Appendix for Improving Gradient Paths for Binary Convolutional Neural Networks

Baozhou Zhu[1]
B.Zhu-1@tudelft.nl
Peter Hofstee[1,2]
hofstee@us.ibm.com
Jinho Lee[3]
leejinho@snu.ac.kr
Zaid Alars[1]
z.al-ars@tudelft.nl

[1] Delft University of Technology, Delft, The Netherlands
[2] IBM Austin, Austin, TX, USA
[3] Seoul National University, Seoul, Korea

Abstract

This file is the appendix for Improving Gradient Paths for Binary Convolutional Neural Networks. The first section shows the gradient paths in BinaryDenseNet variants. In the second section, we illustrate the digrams of our proposed architectures. The third section presents the binarization function. In the fourth section, we describe the experimental details.

1 Gradient paths in binary DenseNet variants

Block	$N_{1st}\|L_{1st}$	$N_{2nd}\|L_{2nd}$	$N_{3rd}\|L_{3rd}$
DenseNet	$0\|0$	$N^{D}\|1$	$-\|-$
BinaryDenseNet	$0\|0$	$N_1^{BD}\|1$	$-\|-$
BinaryDenseNet	$0\|0$	$N_1^{BD}\|1$	$-\|-$
EBinaryDenseNet	$0\|0$	$N_1^{ED}\|1$	$-\|-$

Table 1: Evaluation of gradient path quality for binary model blocks. $(\cdot|\cdot)$ refers to the smallest number of operations to compute gradient backpropagation for a gradient path and the shortest gradient path length. For example, $0|0$ indicates that the smallest operation number and the shortest gradient path length for a binary model block are 0. If N_{1st} is the same for two different model blocks, we compare N_{2nd}. Similar, if N_{1st} and N_{2nd} are the same, we compare N_{3rd}. $N_1^{BD} \approx N_2^{BD}$. $N_1^{ED} \approx N_2^{ED} \approx N_3^{ED} \approx N_4^{ED}$.

We present the gradient paths in the blocks of the binary DenseNet variants in Figure 1 and the evaluation of gradient path quality in Table 1.

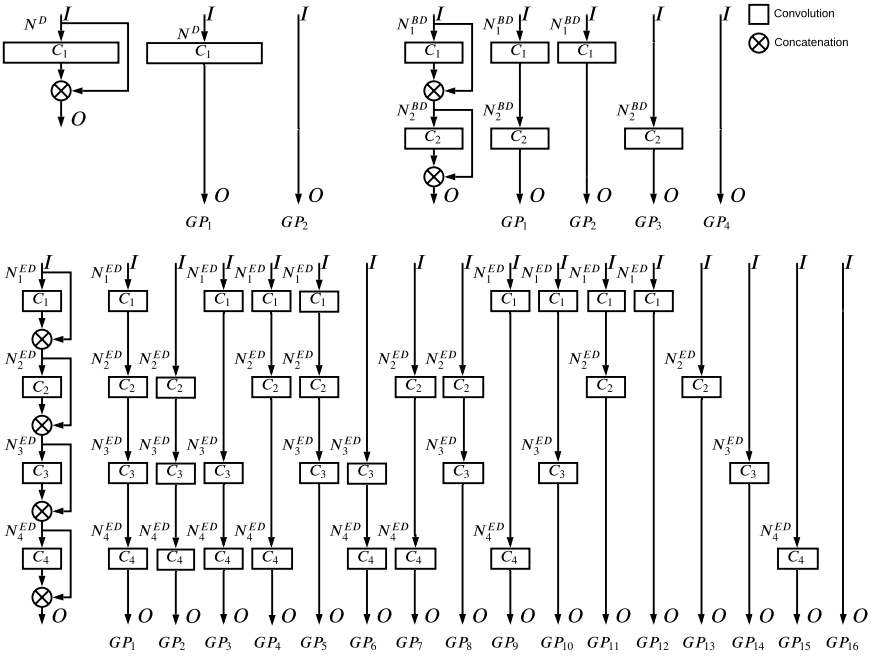

Figure 1: Gradient paths in binary DenseNet variants. **Top left:** Gradient paths in DenseNet block. **Top right:** Gradient paths in BinaryDenseNet block. **Bottom:** Gradient paths in EBinaryDenseNet block. *GP* refers to gradient path. The number of operations to compute gradient backpropagation for a binary convolution layer is N^D in DenseNet, N^{BD} in BinaryDenseNet, and N^{ED} in EBinaryDenseNet, respectively.

1.1 DenseNet vs BinaryDenseNet.

N_{1st} is 0 for BinaryDenseNet and binary DenseNet blocks. N_{2nd} is N_1^{BD} for BinaryDenseNet and N^D for binary DenseNet blocks. we set the computational complexity of different model blocks to be roughly the same, i.e., $N^D \approx N_1^{BD} + N_2^{BD} \approx N_1^{ED} + N_2^{ED} + N_3^{ED} + N_4^{ED}$. $N_1^{BD} \approx N_2^{BD}$. $N_1^{ED} \approx N_2^{ED} \approx N_3^{ED} \approx N_4^{ED}$. Then, $N_1^{BD} < N^D$. Thus, the gradient path quality for BinaryDenseNet block is better than that for binary DenseNet block. Then, it is reasonable that the error of BinaryDenseNet is lower than that of binary DenseNet.

1.2 BinaryDenseNet vs EBinaryDenseNet.

N_{1st} is 0 for BinaryDenseNet and EBinaryDenseNet blocks. N_{2nd} is N_1^{BD} for BinaryDenseNet and N_1^{ED} for EBinaryDenseNet blocks. Then, $N_1^{ED} < N_1^{BD}$. Thus, increasing dense connections further for BinaryDenseNet improves gradient paths. However, the error of EBinaryDenseNet is not lower than BinaryDenseNet. We speculate that the trianing difficulty caused by depth in EBinaryDenseNet is larger than that in BinaryDenseNet, which leads to the fact that EBinaryDenseNet does have better gradient paths and does not have lower error than BinaryDenseNet. In our work, we consider gradient paths for BCNNs and have some assumptions. In particular, we assume that the training difficulties of BCNNs remain unchanged when we design architectures by improving gradient paths. For example, the

training difficulty caused by depth remains unchanged if the depth of the model (Here we consider binary convolutional layers and ignore full-precision layers.) does not double, i.e., $D_{d=18} \approx D_{d=l}$ when $l < 34$ for ResNet18 and $D_{d=51} \approx D_{d=l}$ when $l < 97$ for DenseNet51. The depth of EBinaryDenseNet is twice as large as that of BinaryDenseNet, and the trianing difficulty caused by depth in EBinaryDenseNet is larger than that in BinaryDenseNet, i.e., $D_{d=51} < D_{d=91}$ for DenseNet51.

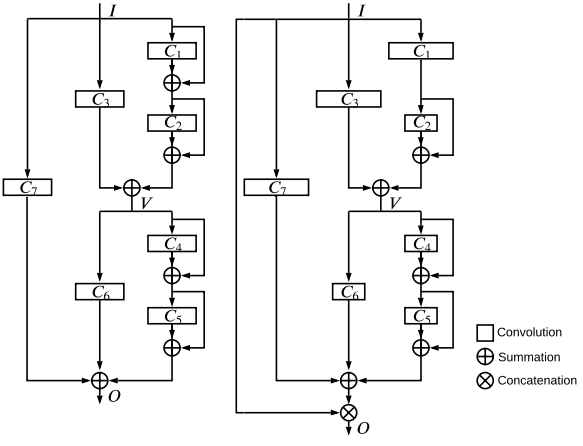

Figure 2: The diagrams of the blocks of proposed architectures with $M = 7$. **Left:** The block description of IGP-ResNet. **Right:** The block description of IGP-DenseNet.

2 Proposed architectures

In this section, we present the architectures of IGP-ResNet and IGP-DenseNet blocks. Then, we show the overview of our proposed architectures.

Proposed architectures with other network architecture configurations. The IGP-ResNet block architecture is shown to the left of Figure 2, where $M = 7$. $M = 7$ indicates that the number of columns and depth is 3 and 4, respectively. Similarly, the IGP-DenseNet block architecture is shown to the right of Figure 2, where $M = 7$.

The IGP-ResNet block architecture is shown to the left of Figure 3, where $M = 15$. $M = 15$ indicates that the number of columns and depth is 4 and 8, respectively. Similarly, the IGP-DenseNet block architecture is shown to the right of Figure 3, where $M = 15$.

Overview of proposed architectures. As shown in Figure 4, we describe the overall view of our proposed architectures with the input images of size 224×224. To ensure a fair comparison when we build our proposed architectures, we scale the number of base channels or the growth rate of our proposed architectures to have almost the same computational complexity as Bi-Real ResNet and BinaryDenseNet.

The left two columns are the IGP-ResNet, i.e., IGP-ResNet21(53) and IGP-ResNet41(48), respectively. 21 and 41 represent the depths of proposed architectures, while 53 and 48 refer to their base number of channels, which are scaled to match the computational com-

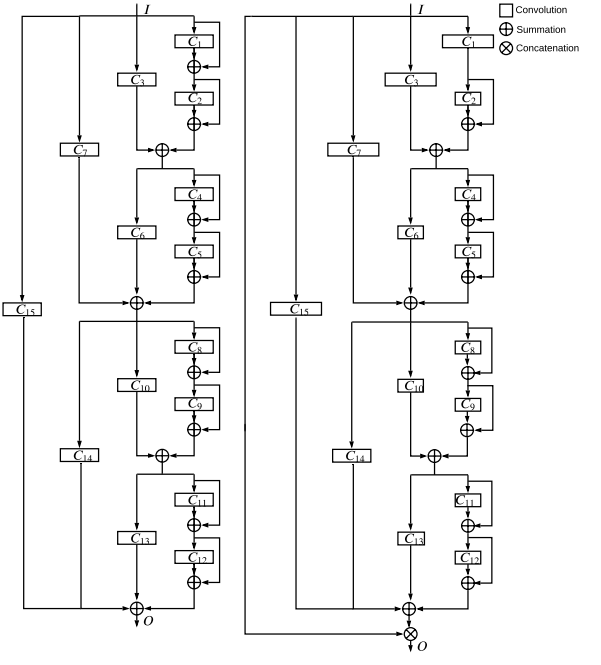

Figure 3: The diagrams of the blocks of proposed architectures with $M = 15$. **Left:** The block description of IGP-ResNet. **Right:** The block description of IGP-DenseNet.

plexity of ResNet18 and ResNet34 after binarization, respectively. Similarly, we build IGP-DenseNet51(53) and IGP-DenseNet69(48) to match the computational complexity of DenseNet51(32) and DenseNet69(32) after binarization [2], respectively. 51 and 69 refer to the depths of proposed architectures, while 53 and 48 refer to the growth rate after scaling. We calculate the model depth with the criteria that every convolutional layer is recognized as one layer, which is different from that in [2] (i.e., every block is recognized as a layer). To ensure consistency, BinaryDenseNet28(64) and BinaryDenseNet37(64) in [2] are renamed as BinaryDenseNet51(32) and BinaryDenseNet69(32) in our paper. The right two columns present the composition of the initial layers, transition block, and final layers in proposed architectures.

3 Binarization function

We describe the binarization function that we adopt for our proposed architectures, including the binarization of weights [7] and activations [6].

Binarization of weights. The forward propagation and backward propagation to binarize the weights are calculated as follows. E and L refer to the mean of the absolute value of the weights and the loss of the model, respectively. W and W_b represent the full precision weights and binary weights. We adopt the straight-through estimator (STE) [1] to approximate the

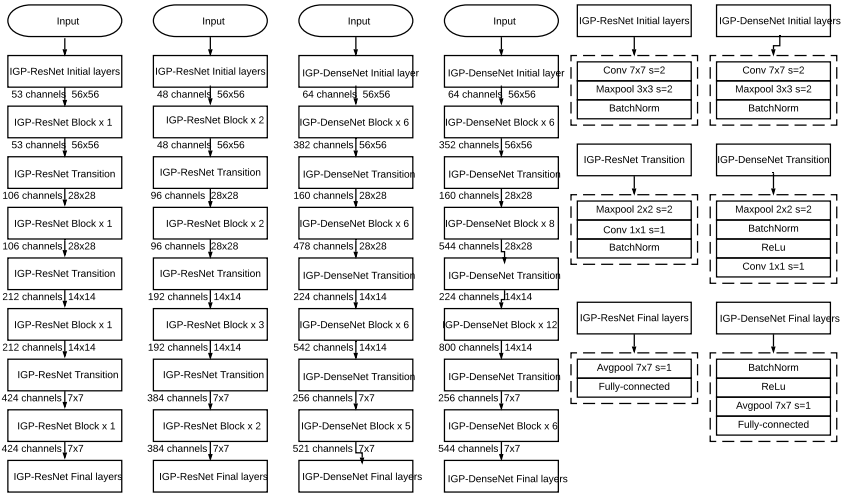

Figure 4: The building blocks and an exemplary network structure of our proposed architectures. **Left two columns:** The IGP-ResNet uses a network architecture configuration of $M = 7$. In an IGP-ResNet block, there are 3 columns and 4 convolutional layers. **Middel two columns:** The IGP-DenseNet uses a network architecture configuration of $M = 3$. In an IGP-DenseNet block, there are 2 columns and 2 convolutional layers.

gradient calculation for sign$(\cdot)$ function.

$$\begin{aligned} &\text{Forward: } W_b = E \times \text{sign}(W) \\ &\text{Backward: } \frac{\partial L}{\partial W} = \frac{\partial L}{\partial W_b} \times \frac{\partial W_b}{\partial W} \approx E \times \frac{\partial L}{\partial W_b} \end{aligned} \tag{1}$$

Binarization of activations. The forward propagation and backward propagation to binarize the activations are calculated as follows. A and A_b represent the full precision activations and binary activations, respectively.

$$\begin{aligned} &\text{Forward: } A_b = \text{sign}(A) \\ &\text{Backward: } \frac{\partial L}{\partial A} = \frac{\partial L}{\partial A_b} \times \frac{\partial A_b}{\partial A} \\ &\text{where} \frac{\partial A_b}{\partial A} = \begin{cases} 2+2A, -1 < A < 0 \\ 2-2A, 0 \leq A < 1 \\ 0, \text{otherwise} \end{cases} \end{aligned} \tag{2}$$

4 Clarifications for our proposed architectures

Taking $M = 3$ as an example, Figure 5 (a) and (b) are candidate architectures of IGP-ResNet21(50) block with a depth of 2. Heterogeneous branches of summation aggregation are helpful to break symmetry and learn different features, and we suggest Figure 5 (b) for IGP-ResNet. To explain our suggestion, we revisit the constant initialization scheme of neural networks. The neurons with constant initialization adopt a weighted sum in the forward

path, and they have identical gradients in the backward path. Throughout the training, neurons evolve symmetrically, which prevents the neurons from learning different features. In Figure 5 (a), $V = C_1(I, W_1) \oplus C_3(I, W_3)$, where W_1 and W_3 are parameters (or weights) of C_1 and C_3, respectively. Here we consider architectural hyperparameters and assume $W_1 = W_3$. Then, two branches of summation aggregation, i.e., C_1 and C_3, have the same architectures in the forward path, and they have identical gradients in the backward path. In other words, C_1 and C_3 use constant initialization for architectural hyperparameters. In Figure 5 (b), each branch of summation aggregation has a unique architecture, which is helpful for branches to break symmetry and learn different features.

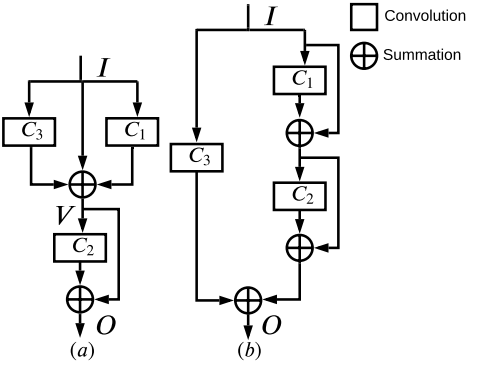

Figure 5: Candidate architectures of IGP-ResNet block.

More importantly, we would like to clarify that our IGP-ResNet and IGP-DenseNet do not cover all the architectures, which can be designed by improving gradient paths. We expect that other advanced architectures will be designed by improving gradient paths for BCNNs in future work.

5 Experimental details

We show all the experimental details in this section, including the data augmentation for CIFAR-100 and ImageNet, the training settings for ResNet on CIFAR-100 and ImageNet, and DenseNet variants on CIFAR-100 and ImageNet.

It is worth clarifying that we design IGP-ResNet and IGP-DenseNet since binary ResNet and DenseNet variants have different strengths. The inference of binary ResNet variants is more efficient (i.e., less computational complexity) than that of binary DenseNet variants. However, the training of binary DenseNet variants is more efficient than that of binary ResNet variants since binary DenseNet variants are trained from scratch while binary ResNet variants need their full-precision pre-trained counterparts as initialization.

Data augmentation for CIFAR-100 classification. CIFAR-100 classification dataset [5] contains 50K training images and 10K test images and consists of 32×32 color images drawn from 100 classes. Images are first zero-padded on each side with four pixels. A random 32×32 patch is cropped from its padded image or its horizontal flip before applying mean/std normalization. During testing, we use only mean/std normalization.

Training setting for ResNet variants on CIFAR-100 classification. For ResNet variants, we use the batch size of 128 and train 200 epochs in total. The learning rate is started at 0.1 and decays by a factor of 0.2 at the step of 60, 120, and 160. Since binarization can be regarded as a regularization, the weight decay is set at 0.0 for BCNNs.

Training setting for DensneNet variants on CIFAR-100 classification. For DenseNet variants, we use the batch size of 128 and train from scratch for 200 epochs with an adam optimizer and a weight decay of 0.0. The learning rate starts at 0.002 and decreases using a cosine annealing schedule until 0.0. We use the method in [2] to initialize the weights. The Relu layer is removed from the "Bn-Relu-Conv" layers.

Data augmentation for ImageNet classification. ImageNet classification dataset [3] consists of 1.2 million images in the training dataset and 50K images in the validation dataset. A 224×224 crop is randomly sampled from an image or its horizontal flip, with the per-pixel mean subtracted. When validating, we perform data augmentation by resizing a shorter edge to 256 and center-cropping to 224×224 pixels, and similarly, normalizing the input images with mean channel subtraction.

Training setting for ResNet variants on ImageNet classification. For ResNet variants, we train a full precision model as an initialization for the BCNNs. During finetuning, the weights and activations are binarized, while the downsampling convolution layer or transition block remains in full-precision in BCNNs. When training the full-precision model, we reorder the layers from the order of "Conv-Bn-Relu" to the order of "Conv-Relu-Bn". Regarding the training settings, we train a full-precision model using a momentum optimizer and a weight decay of $1e-4$. We train 100 epochs in total. The learning rate starts at 0.1 and decays with a factor of 0.1 at the step of 30, 60, and 90. The Tanh function is inserted for the input activations of the convolution. During finetuning, we adopt an adam optimizer and a weight decay of 0.0. We train 50 epochs in total. The learning rate starts at $5e-4$ and decays at the step of 30 and 40. The Tanh function is replaced with the binarization function. We use a batch size of 256.

Training setting for DenseNet variants on ImageNet classification. For DenseNet variants, we train from scratch for 100 epochs with an adam optimizer and a weight decay of 0.0. The learning rate starts at 0.002 and decreases using a cosine annealing schedule until 0.0. We use the method in [4] to initialize the weights. The Relu layer is removed from the "Bn-Relu-Conv" layers.

6 Computational complexity analysis

To guarantee a fair comparison, we scale the number of base channels or the growth rate of our proposed architectures to match the computational complexity of Bi-Real ResNet and BinaryDenseNet baselines. The computational complexity is analyzed in terms of storage in Mbit, computation in Flops, and run-time memory in MB. Given a computation complexity budget, we scale the number of base channels or the growth rate for our proposed architectures with various network architecture configurations to show that our proposed

architectures are robust to architectural hyperparameters of neural networks. Taking IGP-ResNet41(48) and IGP-DenseNet51(53) as examples, 41 and 51 are the depths of our proposed architectures, while 48 and 53 refer to the number of base channels and the growth rate after scaling, respectively.

Storage and computation We adopt the number of parameters as the metric for storage usage, and the number of Flops as the metric for computational efficiency. The number of parameters is measured as the summation of 32bits times the number of floating-point parameters and 1bit times the number of binary parameters in the model. The XNOR and Popcount bitwise operations can be executed by the current CPUs with a parallelism of 64. Therefore, the Flops is calculated by the number of floating-point multiplications plus $1/64$ of the number of binary multiplication.

Run-time memory consumption To estimate the run-time memory requirement, we calculate the size of weights and activations for a layer or an operation, plus all the activations of shortcuts that cross past that layer or operation. The type of that operation can be either a summation or a concatenation. We report the largest among such values for each model, which would serve as a lower bound for the run-time memory disregarding the layer or operation schedule even though the actual usage would largely depend on individual hardware and framework implementation. The reported run-time memory is estimated with a batch size of 64.

Computational complexity of shortcuts and summations There are more shortcuts and summations in our proposed architectures than in Bi-Real ResNet and BinaryDenseNet. Thus, the shortcuts and summations in our proposed architectures need more run-time memory and more computation than those in Bi-Real ResNet and BinaryDenseNet. However, the binary convolutional layers in our proposed architectures consume less run-time memory and less computation than those in Bi-Real ResNet and BinaryDenseNet. To ensure a fair comparison, we scale the number of base channels or the growth rate of our proposed architectures to match the computational complexity of Bi-Real ResNet and BinaryDenseNet. For example, we build IGP-ResNet21 with a scaled number of base channels equal to 53 to match the computational complexity of Bi-Real ResNet18 with the number of base channels equal to 64. Thus, our proposed architectures require almost the same run-time memory and computation as Bi-Real ResNet and BinaryDenseNet.

7 Experimental results on CIFAR-100

ResNet variants on CIFAR-100 As shown in Table 2, we present the error of our proposed architectures for binarizing ResNet18 and ResNet34. IGP-ResNet variants with various network architecture configurations consistently outperform Bi-Real ResNet baselines. Compared with Bi-Real ResNet18(64) and Bi-Real ResNet34(64), the Top-1 error of our IGP-ResNet21(50) and IGP-ResNet41(45) are improved by 2.14% and 2.57%, respectively. Considering the computational complexity, our IGP-ResNet21(50) use 0.11Mbit less for storage, 0.14×10^7 Flops less for computation, and 2.10MB more for run-time memory than Bi-Real ResNet18(64). Our IGP-ResNet41(45) saves the storage by 0.64Mbit, the computation by 0.33×10^7 Flops, and the run-time memory by 3.14MB compared with Bi-Real

Model	Top-1	Top-5	Storage	Computation	Run-time memory
Bi-Real ResNet18(64)	28.48%	8.65%	18.18Mbit	1.67×10^7Flops	50.33MB
IGP-ResNet21(50)	**26.34%**	**7.89%**	18.07Mbit	1.53×10^7Flops	52.43MB
IGP-ResNet37(36)	26.67%	7.51%	17.57Mbit	1.42×10^7Flops	47.19MB
IGP-ResNet69(26)	26.85%	7.57%	17.63Mbit	1.38×10^7Flops	40.89MB
Bi-Real ResNet34(64)	27.93%	8.37%	28.28Mbit	2.61×10^7Flops	50.33MB
IGP-ResNet41(45)	**25.36%**	**7.26%**	27.64Mbit	2.28×10^7Flops	47.19MB
IGP-ResNet77(32)	25.57%	6.86%	27.94Mbit	2.24×10^7Flops	41.94MB
IGP-ResNet149(22)	26.38%	7.83%	26.35Mbit	2.08×10^7Flops	34.60MB
BinaryDenseNet51(32)	27.16%	7.77%	17.65Mbit	5.13×10^7Flops	117.44MB
IGP-DenseNet51(48)	**26.72%**	**7.51%**	17.51Mbit	5.32×10^7Flops	92.27MB
BinaryDenseNet69(32)	26.88%	7.52%	23.70Mbit	5.50×10^7Flops	117.44MB
IGP-DenseNet69(44)	**26.38%**	**7.32%**	23.33Mbit	5.67×10^7Flops	85.98MB

Table 2: Binary ResNet and DenseNet variants on CIFAR-100. There are four blocks in this Table. **First block:** ResNet18(64) and IGP-ResNet variants to compare with ResNet18(64). **Second block:** ResNet34(64) and IGP-ResNet variants to compare with ResNet34(64). **Third block:** BinaryDenseNet51(32) and IGP-DenseNet variants to compare with BinaryDenseNet51(32). **Fourth block:** BinaryDenseNet69(32) and IGP-DenseNet variants to compare with BinaryDenseNet69(32).

ResNet34(64), respectively.

DenseNet variants on CIFAR-100 As shown in Table 2, we present the error of our proposed architectures for binary DenseNet51(32) and DenseNet69(32). The Top-1 error of our proposed IGP-DenseNet51(48) and IGP-DenseNet69(44) are 0.44% and 0.50% lower than those of BinaryDenseNet51(32) and BinaryDenseNet69(32), respectively. The increased number of Flops required for our proposed IGP-DenseNet51(48) and IGP-DenseNet69(44) is 0.19×10^7 and 0.17×10^7, respectively, while the decreased number of parameters required for them is 0.06Mbit and 0.37Mbit, respectively, and the decreased run-time memory size needed for them is 25.17MB and 31.46MB, respectively, compared with BinaryDenseNet51(32) and BinaryDenseNet69(32).

8 Ablation study

We have shown that our proposed architectures have better gradient paths and achieve lower error than Bi-Real ResNet and BinaryDenseNet. In this section, we verify the essential role of the gradient path that requires the smallest number of operations to compute gradient backpropagation. The architectures of GP-ResNet are obtained by removing all the residual connections from IGP-ResNet. Thus, $N_{1,st}$ is 0 for IGP-ResNet and N^{IR} for GP-ResNet block. As shown in Table 3, the Top-1 error of GP-ResNet21(50) and GP-ResNet41(45) is 5.48% and 14.78% higher compared to their IGP-ResNet counterparts.

References

[1] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint*

Model	Top-1	Top-5
IGP-ResNet21(50)	26.34%	7.89%
GP-ResNet21(50)	31.82%	9.70%
IGP-ResNet41(45)	25.36%	7.26%
GP-ResNet41(45)	40.14%	15.19%

Table 3: Results of ResNet variants on CIFAR-100.

arXiv:1308.3432, 2013.

[2] Joseph Bethge, Haojin Yang, Marvin Bornstein, and Christoph Meinel. Binarydensenet: Developing an architecture for binary neural networks. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

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

[6] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European conference on computer vision (ECCV)*, pages 722–737, 2018.

[7] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.