Pyramidal Combination of Separable Branches for Deep Short Connected Neural Networks

Yao Lu, Guangming Lu, Rui Lin and Bing Ma

August 29, 2018

Pyramidal Combination of Separable Branches for Deep Short Connected Neural Networks

Yao Lu, Guangming Lu[⋆], Rui Lin, Bing Ma

Harbin Institute of Technology (ShenZhen), ShenZhen, China.
yaolu_1992@126.com, luguangm@hit.edu.cn,
linrui_1995@163.com, mabing_campus@163.com

Abstract. Recent works have shown that Convolutional Neural Networks (CNNs) with deeper structure and short connections have extremely good performance in image classification tasks. However, deep short connected neural networks have been proven that they are merely ensembles of relatively shallow networks. From this point, instead of traditional simple module stacked neural networks, we propose Pyramidal Combination of Separable Branches Neural Networks (PCSB-Nets), whose basic module is deeper, more delicate and flexible with much fewer parameters. The PCSB-Nets can fuse the caught features more sufficiently, disproportionately increase the efficiency of parameters and improve the model's generalization and capacity abilities. Experiments have shown this novel architecture has improvement gains on benchmark CIFAR image classification datasets.

Keywords: Deep Learning · CNNs · PCSB-Nets.

1 Introduction

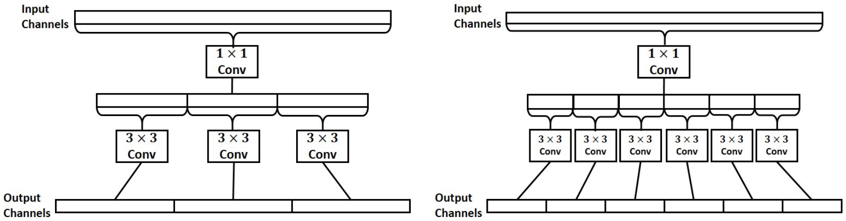
Deep Convolutional Neural Networks (DCNNs) have obtained a number of significant improvements in many computer vision tasks. The famous LeNet-style models [10] mark the beginning of the CNNs era. Nevertheless, AlexNet [8], possessing more convolutional layers stacked with different spatial scales, makes a big breakthrough in *ILSVRC 2012* classification competition.

1.1 Inception and Xception module

Following this trend, some new branchy models have been emerged, for instance, the family of Google Inception-style [12, 6, 13] and Xception [1] models. In fundamental Inception module (see Fig. 1(a)), where all input channels are embedded into a low dimension space through "1×1" convolution (also called "pointwise convolution"operation). Then the embedding feature maps will be equally divided into several branches as the corresponding input of the same number convolutional operations respectively. Specially, this operation can also be called

[⋆] Corresponding author.

“grouped convolutions", which is first used in AlexNet and implemented by such as Caffe [7]. At last, output feature maps from all groups are concatenated together. An “extreme" version of Inception module is shown in Fig. 1(b). In this basic module, the first layer also adopts “point-wise convolution" operation, but at the second stage, the convolutions with the same size filters are used on per input channel (also called “depth-wise convolution" in Tensorflow) and concatenate all output feature maps. This lack communication required between different branches before being merged. Moreover, feature information from separable paths can’t be sufficiently fused resulting in catching poor features.



(a) Inception Module

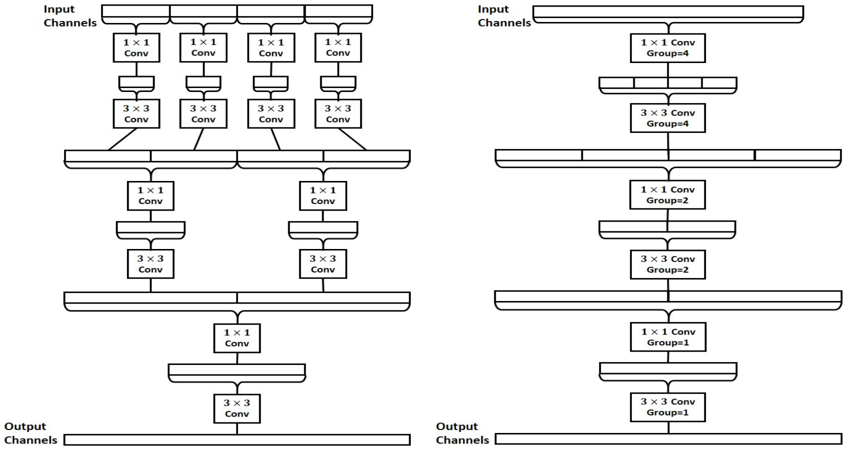
(b) Xception

Fig. 1: Inception module and Xception module.

1.2 Short connected convolutional neural networks

With these models going deeper, it is increasingly difficult to train because of the vanishing gradient. As for this point, a novel kind of architectures with skip connections ResNets [3] have been designed, who employ identity mapping to bypass layers contributing to obtaining great gradient flow and catching better features from objects. Following up this style of architectures, DenseNets [4] are proposed, where each basic module obtains additional inputs from all preceding modules and passes on its own feature maps to all subsequent modules. Although these models seem to have more layers, they have been proven that they don’t increase the depth actually but are only ensembles of relatively shallow networks [9]. Furthermore, the basic residual block is mainly composed of some traditional operation layers stacks, despite they can add more layers to the block to make it deeper, it will have much more parameters at the same time and can’t deal with input message delicately.

Given the importance of these issues of above models, we introduce a novel basic module called Pyramidal Combination of Separable Branches module. And the corresponding networks, referred to PCSB-Nets, are constructed by repeating this new module multiple times. Specially, as shown in Fig. 2, we equip several layers in one PCSB module, and every layer is composed of some branches. Furthermore, the number of branches in every layer decreases exponentially from the input layer to the output layer, accordingly, the shape of these basic novel modules constructed in this way looks like an inverted pyramid. Moreover, in every branch, the corresponding input features are firstly projected into a low dimension space to reduce the model's parameters.



(a) PCSB Module

(b) PCSB equivalent module

Fig. 2: PCSB module.

Our architecture's superiorities are notable. It can make every module more deeper and utilizes much fewer parameters more efficiently, because of both the branchy structure and the low-dimension embedded information of every branch. Additionally, in the feed-forward phase, every small branch only needs to deal with little information at the bottom layer. Then, these small branches are combined gradually to catch a larger spatial information until they become only one branch. This structure can make the features be fused sufficiently and the global task will be assigned to different branches smoothly. Therefore, the small task can be readily fulfilled by each branch. Meanwhile, in the backward phase, the gradient information from upper layers will guide every branch to update the parameters efficiently.

2 PCSB-Nets

2.1 Pyramidal combination of separable branches

The structure of the PCSB-Net's module is shown in Fig. 2(a). In the next illustration, we define a module possessing a specific number of regular operational layers, such as convolutional and nonlinearity activation layers, and it can have several same operational layers. Suppose the module M has L layers. $\boldsymbol{x}_0$ and

$\boldsymbol{x}_{l-1}$ refer to the input of M and the l^{th} layer (or the output of the $(l-1)^{th}$ layer) respectively. The number of branches (can also be called groups) in l^{th} layer denoted by G_l is calculated as bellow:

$$G_l = \alpha^{L-l}, \quad l = (1, 2, ..., L) \tag{1}$$

where α ($\alpha \geq 1$) is the combination factor of branches. Then the non-linear transformation function which consists of a series of operations including Batch Normalization (BN) [6], Rectified Linear Units (ReLU) [2] and Convolution (Conv) of i^{th} branch in the l^{th} layer can be represented as $\mathscr{F}_l^i$. We assume the convolutions contain "point-wise convolution" and ordinary convolutional operations and perform the former first by default. Consequently, the output $\boldsymbol{x}_l$ of the l^{th} layer can be obtained by Eqn. 2.

$$\boldsymbol{x}_l = Concat_i \ \mathscr{F}_l^i(\boldsymbol{x}_{l-1}^i), \quad i = (1, 2, ..., G_l) \tag{2}$$

Where $\boldsymbol{x}_{l-1}^i$ refers to the input of the i^{th} branch in the l^{th} layer, which means the input channels of the l^{th} layer are split to G_l parts equally. Furthermore, $Concat$ is the concatenation of all output feature maps from the l^{th} layer's branches. At last, the output of the module M is $Concat\{\boldsymbol{x}_0, \boldsymbol{x}_L\}$.

From Eqn. 1, when $\alpha > 1$, the number of branches grows exponentially with the number of α. When $\alpha = 1$, it will become the traditional residual module with merely one branch in every layer. More interestingly, when $L = 1$, it degrades to bottleneck [3] architecture, which consists of a "point-wise convolution" and a regular convolutional layer.

In every layer, all the convolutional operations have the same filter size of each branch, so that they can be implemented by "grouped point-wise convolution" and "grouped convolution" operations, when the filter size is equal to 1, the former will become the special case of the latter, therefore, our module can be also equivalently represented as Fig. 2(b). Specially, the groups of residual and Xception module are equal to 1 and the number of feature map channels, respectively. However, there is a notable difference between our module and traditional module in the way of embedding input information. That is our module "grouped point-wise convolution" at every layer. While in traditional modules, they conduct "point-wise convolution" operation first for all the input feature map channels to mix information.

2.2 Network architecture

Following DenseNet (see Fig.3), the PCSB-Net is similar to the overall structure of it with dense connections and repeating PCSB modules multiple times. Furthermore, we have always obeyed the rule that our model's parameters are fewer than or equal to DenseNet. Since the bottom block mainly processes the detail information, there will be more layers in every module in the first dense block contributing to more branches existing at the first layer in each module (*e.g.*, $\alpha = 2$, $L = 3$ and $G_1 = 4$), then in the second block, the number of layers will decrease (*e.g.*, $\alpha = 2$, $L = 2$ and $G_1 = 2$) leading to fewer branches. Finally, in the last dense block, there will be only one layer with only one branch left in each module resulting in bottleneck structure, for the reason that it will catch the high-level features. Fig. 4 shows the final detail structure of the PCSB-Net.

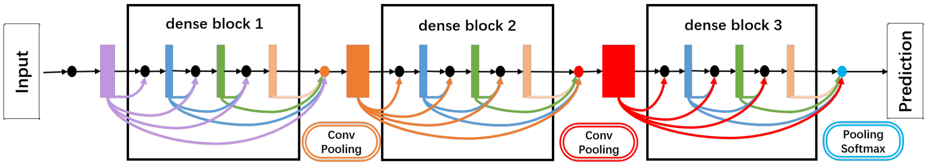


Fig. 3: DenseNets architecture.

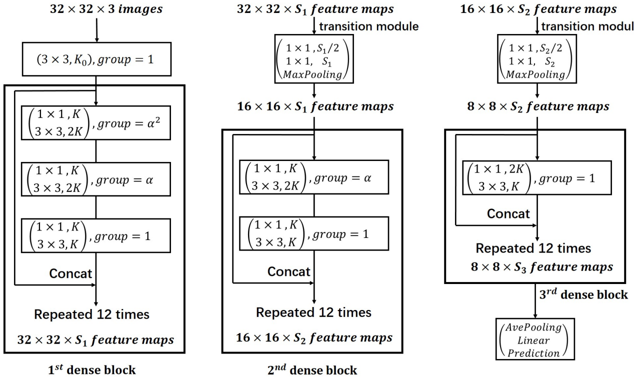

$32 \times 32 \times 3$ *images*
$(3 \times 3, K_0), group = 1$
$\begin{pmatrix}1 \times 1, K\\3 \times 3, 2K\end{pmatrix}, group = \alpha^2$
$\begin{pmatrix}1 \times 1, K\\3 \times 3, 2K\end{pmatrix}, group = \alpha$
$\begin{pmatrix}1 \times 1, K\\3 \times 3, K\end{pmatrix}, group = 1$
Concat
Repeated 12 times
$32 \times 32 \times S_1$ *feature maps*
1^{st} dense block
$32 \times 32 \times S_1$ *feature maps*
transition module
$\begin{pmatrix}1 \times 1, S_1/2\\1 \times 1, S_1\\MaxPooling\end{pmatrix}$
$16 \times 16 \times S_1$ *feature maps*
$\begin{pmatrix}1 \times 1, K\\3 \times 3, 2K\end{pmatrix}, group = \alpha$
$\begin{pmatrix}1 \times 1, K\\3 \times 3, K\end{pmatrix}, group = 1$
Concat
Repeated 12 times
$16 \times 16 \times S_2$ *feature maps*
2^{nd} dense block
$16 \times 16 \times S_2$ *feature maps*
transition module
$\begin{pmatrix}1 \times 1, S_2/2\\1 \times 1, S_2\\MaxPooling\end{pmatrix}$
$8 \times 8 \times S_2$ *feature maps*
$\begin{pmatrix}1 \times 1, 2K\\3 \times 3, K\end{pmatrix}, group = 1$
Concat
Repeated 12 times
$8 \times 8 \times S_3$ *feature maps*
3^{rd} dense block
$\begin{pmatrix}AvePooling\\Linear\\Prediction\end{pmatrix}$

Fig. 4: PCSB-Net architecture: "1×1" and "3×3" represent the convolutional operations, every convolutional operation is followed by the number of output channels. S_1, S_2 and S_3 denote the number of accumulating output feature maps at the end of every dense block.

2.3 Implementation details

DenseNet, and the mini-batch size is 64. Also, the number of training epochs is 300. Additionally, all the PCSB-Nets are trained by the stochastic gradient descent (SGD) method. Finally, Nesterov momentum [11] with 0 dampening is employed to optimize. The initial learning rate starts from 0.1, which is divided by 10 at 150 and 225 training epochs. The momentum is 0.9 and weight decay is $1e - 4$.

3 Experimental results and analysis

3.1 Datasets

CIFAR datasets includes CIFAR-10 (C10) and CIFAR-100 (C100). They both have $60,000$ colored nature scene images in total and the images' size is "32×32". There are $50,000$ images for training and $10,000$ images for testing in 10 and 100 classes. Data augmentation is the same with the common practices. The augmented datasets are marked as C10+ and C100+, respectively.

3.2 Comparisons with state-of-the-art models

Performance We place special emphasis on verifying PCSB-Nets have a better performance in spirit of utilizing fewer or same parameters than all the competing architectures. Since DenseNets have much fewer parameters than other traditional models, the PCSB-Nets are implemented based on DenseNets to make smaller networks. First of all, the number of parameters of DenseNets and the PCSB-Nets is set equally to test the capability of this novel architecture and the results are listed in Table 1. Compared with NiN, FratalNets and different versions of ResNets, the PCSB-Nets achieve relatively better performance on all the datasets, especially on C10 and C100 datasets. Besides that, the proposed models have much fewer parameters than them.

Then the PCSB-Nets are compared with DenseNets. For the models with $1.0M$ parameters, the PCSB-Net ($\alpha = 2$, $K = 20$) achieves relatively better performance than DenseNet (*e.g.*, the error rate is 5.62% vs 7.11% on C10, 5.56% vs 5.82% on C10+, 24.73% vs 29.26% on C100, and 24.32% vs 26.96% on C100+). Furthermore, when the number of parameters increases to $2.2M$, the PCSB-Net ($\alpha = 4$, $K = 32$) has a better performance on the reduction of error rate than DenseNet (*e.g.*, the error rate is 5.12% vs 5.80% on C10, 3.92% vs 5.08% on C10+, 24.16% vs 26.86% on C100 and 21.80% vs 25.46% on C100+). Especially, the other PCSB-Nets all have obtained better results on some relevant datasets, for example, compared with DenseNet ($2.2M$ parameters), the PCSB-Net ($\alpha = 2$, $K = 28$, $1.86M$ parameters), utilizing intermediate activation (we will talk about its effect later), has error rate reduction of 4.26% and 3.70% on C100 and C100+ with fewer parameters.

Parameter efficiency In order to test the efficiency of parameters, a tiny PCSB-Net is designed, which has a little difference from PCSB-Net ($\alpha = 2$, $K = 20$). To design a tiny model using fewer parameters and obtain a competitive result, the output channels of the last layer's bottleneck in each module is set to K/2, and the two transition module's output is respectively set to 160 and 304. This tiny network has only $0.44M$ parameters and obtains a comparable performance (the error rate is shown in Table 1) to that of DenseNet (with $1.0M$ parameters). In addition, in comparison with other architectures, although some PCSB-Nets have much fewer parameters, they all get better performance.

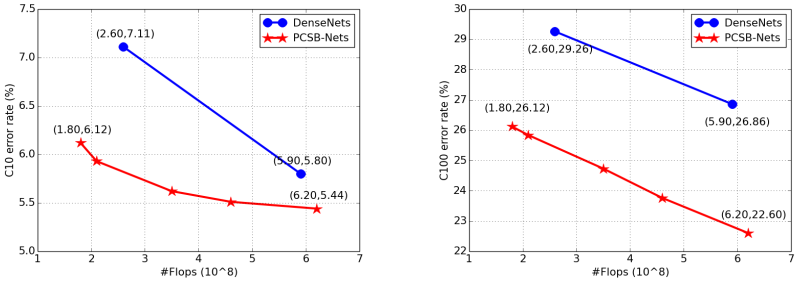
Table 1: Test error rate (%) on CIFAR datasets: α denotes the combination factor and K denotes the number of output feature map channels in every basic module. Results that are better than all competing methods are marked **bold** and the overall best results are **blue**. “+" indicates the data augmentation (translation and/or mirroring) of datasets. “$*$" indicates results obtained by our implementations. All the results of DenseNets are run with Dropout. The proposed models (*e.g.*, tiny PCSB-Net) achieve lower error rates while using fewer or equal parameters than DenseNet.

Method	Version and Structure Settings	Depth	Params	C10	C10+	C100	C100+
ResNet	[3]	110	1.7M	-	6.61	-	-
	reported by [5]	110	1.7M	13.63	6.41	44.74	27.22
	with Stochastic Depth [5]	110	1.7M	11.66	5.23	37.80	24.58
		1202	10.2M	-	4.91	-	-
	pre-activation (reported by [4])	164	1.7M	11.26	5.46	35.58	24.33
		1001	10.2M	10.56	4.62	33.47	22.71
	Wide ResNet[14]	16	11.0M	-	4.81	-	22.07
DenseNet	$K = 12$	40	1.0M	7.11$*$	5.82$*$	29.26$*$	26.96$*$
		58	2.2M	5.80$*$	5.08$*$	26.86$*$	25.46$*$
PCSB-Net	$\alpha = 2$, $K = 20$	150	1.0M	**5.62**	5.56	**24.73**	24.32
	$\alpha = 2$, $K = 28$	150	1.86M	**5.44**	5.08	**25.12**	22.36
	$\alpha = 3$, $K = 27$	150	1.6M	**5.08**	4.96	**24.56**	**22.76**
	$\alpha = 4$, $K = 32$	150	2.2M	**5.12**	**3.92**	**24.16**	**21.80**
	$\alpha = 2$, $K = 28$, with inter-active	150	1.86M	**5.32**	4.88	**22.60**	**21.76**
tiny PCSB-Net	-	150	**0.44M**	6.12	5.96	**26.12**	25.45

Since the pyramidal combination of separable branches implemented by “grouped convolution" operation is utilized in every module, fewer parameters can be used to learn the object features, so that the parameters’ ability of catching information is largely improved. From these contrast experiments, the novel models have significantly improved the capability of parameters.

Computational complexity As shown in Fig. 5, the complexity of computations is compared between DenseNets and the PCSB-Nets. The error rates reveal that the PCSB-net can achieve generally better performance than DenseNets with likely times of flop operations (“multiply-add" step). Especially in Fig. 5(b), the PCSB-net with fewest flops obtains the lower C100 error rate than DenseNet with most Flops ($(1.8 \times 10^8,\ 26.12\%\ )$ vs $(5.9 \times 10^8,\ 26.86\%\ )$). The less computational complexity is attributed to our branchy structure in the basic module.

This branchy structure results in the input feature maps and convolutional filters are all divided into a specific number of groups. In other words, each filter only performs on much less input channels in the corresponding group, which is contributing to much fewer flop operations.



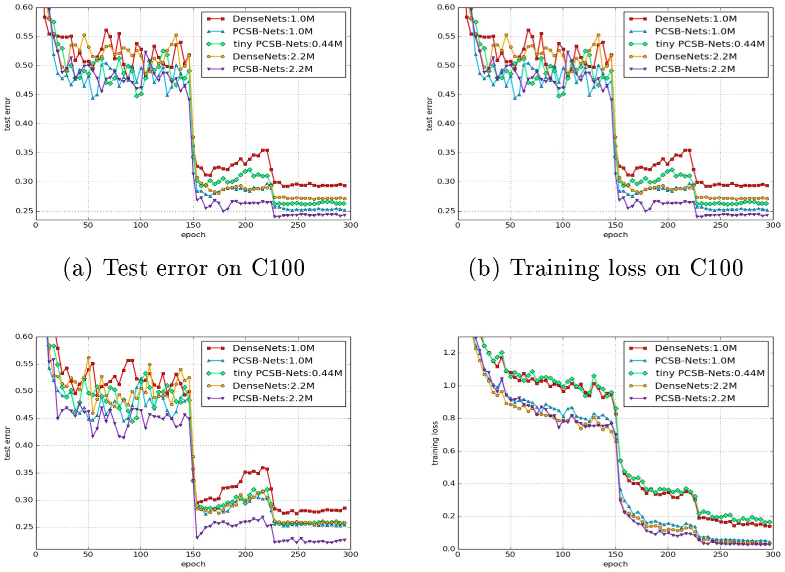
(a) Comparison on C10

(b) Comparison on C100

Fig. 5: Computational complexity comparison.

Model's capacity and overfitting As the increasing of α and K, our model can achieve a better performance, which may result from more parameters. Especially, this trend can be evidently seen from the Table 1, compared with PCSB-Net ($\alpha = 2$, $K = 20$), the error rate obtained from PCSB-Net ($\alpha = 4$, $K = 32$) decreases by 1.64% and 2.52% on the C10+ and C100+ datasets respectively. Based on these different PCSB-Nets' results, our model can catch the features better when the model has more branches and output feature maps in every module at the first and second dense blocks.

In addition, it also suggests that the bigger networks do not get into any troubles of optimization and overfitting. Due to C100 and C100+ datasets have an abundance of object classes, who can validate the performance of models more strongly. The error rate and training loss are plotted in Fig. 6, which are obtained by DenseNets and PCSB-Nets on these two datasets in the course of the training. From Fig. 6(a) and 6(c), our models have lower error rate converges than DenseNets, despite the tiny PCSB-Net has only $0.44M$ parameters. And with the increase of the number of parameters, the model will have a lower error rate. However, in Fig. 6(b) and 6(d), the training loss obtained by PCSB-Net (with $1.0M$ parameters) converges lower than DenseNet (with $1.0M$ parameters). While to the models with $2.2M$ parameters, PCSB-Net and DenseNet have reached an almost identical training loss converges, especially, PCSB-Net converges even slightly higher than DenseNet on C100, which implies our model can avoid overfitting problem better, when the model has a bigger size without adequate training data. We argue although the bigger networks have much more parameters, they also have many branches in every module leading to distributing the parameters to these branches. Accordingly, this architecture can be regarded as an ensemble of many small networks from both horizontal and vertical aspects contributing to alleviating the trouble of overfitting well.



(a) Test error on C100

(b) Training loss on C100

(c) Test error on C100+

(d) Training loss on C100+

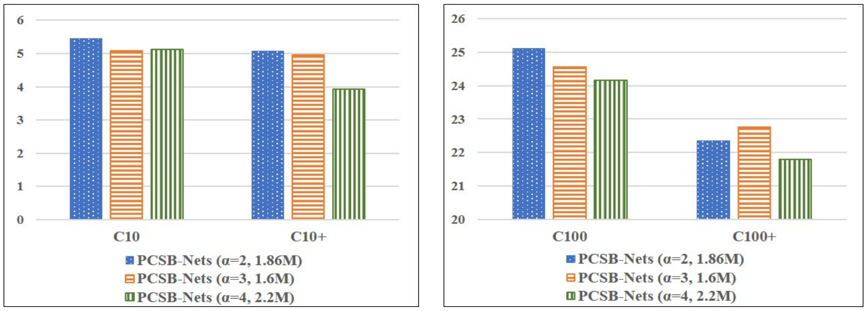
Fig. 6: Training profile on C100 and C100+: comparison of error rate and training loss in the process of training about different models with different number of parameters on C100 and C100+.

3.3 Affected factors of PCSB-Nets' optimization

In this part, we will discuss some affected factors by selecting appropriate hyper-parameters and some regular techniques to optimize the PCSB-Nets.

Effect of combination factor In order to explore the efficiency of the PCSB-Net's module, we observe the results of different PCSB-Nets with combination factor $\alpha = 2$, $\alpha = 3$ and $\alpha = 4$ (shown in Table 1). Because of the limitation of the implementation of "grouped convolution", the number of output channels must be the integer times of groups, hence we set $K = 28$, $K = 27$ and $K = 32$ respectively. These three PCSB-Nets are without the intermediate activation operation. Finally, in Fig. 7, we find that with the increase of combination factor α, the error rate approximately displays a decline trend, for instance, the PCSB-Net with $\alpha = 4$ achieves a much better performance on C10+, C100 and C100+.

It also gets a comparable error rate on C10, most probably in that this model is the most complex one and C10 dataset merely has 10 object classes without any data augmentation. Furthermore, the model with $\alpha = 3$ is also superior to the networks with $\alpha = 2$ except on C100+. This is because these two architectures have almost the same output feature map channels of every module and the former has much more branches at the bottom layer, which results in PCSB-Net ($\alpha = 3$) has smaller parameters and obtains better performance on those less abundant datasets. However, in Fig. 7(b), C100+ is the most abundant dataset, consequently, PCSB-Net ($\alpha = 3$) has a relatively poor generalization, but it also has a competitive result (22.76% vs 22.36%). On account of the comparison of these three networks, the models with more branches will gain a better accuracy and improve the efficiency of parameters, since the networks with more branches will catch the images' information from much more perspectives and fuse the information more thoroughly and carefully.



(a) Error rate (%) on C10 and C10+ (b) Error rate (%) on C100 and C100+

Fig. 7: Effects of combination factor α.

Effect of intermediate activation In the process of the implementation of the above PCSB-Net, we didn't utilize any intermediate activations between the “grouped point-wise convolution" and “ grouped convolution" operations. But traditional models always take advantage of this operation to improve performance and we discover that using the intermediate activation properly can obtain surprising results. For example, PCSB-Net ($\alpha = 2, K = 28$) with intermediate activation can drop the error rate to 21.76% on C100+ dataset. This model has fewer parameters and gets even slightly better performance than PCSB-Net ($\alpha = 4, K = 32$) without intermediate activation.

However, not all the models are suitable for this operation, in order to use it more reasonably, we perform four different sizes of networks with the same number of branches ($\alpha = 2$) and different output feature maps in every module on C100 dataset, and the results have been shown in Table 2. From the results, similar to traditional models, when the model has intermediate activation, the error rate will decrease as the increasing of output channels. But, when the model doesn't have intermediate activation, the error rate will decrease first and increase at the next stage as the growth of the output channels, for example, ($K = 20$, $error = 24.74\%$) is the turning point. Furthermore, when the number of output channels is small, the model utilizing this operation will obtain a lower accuracy than the model without this operation, however, when the number of output channels is relatively large, the model with intermediate activation will perform much better in the experiments.

Table 2: Intermediate activation effect on different sizes PCSB-models.

model	with inter-active C100 err.	without inter-active C100 err.
PCSB-Nets($\alpha = 2$, $K = 16$)	26.08	25.84
PCSB-Nets($\alpha = 2$, $K = 20$)	25.62	**24.74**
PCSB-Nets($\alpha = 2$, $K = 24$)	23.76	25.08
PCSB-Nets($\alpha = 2$, $K = 28$)	**22.60**	25.12

As being shown by Chollet in [1], Xception model doesn't utilize the intermediate activation, since in shallow deep feature spaces (one channel per branch), the non-linearity may be detrimental to the final performance most probably because of the loss of information. For PCSB-Net, a very small number of output feature maps will lead to every branch also with fewer feature maps, which can be seen close to the Xception model, accordingly, our models have nearly same properties with it and don't need the intermediate activation. Moreover, when the number of output feature maps increases in a small range, the performance will be improved because the capacity growth of networks brings more positive impact than the negative impact from not utilizing intermediate activation to the final results. On the contrary, when every branch has relative more input channels, it is best to employ this operation in the model, since every branch will have much ampler information.

4 Conclusions

A novel kind of network architecture, PCSB-Net, is introduced in this paper. This architecture can deal with bottom information from many aspects and fuse the features gradually by the exponentially pyramidal combination structures. Additionally, we also explore some affected factors of the PCSB-Net's optimization, such as the combination factor and intermediate activation operation. This kind of structures are evaluated on CIFAR datasets. The experimental results show that even the tiny PCSB-Net can achieve a relatively better and competitive performance with just $0.44M$ parameters (about half of the parameters of the smallest DenseNets). In comparison with other models, all the PCSB-Nets obtain better performances with much fewer parameters. Consequently, the PCSB-Nets can largely improve the parameters' efficiency without performance penalty.

Acknowledgement

The work is supported by the NSFC fund (61332011), Shenzhen Fundamental Research fund (JCYJ20170811155442454, GRCK2017042116121208), and Medical Biometrics Perception and Analysis Engineering Laboratory, Shenzhen, China.

References

1. Chollet, F.: Deep learning with separable convolutions. arXiv preprint arXiv:1610.02357 (2016)
2. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. Journal of Machine Learning Research **15** (2011)
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (June 2016)
4. Huang, G., Liu, Z., Weinberger, K.Q.: Densely connected convolutional networks (2016)
5. Huang, G., Sun, Y., Liu, Z., Sedra, D., Weinberger, K.: Deep networks with stochastic depth. In: European Conference on Computer Vision. pp. 646–661. Springer International Publishing (2016)
6. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. Computer Science (2015)
7. Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan: Caffe: Convolutional architecture for fast feature embedding. Eprint Arxiv pp. 675–678 (2014)
8. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems **25**(2), 2012 (2012)
9. Larsson, G., Maire, M., Shakhnarovich, G.: Fractalnet: Ultra-deep neural networks without residuals (2016)
10. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
11. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: International Conference on Machine Learning (2013)
12. Szegedy, C., Liu, W., Jia, Y., Sermanet, P.: Going deeper with convolutions. In: Computer Vision and Pattern Recognition. pp. 1–9 (2014)
13. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. Computer Science (2015)
14. Zagoruyko, S., Komodakis, N.: Wide residual networks. arXiv preprint arXiv:1605.07146 (2016)