IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 22 December 2022, accepted 9 January 2023, date of publication 10 February 2023, date of current version 28 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3244072

RESEARCH ARTICLE

EUNNet: Efficient UN-Normalized Convolution Layer for Stable Training of Deep Residual Networks Without Batch Normalization Layer

KHANH-BINH NGUYEN[1], (Student Member, IEEE), JAEHYUK CHOI[2], (Senior Member, IEEE), AND JOON-SUNG YANG[3,4], (Senior Member, IEEE)

[1]Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, South Korea
[2]Department of Semiconductor Systems Engineering, Sungkyunkwan University, Suwon 16419, South Korea
[3]School of Electrical and Electronic Engineering, Yonsei University, Seoul 03722, South Korea
[4]Department of Systems Semiconductor Engineering, Yonsei University, Seoul 03722, South Korea

Corresponding author: Joon-Sung Yang (js.yang@yonsei.ac.kr)

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Korean Government through the Ministry of Science and ICT (MSIT) under Grant 2021-0-00106 and Grant 2022-0-00971.

ABSTRACT Batch Normalization (BN) is an essential component of the Deep Neural Networks (DNNs) architectures. It helps improve stability, convergence, and generalization. However, studies are showing that BN might introduce several concerns. Although there are methods for training DNNs without BN using proper weight initialization, they require several learnable scalars or accurate fine-tuning to the training hyperparameters. As a result, in this study, we aim to stabilize the training process of un-normalized networks without using proper weight initialization and to minimize the hyperparameters fine-tuning step. We propose EUNConv, an **E**fficient **UN**-normalized **Conv**olutional layer, which helps train un-normalized Deep Residual Networks (ResNets) by using hyperparameters of the normalized networks. Furthermore, we introduce **E**fficient **UN**-normalized Neural **Net**work (EUNNet), which replaces all of the conventional convolutional layers of ResNets with our proposed EUNConv. Experimental results show that the proposed EUNNet achieves the same or even better performance than previous methods in various tasks: image recognition, object detection, and segmentation. In particular, EUNNet requires less fine-tuning and less sensitivity to hyperparameters than previous methods.

INDEX TERMS Deep neural networks, batch normalization, overfitting, computer vision, image classification, object detection.

I. INTRODUCTION

Deep Neural Networks have made significant strides in recent years, achieving an extraordinary performance on many tasks such as image recognition [1], speech recognition [2], and natural language processing [3]. The primary contributor to this achievement is the development of novel neural network designs and training methodologies. However, DNNs face an issue of vanishing/exploding gradient, which often occurs during training through backpropagation [4], [5], [6]. Batch Normalization (BN) [7] helps resolve this problem by normalizing signals throughout the batch dimension. The motivation for BN is to reduce the internal covariate shift. Santurkar et al. [6] also shows that BN could have benefits in terms of regularizing effects. The regularizing effects allow efficient training with a larger batch size, eliminate the mean-shift issue, and downscale the residual branch. Thus, BN stabilizes the learning process, enables higher learning rate training, allows faster convergence, and improves generalization. As a result, many variants have been introduced, such as Layer Normalization for recurrent neural networks [8], Instance Normalization (IN) [9] for stylization, Group Normalization (GN) [10] for faster training without dependency on batch size, etc. They have become the core components of the state-of-the-art architectures for many tasks [11].

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

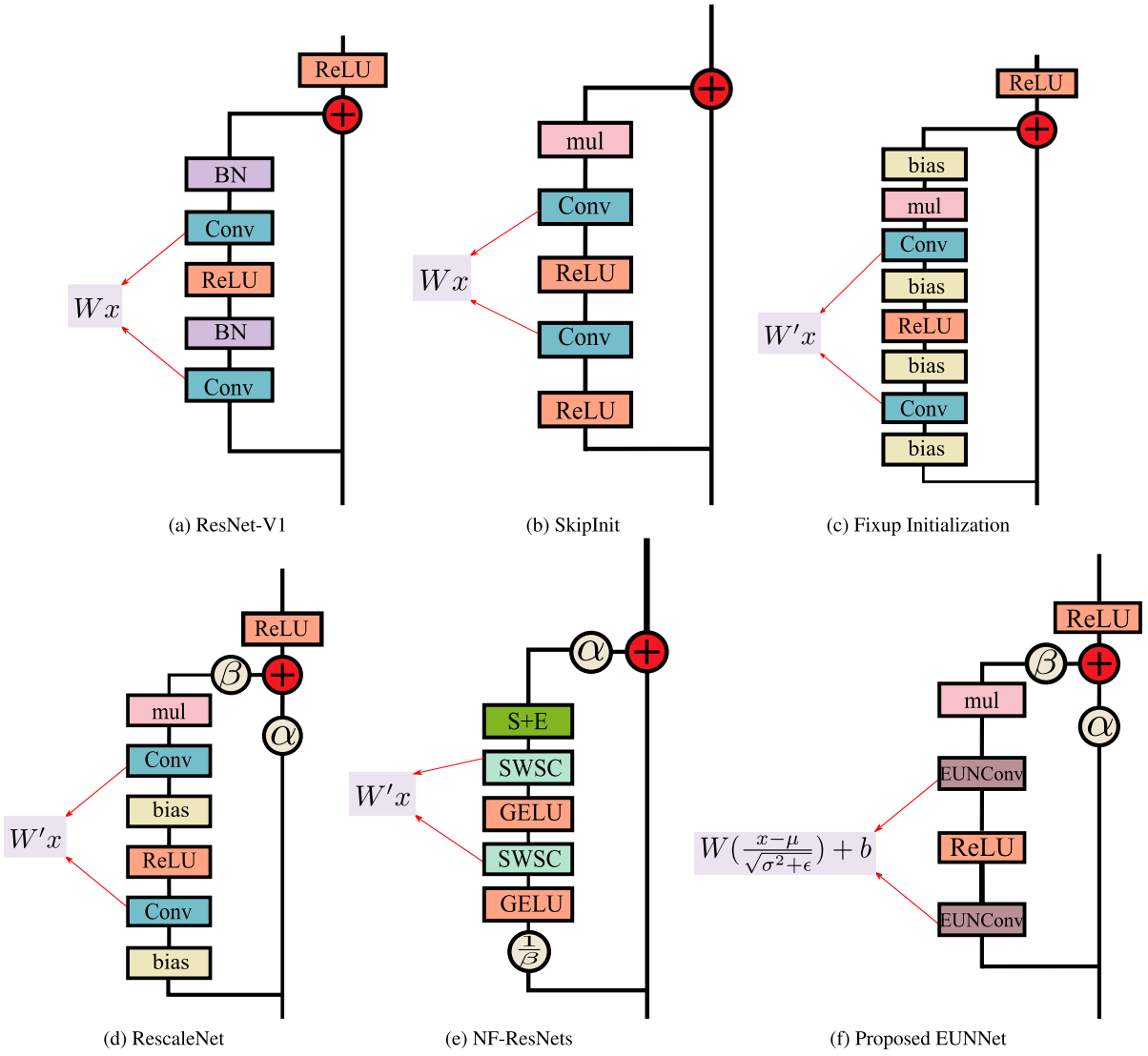


FIGURE 1. **Basic block visualization of ResNets, SkipInit, Fixup, RescaleNet, NF-ResNets and our proposed method - EUNNet. *W* is original weights of the convolutional layers, *W'* is the downscaled weights of the convolutional layers in the Fixup and RescaleNet methods, *b* is convolutional layers bias, α and β are constants. 'SWSC' indicates Scaled Weight Standardization Convolutional layer. 'S+E' is the Squeeze and Excite module. Best view with color.**

TABLE 1. **Prior approaches comparison.**

Method	Approach
SkipInit [12]	learnable multiplier scalar
Fixup [13]	learnable bias + multiplier scalar + weight downscaling
RescaleNet [14]	learnable bias + multiplier scalar + α, β + weight downscaling
NF-ResNets [15]	AGC + multiplier scalar + weight downscaling
EUNNet (ours)	convolutional bias + multiplier scalar (optional) + α, β (optional)

Despite the empirical success of training DNNs with BN, studies show that BN may perform worse when applied with very small batch sizes [10], [16], [17], [18]. BN also has a memory overhead and is a costly computational primitive. Additionally, it introduces several hidden hyperparameters that need to be tuned, causes numerous implementation problems in distributed training [19], and introduces inconsistent behaviors of the model throughout training. The performance of normalized networks can also suffer from the vanishing/exploding gradient problem if the batch data has a high variance during training.

There has been effort to remove BN from DNNs to lessen these BN-related problems. Fixup Initialization (Fixup) [13] demonstrates that DNNs without BN can be trained with carefully initializing the weights of convolutional layers by downscaling their weight values in residual paths, which results in downscaling the variance of residual blocks. This simulates the impact of BN [12]. Although the results of Fixup prove their theorem with competitive performance to normalized networks, Fixup also introduces several settings

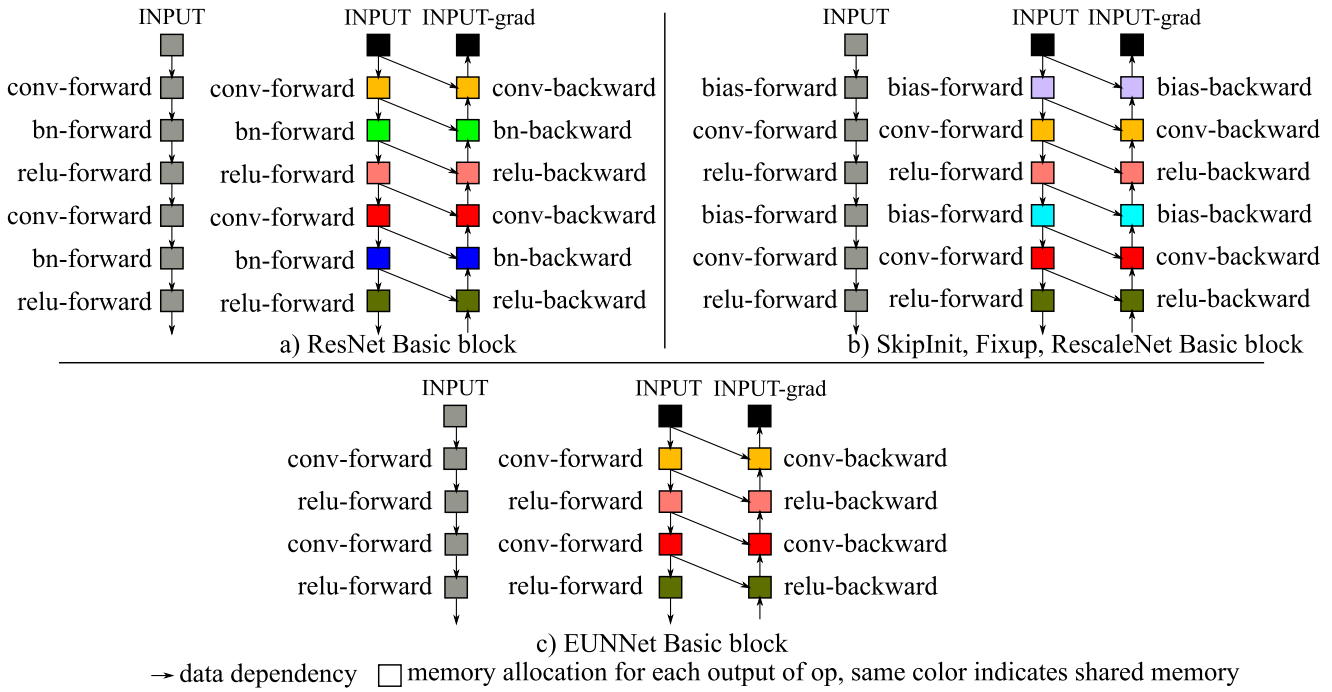


FIGURE 2. **Memory allocation flow of forward and backward propagation for residual block during training. Best view with color.**

to make it work (*e.g.*, biases, custom learning rate for different parameters, and etc). In [12], authors introduce a related work called SkipInit. It simply removes BN layers, then places a multiplier scalar at the end of each residual path (ResNets). Although this method is significantly easier than Fixup and can train un-normalized networks, its performance is unsatisfactory. When used with a very large batch size, SkipInit becomes less efficient. A different approach, RescaleNet, is suggested by [14]. The fundamental concept is to employ several scales for the convolutional layer weights together with additional constants α and β. The value of residual blocks is defined as:

$$x_l = \alpha_l x_{l-1} + \beta_l \mathcal{F}_l(x_{l-1}), \alpha_l + \beta_l = 1 \tag{1}$$

where l is the layer index, this ensures the variance of the residual block is unchanged and ranges between 0 to 1. Another study leverages the signal propagation at initialization on the forward pass in ResNets [15]. The authors propose Scaled Weight Standardization (SWS), a variant of the Weight Standardization [20] to remove the mean-shift. They also introduced "Normalizer-Free" ResNets (NF-ResNets) which match the performance of normalized ResNets. However, NF-ResNets are not stable at large batch sizes and are inefficient with some architectures, such as EfficientNets [21]. In recent work [22], the authors propose Adaptive Gradient Clipping (AGC), which clips the gradients based on the *unit-wise ratio of gradient norms to parameter norms*. AGC helps train "Normalizer-Free" networks stably with larger batch sizes. Moreover, the author designs a family of networks called NFNets. NFNets set the new state-of-the-art on ImageNet, as they score higher accuracy than normalized networks and are significantly faster. NFNet is the first un-normalized network that achieves state-of-the-art with a result greater than 80% validation accuracy on the ImageNet dataset. We summarize the differences between prior approaches in Table 1.

We categorize SkipInit, Fixup, and RescaleNet as *weight initialization downscaling* techniques in this study. Even though downscaling initial weight values may help train un-normalized networks to match the performance of normalized networks, there is still a performance gap in un-normalized networks. Downscaling initial weight values is equivalent to downscaling the initial gradients. It could limit the learning capability of the network in the early stages. Fixup, SkipInit, and RescaleNet overcome this limitation by using a multiplier scalar at the end of each residual path to recover the gradients while maintaining an identical variance for residual blocks. When calculating the gradients, it creates another problem. Input values generally need to be saved in both forward and backward propagation when passing them via a layer. As a result, the network needs more memory to store the values as the number of layers increases. Therefore, adding a learnable scalar as an individual network component is inefficient for the networks, which results in several memory shortages during training. Additionally, many variables, such as learning rate, weight decay, etc., must be precisely adjusted to make their approaches work effectively.

In this paper, we propose EUNConv, a method for effectively training un-normalized networks. EUNConv does not clip the gradients or downscale the initial weight values, which contrasts to the prior methods. EUNConv lowers the memory footprint overhead for the training process by

employing convolutional layer biases rather than adding individual learnable scalars. Furthermore, we present our EUNNet architecture that is based on ResNets without BN layers, where we replace the traditional convolutional layers with EUNConv. With minimal effort for fine-tuning the hyperparameters, EUNNet can perform comparably to normalized networks. We conduct extensive experiments for a wide range of tasks, such as segmentation, object identification, and image classification, to validate our methodology. The experiment results demonstrate that our suggested solution outperforms SkipInit, Fixup, and RescaleNet in terms of performance and stability. We also verify the contribution of learnable scalars for each method by sequentially removing them from the architecture. The results show that, in contrast to other methods, ours does not significantly rely on the learnable scalars.

The contributions of the proposed method are:

- We propose a new approach to train un-normalized deep residual networks in a much more stable and effective way than conventional methods.
- We reduce the need for fine-tuning hyperparameters, which is very sensitive and time-consuming. As a result, we can directly train the un-normalized network with the optimal hyperparameters of the normalized networks.
- We discuss the role of additional learnable scalars and their usage as individual layers. In addition, we prove that the proposed method can remove them to save the memory allocation without hurting performance.

The rest of the paper is organized as follows. Section II and III gives a background and motivation of this work. The proposed method is described in Section IV. The evaluation and results of the proposed work are presented in Section V, VI, and Section VII concludes the paper.

II. RELATED WORK

A. NORMALIZATION METHODS

As introduced, Batch Normalization (BN) [7] has become an essential component of DNN architectures. BN normalizes weights or activations through mini-batch sample statistics (mean and standard deviation), then scales and shifts the normalized outputs (using λ and β). Batch Normalization consists of 2 steps:

1) Normalizing a batch of input by first subtracting its mean μ, then dividing it by its standard deviation σ
2) Scaling the normalized output by a factor γ and shifting it by a factor β, where γ and β are the parameters of the BN layer.

The Batch Normalization formula is $y_l = BN(x_l) = \gamma\hat{x}_l + \beta = \gamma\frac{x_l-\mu_\mathcal{B}}{\sqrt{\sigma^2_\mathcal{B}+\epsilon}} + \beta$, with the mean and variance of the layer input for current batch $\{x_l\}_{l=1}^m$ are:

$$\mu_\mathcal{B} = \frac{1}{m}\sum_{l=1}^{m} x_l, \quad \sigma^2_\mathcal{B} = \frac{1}{m}\sum_{l=1}^{m}(x_l - \mu_\mathcal{B})^2 \tag{2}$$

where ϵ is a small constant for numerical stability and m is the batch size. Layer Normalization (LN) [8] computes normalization statistics by layer dimension and is popular for use within recurrent neural networks (*e.g.*, Transformers [23]). Instance Normalization (IN) is introduced in [9] to help remove instance-specific contrast information from the content image, which usually happens in stylization tasks. Group Normalization (GN) [10] computes normalization statistics over groups of channels, making it flexible to switch into LN or IN by configuration.

B. INITIALIZATION

Initialization is essential in training DNNs. A proper initialization could boost the performance of DNNs further than random initialization. Xavier Initialization [24] calculates the initial values based on the number of input and output channels in a layer, which results in a better performance than the random initialization. Kaiming Initialization (He Initialization) [1] is an initialization for DNNs that contains the non-linearity of activation functions such as ReLU activations [25] or its variants (Leaky ReLU, GELU, etc). Despite the significant gains of Xavier and Kaiming Initialization, they fail to train Deep Residual Networks [13], [26], [27] without the Batch Normalization layers.

III. PRELIMINARIES

The intuition of ResNets is to compute the output activations through a sequence of residual blocks that downscale but still maintain the characteristics of features from the previous layer. Let x_0 be the input to the network, x_l be the output after the l^{th} residual path, and $\mathcal{F}_l$ be the corresponding residual block. The l^{th} residual path is formulated as $x_l = x_{l-1} + \mathcal{F}_l(x_{l-1})$. Each residual block $\mathcal{F}_l$ in turn is a multi-layer network. It could be a convolutional network or a multi-layer perceptron (MLP). Besides, ResNets can be an ensemble of many small shallow networks [28]. Activations are generally ReLUs, as will be assumed in this paper. We will refer to ResNets that do not include BN layers as un-normalized ResNets. Overall, the following two key challenges in training ResNets motivate our work.

A. THE PROBLEM OF EXPLODING VARIANCE

Random weight initialization could result in unstable variances (vanishing/exploding gradients) of the activations for very deep neural networks. There are a variety of proposed initialization strategies to deal with this issue [24], [29]. Xavier Initialization [24] estimates the standard deviation of initial parameters based on the number of input and output channels in a layer. However, Xavier Initialization has problems when initializing networks with a ReLU activation function [1]. Therefore, for ReLU activation networks, Kaiming Initialization [1] is proposed. Kaiming Initialization stabilizes the variances and keeps them constant through the layers of DNNs. In ResNets, the variance of the input and output of residual blocks would remain identical by using this

initialization, i.e., $Var(\mathcal{F}_l(x_{l-1})) = Var(x_{l-1})$, where $Var(x)$ denotes the variances for each component of x. However, since the correlations between x_l and $\mathcal{F}_l(x_{l-1})$ are small, then after each residual path we get:

$$\begin{aligned} Var(x_l) &= Var(x_{l-1} + \mathcal{F}_l(x_{l-1})) \\ &\approx Var(x_{l-1}) + Var(\mathcal{F}_l(x_{l-1})) \\ &= 2Var(x_{l-1}) \end{aligned} \tag{3}$$

Thus, the variance doubles after each residual block, and increases exponentially as we stack up multiple residual blocks, leading to an identical problem in the backward pass during training.

B. TRAINING UN-NORMALIZED NETWORKS

Fixup proposes training un-normalized networks by downscaling the weights of the convolutional layers in the residual path and initializing the residual path to zero at the beginning of the training phase. SkipInit [12] also initializes the residual path to zero by simply adding a small-value multiplier scalar at the end of the residual path (normally zero), represented by $x_l = x_{l-1} + \alpha\mathcal{F}_l(x_{l-1})$, where $\mathcal{F}_l$ is the function computed by the l^{th} residual branch. Unlike Fixup and SkipInit, RescaleNet [14] addresses this by solving the "Dead" ReLU problem. Using α and β scalars, they migrate the formula of residual blocks into:

$$x_l = \alpha_l x_{l-1} + \beta_l \mathcal{F}_l(x_{l-1}), \quad \alpha_l^2 + \beta_l^2 = 1 \tag{4}$$

Therefore, the variance of the output of residual blocks becomes:

$$\begin{aligned} Var(x_l) &\approx \alpha_l^2 Var(x_{l-1}) + \beta_l^2 Var(\mathcal{F}_l(x_{l-1})) \\ &\approx Var(x_{l-1}) \end{aligned} \tag{5}$$

RescaleNet guarantees stable variance during training and the results are much better than Fixup and SkipInit.

Despite its simplicity and efficiency, the performance of SkipInit is not satisfactory when used with a very large batch size. Besides, Fixup and RescaleNet require further modifications to the learning rate and weight decay, as well as using extra learnable scalars. Without modifications, the training procedure could fail, and the network might not be trained properly since the vanishing/exploding gradient still occurs. Brock et al. [15] proposes NF-ResNets, a "Normalizer-Free" ResNets. NF-ResNets employ a residual block with the formula of $x_l = x_{l-1} + \alpha\mathcal{F}_l(x_{l-1}/\beta_{l-1})$, where x_l indicates the input to the l^{th} residual block, $\mathcal{F}_l$ indicates the function computed by the l^{th} residual branch. The scalar α specifies the rate at which the variance of the activations increases after each residual block, typically set to a small value of 0.2. The scalar β_l is determined by predicting the standard deviation of the input to the l^{th} residual block, $\beta_l = \sqrt{Var(x_l)}$, where $Var(x_l) = Var(x_{l-1}) + \alpha^2$. However, there is an exception for transition blocks (where spatial downsampling occurs), the skip path operates on the downscaled input of x_l/β_l, and the expected variance is reset after the transition block to $Var(x_l) = 1 + \alpha^2$. In addition, they propose Scaled Weight Standardization to re-parameterize the convolutional layers with:

$$\begin{aligned} \hat{W}_{i,j} &= \gamma \frac{W_{i,j} - \mu_i}{\sigma_i \sqrt{N}}, \\ \mu_i &= \frac{1}{N} \sum_j W_{ij}, \\ \sigma_i^2 &= \frac{1}{N} \sum_j \left(W_{ij} - \mu_i\right)^2, \end{aligned} \tag{6}$$

where N denotes the fan-in of convolutional filters. NF-ResNets show significant results and outperform their normalized counterparts when the batch size is very small. However, the performance gets worse when the batch size increases. The training process of NF-ResNets is also unstable, and their proposed methods are not compatible with state-of-the-art networks such as EfficientNets [21].

Brock et al. [22] proposes Adaptive Gradient Clipping (AGC) to clip the gradients and it enables NF-ResNets to be trained efficiently with large batch sizes as well as to stabilize the training process. Specifically, the gradient vector $G = \partial L/\partial\theta$ would be clipped as below before updating θ:

$$G \rightarrow \begin{cases} \lambda \dfrac{G}{\|G\|}, & \text{if } \|G\| > \lambda \\ G & \text{otherwise.} \end{cases} \tag{7}$$

where G is the gradient vector, L is the loss, θ is a vector with all model parameters, and γ is clipping threshold, a hyperparameter which must be carefully fine-tuned. Although their proposed NFNets achieve state-of-the-art for the un-normalized networks, it introduces a lot of work and hyperparameters, which require careful tuning, such as clipping threshold γ, multiplier scalars α and β or else it would lead to unstable training. It also has a significantly large number of floating-point operations (FLOPs) and a large number of parameters.

IV. EFFICIENT UN-NORMALIZED NEURAL NETWORK

This section introduces EUNConv, a convolutional layer variant we propose to stabilize residual block variance and prevent gradient issues like vanishing/exploding. Additionally, we present EUNNet, our adaptation of the un-normalized ResNet architecture. The initial weight values of EUNNet are similar to the normalized network instead of being downscaled like previous methods. Because it can take advantage of the optimal hyperparameters from normalized networks, EUNNet is superior to earlier techniques.

A. EFFICIENT UN-NORMALIZED CONVOLUTIONAL LAYER

Feature standardization is a technique that centers the input into zero-mean and unit variance. It is widely used in many machine learning algorithms (*e.g.*, support vector machines, logistic regression, and artificial neural networks). Standardization is one of the core operations of BN, which has been proven to speed up and stabilize the learning process of neural networks [4], [5], [6].

TABLE 2. Results on CIFAR-10 with ResNet-110 (mean/median of 5 runs).

Dataset	ResNet-110	Normalization	Weight downscaling	Bias type	Acc (%)
CIFAR-10	w/BatchNorm	✓	✗	BN biases	93.39
	w/SkipInit	✗	✗	None	90.48
	w/Fixup	✗	✓	Learnable scalars	92.76
	w/RescaleNet	✗	✓	Learnable scalars	92.88
	w/EUNNet	✓	✗	Convolution biases	**93.42**

EUNConv is created by fusing feature standardization into convolutional layers. We first normalize the input of the convolutional layer into zero-mean and unit variance to keep the effect of reducing internal covariate shift from BN. Then we enable the convolutional bias instead of using the learnable scalars. Hence, our EUNConv output formula becomes:

$$\begin{aligned} y_l &= W_l \hat{x_l} + b_l \\ &= W_l \frac{x_l - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} + b_l \\ &= \frac{W_l}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} x_l - \frac{W_l \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} + b_l, \end{aligned} \tag{8}$$

where $\mu_{\mathcal{B}}$ and $\sigma^2_{\mathcal{B}}$ are the mean and variance of the input, ϵ is a small constant for numerical stability.

Assuming that the weights of each convolutional layer is initialized by Kaiming Initialization [29], we can easily obtain that:

$$\begin{aligned} \mathbb{E}(\mathcal{F}_l(\hat{x}_{l-1})) &= \mathbb{E}(\hat{x}_{l-1}) = 0, \\ Var(\mathcal{F}_l(\hat{x}_{l-1})) &= Var(\hat{x}_{l-1}) = 1 \end{aligned} \tag{9}$$

The variance of the residual block using EUNConv can be expressed as:

$$\begin{aligned} Var(x_l) &\approx Var(x_{l-1}) + Var(\mathcal{F}_l(\hat{x}_{l-1})) \\ &\approx Var(x_{l-1}) + 1 \\ &\ll 2Var(x_{l-1}) \end{aligned} \tag{10}$$

As a result, the model can still be trained normally without experiencing vanishing/exploding gradient even after replacing the conventional convolutional layers by EUNConv. In addition, we use Equation 4 to ensure that $Var(x_l)$ and $Var(x_{l-1})$, respectively, are the same. As a result, the variance is as follows:

$$Var(x_l) \approx \alpha_l^2 Var(x_{l-1}) + \beta_l^2 \tag{11}$$

Notably, our work is different from the scheme of placing BN layer before the convolutional layer. Here, we conduct the standardization step without using γ and β scalars of BN. In addition, EUNConv does not store the training statistics for inference such as BN. Therefore using EUNConv is not similar to using BN before convolutional layer.

B. MEMORY FLOW IN DEEP LEARNING TRAINING

There are two main types of information that need to be stored during training. Firstly, the information necessary to backpropagate the error (gradients of the activation w.r.t the loss). Secondly, the information necessary to compute the gradient of the model parameters.

Specifically, the layer with learnable parameters stores its input until the backward pass. Therefore, every layer will need to store its input until it can compute the gradient of its parameters. In addition, because we need to backpropagate the error to the input by the chain rule, layers with learnable parameters also need to store their data. Thus, each layer stores data twice for both forward and backpropagation.

Since the cost of storing feature maps and their gradients scales linearly with the depth of the network [30], this raises an issue with Fixup and RescaleNet as they use biases as individual layers. Hence, this creates a memory bottleneck for forward and backward propagation operations, as shown in Figure 2.

C. RESIDUAL BLOCK

In Figure 1, we visualize the residual block of our proposed EUNNet compared with SkipInit, Fixup, RescaleNet, and NF-ResNets. Through extensive experiments, we found that using α, β, and *scale* scalars of RescaleNet inside our architecture results in better performance. Notably, our architecture can still be trained normally without these scalars. We explore this in Section V-A4, where we sequentially remove each of these scalars and evaluate the model on the CIFAR-100 dataset.

V. EVALUATION

In this section, we conduct experiments for image classification task on 2 datasets: CIFAR-10 [31] and ImageNet [32]. Then, we perform the experiments for the downstream tasks, which are object detection and segmentation, on the COCO dataset [33].

A. IMAGE CLASSIFICATION

1) CIFAR-10

First, the CIFAR-10 dataset is tested using ResNets-110 with default hyperparameters.[1] Results are shown in Table 2.

[1] https://github.com/pytorch/vision

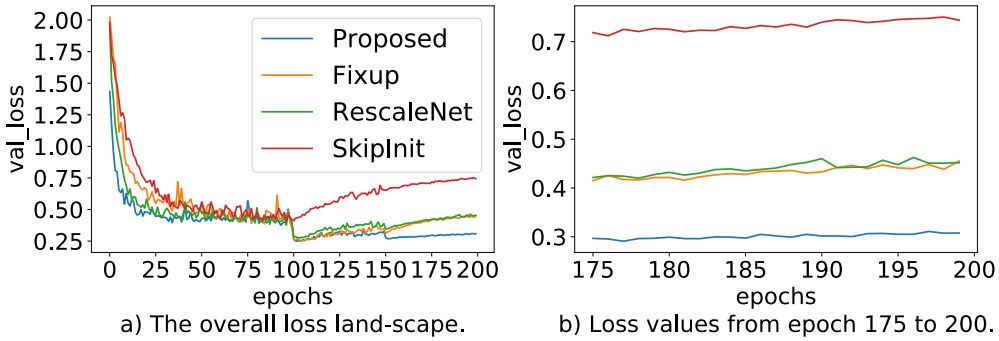


FIGURE 3. Validation loss comparison of the proposed method and Fixup [13], RescaleNet [14], SkipInit [12] on CIFAR-10 dataset. Best view with color.

TABLE 3. Results on ImageNet with ResNet-50 (higher is better). Results are captured from the original paper. '*' and '**' indicate results that are conducted with batch size of 1024 and 4096, respectively.

Model	Method	Regularization	Input normalization	Weight downscaling	Gradient downscaling	Top-1 (%)
ResNet-50	BatchNorm	None	✓			76.4
		Mixup				76.7
	GroupNorm	None	✓			76.1
	SkipInit	None				74.9
		Dropout				75.6
	Fixup	None		✓		72.4
		Mixup				76.0
		Dropout				75.5
	NF-ResNets*	None			✓	75.8
		Dropout				**76.8**
	RescaleNet*	None		✓		74.3
		Mixup				76.4
		Dropout				**76.6**
	EUNNet	None	✓			75.4
		Mixup				76.4
		CutMix				**76.6**
SE-ResNeXt-50	NFNet-F0**	None				80.4
		Mixup				82.2
		CutMix				83.6

The CIFAR-10 [31] dataset contains 50k training and 10k validation images of low-resolution (32×32) which are categorized into 100 categories. The experiments are conducted on 5 runs for each model, and then take the mean/-median of 5 runs for comparison.

The hyperparameters follow the same settings as reported in [14] as a stochastic gradient descent (SGD) optimizer with a momentum coefficient of $m = 0.9$, weight decay coefficient of 5×10^{-4}. We apply standard data augmentation techniques, including normalization, padding, random cropping, and vertical flipping. Networks are initialized with Kaiming Initialization. We train the model for a total of 200 epochs. An initial learning rate of 0.1 is used and a step learning rate scheduler reduces the learning rate by 10 at epoch 100 and 150. A batch size of 128 is used for both the training and validation phases. However, each of the methods has its different settings, as follows:

a: SkipInit

SkipInit uses the same default hyperparameters as ResNets. However, because it removes BN layers, it is configured to reduce a learning rate by a factor of 2 every 5 epochs in the middle of training.

b: Fixup

Similar to SkipInit, weights are initialized by Kaiming Initialization [1], however, the weights on residual paths are downscaled by $L^{-\frac{1}{2m-2}}$ where L is the number of residual blocks in the network and m is a very small integer. The learning rate of the learnable scalars is $\frac{1}{10}$ of the initial learning rate.

c: RescaleNet

Unlike SkipInit and Fixup, RescaleNet uses the same learning rate for all parameters in the training, however, it sets the weight decay of biases and multiplier scalars to 0. It also

TABLE 4. Results on CIFAR-100 with ResNet-110 when sequentially removing *scale*, α and β scalars.

Dataset	ResNet-110	scale	α & β	Accuracy (%)
CIFAR-100	w/SkipInit	✗		-
	w/Fixup	✓		72.01
		✗		70.13
	w/RescaleNet	✓	✓	71.34
		✗	✓	71.05
		✗	✗	-
	w/EUNNet	✓	✓	**72.39**
		✗	✓	71.89
		✗	✗	71.72

downscales the weights on residual paths by $(\frac{L}{k+L})^{\frac{2}{N}}$ where L is the number of residual blocks in the network, k is the index of the layer, and N is the number of layers in the residual path.

The results show that our proposed EUNNet performance is close to the BN model and better than the models trained with Fixup, SkipInit, or RescaleNet. EUNNet can recover the normalization effect of BN to train un-normalized networks and surpasses the best performance of RescaleNet. This provides us with a solid evidence that downscaling initial weight values is sub-optimal and is not the only solution to train un-normalized networks.

2) ImageNet

ImageNet is a dataset that contains 128k training images and 50k validation images with 1000 categories. On the ImageNet dataset [32], we evaluate EUNNet-50 and compare it with Fixup, SkipInit, and RescaleNet using the ResNet-50 architecture.

Models are trained for 100 epochs with the same hyperparameters as used in the CIFAR-10 experiments. The learning rate is decayed by 10 at epoch 30, 60, and 90, respectively. We use weight decay with a coefficient of 10^{-4}. The initial learning rate is 0.1, gradually warming up for 5 epochs, and a batch size of 256 is set for training. For augmentation, in the training step, images are randomly resized and cropped to a size of 224 $\times$ 224, then randomly flipped in the horizontal direction. In the validation step, images are center cropped to a size of 224 $\times$ 224. Experiments are conducted on two NVIDIA TITAN RTX GPUs. We report the top-1 classification accuracy for comparison in Table 3.

3) OVERFITTING

We visualize a validation loss of experiments on CIFAR-10 from Section V-A1 in Figure 3. Since normalization has a strong regularization effect, removing BN makes un-normalized models suffer from overfitting. It can be seen that SkipInit has a strong overfitting effect as the loss increases gradually from epoch 100. Both Fixup and RescaleNet also suffer from this overfitting issue, even though it is less severe than SkipInit. This overfitting problem is also addressed in Fixup [13]. Besides the better accuracy reported in Table 2,

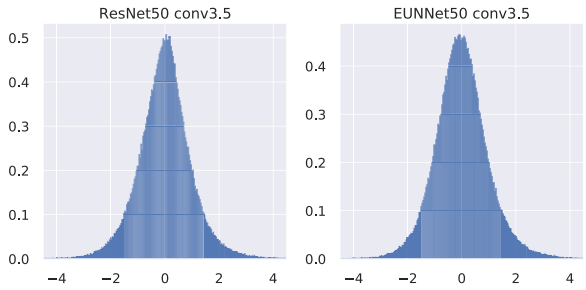


FIGURE 4. Weight distribution of the last convolutional layer of third block from ResNet-50 [1] and EUNNet-50.

our method also stabilizes the training process. Unlike the previous approaches, EUNNet converges faster, and the loss curve is smoother and more stable, while the other methods show an increasing loss after 100 epochs.

It is found that the aforementioned methods are suffering from overfitting issues, which is shown in Figure 3. We can see that the loss curve of previous methods all increases gradually in the later epochs of the training phase, while the proposed EUNNet is stable. To further boost the performance of our method, we apply regularization techniques such as Mixup [34] and CutMix [35]. A Mixup rate of 0.5 and a CutMix rate of 1.0 are used.

Notably, even though NF-ResNets report the best performance in Table 3, the training is unstable and sometimes collapses during training, as mentioned in [15]. The better performance of NF-ResNets is recorded in [22] when applying AGC; however, this is achieved under transfer learning on a large dataset of 300 million labeled images (JFT-300 dataset). In addition, although NFNets have a higher performance than other methods, they are based on the SE-ResNeXt [36] architecture, which has the better performance than ResNets.

4) LEARNABLE SCALARS

In this section, we study the effects of learnable scalars on SkipInit, Fixup, RescaleNet, and EUNNet. We train ResNet-110 on CIFAR-100 with the same hyperparameters used in Section V-A1. The multiplier scalar, *scale*, is removed from the model architecture. For RescaleNet and our method, we further test when α and β scalars are removed. Experiment results are given in Table 4.

Without the *scale* scalar, SkipInit fails to train the model, owing to the vanishing/exploding gradient problem. EUNNet achieves the best overall performance of 71.89%, compared with Fixup of 70.13% and RescaleNet of 71.05%. In addition, when we further remove α and β scalars from the model architecture, RescaleNet fails to train the model, while our method still trains the model normally and achieves an accuracy of 71.72%. The results strongly demonstrate that our method does not depend on learnable scalars.

5) WEIGHT DISTRIBUTION

Figure 4 shows histograms of the weight distribution. As shown in the figure, both weights in EUNNet-50 and

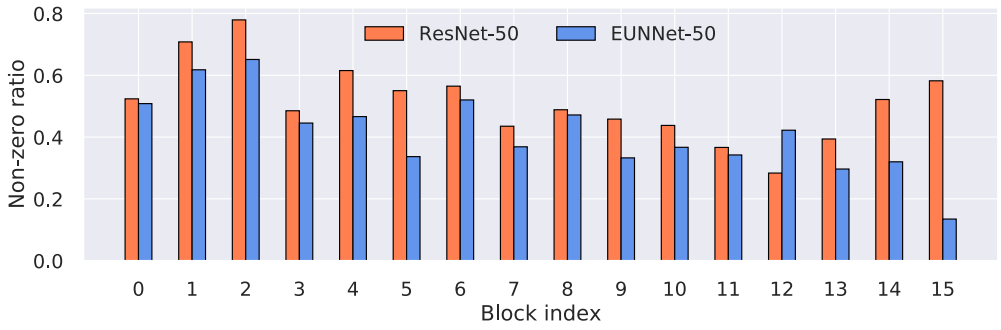


FIGURE 5. Non-zero ratio comparison between ResNet-50 [1] and EUNNet-50.

TABLE 5. Detection and segmentation results in COCO dataset using Mask R-CNN and FPN (higher is better). '*' denotes RescaleNet applying pre-bias to all convolutional layers and linear layers in the heads.

Model	Method	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
ResNet-50	BatchNorm	38.6	59.8	42.1	34.5	56.4	36.3
	GroupNorm	40.3	**61.0**	44.0	35.7	**57.9**	37.7
	RescaleNet	39.2	60.7	43.4	35.7	57.5	38.0
	RescaleNet*	**40.4**	60.8	**44.1**	35.9	57.7	37.9
	EUNNet	40.1	60.3	44.0	**36.1**	57.7	**38.5**
ResNet-101	BatchNorm	40.9	61.9	44.8	36.4	58.5	38.7
	GroupNorm	41.8	**62.7**	45.2	37.0	59.2	39.3
	RescaleNet	41.6	62.4	45.4	36.6	58.2	39.3
	RescaleNet*	41.8	62.7	45.2	37.0	59.2	39.3
	EUNNet	**41.8**	62.4	**45.5**	**37.5**	**59.2**	**40.0**

TABLE 6. Comparison of the number of parameters, FLOPs, memory consumption and inference time of different methods. Results are obtained when running the model with *torchinfo*[2] framework.

Model	Method	Params (M)	FLOPs (B)	F/B pass size (GB)	Throughput (s/epoch)
ResNet-50	BatchNorm	25.56	4.12	22.94	35.16
	Fixup	25.50	4.10	11.56	42.96
	RescaleNet	25.53	4.09	22.17	43.80
	EUNNet	25.53	4.09	11.56	43.80
SE-ResNeXt-50	NFNet-B0	71.50	12.38	-	-

ResNet50-BN models have a healthy distribution. In addition, it also shows that EUNNet maintains the effect of BN, keeping the distribution in the same range as ResNet. We also compare the non-zero ratio for the output values of each residual block and visualize it in Figure 5. It shows that using EUNConv can keep the output values of a residual block from being zeroed (known as "Dead" ReLU issue), which is the main reason for the vanishing gradient issue.

B. OBJECT DETECTION AND SEGMENTATION

We further evaluate the effect of EUNNet in object detection and segmentation tasks with the COCO dataset [33]. COCO is a large-scale object detection, segmentation, and captioning dataset of high-resolution and dense objects. COCO contains more than 200k images and 80 object categories.

[2]https://github.com/tyleryep/torchinfo

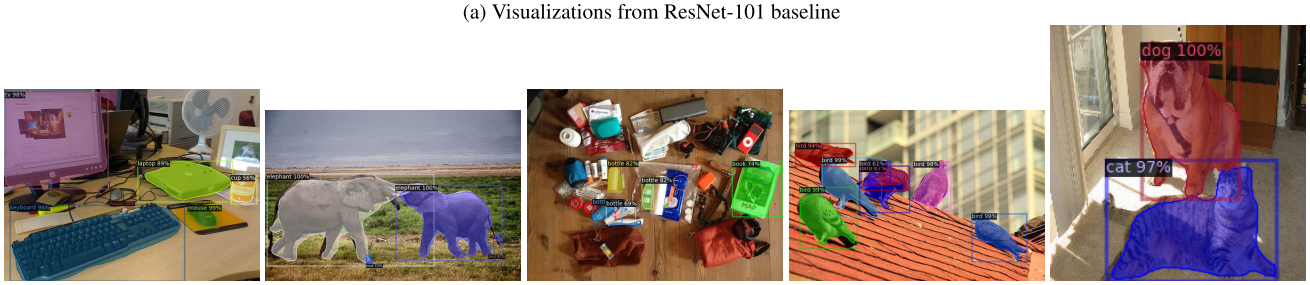
We conduct experiments using Mask R-CNN with a Feature Pyramid Network (FPN) as a detection model. ResNet-50 and ResNet-101 are pre-trained on ImageNet and then used as a feature extraction backbone. Because the object detection and segmentation tasks take higher resolution images as input, models are trained for 24 epochs with a very small batch size (4 images per GPU) on 2 TITAN RTX. We train the detection model using a learning rate of 0.005. For a fair comparison with RescaleNet, we only replace the convolutional layer with EUNConv for the backbone model. We report the Average Precision (AP, AP50, AP75) for the bounding box (AP^{bbox}), and instance segmentation (AP^{mask}) as standard metrics for this task.

As shown in Table 5, our proposed EUNNet achieves a competitive or better performance compared with the BN/GN and RescaleNet models. We can see that the results of un-normalized networks are better than normalized ones.

TABLE 7. Comparison of the multiplier scalar values by layer.

Model	1	2	3	4	5	6	7	8
RescaleNet-50	11.073	10.339	10.310	14.467	10.082	13.121	14.276	16.991
EUNNet-50	2.516	3.127	3.621	4.014	3.708	3.334	3.109	3.231
Model	9	10	11	12	13	14	15	16
RescaleNet-50	12.647	15.192	15.548	17.261	18.033	27.445	23.516	27.258
EUNNet-50	3.036	3.023	2.637	2.390	2.120	22.068	15.986	17.371

(a) Visualizations from ResNet-101 baseline



(b) Visualizations from EUNNet-101

FIGURE 6. Object detection and segmentation visualizations on COCO dataset examples. The visualizations of the model using ResNet-101 as backbone are in Figure 6a, and the ones of the model using proposed EUNNet-101 are in Figure 6b.

This is due to the weaknesses of the BN layer toward small batch size settings (2 per GPU in this experiment). As a result, EUNNet can not only maintain the performance of the normalized networks, but it also boosts the performance further than the best result that we achieve from the GN model.

VI. ABLATION STUDY

A. MEMORY OVERHEAD ANALYSIS

We make a comparison of the number of parameters, FLOPS, forward and backward pass sizes, and the total size that the neural network models need on the GPU. The results are reported in Table 6.

As can be seen, our EUNNet has a slightly higher number of parameters compared with Fixup; however, it is still lower than the model with BN. EUNNet has the same number of FLOPs as RescaleNet since we adopted the α and β from RescaleNet, but only requires half of the memory pass size during training. This achieves 45.30% and 47.14% improvement over the memory allocation size of RescaleNet and ResNets with BN, respectively. The results show that reducing the number of layers can lead to a reduction in the total memory size required for the model, thus proving our hypothesis. In addition, EUNNet has the same inference time as RescaleNet and Fixup, as shown in Table 6. We can see that even though we remove BN from ResNet, the inference time is higher than the conventional ResNet. This is because the batch statistic calculation step of the conventional ResNet is optimized for the hardware by CUDA, thus having a better throughput.

B. MULTIPLIER SCALAR VALUES

We schematically show the per-layer learned scalar multiples for RescaleNet-50 and EUNNet-50 (16 scalars from 16 residual blocks, shallow to deeper ones):

As we can see, the multiplier scalars *scale* of EUNNet are much smaller than RescaleNet (∼5-9 times smaller). This shows that RescaleNet depends strongly on the multiplier scalars to recover the magnitude of gradients because it downscales the value of the initial weights. In addition, the convolution weights shrink in the training due to weight decay, thus the large scalars can compensate for the shrink to keep the output variance. Furthermore, the learned scalars up-weights deeper blocks, which have more parameters and higher-level features. Unlike RescaleNet, since EUNNet does not downscale the value of the initial weights, it does not need

to recover the magnitude of gradients. Therefore, when we remove multiplier scalars, the performance drop is negligible as shown in Table 4.

C. RESIDUAL BLOCK OUTPUT VARIANCE

In this section, we visualize the variance of residual block output for EUNNet and ResNet.

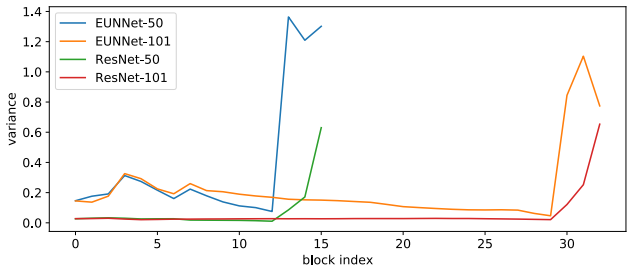


FIGURE 7. Variance of residual block output for EUNNet and ResNet.

From Figure 7, we see that the variance of EUNNet stays identical throughout the model. Even though the variance of EUNNet is higher than that of ResNet, it is still in a proper range in the final model (0.1 to 1.4) and does not double after each residual block.

Experimental results show that the variances of EUNNet range from 0.1 to 1.4, compared with that in the range from 0.0 to 1.2 of ResNet. In addition, the variance of EUNNet does not increase exponentially even when BN is removed. Therefore, the issue of the exploding variance is avoided.

D. VISUALIZATION

To further study the impact of the proposed method on object detection and segmentation tasks, we visualize the output of models using ResNet-101 and EUNNet-101 as the backbones in Figure 6. It can be seen that the model using EUNNet as a backbone produces similar segmentation regions to the model using ResNet. This proves that our proposed method can help an un-normalized model achieve the same result as the normalized model. In addition, both models give mostly the same segmentation and object class regardless of the size and density of the object.

VII. CONCLUSION

In this paper, we present an alternative convolutional layer, EUNConv, which can effectively stabilize the variance of a residual block in Deep Residual Networks (ResNets). Unlike previous methods, EUNConv simplifies and helps reduce the memory allocation during training as it does not introduce any additional learnable scalars as separate layers. In addition, we propose EUNNet, a ResNet variant where we replace convolutional layers with EUNConv and remove BN layers. While allocating less memory during training, EUNNet delivers competitive outcomes to earlier approaches. Furthermore, because EUNNet does not require hyperparameter fine-tuning, it can use the optimal hyperparameters from normalized networks.

VIII. FUTURE WORK

In future work, we would like to explore the strengths of our proposed method, EUNNet, under different architectures such as GCN, attention-based networks, and ViT-based networks. Broadly speaking, since EUNNet helps improve the performance with lesser effort for fine-tuning hyperparameters, which is very sensitive, other state-of-the-art architectures such as EfficientNet and Vision Transformer will be evaluated.

APPENDIX
IMPLEMENTATION CODE OF EUNCONV

```
import torch.nn as nn
import torch.nn.functional as F


class Conv2d(nn.Conv2d):
    def __init__(self, in_channels,
        out_channels, kernel_size, stride=1,
        padding=0, dilation=1, groups=1, bias=
        True):
        super(Conv2d, self).__init__(
            in_channels, out_channels,
            kernel_size, stride, padding,
            dilation, groups, bias)

    def forward(self, input):
        input = F.normalize(input, p=2, dim=1)
        return F.conv2d(input, self.weight,
            self.bias, self.stride, self.
            padding, self.dilation, self.groups
            )
```

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[2] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, and G. Chen, "Deep speech 2: End-to-end speech recognition in English and Mandarin," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 173–182.

[3] R. Socher, Y. Bengio, and C. D. Manning, "Deep learning for NLP (without magic)," in *Proc. Tutorial Abstracts ACL*. 2012, p. 5.

[4] J. Bjorck, C. Gomes, B. Selman, and K. Q. Weinberger, "Understanding batch normalization," 2018, *arXiv:1806.02375*.

[5] P. Luo, X. Wang, W. Shao, and Z. Peng, "Towards understanding regularization in batch normalization," 2018, *arXiv:1809.00846*.

[6] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–11.

[7] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[8] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*.

[9] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," 2016, *arXiv:1607.08022*.

[10] Y. Wu and K. He, "Group normalization," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 3–19.

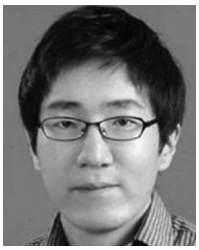
[11] L. Huang, J. Qin, Y. Zhou, F. Zhu, L. Liu, and L. Shao, "Normalization techniques in training DNNs: Methodology, analysis and application," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Feb. 28, 2023, doi: 10.1109/TPAMI.2023.3250241.

[12] S. De and S. Smith, "Batch normalization biases residual blocks towards the identity function in deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 19964–19975.

[13] H. Zhang, Y. N. Dauphin, and T. Ma, "Fixup initialization: Residual learning without normalization," 2019, *arXiv:1901.09321*.

[14] J. Shao, K. Hu, C. Wang, X. Xue, and B. Raj, "Is normalization indispensable for training deep neural network?" in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 13434–13444.
[15] A. Brock, S. De, and S. L. Smith, "Characterizing signal propagation to close the performance gap in unnormalized ResNets," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–30. [Online]. Available: https://openreview.net/forum?id=IX3Nnir2omJ
[16] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: Closing the generalization gap in large batch training of neural networks," 2017, *arXiv:1705.08741*.
[17] S. Ioffe, "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–9.
[18] D. Gaur, J. Folz, and A. Dengel, "Training deep neural networks without batch normalization," 2020, *arXiv:2008.07970*.
[19] H. V. Pham, T. Lutellier, W. Qi, and L. Tan, "CRADLE: Cross-backend validation to detect and localize bugs in deep learning libraries," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*, May 2019, pp. 1027–1038.
[20] S. Qiao, H. Wang, C. Liu, W. Shen, and A. Yuille, "Micro-batch training with batch-channel normalization and weight standardization," 2020, *arXiv:1903.10520*.
[21] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.
[22] A. Brock, S. De, S. L. Smith, and K. Simonyan, "High-performance large-scale image recognition without normalization," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 1059–1071.
[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017, *arXiv:1706.03762*.
[24] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.
[25] A. Fred Agarap, "Deep learning using rectified linear units (ReLU)," 2018, *arXiv:1803.08375*.
[26] D. Balduzzi, M. Frean, L. Leary, J. Lewis, K. W.-D. Ma, and B. McWilliams, "The shattered gradients problem: If ResNets are the answer, then what is the question?" in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 342–350.
[27] B. Hanin, "Which neural net architectures give rise to exploding and vanishing gradients?" 2018, *arXiv:1801.03744*.
[28] A. Veit, M. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1–9.
[29] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis.*, Jun. 2015, pp. 1026–1034.
[30] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," 2016, *arXiv:1604.06174*.
[31] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, Tech. Rep. TR-2009, 2009.
[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.
[33] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, and C. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2014, pp. 740–755.
[34] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," 2017, *arXiv:1710.09412*.
[35] 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.
[36] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5987–5995.

KHANH-BINH NGUYEN (Student Member, IEEE) received the B.S. degree in computer science from the Ho Chi Minh University of Technology (HCMUT), Vietnam, in 2018. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Sungkyunkwan University (SKKU), South Korea. His research interest includes deep learning architecture development focusing on computer vision.



JAEHYUK CHOI (Senior Member, IEEE) received the B.S. degree in electrical engineering from Yonsei University, Seoul, South Korea, in 2004, the M.S. degree in electrical engineering and computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2006, the M.S. degree in electrical and computer engineering from the University of Minnesota, Minneapolis, MN, USA, in 2008, and the Ph.D. degree from the University of Michigan, Ann Arbor, MI, USA, in 2013. From 2013 to 2015, he was a Research Staff Member with the Samsung Advanced Institute of Technology (SAIT), Samsung Electronics, Suwon, South Korea, where he was engaged in researching depth sensors and low power image sensors. In 2016, he joined the Department of Semiconductor Systems Engineering, Sungkyunkwan University, Suwon, as an Assistant Professor. His research interests include low-power circuits, CMOS sensors, and mixed-signal integrated circuits.

JOON-SUNG YANG (Senior Member, IEEE) received the B.S. degree in electrical and computer engineering from Yonsei University, Seoul, South Korea, in 2003, and the M.S. and Ph.D. degrees in electrical and computer engineering from The University of Texas at Austin, Austin, TX, USA, in 2007 and 2009, respectively. After graduation, he was with Intel Corporation, Austin, TX, USA, for four years. He was with Sungkyunwan University. He is currently an Associate Professor with Yonsei University. His research interests include memory architectures and efficient deep learning architecture development. He was a recipient of the Korea Science and Engineering Foundation (KOSEF) Scholarship, in 2005. He received the Best Paper Award at the 2008 IEEE International Symposium on Defect and Fault Tolerance in VLSI systems and at the 2016 IEEE International SoC Design Conference. He was nominated for the Best Paper Award at the 2013 IEEE VLSI Test Sympossium.