Enriching Variety of Layer-wise Learning Information by Gradient Combination

Chien-Yao Wang[1], Hong-Yuan Mark Liao[1], Ping-Yang Chen[2], and Jun-Wei Hsieh[3]
[1]Institute of Information Science, Academia Sinica, Taiwan
[2]Department of Computer Science, National Chiao Tung University
[3]College of Artificial Intelligence and Green Energy, National Chiao Tung University
x102432003@yahoo.com.tw, liao@iis.sinica.edu.tw,
pingyang.cs08g@nctu.edu.tw, and jwhsieh@nctu.edu.tw

Abstract

This study proposes to use the combination of gradient concept to enhance the learning capability of Deep Convolutional Networks (DCN), and four Partial Residual Networks-based (PRN-based) architectures are developed to verify above concept. The purpose of designing PRN is to provide as rich information as possible for each single layer. During the training phase, we propose to propagate gradient combinations rather than feature combinations. PRN can be easily applied in many existing network architectures, such as ResNet, feature pyramid network, etc., and can effectively improve their performance. Nowadays, more advanced DCNs are designed with the hierarchical semantic information of multiple layers, so the model will continue to deepen and expand. Due to the neat design of PRN, it can benefit all models, especially for lightweight models. In the MSCOCO object detection experiments, YOLO-v3-PRN maintains the same accuracy as YOLO-v3 with a 55% reduction of parameters and 35% reduction of computation, while increasing the speed of execution by twice. For lightweight models, YOLO-v3-tiny-PRN maintains the same accuracy under the condition of 37% less parameters and 38% less computation than YOLO-v3-tiny and increases the frame rate by up to 12 fps on the NVIDIA Jetson TX2 platform. The Pelee-PRN is 6.7% mAP@0.5 higher than Pelee, which achieves the state-of-the-art lightweight object detection. The proposed lightweight object detection model has been integrated with technologies such as multi-object tracking and license plate recognition, and is used in a commercial intelligent traffic flow analysis system as its edge computing component. There are already three countries and more than ten cities have deployed this technique into their traffic flow analysis systems.

1. Introduction

Since AlexNet [6] won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [1] in 2012, the era of Deep Convolutional Networks (DCN) has begun. After that, the architecture of Plain networks (PlainNet) makes VGG-16 and VGG-19 [16] achieve excellent performance. But the researchers also found that when a DCN reaches a certain depth, the accuracy of PlainNet begins to decrease as the depth increases. This problem is uneasy to solve, even if GoogLeNet [18] adopts the strategy of auxiliary loss, it still cannot solve the problem. In the design of Residual networks (ResNet), He *et al.* [2] introduce the concept of identity shortcut connection, which attempts to make gradients more efficiently propagate to all layers. Using this concept, they successfully built a number of very deep architectures, such as ResNet-50, ResNet-101, and ResNet-152. The highway networks [17] published before ResNet and the stochastic depth [5] developed after ResNet all adopted the shortcut concept. Also want to let gradients quickly propagate to the various layers, Larsson *et al.* [7] used the characteristics of fractal to achieve the goal. They also proposed the concept of drop path to increase the efficiency of information propagation. In [4] and [3], Huang *et al.* proposed dense networks (DenseNet) and condense networks (CondenseNet), respectively, to directly let the loss layer link to all layers. However, the sparse networks (SparseNet) proposed by Zhu *et al.* [24] have confirmed that DCN can achieve better learning results under a designed topology architecture.

From the above state-of-the-art work, we found that the strategy to improve DCN performance is nothing more than how to combine features and propagate to subsequent layers, and how to make gradients more efficiently propagate to all layers. After an in-depth examination of the learning process of the above mentioned models, we propose a new perspective, i.e., "How to combine gradients of each layer in the training process to achieve better learning results?" In this study, we propose the concept of partial residual networks (PRN). We converted the residual connection into a path that produces the combinations of gradients and designed several PRN-based models to verify the proposed concept. Since the strategy for designing PRN is no longer to propagate combination of features, but rather the combination of gradients, it can be more suitable for lightweight networks. This is mainly because the combination of features will produce new layers, while the combination of gradients will not. Because PRN has a lightweight nature, it can also be effectively applied to real-time inferences in embedded devices. Since the performance of lightweight networks is often limited by the learning power of a network, PRN's design concept highlights its advantages in a lightweight network architecture.

Since object detection is the basic technology that many real-world systems need to use, we choose the object detection task to verify the performance of the PRN and prove that it works perfectly for lightweight models. We compare PRN with other state-of-the-art lightweight object detection methods, such as YOLO-v3-tiny [14], Pelee [20], and tiny-DSOD [22]. The main baseline architecture of PRN is YOLO-v3-tiny. Pelee combined DenseNet and SSD [12] and form a 2-way dense layer. The architecture of tiny-DSOD is based on DSOD proposed in [15]. It combines depth-wise convolution and proposes to use depth-wise dense blocks to make the network lightweight. We compared the execution time of CPU, GPU, and Jetson TX2 on different lightweight models, and found that our proposed PRN has significant advantages over other models. The contributions of this work are summarized as follows:

- PRN enables shallow networks to learn abundant information.
- PRN can combine layers with different number of channels, thus it has excellent flexibility in architecture design.
- The inference process of PRN occupies very little resources and is suitable for deployment in highly restricted embedded systems.
- Compare SparseNet(+) with PRN, the former is sparse in layer, and the latter is sparse in channel, so PRN is more suitable for shallow network architecture.
- PRN runs very fast, which is helpful for tasks that require real-time processing.

The proposed PRN-based object detection model has been deployed into commercial traffic flow analysis system. With this AI software-embedded device, we are able to perform real-time analysis on many traffic-related parameters.

2. Partial Residual Networks

In this section we will explore the structure and characteristics of PRN. In Section 3, we will use the concept of gradients to explain why PRN, ResNet, DenseNet, and SparseNet can work.

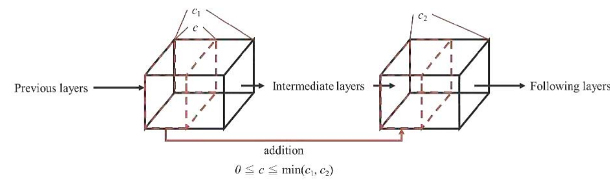


Figure 1. Partial residual connection.

The PRN we propose is a stack of partial residual connection blocks, and the structure of partial residual connection is shown in Figure 1. When we apply partial residual connection on the feature maps of the c_2 channels of the l^{th} layer and the c_1 channels of the $(l-k)^{th}$ layer, we will apply the addition operation to the first c ($c \leq c_1$) channels in the $(l-k)^{th}$ layer with the shortcut connected to the first c ($c \leq c_2$) channels of the l^{th} layer. After the current c channels have done addition, the corresponding feature map will feedforward to subsequent layers. The way how partial residual connection operates can be expressed as follows:

$$x_l = [H_l(x_{l-1})_{[0:c-1]} + x_{l-k}{}_{[0:c-1]}, H_l(x_{l-1})_{[c:c_l]}] \quad (1)$$

where x_l represents the feature map which is the output of the l^{th} layer, H_l represents the non-linear transformation function of the l^{th} layer, and $[a, b, ...]$ means to perform concatenation on $a, b, ...$. Therefore, $[a : b] = [a, a+1, ...b]$ represents the corresponding feature map ranging from channel a to channel b. From the above equation we can see that PlainNet is a special case of PRN. That is, when $c = 0$, Equation 1 can be written as $x_l = H_l(x_{l-1})$. This is equivalent to making a non-linear transformation layer-by-layer. For the case of ResNet, Equation 1 can be rewritten as $x_l = H_l(x_{l-1}) + x_{l-k}$. This is equivalent to a special case of PRN when $c = c_1 = c_2$.

PRN's architecture is highly flexible in design, allowing feature maps of arbitrary layers and arbitrary number of channels to be combined. The PRN architecture is very different from that of ResNet, because the latter limits the blocks within the same stage to have the same number of channels. The operation mechanism of PRN is not like the way of skip connection. The skip connection needs to combine feature maps formed by different number of channels. The method of skip connection requires an additional transition layer to integrate feature maps of the above mentioned sort. The design and inference of PRN proves that this is unnecessary. In addition, during the inference phase, the shortcut connection and the concatenate layer need to save the shortcut or the concatenate feature map that is still needed in the memory. By the design of PRN, when the number of channels for partial residual connection is 50% of the total channel number, we can save up to half of the memory space required for the inference phase.

This study proposes three architectures to highlight and validate the characteristics and benefits of PRN, which are described as follows:

- Characteristic 1: Without changing the topology of ResNet, replace only the residual connection with the partial residual connection of sparsity = ρ, as shown in Figure 2. The computational complexity of this architecture is almost identical to that of ResNet, which we use to compare the learning power of PRN and ResNet.

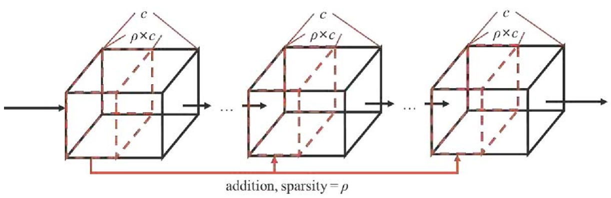


Figure 2. Characteristic 1: maintain the topology of ResNet, change offset, use sparsity = ρ as the basis for partial residual connection.

- Characteristic 2: The number of channels of the block in the same stage is linearly decremented from c to $\gamma \times c$, as shown in Figure 3 for the example of reduce rate = γ. This architecture can be used to demonstrate the flexibility of PRN when combining different channel numbers, and the stage to be modular PRN.

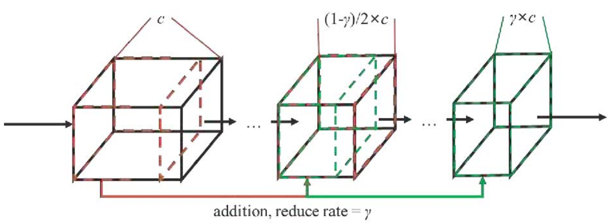


Figure 3. Characteristic 2: Set reduce rate = γ, the number of channels linearly decremented to $\gamma \times c$, partial residual connection operates on feature maps with different channel numbers in the same stage.

- Characteristic 3: The architecture of Figure 4 shows that PRN can be highly flexible because it can be combined with multiple sets of distinct-channel feature maps.

3. Combination of Gradients

The propagated gradients mainly consist of two parts: one is the gradient source, and the other is the timestamp of gradient. We use G_t^s to represent it, where s is the source and t is the timestamp. We will use timestamp and source, respectively, to observe how the gradient is composed during the training process. We will also explain why ResNet, ResNext [21], DenseNet, SparseNet, and the proposed PRN can learn more effectively based on the combination of gradients.

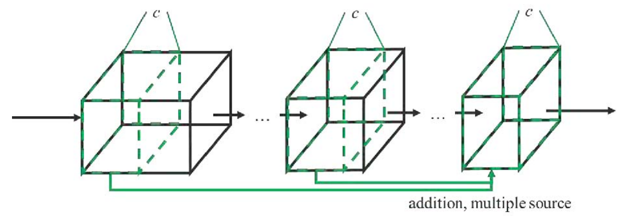


Figure 4. Characteristic 3: Receive partial residual connection from different sources simultaneously.

3.1. Timestamp

The analysis of gradient timestamp for residual connection, partial residual connection, and dense connection is shown in Figure 5, where ResNet, PRN, DenseNet, and SparseNet are all five-layer DCN. As can be seen from Figure 5, ResNet is equivalent to the integration of m k-layer shallow networks, and the total number of layers is $L = m \times k + 1$. The gradient flow will form a cycle for every k timestamps along the time axis and continue to propagate to the bottom layer, and this propagation will last for m cycles. Another characteristic of PRN is that it has only part of the channels connected to the subsequent layers during partial residual connection. It is this reason that makes it possible to naturally produce a variety of gradient combinations, in total m sets of combinations, when propagating backwards. If PRN further combines partial residual connections from multiple sets with different number of channels, more combinations can be obtained.

As to DenseNet, it links all layers directly to the loss layer, so for the first layer, it receives all gradient flows from the timestamps sitting in the $1 \leq t \leq L - l + 1$ duration. For the case of SparseNet, it does the sparse connection at a power of 2 when doing the concatenation. Compared with DenseNet, SparseNet uses the same gradient combinations along the time axis. It increases the level of gradient combinations along the time axis, and this operation makes its number of gradient combinations equal to that of a DenseNet.

3.2. Source

We use Figure 6 to show how gradient source can be analyzed for residual connection, partial residual connection, and dense connection. In Figure 6 we only draw the gradient source for $G_{t=1}^s$. It is obvious that the DenseNet and the SparseNet are concatenate-based, which is different from the ResNet that is shortcut-based. In the gradient propagation of a concatenate-based architecture, the gradients propagated by the same layer with the same timestamp are split and not intersected. As to the ResNet architecture, it is to propagate exactly the same gradient to all layers of the target. This explains why ResNet is easy to learn a lot of redundant information. In order to solve the above mentioned problem, PRN separates part of the gradients to produce a richer gradient

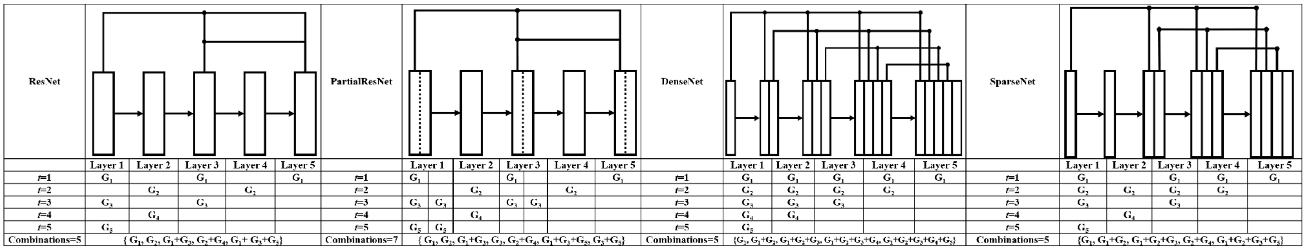


Figure 5. Gradient timestamp and routes for ResNet, PRN, DenseNet and SparseNet.

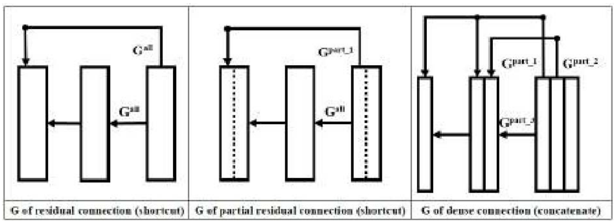


Figure 6. Shortcut connection and concatenate have different gradient propagation sources.

combinations. As for ResNext, it uses the split-transform-merge strategy when propagating forward, so it gets similar outcome as what PRN gets.

3.3. Summary

Through the analysis of the timestamp and source generated by the gradients in the backpropagation process, we can clearly explain how ResNet, PRN, DenseNet, and SparseNet learn the information and how efficient the parameters can be utilized. In ResNet, different layers share many gradients coming from the same timestamp and the same source, while DenseNet propagates gradients coming from different sources but have the same timestamp. This explains why DenseNet can avoid the problem of learning a lot of redundant information like ResNet. The sparse connection strategy of SparseNet splits the gradient timestamp into different layers, which reduces the gradient source but increases the number of changes on gradient timestamp, and also reduces the amount of computation. As for the proposed PRN, it performs partial residual connection on the channel, which increases the number of combinations of the gradients in the timestamp without increasing the number of layers, and causes the gradient source to generate the shunt, and also increases the variability of the gradient source.

4. Experimental Results

4.1. Experimental setting

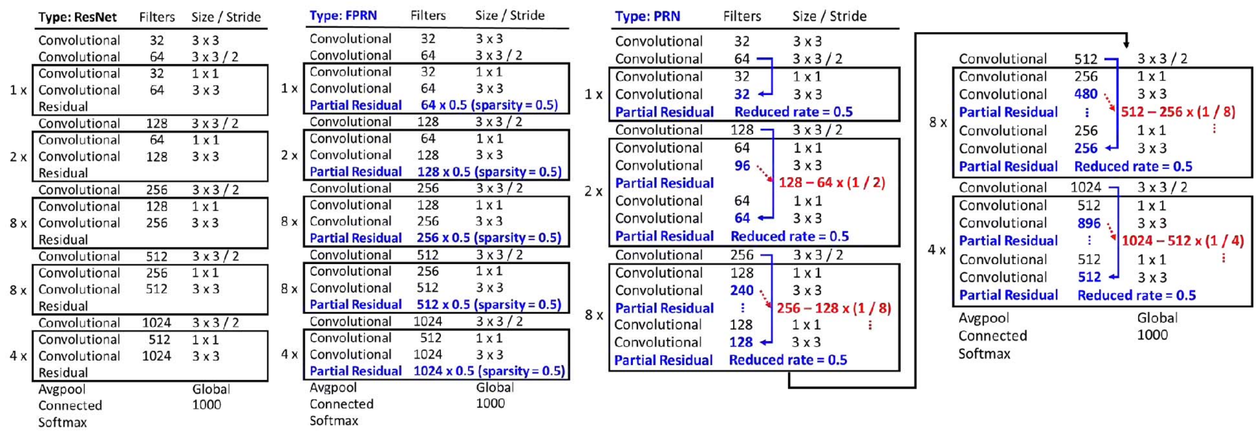
We use MSCOCO object detection dataset [10] to evaluate PRN. We implement PRN on top of the Darknet framework [13]. In order to reduce the impact caused by adjusting hyperparameters, all settings follow the specifications of YOLO-v3. The training set adopted is the trainval35k subset. We design four different PRN models to evaluate their performances. The four models and their corresponding settings are illustrated in Table 1. Among the four PRN models, YOLO-v3-FPRN (Figure 7.(b)) and YOLO-v3-PRN (Figure 7.(c)) are based on YOLO-v3 (Figure 7.(a)), but they are equipped with characteristic 1 and characteristic 2 of PRN, respectively. During the training process, the hyperparameters used by these two models are exactly the same as those used by YOLO-v3. As to v3-tiny-PRN and Pelee-PRN, their architectures are respectively based on YOLO-v3-tiny and Pelee models plus PRN head (Figure 8). During the training process, the hyperparameters used by v3-tiny-PRN and Pelee-PRN are exactly the same as those adopted by YOLO-v3-tiny.

Table 1. PRN models for evaluation.

Models	
Name	Description
YOLO-v3-FPRN	The network topology and the number of convolutional kernels are exactly the same as YOLO-v3. One needs to set the threshold of sparsity to 0.5 when the shortcut connection is accumulated.
YOLO-v3-PRN	Based on YOLO-v3, the five stages of the Darknet-53 backbone gradually decrement the channel to reduce-rate = 0.5. The partial residual connection we designed can work on feature maps with different channel numbers.
v3-tiny-PRN	Add multiple sets of partial residual connection to the YOLO-v3-tiny head, including those partial residual connections combining from multiple sources.
Pelee-PRN	On the PeleeNet backbone, apply the head of v3-tiny-PRN.

4.2. Analysis on effect of combination of gradient

We have designed several architectures using partial residual connection, which use combination of gradients with different degree of richness. The purpose of this experiment is to verify that when we try to reduce the computational complexity of the model, if our strategy is to maximize the utility of the gradient combination, can we get the most benefit? We validate our results using the validation set used by ImageNet at ILSVRC 2012. In the meanwhile, we also



(a) Darknet53 (b) Darknet53-FPRN (c) Darknet53-PRN

Figure 7. Architectures of Darknet53, Darknet53-FPRN, and Darknet53-PRN.

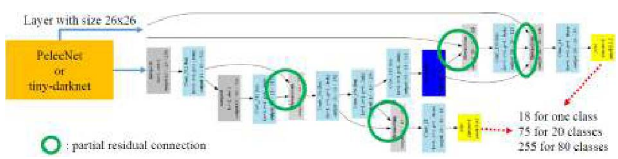


Figure 8. Architecture of tiny PRN head.

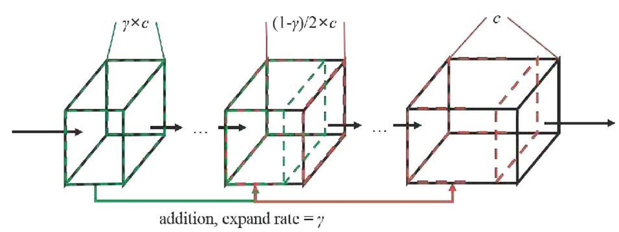


Figure 9. Architecture of PRN-expand.

analyze how recognition rate is affected when we adjust γ to control the amount of computation.

The PRN mentioned in Figure 3 at Section 2 may be gradually shrunk as the number of channels increases with the number of layers. Therefore, in this section we call it PRN-shrink. When the gradients of PRN-shrink pass forward to each of the subsequent layers, distinct number of channels will be affected. Therefore, for a k-layer PRN-shrink, the total number of combination of gradient timestamp is $1 + 2 + ... + k = \frac{(1+k)\times k}{2}$. The channel that is updated by gradient combination sent at different points in time will be considered as a different source and the total number of gradient source is $1 + 2 + ... + k = \frac{(1+k)\times k}{2}$.

In most CNN architectures, the number of convolutional filter channels contained in a stage tends to increase as the stage is getting deeper. Here we increase the number of channels of the blocks of the same stage as block is getting deeper, and introduce the proposed partial residual connection into the architecture. As illustrated in Figure 9, we call the modified architecture PRN-expand. When the gradients under this structure are backward propagating, the total number of combination of gradient timestamp is equal to $1+1+...+1 = k$. In the forward propagation phase, the partial residual connection will choose to add unequal channel numbers for different feature maps. Therefore, the source of the total number of gradient source is $1 + 2 + ... + k = \frac{(1+k)\times k}{2}$.

In the final phase, we adopt the encoder-decoder architecture that auto-encoder often uses, and incorporate the concept of partial residual connection into this architecture. Since the number of middle layer channels in this architecture is extremely small, we call this part PRN-bottleneck, as shown in 10. When the gradient flow passes through the bottleneck layers, these gradients will be integrated into one gradient flow. So we can think of the above integration as a stack of PRN-shrink and PRN-expand. The total number of combination of gradient timestamp of this part, under these circumstances, is $\frac{(1+k/2)\times(k/2)}{2} + (k/2) = \frac{(1+k/2)\times(k/2)+k}{2}$, it is sitting between PRN-shrink and PRN-expand. The total number of gradient sources is $1 + 2 + ... + (k/2) + (k/2) + ((k/2) - 1) + ... + 1 = 2 \times \frac{(1+k/2)\times(k/2)}{2} = (1 + k/2) \times (k/2)$, it is roughly about one half of PRN-shrink and PRN-expand.

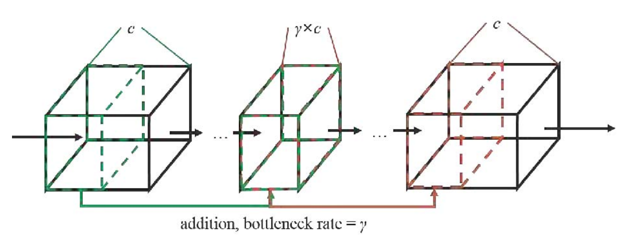


Figure 10. Architecture of PRN-bottleneck.

Figure 11 shows the results of PRN testing on ImageNet, designed according to three strategies. From results shown in Figure 11, when the architecture is designed to reduce the amount of computation, the PRN-shrink designed by maximizing the gradient combination receives the best results. It receives 76.8% and 76.4% recognition rates when the γ is set 0.75 and 0.5, respectively. For the same γ settings, PRN-expand receives 75.7% and 72.7% recognition rates, respectively, and PRN-bottleneck receives 75.5% and 72.7% recognition rates. From the outcomes received by applying PRN-bottleneck and PRN-expand, it is obvious that the benefit of increasing the gradient source combination is higher than that of increasing the gradient timestamp combination. In addition, we observed the effect of changing the γ value and found that the recognition rate of PRN-shrink which is designed to follow the maximal gradient combination linearly and gradually decreases as γ linearly decreases. However, for a PRN which is not designed according to the rule, its recognition rate declines exponentially.

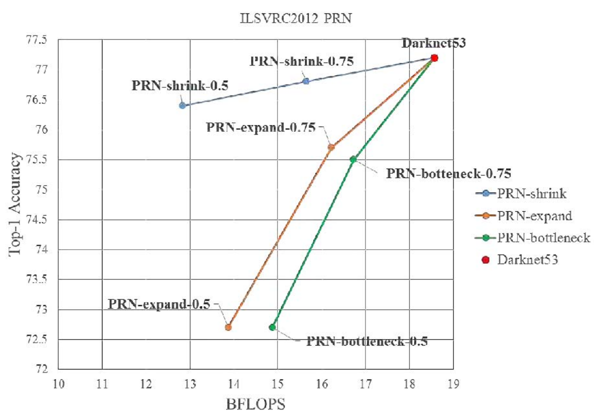


Figure 11. Results of PRN-shrink, PRN-expand, and PRN-bottleneck on ImageNet.

4.3. Comparison with GPU-based real-time models

Table 2 illustrates the comparison of our PRN model with those models with GPU running speed over 30 fps in the literature. First, we compare our PRN with the baseline YOLO-v3 model. For YOLO-v3-FPRN, we only change the offset of all shortcut connections in YOLO-v3, and make a partial residual connection to the subsequent layers with 50% of the channel in the feature map. With only the above simple changes, we increase 1% and 0.5% mAP@.5:.95 on 320×320 and 416×416 input image, respectively. By gradually reducing the number of feature map channels per stage to 50% for YOLO-v3-PRN, we can achieve the same accuracy with 35% less computation and twice the speed of computation. Compared with other state-of-the-art models, YOLO-v3-PRN performs better. For example, YOLO-v3-PRN can speed up 2ms computation time than RBFNet [11], but at the same time gain 0.5% mAP@.5:.95 over RBFNet. The above experiments confirm that under exactly the same conditions, we only need to increase the number of gradient combinations and enrich their content to get better learning results. That is to say, we can use the parameters more effectively and achieve the same learning effect with a smaller number of parameters.

Table 2. Compare with state-of-the-art GPU real time models on MSCOCO test-dev set.

Model	Input size	BFlops	mAP@.5:.95	GPU time
SSD [12]	300	-	25.1%	12 ms
RFBNet [11]	300	-	30.3%	15 ms
RefineDet [23]	320	-	29.4%	26 ms
YOLO-v3 [14]	320	38.973	28.2%	22 ms
YOLO-v3-FPRN **[ours]**	320	38.973	29.2%	21 ms
YOLO-v3-PRN **[ours]**	320	25.106	28.4%	**11 ms**
YOLO-v3 [14]	416	65.864	31.0%	29 ms
YOLO-v3-FPRN **[ours]**	416	65.864	**31.5%**	27 ms
YOLO-v3-PRN **[ours]**	416	42.429	30.8%	13 ms

4.4. Comparison with state-of-the-art lightweight models

In this set of experiments, we will demonstrate the high flexibility of PRN and its advantages in lightweight models. First, we found that when integrating features from multiple stages, either SSD or YOLO-v3 uses concatenate technique. Models such as feature pyramid networks[8]-based (FPN-based) RetinaNet [9] require an additional transition layer to convert the features of different stages into the same number of channels for shortcut connection. These extra operations may be insignificant in large models, but for lightweight models, this is enough to introduce very significant computational cost. The results shown in Table 3 show that our proposed PRN is highly flexible. It has the ability to reduce computing cost and computing time, as well as improving the performance of the model.

Adding PRN to YOLO-v3-tiny (v3-tiny-PRN of Table 3) can reduce 38% of computation, 37% CPU computation time, and 19% GPU computation time, while maintaining the accuracy of 33.1% mAP@0.5. With the NVIDIA Jetson TX2 platform, the above system is 12 fps faster in the object detection task. From the above data, the PRN introduced will undoubtedly result in a full range of performance improvements.

If we compare Pelee and Pelee-PRN, the latter has 47% less parameter. In addition, Pelee-PRN has a much smaller overhead than Pelee, it is far better than Pelee in terms of computing speed. When the input image size is 416×416, Pelee-PRN is 6.7% mAP@0.5 better than Pelee. If we compare the computation time spent on GPU, CPU, and TX2, then Pelee-PRN is 2.9 times, 2.4 times, and 1.8 times faster than Pelee, respectively. If the input image size is 320×320, Pelee-PRN reduces the computation by 7% compared to Pelee, but increases 2.6% mAP@0.5. If we compare the computation speed spent on GPU, CPU, and TX2, then Pelee-PRN is 3 times, 4.3 times, and 2.4 times faster than Pelee.

Overall, the proposed PRN can effectively improve the accuracy of the existing lightweight models with only a small number of parameters. The PRN-based architectures can combine multiple sets of features from different channels to significantly reduce the extra computational cost caused by combining different pyramid features based on concatenate or transition layers.

Table 3. Compare with state-of-the-art lightweight models on MSCOCO test-dev set.

Model	v3-tiny [14]	v3-tiny -PRN **[ours]**	Pelee [20]	Pelee -PRN **[ours]**	Pelee -PRN **[ours]**	Tiny -DSOD [22]	ENet-B0 -PRN **[ours]**
Input size	416	416	304	320	416	300	320
BFlops	5.57	3.47	2.58	2.39	4.04	2.24	2.21
#Parameters	8.86M	4.95M	5.98M	3.16M	3.16M	1.15M	-
MAP@0.5	33.1	33.1	38.3	40.9	45.0	40.4	41.0
GPU time	3.3ms	2.7ms	26ms	8.7ms	9.0ms	21ms	-
CPU time	125ms	78ms	394ms	92ms	166ms	704ms	-
TX2 time	28ms	21ms	68ms	28ms	38ms	71ms	-
TX2 system	37 fps	49 fps	14 fps	37 fps	27 fps	-	-
+visualize	23 fps	28 fps	-	24 fps	19 fps	-	-
+tracking	22 fps	28 fps	-	23 fps	19 fps	-	-
TX2 tensorRT	71 fps	91 fps	49 fps	70 fps	48 fps	-	-

§ GPU: titanX; CPU: intel i7; TX2: Nvidia Jetson TX2.

§ GPU time, CPU time and TX2 time consider model inference time.

§ TX2 system including preprocessing, model inference, and post-processing time.

§ TX2 tensorRT including preprocessing, model inference time.

§ tensorRT can significantly increase the speed, but at the same time lower down the accuracy.

§ Pelee merges convolution layer and batch normalization layer, the time spent on GPU and CPU are 13ms and 271ms, respectively. The accuracy will drop a little bit.

§ SSD-300: GPU 46fps, MAP@0.5 41.2; Mobilenet-v1-SSD-512: GPU 33fps, MAP@0.5 39.2. Pelee-PRN: GPU 108fps, MAP@0.5 45.0.

§ ENet-B0 is EfficientNet-B0 proposed in [19].

4.5. Applications in the real world

The object detection model trained by our PRN architecture has actually worked with the Ministry of Transportations in several countries and established real-time traffic flow analysis platforms in those countries. Our system can instantly analyze information such as vehicle type, number of steering vehicles, vehicle speed, road occupancy, etc. These collected data can be used as the basis for the conversion of traffic signals. Figure 12 shows the intelligent traffic flow analysis machine box of the actual device at the intersection, which houses the embedded computing device we developed and a 1TB SSD for recording videos. Figure 13 shows the fisheye lens-based traffic analysis system at the intersection, which can be used to analyze vehicle type, vehicle color, vehicle speed, and steering traffic flow. Figure 14 shows the traffic tracking system, which can be used to analyze the instantaneous vehicle speed, the length of vehicle queue, etc. Figure 15 shows the road license plate recognition system. In addition to detecting and tracking vehicles, the system can also recognize the license plate.

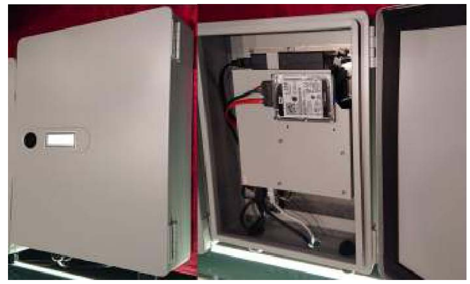


Figure 12. Intelligent traffic flow analysis machine box.

Figure 13. Fisheye lens-based traffic analysis system.

Figure 14. Traffic tracking system.

5. Conclusions

In this study we proposed to use the gradient combination concept to enhance the learning process. We designed partial residual connection and four different PRN architectures to verify our assertion. The proposed PRN has a high degree of flexibility, learning ability, and execution efficiency. Besides, PRN is highly compatible with lightweight models, and it can be easily applied to many advanced DCN architectures,

Figure 15. Road licence plate recognition system.

such as ResNet, FPN, etc. We used MSCOCO dataset for object detection and the experimental results confirmed that YOLO-v3-PRN maintains the same detection accuracy as YOLO-v3 with a 35% reduction of computation in YOLO-v3, while simultaneously increased the speed of execution by twice. In the experiments designed for lightweight models, YOLO-v3-tiny-PRN maintained the same accuracy under the condition of 37% less parameters and 38% less computation than YOLO-v3-tiny. For computational speed, our PRN-based object detection system can increase the frame rate by up to 12 fps on the NVIDIA Jetson TX2 platform. As to the performance of Pelee-PRN, it is 6.7% mAP@0.5 higher than Pelee, which improves the state-of-the-art of the lightweight object detection models.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. IEEE, 2009.

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

[3] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2752–2761, 2018.

[4] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700–4708, 2017.

[5] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In European conference on computer vision, pages 646–661. Springer, 2016.

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

[7] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. arXiv preprint arXiv:1605.07648, 2016.

[8] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2117–2125, 2017.

[9] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In Proceedings of the IEEE international conference on computer vision, pages 2980–2988, 2017.

[10] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In European conference on computer vision, pages 740–755. Springer, 2014.

[11] S. Liu, D. Huang, et al. Receptive field block net for accurate and fast object detection. In Proceedings of the European Conference on Computer Vision (ECCV), pages 385–400, 2018.

[12] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In European conference on computer vision, pages 21–37. Springer, 2016.

[13] J. Redmon. Darknet: Open source neural networks in c. 2013.

[14] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.

[15] Z. Shen, Z. Liu, J. Li, Y.-G. Jiang, Y. Chen, and X. Xue. Dsod: Learning deeply supervised object detectors from scratch. In Proceedings of the IEEE International Conference on Computer Vision, pages 1919–1927, 2017.

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

[17] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. arXiv preprint arXiv:1505.00387, 2015.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1–9, 2015.

[19] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv preprint arXiv:1905.11946, 2019.

[20] R. J. Wang, X. Li, and C. X. Ling. Pelee: A real-time object detection system on mobile devices. In Advances in Neural Information Processing Systems, pages 1963–1972, 2018.

[21] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1492–1500, 2017.

[22] J. L. Yuxi Li, Jianguo Li and W. Lin. Tiny-DSOD: Lightweight object detection for resource-restricted usage. In BMVC, 2018.

[23] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li. Single-shot refinement neural network for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4203–4212, 2018.

[24] L. Zhu, R. Deng, M. Maire, Z. Deng, G. Mori, and P. Tan. Sparsely aggregated convolutional networks. In Proceedings of the European Conference on Computer Vision (ECCV), pages 186–201, 2018.