Exploration of Linear Bottleneck Architecture for RRAM Fault-tolerant Deep Learning

Authored by: Mengzhan Liufu

Mentored by: Yanjing Li, Mingyuan Xiang

Date: March 11, 2024

Abstract

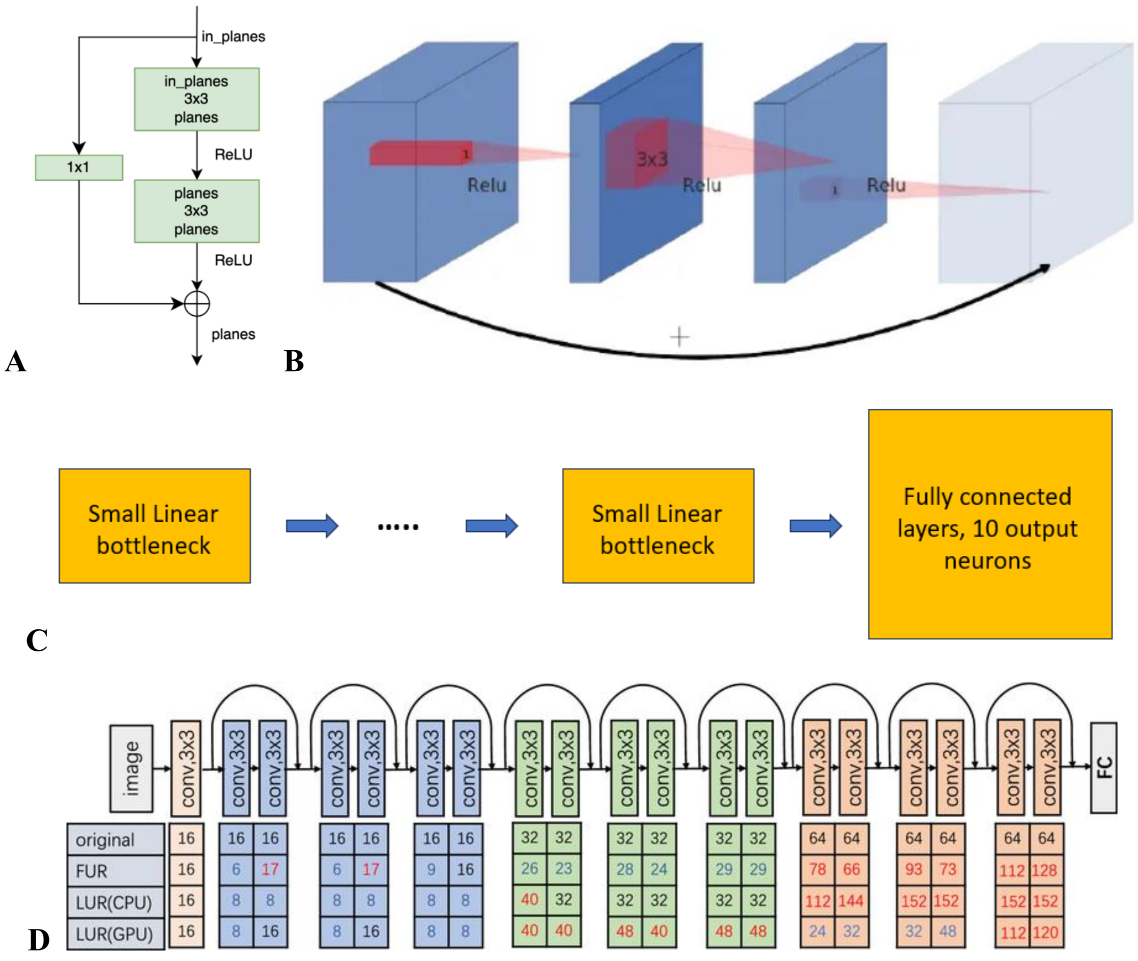
Resistive random-access memory (RRAM) has emerged as an efficient hardware platform for deep learning applications, but its currently high defect rate limits its practical applications. Previous work established Drop-Connect as an efficient model training method to make models tolerant to random faults, and it was hinted that the linear bottleneck architecture made MobileNetV2 more robust to random faults compared to other models. In this study, we investigated this hypothetical fault-tolerance property of linear bottleneck with two experiments. First, we built mini mobilenets out of a small number of linear bottlenecks and quantified their performance on MNIST digit classification task under various weight drop rate. Then, we modified ResNet20 by replacing parts of the model with linear bottlenecks and quantified the performance of the modified model on CIFAR10 under random weight drops. We concluded that linear bottleneck doesn't have significantly advantageous fault tolerance compared to other architectures. Further investigation into the architectural nuances of MobileNetV2 is needed to understand its robustness, and other machine learning and system techniques may be applied to develop DNN applications with reliable RRAM fault tolerance.

Introduction

Deep learning with deep neural networks has shown unparalleled performance in various scenarios such as computer vision and natural language processing. As the size of deep neural networks grows, specialized hardware platforms for the efficient training and deployment of deep neural networks have emerged. Among all that have been proposed, resistive random-access memory (RRAM) is a promising and innovative compute-in-memory solution [1], but the hardware defects of current RRAM devices limit their practical use [2-4].

To train deep neural networks that are robust to any given RRAM fault distribution, Xiang et.al [5] applied a machine learning technique called Drop-Connect. By randomly setting a subset of network weights to zero during model training, they significantly reduced the amount of performance degradation when models were applied on visual classification tasks under random faults [5]. Although not highlighted in the study, it's noticeable that across all models MobileNetV2 [6] was able to recover the most performance with Drop-Connect (Figure 1, left panel).

We hypothesize that the linear bottleneck architecture [6] of MobileNetV2 works best with Drop-Connect compared to other model architectures. Should this be true, future deep learning applications on faulty RRAM can integrate linear bottleneck into their model design, followed by training with Drop-Connect to further preserve performance. To test this hypothesis, we first experimented with several small network models (referred to as "mini mobilenets"), featuring different numbers of linear bottlenecks (Figure 1C). Each model was trained using the Drop-Connect method, then their inference-time performance against random weight drops was quantified with MNIST dataset [8]. In the second stage, we progressively replaced the convolutions in ResNet20 [7] with linear bottlenecks and trained model with Drop-Connect. Performance of the trained models was quantified with CIFAR10 dataset [9] and compared to original ResNet20 as the baseline.



original	16	16	16	16	16	16	16	32	32	32	32	32	32	64	64	64	64	64	64
FUR	16	6	17	6	17	9	16	26	23	28	24	29	29	78	66	93	73	112	128
LUR(CPU)	16	8	8	8	8	8	8	40	32	32	32	32	32	112	144	152	152	152	152
LUR(GPU)	16	8	16	8	16	8	8	40	40	48	40	48	48	24	32	32	48	112	120

Figure 1. Diagrams for architectures used in experiments. **(A)** A Basic Block of ResNet consists of two 3x3 convolutions and a shortcut connection. **(B)** A linear bottleneck of MobileNetV2 has a 3x3 depthwise convolution, a 1x1 pointwise convolution and a shortcut connection (inverted residual). **(C)** Mini mobilenets are small models with varying number of linear bottlenecks

connected to a fully connected layer for MNIST digit classification. **(D)** ResNet20 has nine Basic Blocks (18 3x3 convolutions), and each convolution could technically be replaced by a linear bottleneck.

Methods

MobileNetV2 and Linear Bottleneck

MobileNetV2 is a mobile network designed for visual tasks such as object detection and image classification [6]. A major contribution of this work is a lightweight alternative method of convolution, termed linear bottleneck. Specifically, a linear bottleneck includes three processing steps (Figure 1B). First, pointwise convolution with 1x1 kernels is applied to the input image to expand the number of input dimensions and facilitate feature extraction. The second step is separable convolution, which contains a depth-wise convolution with 3x3 kernels and a pointwise convolution with 1x1 kernels. At the cost of minor expressivity loss, a separable convolution significantly reduces computational complexity and is desirable for mobile applications. Finally, much like the residual operation proposed by [7], the original input is added on top of the output from separable convolution. This operation is termed inverted residual. While MobileNetV2 has 7 linear bottlenecks and 3 additional convolution blocks, we experimented with much smaller mini mobilenets with fewer linear bottlenecks and modified versions of ResNet20 where some convolutions were replaced by linear bottlenecks.

ResNet20 and Basic Block

Proposed by He et.al [7], ResNet is a residual learning framework that solved the training threshold issue. It involves the shortcut connection mechanism, which adds the original input to the output of convolution layers and tackles vanishing gradient issues. The basic architectural unit of ResNet is called Basic Block, which includes two 3x3 convolutions and a shortcut connection (Figure 1A). ResNet20 has nine Basic Blocks and in total 18 convolutions, and each one could technically be replaced by a linear bottleneck (Figure 1D).

Drop-Connect training and Fault-Prone RRAM Simulation

In each iteration of Drop-Connect training, a randomly determined subset of model weights are masked with 0, and the subset makes up a given percentage of total number of model weights ("drop-connect rate"). A similar method is used to evaluate a model's inference-time performance on fault-prone RRAM, but there are two important differences. First, while in Drop-Connect training a new weight mask is generated in every iteration, only one weight mask is generated for simulated inference on fault-prone RRAM. This mask is commonly applied to all forward passes. Second, the percentage of weight drops during inference time is determined by fault rate, which is a separate parameter than drop-connect rate.

Results

Experiment 1: Mini Mobilenets and MNIST

Before testing the mini mobilenets' reaction to weight drops, we first confirmed that small mobilenets with only a few linear bottlenecks could perform well on the MNIST digit classification task. Without weight drop during training and validation, a small mobilenet with two linear bottlenecks was able to reach 98% validation accuracy after 3 epochs of training (Figure 2A).

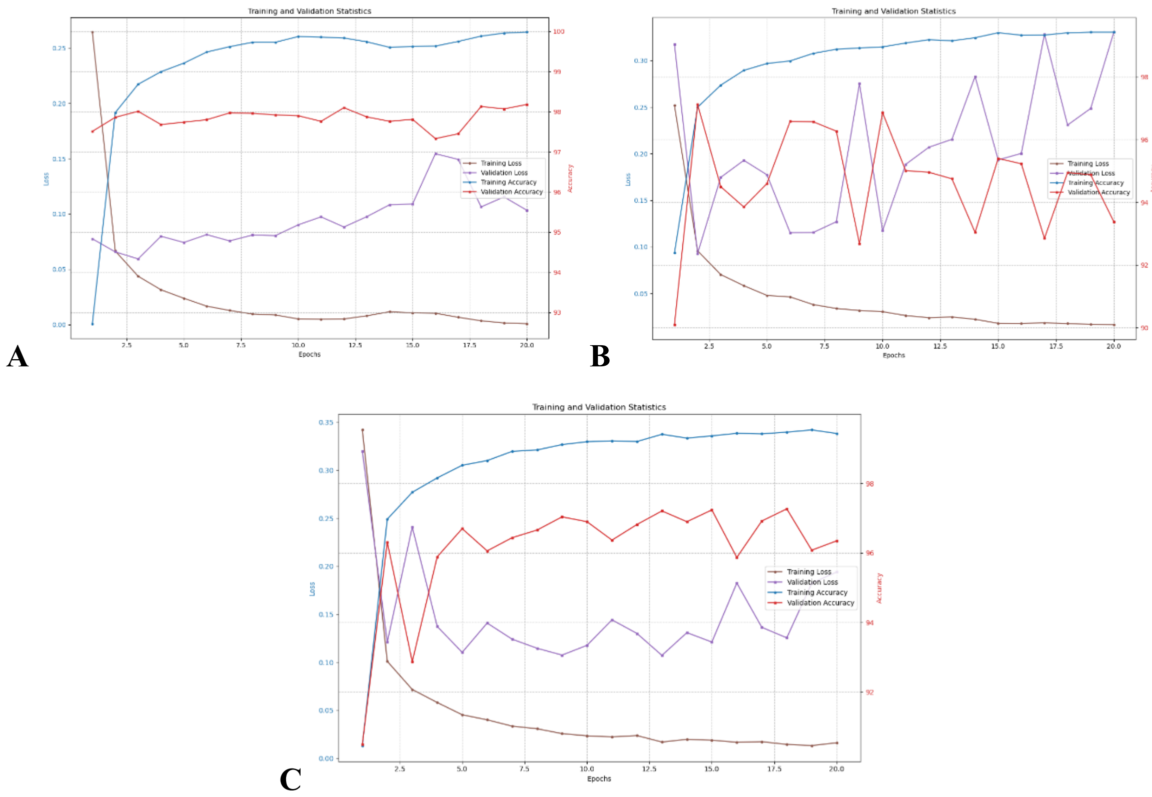


Figure 2. Mini mobilenet performance on MNIST under 0% and 10% drop rate. **(A)** 2-bottleneck mobilnet reached 98% performance when no weight was dropped. **(B)** Performance of 2-bottleneck mobilenet fluctuated significantly around 94% when 10% of weights were dropped. **(C)** The magnitude of fluctuation reduced when the number of linear bottlenecks was increased to 4 (4-bottleneck mobilenet).

Having the model validated, we introduced weight drops progressively. Specifically, we trained mini mobilenets with varying number of linear bottlenecks under 10%, 20% and 30% of drop-connect rate and fault rate. While the optimal drop-connect rate for a given fault-prone RRAM device might not be equal to its fault rate [5], we keep them consistent in both experiments for simplicity and refer to them jointly as drop rate.

When 10% drop rate was applied to the two bottleneck mobilenet we validated, its validation accuracy stayed around 94% but displayed noticeable fluctuations as we trained for more epochs (Figure 2B). Although the average accuracy remained decent, the fluctuations indicate instability caused by random weight drops. Such instability would also manifest in the form of inconsistent performance across different fault-prone RRAM devices, each of which has a unique fault distribution. To compensate for the instability, we made the mini mobilenet deeper by introducing two more linear bottleneck. Validation accuracy of the 4-bottleneck mobilenet started stabilizing to roughly 97% after 5 training epochs (Figure 2C).

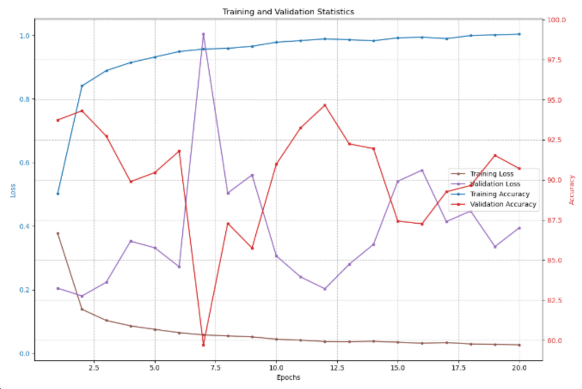
Similar situation was observed when drop rate was increased to 20%. Validation accuracy of the 2-bottleneck mobilenet degraded to an average of 87%, with a high of 95% and a low of 80% (Figure 3A). No trend of improvement was observed with further training up to 20 training epochs. We again increased the number of linear bottlenecks in the model to determine the minimum number of bottlenecks required to account for 20% weight being dropped. Although the 6-bottleneck model reached relatively stable performance around 96% (Figure 3B), performance degraded when the number of bottlenecks was increased to 7 (Figure 3C). This indicates that larger number of linear bottlenecks doesn’t necessarily translate to better and more stable performance under random weight drops.

When drop rate was increased to 30%, addition of extra linear bottlenecks (up to six in total) was not able to effectively recover performance (Supplementary Figure 1). In addition to the lack of improvement with continued training, certain mini mobilenets showed either a declining trend (Supplementary Figure 1B) or catastrophic dip (Supplementary Figure 1A, 1C) in performance across training epochs.

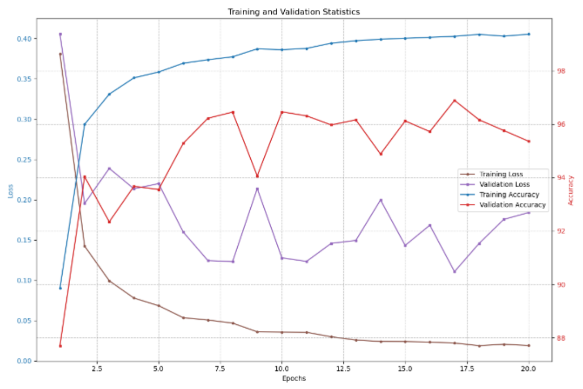
Experiment 2: Modified ResNet20 and CIFAR10

The results of experiment 1 with mini mobilenets complicated the question at hand. We originally aimed to determine the minimum number of linear bottlenecks needed to account for a certain percentage of random weight drops, but this question involves two implicit assumptions. We assumed that a deeper mini mobilenet (i.e larger number of linear bottlenecks) should have better and more stable performance against random weight drops, and that with any given model and drop rate performance should increase with more training epochs. Neither of these assumptions held, which made it challenging to answer the original question.

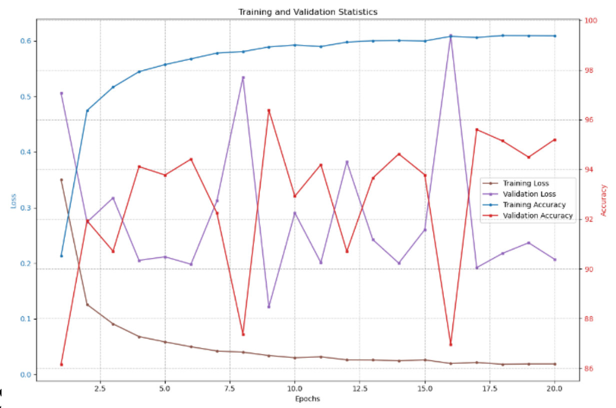
To potentially bypass those issues, we changed our strategy to incrementally replacing parts of a well-established baseline model (ResNet20, Figure 1D) with faulty linear bottlenecks where random weights were dropped. No weights in the unreplaced 3x3 convolution blocks were dropped, and the first two convolutions were kept unchanged because they are commonly believed to be important for feature extraction. This experiment emulates outsourcing fault-



A



B



C

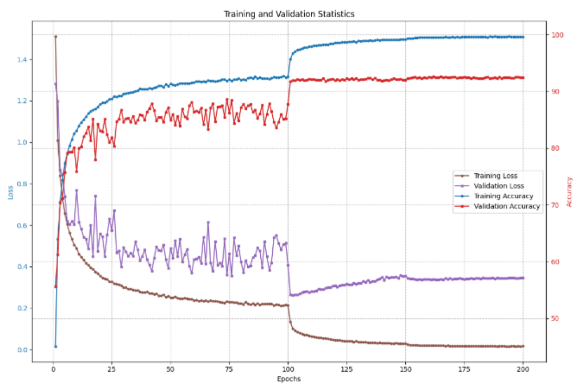
Figure 3. Mini mobilenet performance on MNIST under 20% drop rate. **(A)** Similar to its reaction to 10% drop rate, the 2-bottleneck mobilenet showed significantly degraded and unstable performance when 20% of weights were dropped. **(B)** Performance was recovered when the number of linear bottlenecks was increased to 6 (6-bottleneck model). **(C)** Performance degraded again when 7 bottlenecks were used (7-bottleneck model).

tolerant model components to fault-prone RRAM accelerators, and the goal is to determine the maximum number of convolutions that can be replaced by linear bottlenecks and outsourced to RRAM for maximum acceleration benefits without compromising model performance. To gain better knowledge of how well this method applies to real-life workloads, we used CIFAR10 which is a more challenging visual classification task dataset than MNIST.

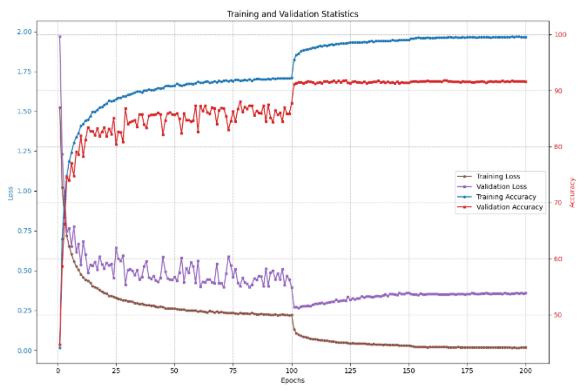
We first established the original ResNet20 as the baseline model for comparison, and it was able to reach 93% accuracy after 100 training epochs (Figure 4A), which was consistent with the common understanding. We then tested if a modified ResNet20 with linear bottleneck(s) was able to perform well on CIFAR10 without weight drops. Without rigorous testing over all possible configurations, we replaced one convolution with linear bottleneck and observed no loss of performance (Figure 4B). Alongside theoretical backing of separable convolution, this shows that the linear bottleneck structure is capable of performing what the original convolution does.

With drop rate = 20%, decent performance was maintained up to six convolutions being replaced. While the original ResNet20 and the modified ones with fewer than six convolutions replaced, validation accuracy curve stays almost perfectly stable from 100 to 200 training epochs. On the other hand, the 6-bottleneck ResNet20 displayed fluctuating accuracy around 90% (Figure 4C), reflecting the uncertainty introduced by random weight drops in the linear bottlenecks. In the extreme case where all convolutions were replaced except for the first one in each layer, validation accuracy significantly fluctuates between 50% to 70% before 100 training epochs and relatively stabilized to 80% from 100 to 200 training epochs (Figure 4D). Although no reliable performance can be claimed with the presence of such fluctuations, we could still observe the upper sloping trend in accuracy with continued training (Figure 4D). This suggests that the model was able to improve with training but likely needed more parameters to account for random weight drops.

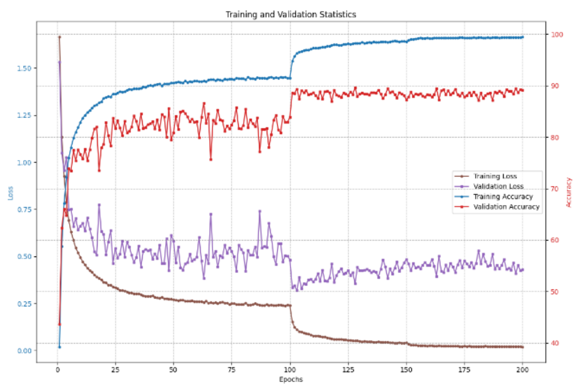
Model performance under 30% drop rate aligns with the interpretation. With a higher drop rate, significant fluctuation in validation accuracy emerged with as few as three convolution replacements (Figure 4E). When six convolutions were replaced by faulty linear bottlenecks (30% drop rate), the level of fluctuation and final accuracy matched the fully replaced model with 20% drop rate (Figure 4E, 4F).



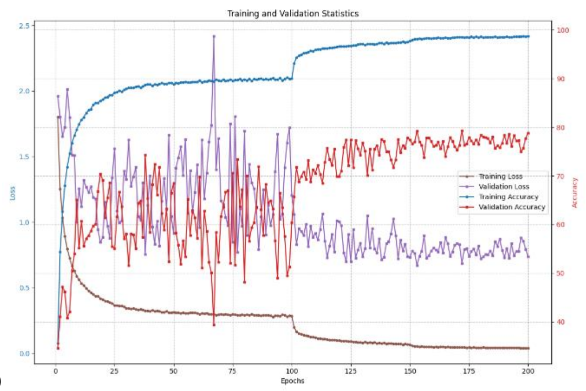
A



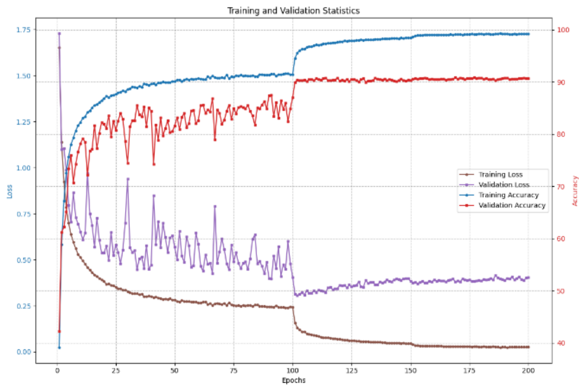
B



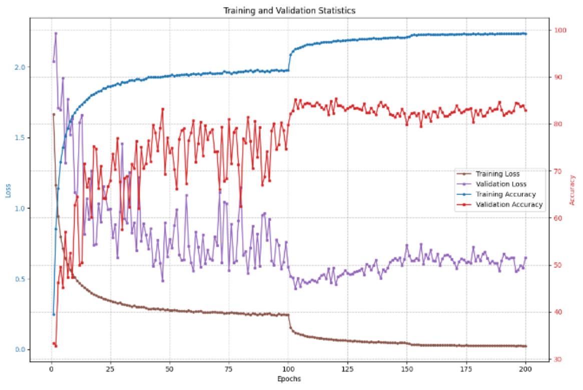
C



D



E



F

Figure 4. Performance of modified ResNet20 with 20% and 30% drop rate for the linear bottlenecks. **(A)** Performance of the original ResNet20. **(B)** One convolution replaced (10th), no weight drop. **(C)** Six convolutions replaced (4th, 6th, 10th, 12nd ,16th, 18th) with bottlenecks of 20% drop rate **(D)** All convolutions replaced except for the first, 7th, and 13th. **(E)** Three convolutions (4th 10th 16th) replaced with bottlenecks of 30% drop rate. **(F)** Six convolutions replaced (4th, 6th, 10th, 12nd ,16th, 18th) with bottlenecks of 30% drop rate.

Discussion

Our second experiment suggests that a fully modified ResNet20 is able to improve its performance against random weight drops, but greater parameter overhead is needed to account for 20% to 30% of random weight drops. Despite the possibility of rescuing accuracy with larger overhead, such approach would defeat the purpose of using linear bottlenecks at all. The original ResNet20 has near 80% accuracy with 20% drop rate in all convolutions, which is comparable to that of the fully replaced ResNet20 (Figure 4D). Because of the expansion step (Figure 1B), a linear bottleneck has more parameters than a normal convolution with the same input and output dimension. That said, a ResNet20 with most parts replaced by linear bottlenecks has significant overhead without improving tolerance against random RRAM faults. This motivates more nuanced investigation of the fault tolerance property of MobileNetV2, if the linear bottleneck architecture doesn't explain it.

It's also very likely that the effect of replacing a particular convolution depends on its position in the model: while some layers may have more redundancy or play a less critical role in representation extraction, others may be very precise and important to the final classification accuracy. In that light, the first type of convolutions should be identified and outsourced to faulty RRAM accelerators, and the second type should be kept on reliable computing components for precise execution. Since such information is probably unique to individual model and task, architecture search techniques should be used to efficiently derive model-specific replacement scheme.

With Drop-Connect, Xiang et.al hoped to bypass the need to characterize fault distribution for each RRAM device by making models tolerant to any given fault distribution, but our experiments suggested that significant amount of resource overhead is required, if the goal is possible at all. Instead of completely ignoring information about individual RRAM devices, it might be favorable to develop efficient characterization of RRAM fault distribution and dynamically customizable models. Hypernetwork and dynamic machine learning architectures may be adapted for such purpose.

Conclusion

In this study, we tested the hypothesis that the linear bottleneck structure from MobileNetV2 has above average fault tolerance that renders it favorable for RRAM-based deep learning applications. Via experiments with mini mobilenets built of a few linear bottlenecks and modified ResNet20-s that incorporated varying number of linear bottlenecks, we were able to better understand the fault tolerance property of the linear bottleneck structure. Our main conclusion is that a linear bottleneck trained with Drop-Connect doesn't have higher built-in fault tolerance than other model architectures such as Basic Block. The robust performance of MobileNetV2 probably lies in nuances such as model hyperparameters. Other machine learning techniques are needed to understand and discover fault-tolerant DNN architectures or make existing ones fault-tolerant.

Code Availability: Model implementations and training code for our experiments can be found at https://github.com/mingyuan-xiang/dropnet/tree/main

References

[1] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in 43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016. IEEE Computer Society, 2016, pp. 14–26

[2] Y. Huang, Y. He, J. Wang, J. Yue, L. Zhang, K. Zou, H. Yang, and Y. Liu, "Bit-aware fault-tolerant hybrid retraining and remapping schemes for rram-based computing-in-memory systems," IEEE Trans. Circuits Syst. II Express Briefs, vol. 69, no. 7, pp. 3144–3148, 2022.

[3] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang, "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar," in Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017, D. Atienza and G. D. Natale, Eds. IEEE, 2017, pp. 19– 24.

[4] C. Liu, M. Hu, J. P. Strachan, and H. H. Li, "Rescuing memristor-based neuromorphic design with high defects," in Proceedings of the 54th Annual Design Automation Conference, DAC 2017, Austin, TX, USA, June 18-22, 2017. ACM, 2017, pp. 87:1–87:6.

[5] Mingyuan Xiang, Xuhan Xie, Pedro Savarese, Xin Yuan, Michael Maire, and **Yanjing Li**, "Drop-Connect as a Fault-Tolerance Approach for RRAM-based Deep Neural Network Accelerators," *Proc. IEEE VLSI Test Symposium (VTS)*, 2024

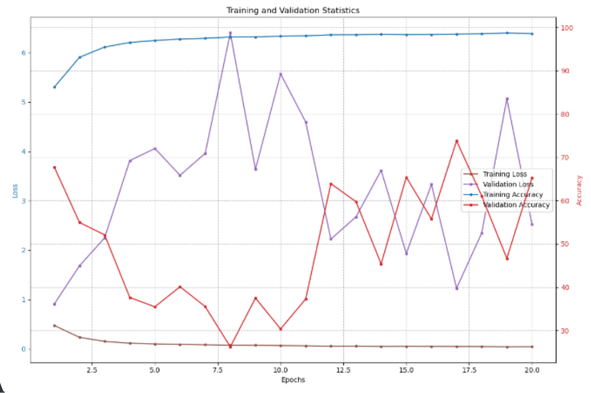
[6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 2018 pp. 4510-4520.

[7] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

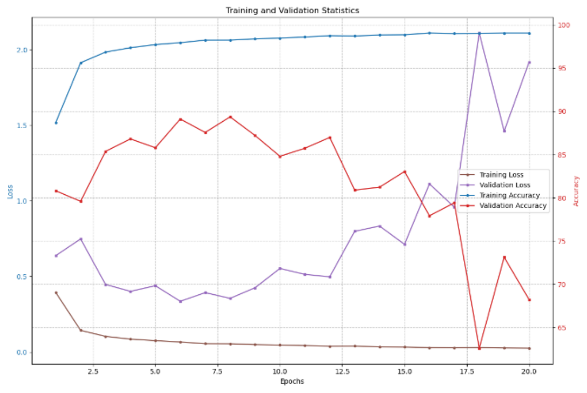
[8] Deng L. The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine. 2012;29(6):141–2.

[9] Krizhevsky, A. & Hinton, G, "Learning multiple layers of features from tiny images", Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

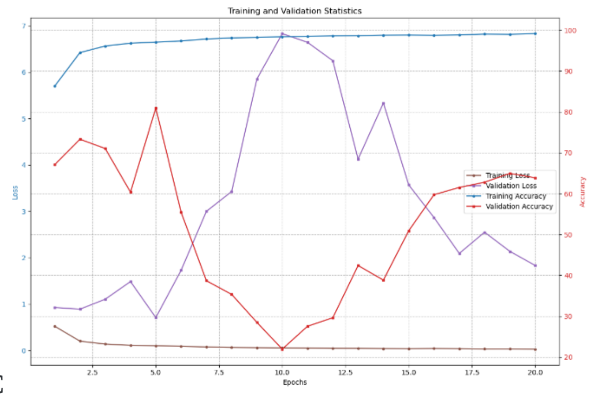
Supplementary Figures



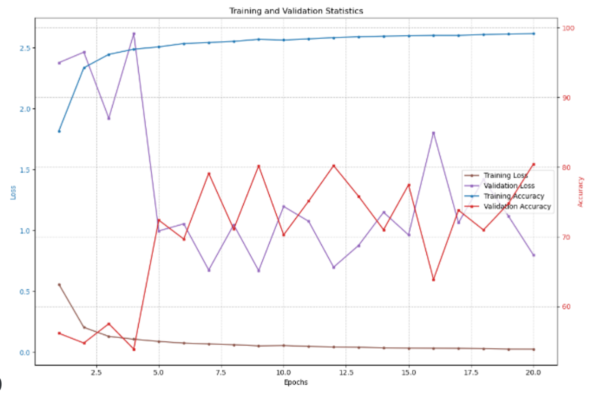
A



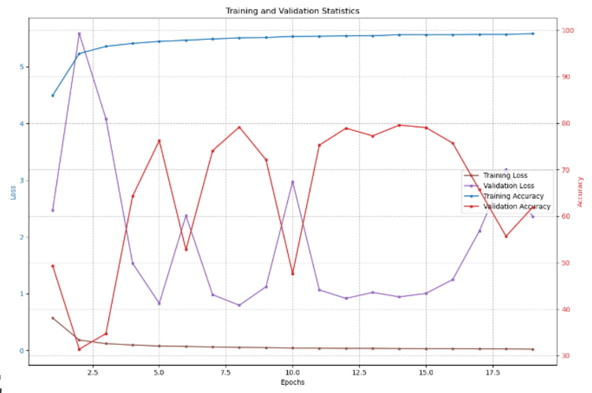
B



C



D



E

Supplementary Figure 1. Under 30% drop rate, no mini mobilenet up to 6 linear bottlenecks (6-bottleneck model) reached stable performance beyond 80% accuracy. **(A)** Performance of 2-bottleneck model against 30% drop rate. **(B)** Performance of 3-bottleneck model against 30% drop rate. **(C)** Performance of 4-bottleneck model against 30% drop rate. **(D)** Performance of 5-bottleneck model against 30% drop rate. **(E)** Performance of 6-bottleneck model against 30% drop rate.