Propagation Mechanism for Deep and Wide Neural Networks

Dejiang Xu Mong Li Lee Wynne Hsu
School of Computing,
National University of Singapore
{xudj,leeml,whsu}@comp.nus.edu.sg

Abstract

Recent deep neural networks (DNN) utilize identity mappings involving either element-wise addition or channel-wise concatenation for the propagation of these identity mappings. In this paper, we propose a new propagation mechanism called channel-wise addition (cAdd) to deal with the vanishing gradients problem without sacrificing the complexity of the learned features. Unlike channel-wise concatenation, cAdd is able to eliminate the need to store feature maps thus reducing the memory requirement. The proposed cAdd mechanism can deepen and widen existing neural architectures with fewer parameters compared to channel-wise concatenation and element-wise addition. We incorporate cAdd into state-of-the-art architectures such as ResNet, WideResNet, and CondenseNet and carry out extensive experiments on CIFAR10, CIFAR100, SVHN and ImageNet to demonstrate that cAdd-based architectures are able to achieve much higher accuracy with fewer parameters compared to their corresponding base architectures.

1. Introduction

After the impressive performance of deep neural network [17] at the ImageNet [3] 2012 competition, there has been a rapid introduction of new neural network architectures with improved performance. These architectures include ResNet [7], Wide-ResNet [32], ResNeXt [31], PyramidNet [6], DenseNet [12], Dual Path Network [2], MobileNet [10], Shake-Shake Net [4], ShuffleNet [33], CondenseNet [11] etc. Recent attempts to use the sheer power of numerous GPU servers to automatically search for good neural network architectures have led to NASNet [34], EAS [1], ENAS [22] and AmoebaNets [24]).

One trend that is consistent across these neural network architectures is that a deeper and wider neural network often yields better performance. However, a deep and wide network suffers from the problem of vanishing gradient as well as a quadratic growth in the number of parameters. Further, the computational complexity and memory requirements also escalate in these architectures which are formidable for scalable learning in real world applications.

It remains non-trivial to design neural architectures that can address the vanishing gradient problem, yet are capable of capturing complex features to significantly lift the performance of the learned models which are also sufficiently small in size to reduce power consumption and potentially be deployable on IoT devices and mobile platforms.

We observe that the depth of a neural architecture is key to its performance. Current neural architectures use identity mappings in the form of skip connections to increase their depth. This allows the gradient to be passed backwards directly thus allowing the increase in depth without the issue of vanishing gradients. The propagation of these identity mappings from one block to the next is achieved either via *element-wise addition* (eAdd) [7] or *channel-wise concatenation* (cCon) [12]. Figure 1 shows these propagation mechanisms. In eAdd, addition is performed on the corresponding elements, hence the input size for each unit remains the same. On the other hand, cCon concatenates the inputs from all the preceding units, thus increasing the input size quadratically for each subsequent units. As a result, cCon can learn more complex features, however, it needs more memory to store the concatenated inputs [23] .

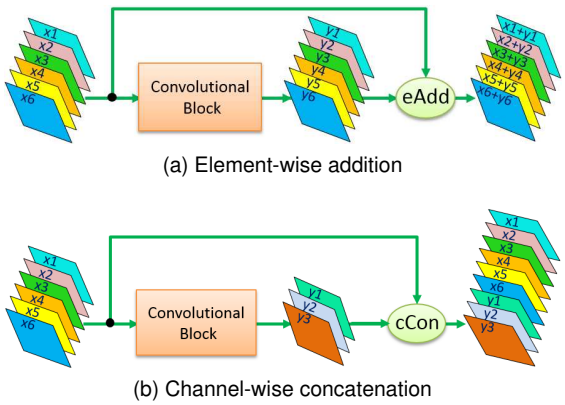


Figure 1. Propagation mechanism of element-wise Addition (eAdd) and channel-wise Concatenation (cCon).

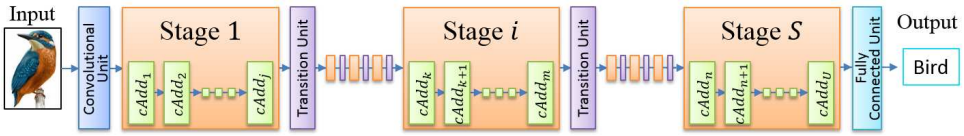


Figure 2. General architecture of a deep neural network using cAdd.

In this work, we propose a novel propagation mechanism, called channel-wise addition (cAdd), that can be easily incorporated into any of the state-of-art neural architectures to reduce the computational and memory requirements while achieving high accuracy. In order to keep the memory requirement small, we sequentially produce small residual part and add them to part of channels of the identity part in one unit. The unit is repeated multiple times until all the channels are added. With this, the depth of a network is increased while the number of parameters is reduced.

Figure 2 shows a general architecture of a neural network using cAdd. It has several stages and the cAdd units within each stage have the same resolution for both input and output feature maps to allow for channel-wise addition. The resolution across the stages may be different as they could be down-sampled by transition units. This design has several advantages:

1. Vanishing gradient can be alleviated since cAdd also has a shortcut that allows the gradient to bypass the unit directly.

2. Less memory is needed since cAdd adds back the output features instead of concatenation, thus keeping the input size the same for each unit.

3. More complex features can be generated as cAdd significantly increases the width and depth of CNNs.

4. Fewer parameters and FLOPs compared to existing neural networks with the same width and height.

Extensive experiments on CIFAR10 [16], CIFAR100 [16], SVHN [21] and ImageNet [3] datasets demonstrate that cAdd-based neural networks consistently achieve higher accuracy with fewer number of parameters compared to the original networks that use either eAdd or cCon.

2. Related Work

Neural Networks using eAdd. Depth is vital for neural networks to achieve higher performance. However, it is hard to optimize deep neural networks. Element-wise addition was introduced in ResNet [7] to significantly deepen the neural network and ease the training process [8]. It has been widely used in many deep neural networks, including Inception-ResNet [29], Wide-ResNet [32], ResNeXt [31], PyramidNet [6], Shake-Shake Net [4], and ShuffleNets [33]. It is also adopted by AlphaGo [26] and the automatically designed architectures, like NASNet [34], ENAS [22], and AmoebaNets [24].

The width of a neural network is also crucial to gain accuracy. Unlike ResNet, which achieves higher performance by simply stacking element-wise addition, Wide-ResNet widens the network by increasing the input channels along the depth. Experimental results show a 16-layer Wide-ResNet can outperform a thousand-layer ResNet in both accuracy and efficiency. For Wide-ResNet, the increase in width occurs only between stages, and the input size within a stage remains the same. PyramidNet gradually increases its width in a pyramid-like shape with a widening step factor, which has been experimentally proven to improve generalization ability. ResNext uses multi-branch element-wise additions, by replacing the only branch with a set of small homogeneous branches. Simply adding more branches can improve the the performance of ResNext. Instead of directly summing up all the small branches, Shake-shake Net uses a stochastic affine combination to significantly improve the generalization ability.

Unlike the manually designed architectures which need human expertise, automatically designed architectures search the entire architecture space to find the best design. Although the learned architectures have many different small branches, the distinct characteristic is that they all use eAdd to sum up the branches. Since eAdd requires the output size to be at least the same or larger than the input size, a neural network can go deeper or wider, but not both when the number of parameters is limited. In contrast, the proposed cAdd can both deepen and widen a neural network for the same number of parameters.

Neural Networks using cCon. Channel-wise concatenation was first used in DenseNet [12]. Features from all preceding units are used as inputs to generate a small number of outputs, which are passed to subsequent units. While this strengthens feature propagation and reuse, not all prior features need to be used as inputs to every subsequent layer. As such, CondenseNet selects only the most relevant inputs through an learned group convolution [11]. It sparsifies the convolutional layer by pruning away unimportant filters during the condensing stage, and optimizes the sparsified model in the second half of the training process. CondenseNet is more efficient than the compact MobileNes [10] and ShuffleNets [33], which are designed for mobile devices using depth-wise separable convolutions [15].

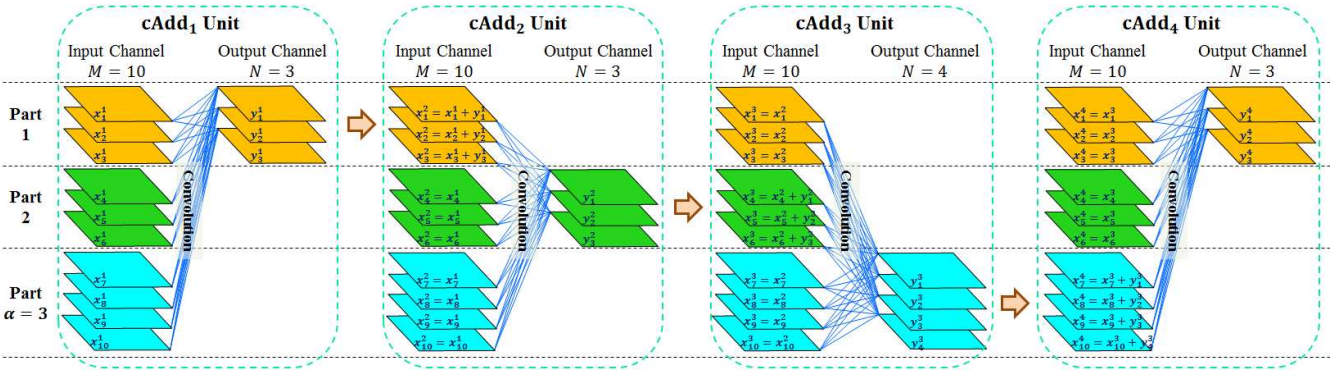


Figure 3. Propagation mechanism within four cAdd units.

The best models obtained from automatically designed architectures all utilize cCon, especially for the combination of all the cell outputs. Since concatenation increases the input size linearly, this also increases the number of parameters and memory requirements. In contrast, the proposed cAdd is able to keep the input size constant by adding back outputs to selected inputs.

3. Channel-wise Addition

The proposed cAdd propagation mechanism combines the benefits of eAdd and cCon to deepen and widen neural networks yet using fewer parameters. The key idea is that each unit must generate a small number of output channels whereby the generated outputs are then added back to the corresponding skipped connections to form the inputs to the next unit. Figure 3 shows the propagation across 4 units using cAdd. The first cAdd unit generates 3 outputs which are then added back to the first 3 skipped connections to form the inputs to the second cAdd unit.

Suppose M is the number of input channels. To ensure all the skipped connections are covered, we group the input channels of each unit into non-overlapping parts. The size of each part is controlled by a parameter α such that each part has exactly $\lfloor M/\alpha \rfloor$ channels except the final part which has $\lfloor M/\alpha \rfloor + R$ channels where R is the remaining channels. In Figure 3, we see that the second input part (green parallelogram) has $\lfloor M/\alpha \rfloor = 3$ channels and these are covered by the addition to the outputs of the cAdd_2 unit, while the third and final input part (blue parallelogram) has $\lfloor M/\alpha \rfloor + R = 4$ channels and they are covered by the addition to the outputs of the cAdd_3 unit.

In order for the addition operation to make sense, the number of generated outputs from a unit must match the number of channels to be covered in the next unit. Mathematically, the number of output channels for the k^{th} cAdd unit is given by:

$$\begin{cases} \lfloor M/\alpha \rfloor, & k\%\ \alpha \neq 0 \\ \lfloor M/\alpha \rfloor + M\%\ \alpha, & otherwise \end{cases} \tag{1}$$

We show that the cAdd propagation mechanism is able to alleviate the vanishing gradient problem. Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_M]$ be the input to a cAdd unit, and $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_N]$ be the output of $\mathbf{X}$ after passing through the non-linear transformation function $\mathcal{F}(\cdot)$ of the convolutional block, which may have different layers consisting of batch normalization [14] (BN), rectified linear units [5] (ReLU), and convolution layers (Conv). that is,

$$\mathbf{Y} = \mathcal{F}(\mathbf{X}) \tag{2}$$

The cAdd unit adds back its outputs $\mathbf{Y}$ into part of its inputs $\mathbf{X}$ to form the inputs $\mathbf{X}'$ for the next unit as follows:

$$\mathbf{X}' = \mathbf{X} + \mathbf{T}\mathbf{Y} \tag{3}$$

where $\mathbf{T}$ is a $M \times N$ sparse matrix, $\mathbf{T}_{ij} = 1$ if $\mathbf{y}_j$ is to be added to $\mathbf{x}_i$.

With Equations 2 and 3, we have:

$$\mathbf{X}' = \mathbf{X} + \mathbf{T} \cdot \mathcal{F}(\mathbf{X}) \tag{4}$$

Let us consider the propagation from cAdd unit s to cAdd unit e whose corresponding inputs are $\mathbf{X}^s$ and $\mathbf{X}^e$ respectively. We have

$$\mathbf{X}^e = \mathbf{X}^s + \sum_{i=s}^{e-1} \mathbf{T}^i \cdot \mathcal{F}(\mathbf{X}^i) \tag{5}$$

Let E be the error loss. The gradient on $\mathbf{X}^s$ can be expressed as:

$$\frac{\partial E}{\partial \mathbf{X}^s} = \frac{\partial E}{\partial \mathbf{X}^e}\frac{\partial \mathbf{X}^e}{\partial \mathbf{X}^s} = \frac{\partial E}{\partial \mathbf{X}^e}\left(1 + \sum_{i=s}^{e-1} \mathbf{T}^i \cdot \frac{\partial \mathcal{F}(\mathbf{X}^i)}{\partial \mathbf{X}^s}\right) \tag{6}$$

Since it is not possible for all the training samples within a batch to have the component $\sum_{i=s}^{e-1} \mathbf{T}^i \cdot \frac{\partial \mathcal{F}(\mathbf{X}^i)}{\partial \mathbf{X}^s}$ in Equation (6) to be always equal to -1, this implies that gradient is unlikely to be 0, thus alleviating the vanishing gradient problem.

4. Architectures using cAdd

The proposed cAdd propagation mechanism can be easily incorporated into existing neural networks. There are two kinds of units in neural networks, namely, basic and bottleneck units. We use the following notations for the different layers in a unit:

- Conv(I, O, L, L). Convolution layer with I input channels, O output channels, and kernel size $L \times L$.
- BN(I). Batch normalization with I input channels.
- ReLU. Rectified linear unit.

We first consider networks that use eAdd propagation mechanism. In the eAdd basic unit, the number of output channels must be the same as that of the input channels for element-wise addition. This constraint is no longer required when we replace eAdd by cAdd. Recall Equation 1 where the number of output channels in cAdd is determined by α. A large α will imply a significant reduction in the number of output channels. Figure 4 shows that the initial convolution layer of cAdd basic unit is Conv(M, M/α, L, L) instead of eAdd basic unit from Conv(M, M, L, L).

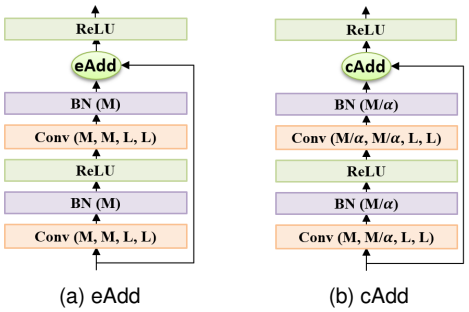


Figure 4. Basic unit using eAdd vs cAdd

The eAdd bottleneck unit uses convolution layer with kernel size 1×1 [19] to spatially combine large numbers of input feature maps with few parameters (see Figure 5(a)). Due to the element-wise addition requirement, an additional convolution layer is needed to expand the size of the output channels back to M. However, this is not needed for channel-wise addition. Figure 5(b) shows the corresponding bottleneck unit that uses cAdd.

Adapting cCon-based neural networks to use the cAdd propagation mechanism is straightforward. Instead of using the growth rate g to determine the number of output channels in both the basic and bottleneck units, we use Equation 1. Figure 6 shows the basic unit using cAdd vs cCon where the convolution layer is Conv(M, M/α, L, L) instead of Conv(M, g, L, L).

Similar adaptations can be made to neural architecture variants such as PyramidNet [6].

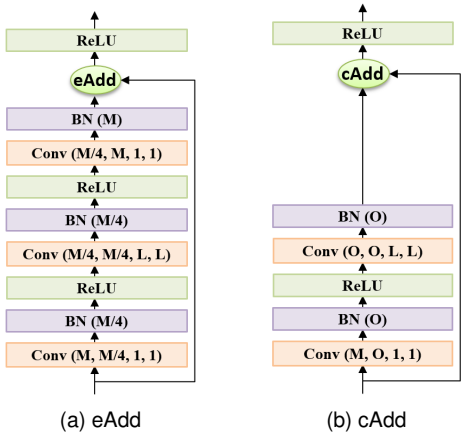


Figure 5. Bottleneck unit using eAdd vs cAdd

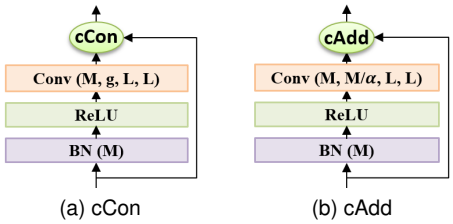


Figure 6. Basic unit using cCon vs cAdd

We analyze the number of parameters required in a neural architecture that uses cAdd vs eAdd or cCon. We assume that the neural architecture has U basic units, and each unit has M input channels with size of $h \times w$. For fair comparison, we assume that the growth rate g for a cCon unit is M/α so that the cCon unit has the same number of output as cAdd. Table 1 gives the number of parameters required.

	Number of Parameters
eAdd basic unit	$2 * U * M^2 * L^2$
corresponding cAdd unit	$U * M^2 * L^2 * (1/\alpha + 1/\alpha^2)$
cCon basic unit	$U * M^2/\alpha * L^2 +$ $(M/\alpha)^2 * L^2 * (U^2 - U)/2$
corresponding cAdd unit	$U * M^2/\alpha * L^2$

Table 1. Comparison of number of parameters required.

We see that a neural network using cAdd has approximately 2α times fewer parameters compared to a network that uses eAdd. That is, with the same number of parameters, the depth of a neural network using cAdd can be increased by 2α, or the width can be increased by $\sqrt{2\alpha}$ compared to using eAdd. Such an increase can improve the generalization ability of the neural networks, thus leading higher accuracy. Clearly, the number of parameters required by cCon in Table 1 have more parameters than cAdd. The residual part of $(M/\alpha)^2 * L^2 * (U^2 - U)/2$ is introduced by concatenation operation.

We also compare the number of FLOPs required Table 2 shows that a neural network using cAdd requires approximately 2α and $(1 + \frac{U-1}{2\alpha})$ times fewer FLOPs compared to a network that uses eAdd and cCon respectively.

	Number of FLOPs
eAdd basic unit	$2M^2L^2hwU + MhwU$
corresponding cAdd unit	$(1/\alpha + 1/\alpha^2)M^2L^2hwU + 1/\alpha MhwU$
cCon basic unit	$1/\alpha M^2L^2hwU + 1/\alpha^2 M^2L^2hw(U^2 - U)/2$
corresponding cAdd unit	$1/\alpha M^2L^2hwU + 1/\alpha MhwU$

Table 2. Comparison of FLOPs required.

5. Experimental Evaluation

We carry out experiments to compare the performances of neural architectures that use cAdd, eAdd and cCon. We incorporate cAdd into three widely used CNN architectures, namely ResNet, WRN and CondenseNet, and call them cResNet, cWRN and cCondenseNet respectively. Each architecture has 3 stages.

We train these networks using stochastic gradient descent with nesterov momentum [28] of 0.9 without dampening, and a weight decay of 10^{-4}. For fair comparison, all the training settings (learning rate, batch size, epochs, and data augmentations) are the same as in the original papers, unless otherwise specified. The following datasets are used:

- CIFAR10 [16]: It has 10 object classes with 6,000 32x32 color images for each class. There are 50,000 images for training and 10,000 for testing.
- CIFAR100 [16]: It has 100 classes with 600 32x32 color images for each class. The training and testing sets contain 50,000 and 10,000 images respectively.
- SVHN [21]: This has over 600,000 32x32 images of real-world house numbers. There are 73,257 images for training, 26,032 for testing, and additional 531,131 for extra training.
- ImageNet [3]: It has 1,000 classes. The training set has 1.2 millon images and validaiton set has 50,000 images.

5.1. ResNet vs cResNet

In this set of experiments, we examine the performance of ResNet with cResNet. Like ResNet, we train all the cResNet ($\alpha = 7$) for 300 epochs with batch size of 128. The learning rate starts from 0.1 and is reduced by 10 after the 150^{th} and 225^{th} epoch. For the 1224-layer cResNet, the initial learning rate is 0.01 for the first 20 epochs, and then go back to 0.1 to continue the training.

Table 3 gives the results of ResNet, pre-activation ResNet, and cResNet on CIFAR10, CIFAR100, and SVHN datasets. ResNet-20 with 0.27 million parameters has a depth of 20, and its width for three stages are 16, 32, and 64 respectively. In contrast, cResNet-86 with comparable number of parameters (0.21 million) has a depth of 86, and its corresponding width are 84, 112, and 140. The *increased width and depth* in cResNet-86 over ResNet-20 enables it to have a much *higher accuracy* on CIFAR10. In fact, the accuracy of cResNet-86 beats ResNet-56 on CIFAR10, CIFAR100 and SVHN datasets, which has four times the number of parameters.

When we increase the width of cResNet-86 to 168-196-308 so that it has comparable number of parameters (0.84 million) as ResNet-56, the gap in accuracy widens significantly. cResNet-86 even outperforms ResNet-110, ResNet-164 and pre-activation ResNet-164, which have twice the number of parameters. We see that cResNet-170 with 1.65 million parameters gives the best results over all the ResNets and pre-activation ResNets.

We observe that ResNet-1202 has 19.4 million parameters, yet its error rate is higher than ResNet-110, possibly due to over-fitting [7]. On the other hand, our cResNet-1224, which is much wider and deeper than ResNet-1202, achieves the lowest top-1 error rate of 4.06 on CIFAR10.

Figure 7 shows the top-1 error rates of the cResNet and ResNet on CIFAR10 dataset as we vary the number of parameters. Clearly, the error rate of cResNet is always lower than ResNet for the same number of parameters. We observe that ResNet at its lowest error rate has 8 times more parameters than cResNet.

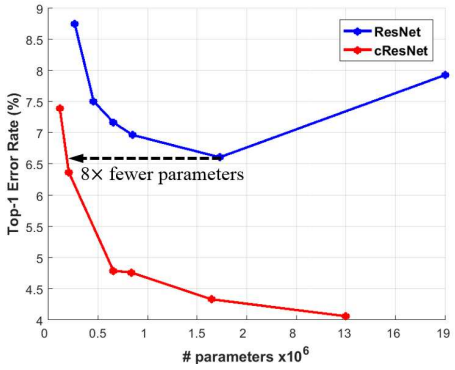


Figure 7. ResNet vs. cResNet on CIFAR10.

5.2. WRN vs cWRN

Next, we compare the performance of WRN with cWRN. Similar to WRN, we train cWRN ($\alpha = 7$) for 200 epochs with batch size of 128. The learning rate starts from 0.1, annealed by a factor of 5 times after the 60^{th}, 120^{th}, and 160^{th} epochs for CIFAR10 and CIFAR100 datasets. For SVHN dataset, cWRN are trained for 160 epochs with batch size of 128, and is optimized by dividing the initial learning rate of 0.01 by 10 after the 80^{th} and 120^{th} epochs.

Architecture	Width	# Params	CIFAR10	CIFAR100	SVHN
ResNet-20 [7]	16-32-64	0.27M	8.75	-	-
ResNet-32 [7]	16-32-64	0.46M	7.51	-	-
ResNet-44 [7]	16-32-64	0.66M	7.17	-	-
ResNet-56 [7]	16-32-64	0.85M	6.97	28.25 *	2.49 *
ResNet-110 [7]	16-32-64	1.73M	6.61 ± 0.16	27.22 †	2.01 †
ResNet-164 [7]	16-32-64	1.70M	-	25.16	-
ResNet-1001 [7]	16-32-64	10.2M	-	27.82	-
ResNet-1202 [7]	16-32-64	19.4M	7.93	-	-
Pre-activation ResNet -164 [8]	64-128-256	1.7M	5.46	24.33	-
Pre-activation ResNet -1001 [8]	64-128-256	10.2M	4.92	22.71	-
cResNet-86	84-112-140	0.21M	6.37 ± 0.09	27.45 ± 0.11	2.09 ± 0.07
cResNet-86	168-196-308	0.84M	4.76 ± 0.07	23.35 ± 0.17	2.04 ± 0.07
cResNet-170	**196-224-280**	**1.65M**	**4.33 ± 0.04**	**21.33 ± 0.20**	**1.92 ± 0.06**
cResNet-1224	196-224-280	13.185M	**4.06**	-	-

Table 3. Top-1 error rate of ResNet and cResNet. Width is the number of input channels in the three stages. * indicates results are from [30] and † shows results are from [13], Results for cResNet are averaged over 5 runs in the format of "mean±std".

Architecture	Width	# Params	CIFAR10	CIFAR100	SVHN
WRN-52-1 [32]	16-32-64	0.76M	6.43	28.89	2.08
WRN-16-4 [32]	64-128-256	2.75M	5.02	24.03	1.85
WRN-40-4 [32]	64-128-256	8.95M	4.53	21.18	-
WRN-16-8 [32]	128-256-512	11.00M	4.27	20.43	-
cWRN-130-2	98-126-154	0.39M	6.32 ± 0.10	26.75 ± 0.20	1.99 ± 0.06
cWRN-130-4	196-252-308	1.52M	4.87 ± 0.09	22.4 ± 0.19	**1.81 ± 0.05**
cWRN-172-6	294-378-462	4.41M	4.34 ± 0.09	20.87 ± 0.13	-
cWRN-172-8	392-504-616	7.80M	4.26 ± 0.07	19.78 ± 0.17	-
cWRN-88-13	**637-819-1001**	**10.90M**	**4.04 ± 0.09**	**19.33 ± 0.13**	-

Table 4. Top-1 error rate of WRN and cWRN. Width is the number of input channels in the three stages. Results for cWRN are averaged over 5 runs in the format of "mean±std".

Table 4 gives the results. All the cWRN are much wider and deeper compared to the corresponding WRN, and are able to achieve lower top-1 error rates with fewer parameters on all three datasets. Specifically, cWRN-130-2 outperforms WRN-52-1 with half the parameters (0.39 million vs. 0.76 million) on all three datasets. Overall, cWRN-88-13 gives the best performance.

Figure 8 shows the top-1 error rates of the cWRN and WRN on CIFAR10 dataset as we vary the number of parameters. We see that cWRN has 1.4 times fewer parameters than WRN for the same error rate.

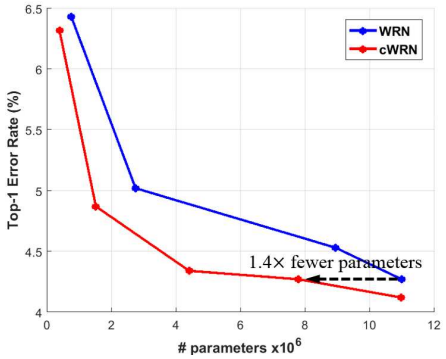


Figure 8. WRN vs. cWRN on CIFAR10.

5.3. CondenseNet vs cCondenseNet

Finally, we examine the performance of using cAdd in CondenseNet. We train all the cCondenseNet ($\alpha = 6$) for 300 epochs with a batch size of 64, and use a cosine-shaped learning rate from 0.1 to 0. For cCondenseNet-254, we train for 600 epochs with a dropout rate [27] of 0.1 to ensure fair comparison with CondenseNet-182.

Table 5 shows the results with cCondenseNet-254 giving the best performance on both CIFAR10 and CIFAR100. It has 456 input channels which is 38 times the width of CondenseNet-182, and 254 convolutional layers which is 1.4 times the depth of CondenseNet-182. We see that cCondenseNet-146 and cCondenseNet-110 are much wider and deeper with fewer parameters compared to their counterparts CondenseNet-86 and CondenseNet-50. In partic-

Architecture	Width	# Params	CIFAR10	CIFAR100
CondenseNet-50 [11]	8-16-32	0.22M	6.22	-
CondenseNet-74 [11]	8-16-32	0.41M	5.28	-
CondenseNet-86 [11]	8-16-32	0.52M	5.06	23.64
CondenseNet-98 [11]	8-16-32	0.65M	4.83	-
CondenseNet-110 [11]	8-16-32	0.79M	4.63	-
CondenseNet-122 [11]	8-16-32	0.95M	4.48	-
CondenseNet-182 [11]	12-24-48	4.22M	3.76	18.47
cCondenseNet-110	96-144-192	0.19M	5.74 $\pm$ 0.08	27.40 $\pm$ 0.15
cCondenseNet-146	168-216-264	0.50M	4.64 $\pm$ 0.08	23.44 $\pm$ 0.11
cCondenseNet-254	**456-504-576**	**4.16M**	**3.40 $\pm$ 0.09**	**18.20 $\pm$ 0.13**

Table 5. Top-1 error rate of CondenseNet and cCondenseNet. Width is the number of input channels or growth rate in the three stages. Results for cCondenseNet are averaged over 5 runs in the format of "mean±std".

Architecture	Width	# Params	Top-1 error rate	Top-5 error rate
CondenseNet-74 (G=C=4) [11]	8-16-32-64-128	4.8M	26.2%	8.3%
cCondenseNet-246 (G=C=4)	192-288-384-552-768	**4.7M**	**25.4%**	**7.7%**
ResNet-50 [7]	64-256-512-1024-2048	25.6M	24.7%	7.8%
ResNet-101 [7]	64-256-512-1024-2048	44.5M	23.6%	7.1%
cResNet-72	280-280-560-1120-2240	**23.3M**	**23.7%**	**7.1%**

Table 6. One-crop validation results on ImageNet. Width is the number of input channels or growth rate in the five stages

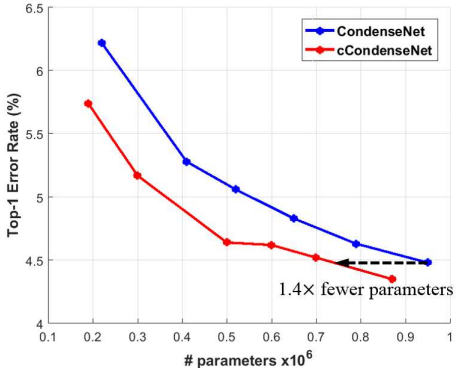


Figure 9. CondenseNet vs. cCondenseNet on CIFAR10.

ular, although cCondenseNet-110 has 0.03 million fewer parameters than CondenseNet-50, its top-1 error rate is smaller than that of CondenseNet-50, 5.74% versus 6.22%.

Figure 9 shows the top-1 error rates on CIFAR10. We see that cCondenseNet has 1.4 times fewer parameters than CondenseNet for the same error rate.

5.4. Experiments on ImageNet

We also compare the performances of the various neural architectures on the ImageNet dataset. Table 6 shows the results. We observe that cResNet-72 achieves much lower top-1 and top-5 error rates compared to ResNet-50 with similar number of parameters. When we compare ResNet-101 and cResNet-72 which has similar top-1 and top-5 error rates, we see that cAdd-based architecture requires only half the number of parameters. Similarly, cCondenseNet-246 with 0.1 million fewer parameters outperforms CondenseNet-74.

5.5. Depth vs. Width

Depth and width are vital dimensions for neural networks to achieve higher performance. Depth controls the complexity of the learned features. A deeper neural network can learn more complex features, while a wider network enables more features to be involved in the final classification.

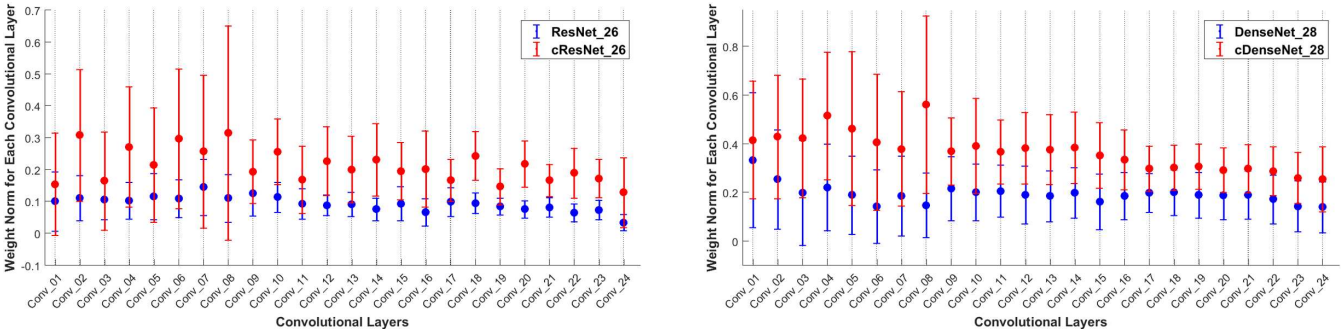
For cAdd based architectures, we have the flexibility of either increasing the depth or the width or both and still retain approximately the same number of parameters. Here, we investigate the impact of the depth and width of a cAdd based architecture on its classification accuracy.

We use ResNet-56 with 0.85 million parameters, and CondenseNet-86 with 0.52 million parameters as the baselines, and design different cResNet and cCondenseNet with approximately the same number of parameters at varying depth and width. Table 7 shows the results on both CIFAR10 and CIFAR100 datasets.

We observe that the best performances are attained when the increase in depth is balanced with the increase in width, indicating that both depth and width are equally important. This makes sense as the performance of a neural net depends both on the number of features as well as the complexity of these features.

Architecture	# Params	Width	Depth	CIFAR10	CIFAR100
ResNet-56 [7] (Base-line)	**0.85M**	**16-32-64**	**56**	**6.97**	**28.25**
cResNet-44	0.86M	280-308-336	44	6.20	27.14
cResNet-86	0.81M	196-224-252	86	5.91	27.09
cResNet-128	0.89M	168-196-224	128	5.84	26.94
cResNet-170	0.88M	140-168-196	170	5.66	27.04
cResNet-212	**0.89M**	**126-154-182**	**212**	**5.50**	**26.93**
cResNet-254	0.88M	112-140-168	254	5.88	27.39
cResNet-296	0.86M	100-128-156	296	5.95	27.77
cResNet-338	0.82M	91-119-147	338	5.94	27.55
CondenseNet-86 [11] (Base-line)	**0.52M**	**8-16-32**	**86**	**5.06**	**23.64**
cCondenseNet-38	0.51M	312-360-384	38	5.08	25.29
cCondenseNet-74	0.49M	240-288-312	74	4.89	24.19
cCondenseNet-110	0.51M	216-240-288	110	4.73	24.02
cCondenseNet-182	**0.51M**	**168-192-240**	**182**	**4.61**	**23.46**
cCondenseNet-218	0.51M	144-192-216	218	4.94	23.56
cCondenseNet-254	0.49M	120-168-216	254	4.89	23.74
cCondenseNet-290	0.51M	120-168-192	290	4.86	24.19
cCondenseNet-326	0.51M	120-144-192	326	5.11	24.24

Table 7. Top-1 error rate of cResNet, and cCondenseNets on CIFAR10, and CIFAR100 datasets.



(a) eAdd vs. cAdd (b) cCon vs. cAdd

Figure 10. Neuron weights in the convolutional layer of architectures using cAdd, eAdd and cCon.

5.6. Norm of Weights

Weight norm measures the activeness of neurons during feature learning [9, 11, 18, 20]. Figure 10 shows the mean and standard deviation of the neuron weights within each convolutional layer of the trained neural networks using cAdd (ResNet-26 and DenseNet-28), eAdd (ResNet-26), and cCon (DenseNet-28). We observe that the neurons in cAdd based networks have larger weights than eAdd and cCon based networks. This indicates that cAdd neurons are more active compared to eAdd and cCon neurons during feature learning. One possible reason could be that many of the weights in eAdd and cCon are close to zero and can be pruned without sacrificing accuracy [9, 18, 20]. With cAdd, we are able to reduce the number of weights, leading to fewer parameters and higher accuracy.

6. Conclusion

In this paper, we have proposed a new channel-wise addition propagation mechanism to deepen and widen neural networks with significantly fewer parameters. We have described how we can adapt state-of-the-art deep neural networks, namely, ResNet, WRN and CondenseNet to use cAdd. Extensive comparative experiments on CIFAR10, CIFAR100, SVHN and ImageNet datasets show that cAdd based neural architectures (cResNet, cWRN and cCondenseNet) consistently outperform their corresponding counterparts with higher accuracy, fewer parameters and lower computational costs. Future work includes investigating how channel-wise addition can be incorporated to further enhance the compact neural architectures for real world deployment.

References

[1] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, 2018.

[2] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. *CoRR*, abs/1707.01629, 2017.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2009.

[4] Xavier Gastaldi. Shake-shake regularization. *ICLR*, abs/1705.07485, 2017.

[5] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, volume 15, pages 315–323, 2011.

[6] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. *the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2015.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *ECCV*, 2016.

[9] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. *CoRR*, abs/1707.06168, 2017.

[10] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[11] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. *CVPR*, 2017.

[12] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CVPR*, 2016.

[13] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*, 2015.

[15] Lukasz Kaiser, Aidan N. Gomez, and François Chollet. Depthwise separable convolutions for neural machine translation. *CoRR*, abs/1706.03059, 2017.

[16] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[18] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016.

[19] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *International Conference on Learning Representations*, 2013.

[20] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. *CoRR*, abs/1708.06519, 2017.

[21] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *NIPS*, 2011.

[22] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *International Conference on Machine Learning*, 2018.

[23] Geoff Pleiss, Danlu Chen, Gao Huang, Tongcheng Li, Laurens van der Maaten, and Kilian Q. Weinberger. Memory-efficient implementation of densenets. *CoRR*, abs/1707.06990, 2017.

[24] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018.

[25] L. Sifre. Rigid-motion scattering for image classification. *Ph. D. thesis*, 2014.

[26] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550,354-359, 2017.

[27] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[28] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[29] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *AAAI*, 2017.

[30] Yan Wang, Lingxi Xie, Chenxi Liu, Ya Zhang, Wenjun Zhang, and Alan L. Yuille. SORT: second-order response transform for visual recognition. *CoRR*, abs/1703.06993, 2017.

[31] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CVPR*, 2017.

[32] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *BMVA*, 2016.

[33] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017.

[34] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017.