 4163–4174.
- [31] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Proc. J. Artif. Intell. Res.*, vol. 11, pp. 169–198, Jan. 1999.
- [32] A. Rame and M. Cord, "DICE: Diversity in deep ensembles via conditional redundancy adversarial estimation," 2021, *arXiv:2101.05544*.
- [33] S. Fort, H. Hu, and B. Lakshminarayanan, "Deep ensembles: A loss landscape perspective," 2019, *arXiv:1912.02757*.
- [34] G. Nam, J. Yoon, Y. Lee, and J. Lee, "Diversity matters when learning from ensembles," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 8367–8377.



JAEHOON PARK (Student Member, IEEE) received the B.E. degree in statistics from Sungkyunkwan University, Seoul, Republic of Korea, in 2021. He is currently pursuing the M.S. degree with the Graduate School of Artificial Intelligence, Korea University, Seoul. His research interests include deep ensembles and self-supervised learning.

...