DIVISION: Memory Efficient Training via Dual Activation Precision

Guanchu Wang[1] **Zirui Liu**[1] **Zhimeng Jiang**[2]
Ninghao Liu[3] **Na Zou**[4] **Xia Hu**[1]

Abstract

Activation compressed training provides a solution towards reducing the memory cost of training deep neural networks (DNNs). However, state-of-the-art work combines a search of quantization bit-width with the training, which makes the procedure complicated and less transparent. To this end, we propose a simple and effective method to compress DNN training. Our method is motivated by an instructive observation: *DNN backward propagation mainly utilizes the low-frequency component (LFC) of the activation maps, while the majority of memory is for caching the high-frequency component (HFC) during the training*. This indicates the HFC of activation maps is highly redundant and compressible, which inspires our proposed <u>D</u>ual Act<u>IV</u>ation Prec<u>ISION</u> (DIVISION). During the training, DIVISION preserves a high-precision copy of LFC and compresses the HFC into a light-weight copy with low numerical precision. This can significantly reduce the memory cost while maintaining a competitive model accuracy. Experiment results show DIVISION has better comprehensive performance than state-of-the-art methods, including over 10× compression of activation maps and competitive training throughput, without loss of model accuracy. The source code is available at `https://github.com/guanchuwang/division`.

1. Introduction

Deep neural networks (DNNs) have been widely applied to real-world tasks such as language understanding (Devlin et al., 2018), machine translation (Vaswani et al., 2017), visual detection and tracking (Redmon et al., 2016). With increasingly larger and deeper architectures, DNNs achieve remarkable improvement in representation learning and generalization capacity (Krizhevsky et al., 2012). Nevertheless, training a larger model requires more memory resources to cache the activation values of all intermediate layers during the backward propagation[1]. For example, training a DenseNet-121 (Huang et al., 2017) on the ImageNet dataset (Deng et al., 2009) requires to cache over 1.3 billion float activation values (4.8GB) during backward propagation; and training a ResNet-50 (He et al., 2016) requires to cache over 4.6 billion float activation values (17GB). Some techniques have been developed to reduce the training cache of DNNs, such as checkpointing (Chen et al., 2016; Gruslys et al., 2016), mix precision training (Vanholder, 2016), low bit-width training (Lin et al., 2017; Chen et al., 2020) and activation compressed training (Georgiadis, 2019; Evans & Aamodt, 2021). Among these, the activation compressed training (ACT) has emerged as a promising method due to its significant reduction of training memory and the competitive learning performance (Liu et al., 2021b).

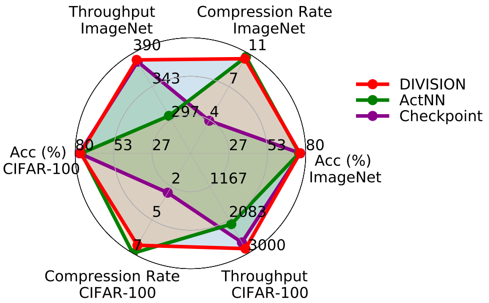


Figure 1: Performance of DIVISION vs baseline methods.

Existing work of ACT relies on quantizing the activation maps to reduce the memory consumption of DNN training, such as BLPA (Chakrabarti & Moseley, 2019), TinyScript (Fu et al., 2020) and ActNN (Chen et al., 2021). Although ACT could significantly reduce the training memory cost, the quantization process introduces noises in backward propagation, which makes the training suffer from undesirable degradation of accuracy (Fu et al., 2020). Due to this reason, BLPA requires 4-bit ACT to

[1] Department of Computer Science, Rice University [2] Department of Computer Science and Engineering, Texas A&M University [3] Department of Computer Science, University of Georgia [4] Department of Engineering Technology, Texas A&M University. Correspondence to: Xia Hu <xia.hu@rice.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[1] The activation map of each layer is required for estimating the gradient during backward propagation.

ensure the convergence to optimal solution on the ImageNet dataset, which has only a $6\times$ compression rate[2] of activation maps (Chakrabarti & Moseley, 2019). Other works propose to search for optimal bit-width to match different samples during training, such as ActNN (Chen et al., 2021) and AC-GC (Evans & Aamodt, 2021). Although they can moderately reduce the quantization noise and achieves optimal solution under 2-bit ACT (nearly $10\times$ compression rate), the following issues cannot be ignored. First, it is time-consuming to search for the optimal bit-width during training. Second, the framework of bit-width searching is complicated and non-transparent, which brings new challenges to follow-up studies on the ACT and its applicability.

In this work, we propose a simple and transparent method to reduce the memory cost of DNN training. Our method is motivated by an instructive observation: *DNN backward propagation mainly utilizes the low-frequency component (LFC) of the activation maps, while the majority of memory is for the storage of high-frequency component (HFC).* This indicates the HFC of activation map is highly redundant and compressible during the training. Following this direction, we propose Dual Activation Precision (DIVISION), which preserves the high-precision copy of LFC and compresses the HFC into a light-weight copy with low numerical precision during the training. In this way, DIVISION can significantly reduce the memory cost. Meanwhile, it will not negatively affect the quality of backward propagation and could maintain competitive model accuracy.

Compared with existing work that integrates searching into learning (Chen et al., 2021), DIVISION has a more simplified compressor and decompressor, speeding up the procedure of ACT. More importantly, it reveals the compressible (HFC) and non-compressible factors (LFC) during DNN training, improving the transparency of ACT. Figure 1 gives the comprehensive performance of DIVISION compared with state-of-the-art methods, which demonstrates the competitiveness of DIVISION in terms of the model accuracy, compression rate, and training throughput. The contributions of this work are summarized as follows:

- We experimentally and theoretically prove DNN backward propagation mainly utilizes the LFC of the activation maps. The HFC is highly redundant and compressible.
- We propose a simple and effective framework called DIVISION to reduce the memory cost of DNN training via removing the redundancy in the HFC of activation maps.
- Experiments on three benchmark datasets demonstrate the effectiveness of DIVISION in terms of memory cost, model accuracy, and training throughput.

2. Preliminary

2.1. Notations

Without loss of generality, we consider an L-layer deep neural network in this work. During the forward pass, for each layer l ($1 \leq l \leq L$), the activation map is given by

$$\mathbf{H}_l = \mathrm{forward}(\mathbf{H}_{l-1}; \mathbf{W}_l), \tag{1}$$

where $\mathbf{H}_l$ denotes the activation map of layer l; $\mathbf{H}_0$ takes a mini-batch of input images; $\mathbf{W}_l$ denotes the weight of layer l; and $\mathrm{forward}(\cdot)$ denotes a feed-forward operation. During the backward pass, the gradients of the loss value towards the activation maps and weights are be estimated by

$$[\hat{\nabla}_{\mathbf{H}_{l-1}}, \hat{\nabla}_{\mathbf{W}_l}] = \mathrm{backward}(\hat{\nabla}_{\mathbf{H}_l}, \mathbf{H}_{l-1}, \mathbf{W}_l), \tag{2}$$

where $\hat{\nabla}_{\mathbf{H}_{l-1}}$ and $\hat{\nabla}_{\mathbf{H}_l}$ denote the gradient towards the activation map of layer l–1 and l, respectively; $\hat{\nabla}_{\mathbf{W}_l}$ denotes the gradient towards the weight of layer l; and $\mathrm{backward}(\cdot)$[3] denotes the backward function which takes $\hat{\nabla}_{\mathbf{H}_l}$, $\mathbf{H}_{l-1}$ and $\mathbf{W}_l$, and outputs the gradients $\hat{\nabla}_{\mathbf{H}_{l-1}}$ and $\hat{\nabla}_{\mathbf{W}_l}$. Equation (2) indicates it is required to cache the activation maps $\mathbf{H}_0, \cdots, \mathbf{H}_{L-1}$ after the feed-forward operations for gradient estimation during backward propagation.

2.2. Activation Compressed Training

It has been proved in existing work (Chen et al., 2020) that majority of memory (nearly 90%) is for caching activation maps during the training of DNNs. Following this direction, the activation compressed training (ACT) reduces the memory cost via real-time compressing the activation maps during the training. A typical ACT framework in existing work (Chakrabarti & Moseley, 2019) is shown in Figure 2. Specifically, after the feed-forward operation of each layer l, activation map $\mathbf{H}_{l-1}$ is compressed into a representation for caching. The compression enables a significant reduction of memory compared with caching the original (exact) activation maps. During the backward pass of layer l, ACT decompresses the cached representation into $\hat{\mathbf{H}}_{l-1}$, and estimates the gradient by taking the reconstructed $\hat{\mathbf{H}}_{l-1}$ into Equation (2): $[\hat{\nabla}_{\mathbf{H}_{l-1}}, \hat{\nabla}_{\mathbf{W}_l}] = \mathrm{backward}(\hat{\nabla}_{\mathbf{H}_l}, \hat{\mathbf{H}}_{l-1}, \mathbf{W}_l)$. Even though the pipeline of compression and decompression is lossy, i.e. $\hat{\mathbf{H}}_l \neq \mathbf{H}_l$ for $1 \leq l \leq L$. It has been proved ACT can limit the reconstruction error flowing back to early layers and enables the training to approach an approximately optimal solution (Chen et al., 2021).

2.3. Discrete Cosine Transformation

Discrete Cosine Transformation (DCT) projects the target data from the spatial domain to the frequency domain via the inner-production of the data and a collection of cosine functions with different frequency (Rao & Yip, 2014). We

[2] A $6\times$ compression rate indicates the memory of cached activation maps is $1/6$ of that of normal training.

[3] We do not focus on the closed from the backward function, which is implemented by `torch.autograd`.

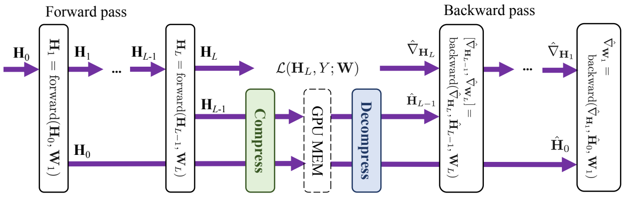

Forward pass
Backward pass
$\mathbf{H}_0$
$\mathbf{H}_1 = \text{forward}(\mathbf{H}_0, \mathbf{W}_1)$
$\mathbf{H}_1$
...
$\mathbf{H}_{L\text{-}1}$
$\mathbf{H}_L = \text{forward}(\mathbf{H}_{L-1}, \mathbf{W}_L)$
$\mathbf{H}_L$
$\mathbf{H}_0$
$\mathbf{H}_{L\text{-}1}$
Compress
GPU MEM
Decompress
$\mathcal{L}(\mathbf{H}_L, Y; \mathbf{W})$
$\hat{\nabla}_{\mathbf{H}_L}$
$\hat{\mathbf{H}}_{L-1}$
$[\hat{\nabla}_{\mathbf{H}_{L-1}}, \hat{\nabla}_{\mathbf{W}_L}] = \text{backward}(\hat{\nabla}_{\mathbf{H}_L}, \hat{\mathbf{H}}_{L-1}, \mathbf{W}_L)$
...
$\hat{\nabla}_{\mathbf{H}_1}$
$\hat{\mathbf{H}}_0$
$\hat{\nabla}_{\mathbf{W}_1} = \text{backward}(\hat{\nabla}_{\mathbf{H}_1}, \hat{\mathbf{H}}_0, \mathbf{W}_1)$

Figure 2: Activation compressed training.

focus on the 2D-DCT in this section, where the target data is the input image and activation maps of DNNs. The cases of 1D/3D-DCT for 1D/3D activation maps are considered in Appendix A. Specifically, for 2D-matrix data $\mathbf{H}$, the frequency-domain feature $\widetilde{\mathbf{H}}$ is estimated by $\widetilde{\mathbf{H}} = \text{DCT}(\mathbf{H})$, where $\mathbf{H}$ and $\widetilde{\mathbf{H}}$ have the same shape of $N \times N$; and each of the element $\tilde{h}_{i,j}$ is given by

$$\tilde{h}_{i,j} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} h_{m,n} \cos\left[\frac{\pi}{N}\left(m + \frac{1}{2}\right)i\right] \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)j\right], \quad (3)$$

where $h_{m,n}$, $0 \le m, n \le N-1$, are elements in the original matrix $\mathbf{H}$. During the training of DNNs, an image or activation map has the shape of $\text{Minibatch} \times \text{Channel} \times N \times N$. In this case, the frequency-domain feature is estimated via operating 2D-DCT for each $N \times N$ matrix in each channel.

With DCT, we could extract the low-frequency/high-frequency component (LFC/HFC) of an image or activation map, using a pipeline of low-pass/high-pass masking and inverse DCT, as shown in Figure 3 (a). To be concrete, the estimation of LFC and HFC is given by

$$\mathbf{H}^{\mathsf{L}} = \text{iDCT}(\widetilde{\mathbf{H}} \odot \mathbf{M}) \quad (4)$$

$$\mathbf{H}^{\mathsf{H}} = \text{iDCT}(\widetilde{\mathbf{H}} \odot (\mathbf{1}_{N \times N} - \mathbf{M})), \quad (5)$$

where $\text{iDCT}(\cdot)$ denotes the inverse DCT (Rao & Yip, 2014); $\mathbf{M} = [m_{i,j} | 1 \le i, j \le N]$ denotes an $N \times N$ low-pass mask satisfying $m_{i,j} = 1$ for $1 \le i, j \le W$ and $m_{i,j} = 0$ for other elements; and $\mathbf{1}_{N \times N} - \mathbf{M}$ indicates the high-pass mask. Intuitively, $\mathbf{H}^{\mathsf{L}}$ has W^2 non-zero float numbers in each channel, in contrast with $N^2 - W^2$ non-zero float numbers in each channel of $\mathbf{H}^{\mathsf{H}}$. Generally, we have $W \ll N$ in practical scenarios, e.g. $W/N = 0.1$ in Figure 3 (a). This indicates the HFC takes the majority of the memory cost in the caching of activation maps.

3. Contribution of LFC and HFC to Backward Propagation

In this section, we experimentally prove the LFC of activation maps has significantly more contribution to DNN backward propagation than the HFC. Meanwhile, we theoretically prove that LFC makes the estimated gradient to be bounded into a tighter range around the optimal value, leading to a more accurate learned model, which is consistent with the experimental results.

3.1. Experimental Analysis

To study the contribution of LFC and HFC to DNN backward propagation, we design three training methods with different backward propagations: **LFC-ACT** takes LFC into the backward function as shown in Equation (6), where $\mathbf{H}_l^{\mathsf{L}}$ is estimated by Equations (4); **HFC-ACT** takes HFC into the backward function as given in Equation (7), where $\mathbf{H}_l^{\mathsf{H}}$ is according to Equation (5); **Normal training** (for comparison) estimates the gradients by Equation (2).

$$[\hat{\nabla}_{\mathbf{H}_{l-1}}, \hat{\nabla}_{\mathbf{W}_l}] = \text{backward}(\hat{\nabla}_{\mathbf{H}_l}, \mathbf{H}_l^{\mathsf{L}}, \mathbf{W}_l), \quad (6)$$

$$[\hat{\nabla}_{\mathbf{H}_{l-1}}, \hat{\nabla}_{\mathbf{W}_l}] = \text{backward}(\hat{\nabla}_{\mathbf{H}_l}, \mathbf{H}_l^{\mathsf{H}}, \mathbf{W}_l). \quad (7)$$

We conduct the experiments on the CIFAR-10 dataset. The implementation details are given in Appendix B. The top-1 accuracy and memory cost of LFC-ACT, HFC-ACT, and normal training are shown in Figure 3 (b) and (c), respectively. Overall, we have the following observations:

- **Accuracy:** According to Figure 3 (b), HFC-ACT suffers from significantly more degradation of accuracy than LFC-ACT. This indicates *DNN backward propagation mainly utilizes the LFC of activation maps*.
- **Memory:** According to Figure 3 (c), the storage of HFC requires significantly more memory than that of the LFC, i.e., *the storage of HFC consumes the majority of memory.*

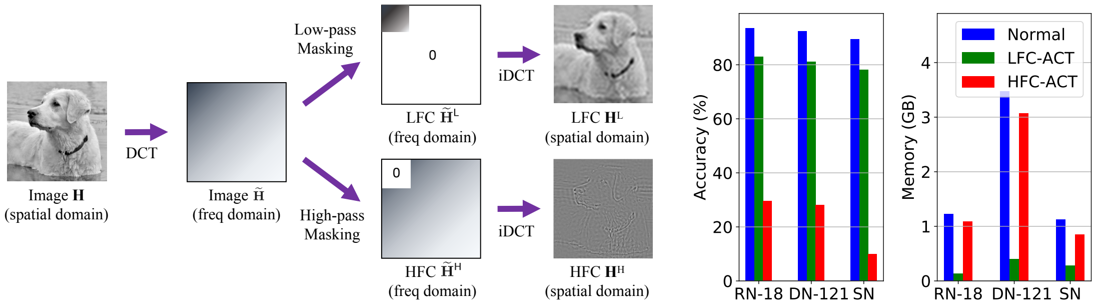
To better understand the results of model accuracy, we theoretically prove the gradient for backward propagation is bounded into a tighter range around the optimal value in LFC-ACT. This enables LFC-ACT to learn a more accurate model than HFC-ACT.

3.2. Theoretical Analysis

We theoretically analyze the gradient estimation error of LFC-ACT and HFC-ACT which adopt Equations (6) and (7) for backward propagation, respectively. Formally, for LFC-ACT and HFC-ACT, let $\hat{\nabla}^{\mathsf{L}}_{\mathbf{W}_l}$ and $\hat{\nabla}^{\mathsf{H}}_{\mathbf{W}_l}$ denote the estimated gradient of layer l, respectively. In this way, $||\hat{\nabla}^{\mathsf{L}}_{\mathbf{W}_l} - \nabla_{\mathbf{W}_l}||_F$[4] and $||\hat{\nabla}^{\mathsf{H}}_{\mathbf{W}_l} - \nabla_{\mathbf{W}_l}||_F$ indicates the gradient estimation errors, taking the complete gradient $\nabla_{\mathbf{W}_l}$ as a reference. To compare the distortion of backward propagation in LFC-ACT and HFC-ACT, let $\text{GEB}_l^{\mathsf{L}}$ and $\text{GEB}_l^{\mathsf{H}}$ denote the gradient error upper bound (GEB), respectively, i.e. $||\hat{\nabla}^{\mathsf{L}}_{\mathbf{W}_l} - \nabla_{\mathbf{W}_l}||_F \le \text{GEB}_l^{\mathsf{L}}$ and $||\hat{\nabla}^{\mathsf{H}}_{\mathbf{W}_l} - \nabla_{\mathbf{W}_l}||_F \le \text{GEB}_l^{\mathsf{H}}$. Intuitively, higher GEB indicates less accurate backward propagation, leading to a less accurate model after the training. To this end, we give Theorem 3.1 to compare $\text{GEB}_l^{\mathsf{L}}$ and $\text{GEB}_l^{\mathsf{H}}$, where a convolutional layer is considered. The proof is given in Appendix M. A similar analysis of GEB for a linear layer applied to MLPs and Transformers is provided in Appendix N.

Theorem 3.1. *During the backward pass of a convolutional*

[4] The Frobenius norm of $n \times n$ matrix $\mathbf{A}$ is given by $||\mathbf{A}||_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2}$.



(a) (b) (c)

Figure 3: (a) Adopting DCT to estimate the low frequency component (LFC) and high frequency component (HFC) of an image. (b) Top-1 accuracy and (c) Memory cost of normal training, LFC-ACT and HFC-ACT, where RN-18, DN-121, and SN refer to the ResNet-18, DenseNet-121, and ShuffleNet-V2, respectively.

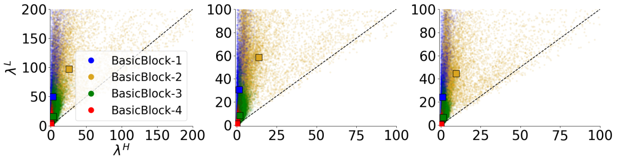
layer l, $\mathrm{GEB}_l^{\mathsf{L}}$ and $\mathrm{GEB}_l^{\mathsf{H}}$ satisfy

$$\mathrm{GEB}_l^{\mathsf{L}} - \mathrm{GEB}_l^{\mathsf{H}} = \Big(\alpha_{l,l}||\mathbf{H}_{l-1}^{\mathsf{T}}||_F + \beta_l\Big)(\lambda_l^{\mathsf{H}} - \lambda_l^{\mathsf{L}}) + ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F \sum_{i=l+1}^{L} \alpha_{l,i}(\lambda_i^{\mathsf{H}} - \lambda_i^{\mathsf{L}}) \prod_{j=l}^{i-1} \gamma_j, \tag{8}$$

where $\alpha_{l,i}, \beta_l, \gamma_l > 0$ for $1 \le l, i \le L$ depend on the model weights before backward propagation (given by Equations (28) in Appendix M); $\lambda_l^{\mathsf{L}} = ||\widetilde{\mathbf{H}}_l \odot \mathbf{M}||_F$; $\lambda_l^{\mathsf{H}} = ||\widetilde{\mathbf{H}}_l \odot (\mathbf{1} - \mathbf{M})||_F$; $\widetilde{\mathbf{H}}_l = \mathrm{DCT}(\mathbf{H}_l)$; and $\mathbf{M}$ denotes the loss-pass mask given by Equation (4).

Theorem 3.1 indicates the GEB difference depends on $\lambda_l^{\mathsf{H}} - \lambda_l^{\mathsf{L}}$ for $1 \le l \le L$ during the training. Following this direction, we estimate λ_l^{L} and λ_l^{H} via $\lambda_l^{\mathsf{L}} = ||\widetilde{\mathbf{H}}_l \odot \mathbf{M}||_F$ and $\lambda_l^{\mathsf{H}} = ||\widetilde{\mathbf{H}}_l \odot (\mathbf{1} – \mathbf{M})||_F$ during the training of ResNet-18 and DenseNet-121 on the CIFAR-10 dataset. Specifically, $\mathbf{H}_l$ takes the activation maps of the BasicBlocks in ResNet-18, and Denseblocks in DenseNet-121. The estimation of λ_l^{L} and λ_l^{H} is based on the checkpoint of ResNet-18 in epoches 20, 40, and 60, and visualized in Figures 4 (a)-(c), respectively. The implementation details and results of DenseNet-121 are given in Appendix C. It is consistently observed that $\lambda_l^{\mathsf{L}} > \lambda_l^{\mathsf{H}}$ for different instances and layers. This leads to $\mathrm{GEB}_l^{\mathsf{L}} < \mathrm{GEB}_l^{\mathsf{H}}$ according to Theorem 3.1. Therefore, HFC-ACT suffers from a worse distortion of backward propagation during the training, eventually leading to less accurate learned model than LFC-ACT.

With both experiments and theoretical analysis, we have proved the HFC of activation maps has less contribution to backward propagation than LFC. However, according to Figure 3 (c), the HFC takes the majority of memory cost during the training, so it is highly redundant and compressible during the training. Motivated by this, we propose DIVISION to compress the activation maps into a dual precision representation: *high-precision LFC* and *low-precision HFC*. On the one hand, both LFC and low-precision HFC requires much less memory to cache. On the other hand, removing redundancy from HFC will not cause much distortion in backward propagation. In this way, DIVISION significantly reduces training memory without affecting model accuracy.



(a) Epoch 20. (b) Epoch 40. (c) Epoch 60.

Figure 4: $\lambda_l^{\mathsf{L}} = ||\widetilde{\mathbf{H}}_l \odot \mathbf{M}||_F$ versus $\lambda_l^{\mathsf{H}} = ||\widetilde{\mathbf{H}}_l \odot (\mathbf{1} - \mathbf{M})||_F$ in training epoches 20, 40, and 60 of ResNet-18. $\mathbf{H}_l$ takes the activation maps of four BasicBlocks in ResNet-18; the y- and x-axis of □ indicates the expectation of λ_l^{L} and λ_l^{H}, respectively.

4. Dual Activation Precision Training

We introduce the proposed Dual ActIVation PrecISION (DIVISION) in this section. The framework of DIVISION is shown in Figure 5. Specifically, after the feed-forward operation of each layer, DIVISION estimates the LFC and compresses the HFC into a low-precision copy such that the total memory cost is significantly reduced. Before the backward propagation of each layer, the low-precision HFC is decompressed and combined with LFC to reconstruct the activation map. To facilitate illustration, the compression and decompression are formalized based on 2D activation maps in this section. A similar processing of 1D/3D activation maps is discussed in Appendix D.

4.1. Activation Map Compression

To compress the activation map $\mathbf{H}_l$ of layer l, DIVISION estimates the LFC $\mathbf{H}_l^{\mathsf{L}}$ and HFC $\mathbf{H}_l^{\mathsf{H}}$ using DCT after the feed-forward operation. However, the high computational complexity of DCT prevents us from directly applying it to real-time algorithms. We thus give Theorem 4.1 to introduce a moving average operation that can approximate the loss-pass filter. The proof is given in Appendix O.

Theorem 4.1. *For any real-valued function $f(x)$ and its moving average $\bar{f}(x) = \frac{1}{2B}\int_x^{x+2B} f(t)\mathrm{d}t$, let $F(\omega)$ and $\overline{F}(\omega)$ denote the Fourier transformation of $f(x)$ and $\bar{f}(x)$, respectively. Generally, we have $\overline{F}(\omega) = H(\omega)F(\omega)$, where $|H(\omega)| = \left|\frac{\sin \omega B}{\omega B}\right|$.*

Remark 4.2. The frequency response of $H(\omega)$ depends on its envelope function $\frac{1}{|\omega B|}$. Note that $\frac{1}{|\omega B|}$ decreases with

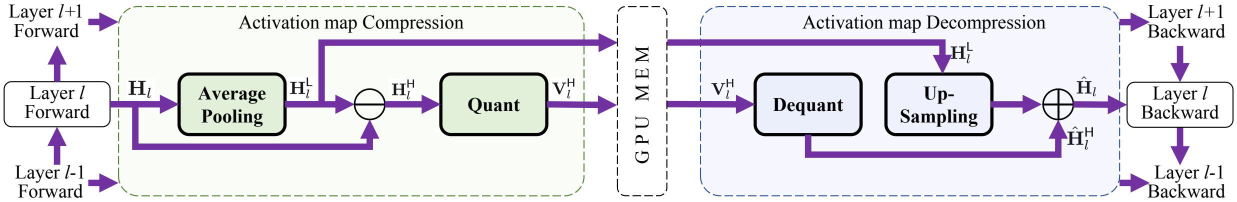

Layer l+1
Forward
Layer l
Forward
Layer l-1
Forward
Activation map Compression
$\mathbf{H}_l$
Average
Pooling
$\mathbf{H}_l^{\mathsf{L}}$
$\mathbf{H}_l^{\mathsf{H}}$
Quant
$\mathbf{V}_l^{\mathsf{H}}$
GPU MEM
Activation map Decompression
$\mathbf{H}_l^{\mathsf{L}}$
$\mathbf{V}_l^{\mathsf{H}}$
Dequant
Up-
Sampling
$\hat{\mathbf{H}}_l$
$\hat{\mathbf{H}}_l^{\mathsf{H}}$
Layer l+1
Backward
Layer l
Backward
Layer l-1
Backward

Figure 5: The proposed framework of Dual Activation Precision Training.

$|\omega|$ such that $\frac{1}{|\omega B|} \to 0$ as $\omega \to \infty$. Hence, $H(\omega)$ is an approximate loss-pass filter.

According to Remark 4.2, we approximate the LFC $\mathbf{H}_l^{\mathsf{L}}$ with the moving average of $\mathbf{H}_l$. Notably, the average pooling operator provides efficient moving average, so DIVISION adopts average pooling to estimate the LFC as $\mathbf{H}_l^{\mathsf{L}} = \mathrm{AveragePooling}(\mathbf{H}_l)$. The value of block-size and moving stride is a unified hyper-parameter B, which controls the memory of $\mathbf{H}_l^{\mathsf{L}}$[5]. Moreover, $\mathbf{H}_l^{\mathsf{L}}$ is cached in the format of `bfloat16` for saving the memory. In our experiments, we found $B = 8$ can provide representative LFC for backward propagation, where the memory cost of $\mathbf{H}_l^{\mathsf{L}}$ is only 0.8% of $\mathbf{H}_l$.

To estimate the HFC, DIVISION calculates the residual given by $\mathbf{H}_l^{\mathsf{H}} = \mathbf{H}_l - \mathrm{UpSampling}(\mathbf{H}_l^{\mathsf{L}})$, where the upsampling operation enlarges $\mathbf{H}_l^{\mathsf{L}}$ to shape $\mathrm{Minibatch} \times \mathrm{Channel} \times N \times N$ via nearest interpolation. Then, DIVISION compress $\mathbf{H}_l^{\mathsf{H}}$ into low-precision because it plays a less important role during the backward propagation but consumes most of the memory. Specifically, DIVISION adopts Q-bit per-channel quantization[6][7] for the compression, where the bit-width Q controls the precision and memory cost of HFC after the compression. Let $\mathbf{V}_l^{\mathsf{H}}$ denote a Q-bit integer matrix, as the low-precision representation of $\mathbf{H}_l^{\mathsf{H}}$. The procedure of compressing $\mathbf{H}_l^{\mathsf{H}}$ into $\mathbf{V}_l^{\mathsf{H}}$ is given by

$$\mathbf{V}_l^{\mathsf{H}} = \mathrm{Quant}(\mathbf{H}_l^{\mathsf{H}}) = \lfloor \Delta_l^{-1}(\mathbf{H}_l^{\mathsf{H}} - \delta_l) \rceil, \tag{9}$$

where δ_l denotes the minimum element in $\mathbf{H}_l^{\mathsf{H}}$; $\Delta_l = (h_{\max} - \delta_l)/(2^Q - 1)$ denotes the quantization step; $h_{\max}$ denotes the maximum element in $\mathbf{H}_l^{\mathsf{H}}$; $\lfloor \bullet \rceil$ denotes the *stochastic rounding*[8][9] (Gupta et al., 2015); and δ_l and Δ_l are cached in the formate of `bfloat16` for saving memory. In this way, the memory cost of $(\mathbf{V}_l^{\mathsf{H}}, \delta_l, h_{\max})$ is $(N^2 Q/8 + 4)$ bytes per channel, in contrast with that of $\mathbf{H}_l$ being $4N^2$ bytes per channel. In our experiments, we found $Q = 2$ can provide enough representation for backward propagation, where the memory cost of $\mathbf{V}_l^{\mathsf{H}}$ is only 8.3% of $\mathbf{H}_l$.

After the compression, as the representation of $\mathbf{H}_l$, the tuple of $(\mathbf{H}_l^{\mathsf{L}}, \mathbf{V}_l^{\mathsf{H}}, \Delta_l, \delta_l)$ is cached to the memory for reconstructing the activation maps during the backward pass.

4.2. Activation Map Decompression

During the backward pass, DIVISION adopts the cached tuples of $\{(\mathbf{H}_l^{\mathsf{L}}, \mathbf{V}_l^{\mathsf{H}}, \Delta_l, \delta_l) \,|\, 0 \le l \le L-1\}$ to reconstruct the activation map layer-by-layer. Specifically, for each layer l, DIVISION dequantizes the HFC via $\hat{\mathbf{H}}_l^{\mathsf{H}} = \Delta_l \mathbf{V}_l^{\mathsf{H}} + \delta_l$, which is the inverse process of Equation (9). Then, the activation map is reconstructed via

$$\hat{\mathbf{H}}_l = \mathrm{UpSampling}(\mathbf{H}_l^{\mathsf{L}}) + \hat{\mathbf{H}}_l^{\mathsf{H}}, \tag{10}$$

where $\mathrm{UpSampling}(\cdot)$ enlarges $\mathbf{H}_l^{\mathsf{L}}$ to the shape of $\mathrm{Minibatch} \times \mathrm{Channel} \times N \times N$ via nearest interpolation. After the decompression, DIVISION frees the caching of $(\mathbf{H}_l^{\mathsf{L}}, \mathbf{V}_l^{\mathsf{H}}, \Delta_l, \delta_l)$, and takes $\hat{\mathbf{H}}_l$ into $[\hat{\nabla}_{\mathbf{H}_{l-1}}, \hat{\nabla}_{\mathbf{W}_l}] = \mathrm{backward}(\hat{\nabla}_{\mathbf{H}_l}, \hat{\mathbf{H}}_{l-1}, \mathbf{W}_l)$ to estimate the gradient for backward propagation.

Algorithm 1 Mini-batch updating of DIVISION

```
Input: Mini-batch samples $\mathbf{x}$ and labels $\mathrm{y}$.
Output: Weight and bias $\{\mathbf{W}_l, \mathbf{B}_l | 1 \le l \le L\}$.
 1: for layer $l := 1$ to $L$ do
 2:     $\mathbf{H}_l = f(\mathbf{W}_l \mathbf{H}_{l-1} + \mathbf{B}_l)$ // $\mathbf{H}_0 = \mathbf{x}$
 3:     $\mathbf{H}_{l-1}^{\mathsf{L}} = \mathrm{AveragePooling}(\mathbf{H}_{l-1})$
 4:     $\mathbf{H}_{l-1}^{\mathsf{H}} = \mathbf{H}_{l-1} - \mathrm{UpSampling}(\mathbf{H}_{l-1}^{\mathsf{L}})$
 5:     $\mathbf{V}_{l-1}^{\mathsf{H}}, \Delta_{l-1}, \delta_{l-1} = \mathrm{Quant}(\mathbf{H}_{l-1}^{\mathsf{H}})$
 6:     Cache $(\mathbf{H}_{l-1}^{\mathsf{L}}, \mathbf{V}_{l-1}^{\mathsf{H}}, \Delta_{l-1}, \delta_{l-1})$
 7: end for
 8: Estimate the loss value and gradient $\hat{\nabla}_{\mathbf{H}_L}$.
 9: for layer $l := L$ to 1 do
10:     $\hat{\mathbf{H}}_{l-1}^{\mathsf{H}} = \mathrm{Dequant}(\mathbf{V}_{l-1}^{\mathsf{H}}, \Delta_{l-1}, \delta_{l-1})$
11:     $\hat{\mathbf{H}}_{l-1} = \mathrm{UpSampling}(\mathbf{H}_{l-1}^{\mathsf{L}}) + \hat{\mathbf{H}}_{l-1}^{\mathsf{H}}$
12:     Estimate $[\hat{\nabla}_{\mathbf{H}_{l-1}}, \hat{\nabla}_{\mathbf{W}_l}]$ and update $\mathbf{W}_l$.
13:     Free $(\mathbf{H}_{l-1}^{\mathsf{L}}, \mathbf{V}_{l-1}^{\mathsf{H}}, \Delta_{l-1}, \delta_{l-1})$.
14: end for
```

4.3. The Algorithm of DIVISION

Algorithm 1 presents a mini-batch updating of DIVISION, which includes a forward pass and backward pass. During the forward pass of each layer, DIVISION first forwards the exact activation map to the next layer (line 2); then, estimates the LFC and HFC (line 3-4); after this, achieves the

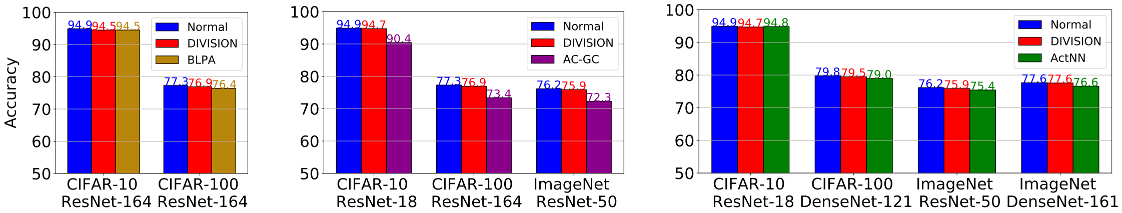
[5] For the case $N < B$, the pooling block-size and stride will be N such that the shape of $\mathbf{H}_l^{\mathsf{L}}$ is $\mathrm{Minibatch} \times \mathrm{Channel} \times 1 \times 1$.

[6] A fixed bit-width is adopted for the quantization of all layers to maximize the efficiency of data processing.

[7] Per-channel quantization is more efficient and light than per-group quantization in state-of-the-art work.

[8] $\lfloor x \rceil$ takes the value of $\lfloor x \rfloor$ with a probability of $x - \lfloor x \rfloor$ and takes $\lceil x \rceil$ with a probability of $\lceil x \rceil - x$.

[9] The stochastic rounding enables the pipeline of quantization and dequantization to be unbiased, i.e. $\mathbb{E}[\mathbf{V}_l^{\mathsf{H}}] = \mathbf{H}_l^{\mathsf{H}}$.



(a) DIVISION vs BLPA

(b) DIVISION vs AC-GC

(c) DIVISION vs ActNN

Figure 6: Top-1 accuracy (%) ↑ of Normal training, DIVISION, BLPA (a), AC-GC (b), and ActNN (c).

low precision copy of HFC (lines 5); finally caches the representation to the memory (line 6). During the backward pass of each layer, DIVISION first decompresses the HFC (line 10); reconstructs the activation map (line 11); estimates the gradients and updates the weights of layer l (line 12); finally frees the caching of $(\mathbf{H}_{l-1}^{\mathsf{L}}, \mathbf{V}_{l-1}^{\mathsf{H}}, \Delta_{l-1}, \delta_{l-1})$ (line 13). For each mini-batch updating, the memory usage reaches the maximum value after the forward pass (caching the representation of activation maps layer-by-layer), and reduces to the minimum value after the backward pass (freeing the cache layer-by-layer). Existing work (Chen et al., 2021) estimates the memory cost of activation maps by

$$\text{MEM Cost} = \text{MEM Util}_{\text{after forward}} - \text{MEM Util}_{\text{after backward}},$$

where existing deep learning tools provide APIs[10] to estimate the memory utilization.

The theoretical compression rate R of DIVISION is given in Appendix P, where general cases of convolutional neural networks and multi-layer perception are considered for the estimation. For the model architectures in our experiments, we have $R_{\text{ResNet-50}}, R_{\text{WRN-50-2}} \geq 10.35$.

5. Evaluation of DIVISION

We conduct experiments to evaluate DIVISION by answering the following research questions. **RQ1:** How does DIVISION perform compared with state-of-the-art baseline methods in terms of the model accuracy, memory cost, and training throughput? **RQ2:** Does the strategy of dual-precision compression contribute to DIVISION? **RQ3:** What is the effect of hyper-parameters on DIVISION?

5.1. Experiment Setup

The experiment settings including the datasets, baseline methods and DNN architectures are given in Appendix E. The implementation details and configuration of computational infrastructure are given in Appendix F and K, respectively. Experiments on MLPs are given in Appendix H.

5.2. Evaluation by Model Accuracy (RQ1)

In this section, we evaluate the training methods in terms of model accuracy on the CIFAR-10, CIFAR-100 and ImageNet datasets. Specifically, DIVISION is compared with BLPA (Chakrabarti & Moseley, 2019), AC-GC (Evans & Aamodt, 2021) and ActNN (Chen et al., 2021) in Figure 6 (a)-(c), respectively, where different model architectures are considered. We do not consider Checkpoint and SWAP in this section because they can reduce the training memory without degradation of model accuracy. Overall, we have the following observations:

- **DIVISION vs Baseline Methods:** Compared with normal training, DIVISION achieves almost the same accuracy, which indicates a loss-less compression of training. In contrast, BLPA and ActNN show a slight degradation; and AC-GC has lower accuracy than other methods.
- **Flexibility of DIVISION:** DIVISION consistently achieves competitive model accuracy in the training of different architectures on different datasets. This indicates DIVISION is a flexible framework that can be applied to different architectures and datasets.
- **Compressibility of HFC:** Note that DIVISION adopts a significantly high compression rate 12× for the HFC during the training, and achieves nearly loss-less accuracy. This result indicates the HFC of activation map is highly redundant and compressible during the training.

5.3. Evaluation by Memory Cost (RQ1)

We evaluate different training methods in terms of the training memory cost on the ImageNet datasest (the configuration of our computational infrastructure is given in Appendix K). Table 1 (a) indicates the training memory cost and practical compression rate of DIVISION and baseline methods. Overall, we have the following observations:

- **DIVISION vs Checkpoint & BLPA:** Checkpoint shows less effective compression because it caches some key activation maps to reconstruct other activation maps during the training. BLPA has lower compression rate than DIVISION because it relies on at least 4-bit compression.
- **DIVISION vs AC-GC:** AC-GC searches the bit-width from an initial maximum value, and finalizes the bit-width as 7.01. The compression rate of activation maps in the last epoch is 3.5×, which is lower than DIVISION.
- **DIVISION vs ActNN:** DIVISION has approximately the same memory cost as ActNN. Beyond the storage of 2-bit

[10] `torch.cuda.memory_allocated` returns the memory occupied by tensors in bytes.

Table 1: (a) Memory cost↓ and compression rate↑. *Total Mem* refers to total memory cost of weights, optimizer, data and activation maps. *Act Mem* refers to memory cost of activation maps. (b) Performance of DIVISION, Checkpoint, and Mesa on the Swin-Transformer. (c) Model accuracy with fixed bit-width quantization of DIVISION w/o LFC, DIVISION w/o HFC, and DIVISION.

(a)

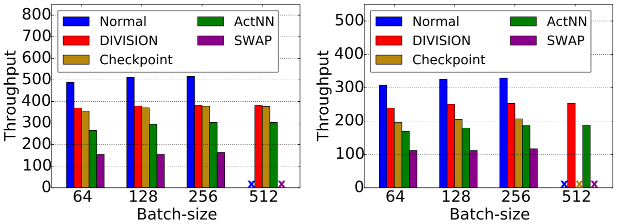
Architecture		ResNet-50				WRN-50-2			
Batch-size		64	128	256	512	64	128	256	512
Total Mem (GB)	Normal	5.46	10.62	20.92	*OOM*	7.52	14.23	27.68	*OOM*
	Checkpoint	2.57 (2.1×)	4.84 (2.2×)	9.39 (2.2×)	18.49	3.05 (2.5×)	5.33 (2.7×)	9.88 (2.8×)	*OOM*
	BLPA	1.15 (4.7×)	2.01 (5.3×)	3.72 (5.6×)	7.14	1.87 (4.0×)	2.96 (4.8×)	5.15 (5.4×)	9.51
	AC-GC	1.80 (3.0×)	3.31 (3.2×)	6.31 (3.3×)	12.33	2.72 (2.8×)	4.66 (3.1×)	8.53 (3.2×)	16.27
	ActNN	**0.81 (6.7×)**	**1.34 (7.9×)**	**2.39 (8.8×)**	**4.47**	**1.44 (5.2×)**	**2.09 (6.8×)**	**3.41 (8.1×)**	**6.03**
	DIVISION	**0.82 (6.7×)**	**1.35 (7.9×)**	**2.41 (8.7×)**	**4.52**	**1.45 (5.2×)**	**2.12 (6.7×)**	**3.44 (8.0×)**	**6.08**
Act. Mem (GB)	Normal	5.14	10.25	20.48	*OOM*	6.70	13.38	26.75	*OOM*
	Checkpoint	2.24 (2.3×)	4.48 (2.3×)	8.95 (2.3×)	17.90	2.24 (3.0×)	4.48 (3.0×)	8.95 (3.0×)	*OOM*
	BLPA	0.82 (6.3×)	1.64 (6.2×)	3.28 (6.2×)	6.56	1.06 (6.3×)	2.11 (6.3×)	4.22 (6.3×)	8.44
	AC-GC	1.47 (3.5×)	2.94 (3.5×)	5.88 (3.5×)	11.75	1.91 (3.5×)	3.81 (3.5×)	7.61 (3.5×)	15.20
	ActNN	**0.49 (10.5×)**	**0.97 (10.6×)**	**1.94 (10.6×)**	**3.89**	**0.62 (10.8×)**	**1.25 (10.7×)**	**2.49 (10.7×)**	**4.97**
	DIVISION	**0.49 (10.5×)**	**0.99 (10.4×)**	**1.97 (10.4×)**	**3.94**	**0.64 (10.5×)**	**1.27 (10.5×)**	**2.52 (10.6×)**	**5.02**

(b)

	Acc (%)	Mem. (GB)	Throughput
Normal	81.2	14.43 (1×)	233.53 ips
Mesa	**81.3**	6.56 (2.20×)	136.70 ips
Checkpoint	81.2	6.72 (2.15×)	**201.73** ips
DIVISION	81.0	**5.07 (2.85×)**	175.48 ips

(c)

	CIFAR-100	ImageNet
Fixed-4bit	75.07	76.05
Fixed-2bit	1	0.1
w/o LFC	60.54	7.79
w/o HFC	69.56	28.51
DIVISION	76.3	75.9



(a) ImageNet. (b) WRN-50-2.

Figure 7: Training throughput ↑ of (a) Resnet-50 and (b) WRN-50-2 on the ImageNet dataset, where ✖ indicates out of memory.

activation maps, DIVISION has overhead for caching the LFC; and ActNN spends almost equal overhead for storing the parameters of per-group quantization.

- **Compression Rate:** The activation map compression rate of DIVISION is consistent with the theoretical results ($R_{\text{ResNet-50}}, R_{\text{WRN-50-2}} \geq 10.35$, see Appendix P), which is not influenced by the mini-batch size. Moreover, the overall compression rate grows with the mini-batch size.
- **Activation Maps:** For the normal training, the storage of activation maps takes the majority of memory cost ($> 90\%$, growing with the mini-batch size), which is consistent with our discussion in Section 1.

5.4. Evaluation by Training Throughput (RQ1)

We now evaluate the training methods in terms of the training throughput on the ImageNet dataset. Generally, the throughput indicates the speed of a method via counting the number of data samples processed per second. Formally, it is given by $\frac{\text{Mini-batch Size}}{T_{\text{batch}}}$, where T_{batch} denotes the time of a single mini-batch updating. According to the training throughput of DIVISION and baseline methods in Figures 7 (a) and (b), we have the following observations:

- **Reasons for Time overhead:** Compared with normal training, the time overhead of DIVISION comes from the estimation of LFC and quantization of HFC. In ActNN, it mainly comes from the the dynamic bit-width allocation and activation map quantization. In Checkpoint, it comes from replaying the forward process of inter-media layers. In SWAP, the overhead mainly derives from the communication cost between the CPUs and GPUs.
- **DIVISION vs ActNN:** DIVISION shows 1.3× acceleration compared to ActNN, as a result of simplified data compression. To be concrete, DIVISION adopts a simple average-pooling to extract the LFC, and a fixed bit-width per-channel quantization to compress the HFC. In contrast, ActNN relies on a searching of optimal bit-width to match different samples, and adopts the searched bit-width for per-group quantization. ActNN has a more complex processing during the training, which leads to its lower throughput than DIVISION.

To summarize, according to Figures 6, 7 and Table 1, state-of-the-art methods AC-GC, Checkpoint and ActNN shows a performance degradation in the aspect of model accuracy, compression rate, and training throughput, respectively. According to a comprehensive comparison in terms of the three evaluation metrics in Figure 1, **DIVISION shows better comprehensive performance than these methods**.

5.5. Performance of DIVISION on Vision Transformer

To further evaluate DIVISION on vision transformers, we conduct experiments of Swin Transformer (Liu et al., 2021a) on the ImageNet dataset in comparison with Mesa (Pan et al., 2021) and Checkpoint (Shoeybi et al., 2019). The model accuracy, memory cost (with batch-size 128) and training throughput are given in Table 1 (b). It is observed that DIVISION can effectively compress the training of the transformer, with almost the same model accuracy with normal training. Although DIVISION shows slightly lower accuracy and throughput than Mesa and Checkpoint, respectively, it can significantly save more memory (nearly 1.5GB than Mesa, and 1.7GB than Checkpoint). Moreover, Mesa is explicitly designed for vision transformers, and Checkpoint relies on manually selection of checkpointed layers. DIVISION is flexible for general vision models, including MLPs, CNNs, and vision transformers.

Table 2: (a) Accuracy of MoblieNet-V2 on the CIFAR-10 and CIFAR-100 datasets. (b) Performance of DIVISION under different hyperparameter settings. $n\times$ refers to the compression rate.

(a)

Method	MN-V2 CIFAR-10	MN-V2 CIFAR-100
Normal	91.9	71.0
DIVISION	91.8	70.6

(b)

	$B=18$ $Q=2$	$B=12$ $Q=2$	$B=8$ $Q=2$	$B=8$ $Q=4$	$B=8$ $Q=8$
Acc(%)	78.70	92.78	94.59	94.84	94.91
$n\times$	10.18	10.18	9.74	5.74	3.25

5.6. Performance of DIVISION on Depthwise and Pointwise Convolutional Layers

To evaluate DIVISION on the depthwise convlution and pointwise convulution layers, we conduct experiments of MobileNet-V2 on the CIFAR-10 and CIFAR-100 datasets. The model accuracy of normal training and DIVISION are given Table 2 (a). It is observed that DIVISION achieve nearly the same accuracy compared with normal training. This indicates the effectiveness of DIVISION deployed to the depthwise convlution and pointwise convulution layers.

5.7. Effect of Dual Precision Strategy (RQ2)

To study the effect of our proposed dual precision strategy, DIVISION is compared with three training methods: **DIVISION w/o HFC**: Merely providing the LFC for backward propagation. **DIVISION w/o LFC**: Merely providing the low-precision HFC for backward propagation. **Fixed Quant**: Compressing the activation maps using a fixed bit-width quantization. The experiments are conducted on the CIFAR-100 and ImageNet dataset using the hyper-parameters given in Appendix L. The model accuracy are given in Table 1 (c). Overall, we have the following insights:

- **LFC & Low Precision HFC:** Removing either HFC or LFC from DIVISION, the training converges to far lower levels of accuracy. This indicates both the LFC and low precision HFC of activation maps are necessary for leading the training to converge to an optimal solution.
- **Benifits of Dual Precision:** For the training with a fixed bit-width quantization, the bit-width should be at least 4. The noise caused by a fixed 2-bit quantization can terribly disturb the backward propagation, leading to a failure of convergence, as shown in Table 1 (c). DIVISION solves this problem by combining a high-precision LFC and a fixed 2-bit quantization for the compression, and achieves nearly loss-less model accuracy.

5.8. Hyper-parameter Tuning for DIVISION (RQ3)

We study the effect of hyper-parameters B (block-size) and Q (bit-width) on the accuracy and compression rate. Specifically, we adopt DIVISION to train ResNet-18 on the CIFAR-10 dataset with $B \in \{8, 12, 18\}$ and $Q \in \{2, 4, 8\}$. The model accuracy and compression rate are given in Table 2 (b). We have the following insights:

- **Effect of Q:** DIVISION shows a stable accuracy (nearly 94.8%) as the precision of HFC reduces (Q reduces from 8 to 2). This indicates it only requires approximate values of HFC during backward propagation.
- **Effect of B:** As B grows from 8 to 18, caching lower precision LFC during the forward pass leads to significant degradation of accuracy. This is because DIVISION relies on a high-precision LFC to reconstruct the activation maps during the backward propagation.
- **Optimal Setting:** DIVISION shows optimal accuracy-compression trade-off taking $B=8$ and $Q=2$, where the degradation of accuracy is less than 0.35%.

Table 3: Performance of DIVISION deployed to ResNet-18 and MoblieNet-V2 with different hyper-parameter settings.

Hyperparameter setting of DIVISION	$B = 18$ $Q = 2$		$B = 8$ $Q = 2$		$B = 8$ $Q = 8$	
Evaluation metric	Acc(%)	$n\times$	Acc(%)	$n\times$	Acc(%)	$n\times$
RN-18 CIFAR-10	78.7	10.18×	94.6	9.74×	94.9	3.25×
RN-18 CIFAR-100	73.2	10.18×	76.9	9.33×	77.0	3.25×
MN-V2 CIFAR-10	10.0	3.26×	91.8	3.20×	91.0	2.00×
MN-V2 CIFAR-100	62.4	3.26×	70.6	3.20×	71.6	2.00×

5.9. Re-utilization of Hyper-parameter Settings Across Different Model Architectures and Datasests (RQ3)

We conduct follow-up experiments to study whether the hyper-parameter setting of DIVISION has a consistent effect on different model architectures and datasets. Specifically, the performance of DIVISION deployed to ResNet-18 and MobileNet-V2 on the CIFAR-10 and CIFAR-100 datasets is shown in Table 3, where the hyper-parameters are selected from $B \in \{8, 18\}$ and $Q \in \{2, 8\}$. It is observed B and Q have a consistent impact on different model architectures and datasets: the accuracy slightly grows with Q and considerably reduces with B. This indicates we can reuse the hyper-parameters of DIVISION on CIFAR-10 to CIFAR-100 with the same model architecture, or reuse the setting across ResNet-18 and MobileNet-V2 on the same dataset. Moreover, $B=8$ and $Q=2$ can be a general and effective setting for most model architectures and datasets.

6. Conclusion

In this work, we propose a simple framework of activation compressed training. Our framework is motivated by an instructive observation: *DNN backward propagation mainly utilizes the LFC of the activation maps, while the majority of memory is for the storage of HFC during the training.* This indicates the HFC of the activation maps is highly redundant and compressible during the training. Following this direction, our proposed DIVISION compresses the activation maps into dual precision representations: high-precision LFC and low-precision HFC, corresponding to their contributions to the backward propagation. This dual precision compression can significantly reduce the memory cost of DNN training without loss of model accuracy.

Different from the existing work of ACT, DIVISION is a simple and transparent framework, where the simplicity enables efficient compression and decompression; and transparency allows us to understand the compressible (HFC) and non-compressible factors (LFC) during DNN training. To this end, we hope our work could provide some inspiration for future work of DNN training.

7. Acknowledgement

The authors thank the anonymous reviewers for their helpful comments. The work is in part supported by NSF grants NSF IIS-2224843. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing any funding agencies.

References

Chakrabarti, A. and Moseley, B. Backprop with approximate activations for memory-efficient network training. *Advances in Neural Information Processing Systems*, 32, 2019.

Chen, J., Gai, Y., Yao, Z., Mahoney, M. W., and Gonzalez, J. E. A statistical framework for low-bitwidth training of deep neural networks. *Advances in Neural Information Processing Systems*, 33:883–894, 2020.

Chen, J., Zheng, L., Yao, Z., Wang, D., Stoica, I., Mahoney, M., and Gonzalez, J. Actnn: Reducing training memory footprint via 2-bit activation compressed training. In *International Conference on Machine Learning*, pp. 1803–1813. PMLR, 2021.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

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

Desai, A., Xu, Z., Gupta, M., Chandran, A., Vial-Aussavy, A., and Shrivastava, A. Raw nav-merge seismic data to subsurface properties with mlp based multi-modal information unscrambler. *Advances in Neural Information Processing Systems*, 34:8740–8752, 2021.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Diniz, P. S., Da Silva, E. A., and Netto, S. L. *Digital signal processing: system analysis and design*. Cambridge University Press, 2010.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Duan, K., Liu, Z., Wang, P., Zheng, W., Zhou, K., Chen, T., Hu, X., and Wang, Z. A comprehensive study on large-scale graph training: Benchmarking and rethinking. *arXiv preprint arXiv:2210.07494*, 2022a.

Duan, K., Liu, Z., Zheng, W., Wang, P., Zhou, K., Chen, T., Wang, Z., and Hu, X. Benchmarking large-scale graph training over effectiveness and efficiency. In *Workshop on Graph Learning Benchmarks*, 2022b.

Evans, R. D. and Aamodt, T. Ac-gc: Lossy activation compression with guaranteed convergence. *Advances in Neural Information Processing Systems*, 34:27434–27448, 2021.

Ferber, A., Huang, T., Zha, D., Schubert, M., Steiner, B., Dilkina, B., and Tian, Y. Surco: Learning linear surrogates for combinatorial nonlinear optimization problems. In *International Conference on Machine Learning*, 2023.

Fu, F., Hu, Y., He, Y., Jiang, J., Shao, Y., Zhang, C., and Cui, B. Don't waste your bits! squeeze activations and gradients for deep neural networks via tinyscript. In *International Conference on Machine Learning*, pp. 3304–3314. PMLR, 2020.

Georgiadis, G. Accelerating convolutional neural networks via activation map compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7085–7095, 2019.

Gruslys, A., Munos, R., Danihelka, I., Lanctot, M., and Graves, A. Memory-efficient backpropagation through time. *Advances in Neural Information Processing Systems*, 29, 2016.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. Deep learning with limited numerical precision. In *International conference on machine learning*, pp. 1737–1746. PMLR, 2015.

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

Horn, R. A. and Johnson, C. R. *Matrix analysis*. Cambridge university press, 2012.

Huang, C.-C., Jin, G., and Li, J. Swapadvisor: Pushing deep learning beyond the gpu memory limit via smart swapping. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1341–1355, 2020.

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

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Lin, X., Zhao, C., and Pan, W. Towards accurate binary convolutional neural network. *Advances in neural information processing systems*, 30, 2017.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021a.

Liu, Z., Zhou, K., Yang, F., Li, L., Chen, R., and Hu, X. Exact: Scalable graph neural networks training via extreme activation compression. In *International Conference on Learning Representations*, 2021b.

Liu, Z., Chen, S., Zhou, K., Zha, D., Huang, X., and Hu, X. Rsc: Accelerating graph neural networks training via randomized sparse computations. *arXiv preprint arXiv:2210.10737*, 2022a.

Liu, Z., Zhou, K., Yang, F., Li, L., Chen, R., and Hu, X. Exact: Scalable graph neural networks training via extreme activation compression. In *International Conference on Learning Representations*, 2022b. URL https://openreview.net/forum?id=vkaMaq95_rX.

Madisetti, V. *The digital signal processing handbook*. CRC press, 1997.

Pan, Z., Chen, P., He, H., Liu, J., Cai, J., and Zhuang, B. Mesa: A memory-saving training framework for transformers. *arXiv preprint arXiv:2111.11124*, 2021.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

Rao, K. R. and Yip, P. *Discrete cosine transform: algorithms, advantages, applications*. Academic press, 2014.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

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

Sundararajan, D. *The discrete Fourier transform: theory, algorithms and applications*. World Scientific, 2001.

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

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.

Vanholder, H. Efficient inference with tensorrt. In *GPU Technology Conference*, volume 1, pp. 2, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, G., Bhat, Z. P., Jiang, Z., Chen, Y.-W., Zha, D., Reyes, A. C., Niktash, A., Ulkar, G., Okman, E., Cai, X., et al. Bed: A real-time object detection system for edge devices. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pp. 4994–4998, 2022a.

Wang, Z., Xu, Z., Wu, X., Shrivastava, A., and Ng, T. E. Dragonn: Distributed randomized approximate gradients of neural networks. In *International Conference on Machine Learning*, pp. 23274–23291. PMLR, 2022b.

Xu, Z., Chen, B., Li, C., Liu, W., Song, L., Lin, Y., and Shrivastava, A. Locality sensitive teaching. *Advances in Neural Information Processing Systems*, 34:18049–18062, 2021a.

Xu, Z., Song, Z., and Shrivastava, A. Breaking the linear iteration cost barrier for some well-known conditional gradient methods using maxip data-structures. *Advances in Neural Information Processing Systems*, 34:5576–5589, 2021b.

Xu, Z., Song, Z., and Shrivastava, A. A tale of two efficient value iteration algorithms for solving linear mdps with large action space. In *International Conference on Artificial Intelligence and Statistics*, pp. 788–836. PMLR, 2023.

Yuan, B., Jankov, D., Zou, J., Tang, Y., Bourgeois, D., and Jermaine, C. Tensor relational algebra for distributed machine learning system design. *Proc. VLDB Endow.*, 14(8):1338–1350, apr 2021. ISSN 2150-8097. doi: 10.14778/3457390.3457399. URL https://doi.org/10.14778/3457390.3457399.

Yuan, B., Wolfe, C. R., Dun, C., Tang, Y., Kyrillidis, A., and Jermaine, C. Distributed learning of fully connected neural networks using independent subnet training. 15, 2022.

Zha, D., Feng, L., Bhushanam, B., Choudhary, D., Nie, J., Tian, Y., Chae, J., Ma, Y., Kejariwal, A., and Hu, X. Autoshard: Automated embedding table sharding for recommender systems. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022a.

Zha, D., Feng, L., Tan, Q., Liu, Z., Lai, K.-H., Bhushanam, B., Tian, Y., Kejariwal, A., and Hu, X. Dreamshard: Generalizable embedding table placement for recommender systems. *arXiv preprint arXiv:2210.02023*, 2022b.

Zha, D., Feng, L., Luo, L., Bhushanam, B., Liu, Z., Hu, Y., Nie, J., Huang, Y., Tian, Y., Kejariwal, A., and Hu, X. Pre-train and search: Efficient embedding table sharding with pre-trained neural cost models. In *Conference on Machine Learning and Systems*, 2023.

Zhong, S., Zhang, G., Huang, N., and Xu, S. Revisit kernel pruning with lottery regulated grouped convolutions. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=LdEhiMG9WLO.

Appendix

A. 1D/3D-DCT for 1D/3D Activation maps

For 1D activation maps $\mathbf{H} \in \mathbb{R}^{N}$, the frequency-domain feature $\widetilde{\mathbf{H}} = \mathrm{DCT}(\mathbf{H})$ has the same shape of $N \times 1$; and each of the element $\tilde{h}_i$ is given by

$$\tilde{h}_i = \sum_{m=0}^{N-1} \cos\left[\frac{\pi}{N}\left(m+\frac{1}{2}\right)i\right],$$

where h_i, $0 \le i \le N-1$, are elements in the original matrix $\mathbf{H}$. For 3D activation maps $\mathbf{H} \in \mathbb{R}^{N \times N \times N}$, the frequency-domain feature $\widetilde{\mathbf{H}} = \mathrm{DCT}(\mathbf{H})$ has a shape of $N \times N \times N$; and each of the element $\tilde{h}_{i,j,k}$ is given by

$$\tilde{h}_{i,j,k} = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\sum_{t=0}^{N-1} h_{m,n,t} \cos\left[\frac{\pi}{N}\left(m+\frac{1}{2}\right)i\right] \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)j\right] \cos\left[\frac{\pi}{N}\left(t+\frac{1}{2}\right)k\right],$$

where $h_{m,n,t}$, $0 \le m, n, t \le N-1$, are elements in the original matrix $\mathbf{H}$. During the training of DNNs, the frequency-domain feature is estimated via operating 1D/3D-DCT for the vector/tensor in each channel according to the shape of the activation map. For 1D/3D activation maps, the LFC and HFC can be extracted given by

$$\mathbf{H}^{\mathsf{L}} = \mathrm{iDCT}(\widetilde{\mathbf{H}} \odot \mathbf{M}) \tag{11}$$

$$\mathbf{H}^{\mathsf{H}} = \mathrm{iDCT}(\widetilde{\mathbf{H}} \odot (\mathbf{1} - \mathbf{M})), \tag{12}$$

where $\mathrm{iDCT}(\cdot)$ denotes the inverse DCT. For 1D activation maps, $\mathbf{1}$ is N-dimensional vector; $\mathbf{M} = [m_i | 1 \le i \le N]$ denotes an N-dimensional low-pass mask satisfying $m_i = 1$ for $1 \le i \le W$ and $m_i = 0$ for other elements. For 3D activation maps, $\mathbf{1}$ is $N \times N \times N$ tensor; $\mathbf{M} = [m_{i,j,k} | 1 \le i, j, k \le N]$ denotes an $N \times N \times N$ low-pass mask satisfying $m_{i,j,k} = 1$ for $1 \le i, j, k \le W$ and $m_{i,j,k} = 0$ for other elements. $\mathbf{1} - \mathbf{M}$ indicates the high-pass mask.

B. Implementation Details of Section 3

We give the details of the experiment in Section 3. Without loss of generality, the experiment is conducted on the CIFAR-10 dataset using ResNet-18, DenseNet-121 and ShuffleNet-V2. During the backward propagation of normal training, the gradient of each layer l is estimated by

$$[\hat{\nabla}_{\mathbf{H}_{l-1}}, \hat{\nabla}_{\mathbf{W}_l}] = \mathrm{backward}(\hat{\nabla}_{\mathbf{H}_l}, \mathbf{H}_l, \mathbf{W}_l) \tag{13}$$

For LFC-ACT, the gradient is estimated by

$$[\hat{\nabla}_{\mathbf{H}_{l-1}}, \hat{\nabla}_{\mathbf{W}_l}] = \mathrm{backward}(\hat{\nabla}_{\mathbf{H}_l}, \mathbf{H}_l^{\mathsf{L}}, \mathbf{W}_l), \tag{14}$$

where HFC-ACT denotes the HFC of $\mathbf{H}_l$; for HFC-ACT, the gradient is estimated by

$$[\hat{\nabla}_{\mathbf{H}_{l-1}}, \hat{\nabla}_{\mathbf{W}_l}] = \mathrm{backward}(\hat{\nabla}_{\mathbf{H}_l}, \mathbf{H}_l^{\mathsf{H}}, \mathbf{W}_l), \tag{15}$$

where $\mathbf{H}_l^{\mathsf{H}}$ denotes the HFC of $\mathbf{H}_l$. Note that Equations (14) and (15) causes the distortion of backward propagation in LFC-ACT and HFC-ACT, respectively. The objective of this experiment is to investigate whether this distortion of backward propagation may be powerful enough to lead training to a non-optimal solution. The hyper-parameter setting of the training is given in Table 4.

Table 4: Hyper-parameters of the experiments in Section 3.

Architecture	ResNet-18	DenseNet-121	ShuffleNet-V2
Epoch	100	100	100
Batch-size	256	256	256
Initial LR	0.1	0.1	0.1
LR scheduler	Step LR	Step LR	Step LR
Weight-decay	0.0005	0.0005	0.0005
Optimizer	SGD	SGD	SGD
SGD Momentum	0.9	0.9	0.9
Ratio of LFC (W/N)	0.3	0.3	0.5

C. λ_l^L versus λ_l^H in DenseNet-121

The training of DNNs follows Table 4. For the estimation of LFC and HFC of activation maps, we take $W/N = 0.5$ for the low-pass $\mathbf{M}$ mask and achieves the results in Figure 4. To further study whether Theorem 3.1 holds for less W, we take $W/N = 0.1$ and $W/N = 0.2$, and achieves the values of λ_l^L and λ_l^H in the training (epoches 20, 40, and 60) of DenseNet-121 in Table 5, where λ_l^L and λ_l^H are estimated based on the input activation maps of the four DenseBlocks. It is consistently observed that $\lambda_l^L > \lambda_l^H$ for $W/N = 0.1$ and $W/N = 0.2$ in different training epochs. This indicates our proposed Theorem 3.1 holds without loss of generality.

D. Compression of 1D and 3D Activation Maps by DIVISION

We give more details about DIVISION considering 1D, 2D and 3D activation maps in this section.

D.1. Activation Map Compression

DIVISION adopts average pooling to estimate the LFC by $\mathbf{H}_l^{\mathsf{L}} = \mathrm{AveragePooling}(\mathbf{H}_l)$. The value of block-size and moving stride is a unified hyper-parameter B, which controls the memory of $\mathbf{H}_l^{\mathsf{L}}$. The average pooling of 1D, 2D and 3D activation maps are considered as follows,

$$\begin{gathered}
\mathrm{Minibatch} \times \mathrm{Channel} \times N \xrightarrow{\mathrm{AveragePooling1}D} \\
\mathrm{Minibatch} \times \mathrm{Channel} \times \lfloor N/B \rfloor \\
\mathrm{Minibatch} \times \mathrm{Channel} \times N \times N \xrightarrow{\mathrm{AveragePooling2}D} \\
\mathrm{Minibatch} \times \mathrm{Channel} \times \lfloor N/B \rfloor \times \lfloor N/B \rfloor \\
\mathrm{Minibatch} {\times} \mathrm{Channel} {\times} N {\times} N {\times} N \xrightarrow{\mathrm{AveragePooling3}D} \\
\mathrm{Minibatch} {\times} \mathrm{Channel} {\times} \lfloor N/B \rfloor {\times} \lfloor N/B \rfloor {\times} \lfloor N/B \rfloor
\end{gathered} \tag{16}$$

To estimate the HFC, DIVISION calculates the resid-

Table 5: Ratio of λ_l^L to λ_l^H on DensNet-121. (a) $W/N = 0.1$ and (b) $W/N = 0.2$.

(a) $W/N = 0.1$

Epoch	20	40	60	Average λ_l^L/λ_l^H
DenseBlock-1	$\lambda_l^L = 298.281$ $\lambda_l^H = 218.605$	$\lambda_l^L = 184.913$ $\lambda_l^H = 138.069$	$\lambda_l^L = 142.668$ $\lambda_l^H = 104.755$	1.36
DenseBlock-2	$\lambda_l^L = 3.245$ $\lambda_l^H = 1.713$	$\lambda_l^L = 1.284$ $\lambda_l^H = 0.689$	$\lambda_l^L = 0.687$ $\lambda_l^H = 0.372$	1.87
DenseBlock-3	$\lambda_l^L = 0.387$ $\lambda_l^H = 0.260$	$\lambda_l^L = 0.160$ $\lambda_l^H = 0.086$	$\lambda_l^L = 0.084$ $\lambda_l^H = 0.048$	1.70
DenseBlock-4	$\lambda_l^L = 0.062$ $\lambda_l^H = 0.009$	$\lambda_l^L = 0.011$ $\lambda_l^H = 0.001$	$\lambda_l^L = 0.006$ $\lambda_l^H = 0.001$	7.56
Average λ_l^L/λ_l^H	2.95	3.12	3.30	3.12

(b) $W/N = 0.2$

Epoch	20	40	60	Average λ_l^L/λ_l^H
DenseBlock-1	$\lambda_l^L = 362.672$ $\lambda_l^H = 154.214$	$\lambda_l^L = 225.543$ $\lambda_l^H = 97.439$	$\lambda_l^L = 173.595$ $\lambda_l^H = 73.828$	2.34
DenseBlock-2	$\lambda_l^L = 3.632$ $\lambda_l^H = 1.326$	$\lambda_l^L = 1.440$ $\lambda_l^H = 0.533$	$\lambda_l^L = 0.774$ $\lambda_l^H = 0.285$	2.72
DenseBlock-3	$\lambda_l^L = 0.445$ $\lambda_l^H = 0.202$	$\lambda_l^L = 0.179$ $\lambda_l^H = 0.067$	$\lambda_l^L = 0.095$ $\lambda_l^H = 0.037$	2.49
DenseBlock-4	$\lambda_l^L = 0.062$ $\lambda_l^H = 0.009$	$\lambda_l^L = 0.011$ $\lambda_l^H = 0.001$	$\lambda_l^L = 0.006$ $\lambda_l^H = 0.001$	7.56
Average λ_l^L/λ_l^H	3.58	3.78	3.97	3.78

ual value $\mathbf{H}_l^{\mathsf{H}} = \mathbf{H}_l - \mathrm{UpSampling}(\mathbf{H}_l^{\mathsf{L}})$, where the $\mathrm{UpSampling}(\cdot)$ enlarges $\mathbf{H}_l^{\mathsf{L}}$ to the shape of $\mathbf{H}_l$ via nearest interpolation. The up sampling of 1D, 2D and 3D activation maps are considered as follows,

$$\begin{aligned}
&\mathrm{Minibatch}{\times}\mathrm{Channel}\times\lfloor N/B\rfloor \\
&\qquad \xrightarrow{\mathrm{UpSampling}1D} \mathrm{Minibatch}\times\mathrm{Channel}\times N \\
&\mathrm{Minibatch}{\times}\mathrm{Channel}\times\lfloor N/B\rfloor\times\lfloor N/B\rfloor \qquad (17)\\
&\qquad \xrightarrow{\mathrm{UpSampling}2D} \mathrm{Minibatch}\times\mathrm{Channel}\times N\times N \\
&\mathrm{Minibatch}{\times}\mathrm{Channel}{\times}\lfloor N/B\rfloor{\times}\lfloor N/B\rfloor{\times}\lfloor N/B\rfloor \\
&\qquad \xrightarrow{\mathrm{UpSampling}3D} \mathrm{Minibatch}\times\mathrm{Channel}{\times}N{\times}N{\times}N
\end{aligned}$$

Then, DIVISION adopts Q-bit per-channel quantization for the compression, where the bit-width Q controls the precision and memory cost of HFC after the compression. Let $\mathbf{V}_l^{\mathsf{H}}$ denote a Q-bit integer matrix, as the low-precision representation of $\mathbf{H}_l^{\mathsf{H}}$. The detailed procedure of compressing $\mathbf{H}_l^{\mathsf{H}}$ into $\mathbf{V}_l^{\mathsf{H}}$ is given by

$$\mathbf{V}_l^{\mathsf{H}} = \mathrm{Quant}(\mathbf{H}_l^{\mathsf{H}}) = \lfloor \Delta_l^{-1}(\mathbf{H}_l^{\mathsf{H}} - \delta_l) \rceil, \qquad (18)$$

where δ_l denotes the minimum element in $\mathbf{H}_l^{\mathsf{H}}$; $\Delta_l = (h_{\max} - \delta_l)/(2^Q - 1)$ denotes the quantization step; $h_{\max}$ denotes the maximum element in $\mathbf{H}_l^{\mathsf{H}}$; $\lfloor \bullet \rceil$ denotes the stochastic rounding.

After the compression, as the representation of $\mathbf{H}_l$, the tuple of $(\mathbf{H}_l^{\mathsf{L}}, \mathbf{V}_l^{\mathsf{H}}, \Delta_l, \delta_l)$ is cached to the memory for reconstructing the activation maps during the backward pass.

D.2. Activation Map Decompression

During the backward pass, DIVISION adopts the cached tuples of $\{(\mathbf{H}_l^{\mathsf{L}}, \mathbf{V}_l^{\mathsf{H}}, \Delta_l, \delta_l) \,|\, 0 \le l \le L{-}1\}$ to reconstruct the activation map layer-by-layer. Specifically, for each layer l, DIVISION dequantizes the HFC via $\hat{\mathbf{H}}_l^{\mathsf{H}} = \Delta_l \mathbf{V}_l^{\mathsf{H}} + \delta_l$, which is the inverse process of Equation (18). Then, the activation map is reconstructed via $\hat{\mathbf{H}}_l = \mathrm{UpSampling}(\mathbf{H}_l^{\mathsf{L}}) + \hat{\mathbf{H}}_l^{\mathsf{H}}$, where $\mathrm{UpSampling}(\cdot)$ enlarges $\mathbf{H}_l^{\mathsf{L}}$ to the shape of $\mathbf{H}_l$ via nearest interpolation. The cases of 1D, 2D and 3D activation maps are considered in Equation (17).

After the decompression, DIVISION frees the caching of $(\mathbf{H}_l^{\mathsf{L}}, \mathbf{V}_l^{\mathsf{H}}, \Delta_l, \delta_l)$, and takes $\hat{\mathbf{H}}_l$ into $[\hat{\nabla}_{\mathbf{H}_{l-1}}, \hat{\nabla}_{\mathbf{W}_l}] = \mathrm{backward}(\hat{\nabla}_{\mathbf{H}_l}, \hat{\mathbf{H}}_{l-1}, \mathbf{W}_l)$ to estimate the gradient for backward propagation.

E. Experiment Setting

We give the experiment setting including the datasets, baseline methods and model architectures in this section.

Datasets. We consider CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009) and ImageNet (Deng et al., 2009) datasets in our experiments. **CIFAR-10:** An image dataset with 60,000 color images in 10 different classes, where each image has 32×32 pixels. **CIFAR-100:** An image dataset with 60,000 color images in 100 different classes, where each image has 32×32 pixels. **ImageNet:** A large scale image dataset which has over one million color images covering 1000 categories, where each image has 224×224 pixels.

Baseline Methods. Normal: Caching the exact activation map for backward propagation. **BLPA:** A systemic implementation of ACT by (Chakrabarti & Moseley, 2019), which only supports ResNet-related architectures. **AC-GC:** A framework of ACT with automatic searched bit-width for the quantization of activation maps (Evans & Aamodt, 2021). **ActNN:** Activation compression training with dynamic bit-width quantization, where the bit-allocation minimizes the variance of activation maps via dynamic processing (Chen et al., 2021). **Mesa:** Mesa is an ACT-based memory-efficient training method explicitly designed for vision transformers (Pan et al., 2021). **Checkpoint:** Caching some key activation maps to reconstruct other activation maps via replaying parts of the forward pass during the backward pass (Chen et al., 2016). **SWAP:** Swapping the activation maps to the CPU during the forward pass the memory consumption of GPU, and reload the activation maps to GPU during the backward pass (Huang et al., 2020).

DNN Architectures. For benchmarking the model accuracy, we consider ResNet-18 (top-1 accuracy 94.89%), ResNet-164 (top-1 accuracy 94.9%), and MobileNet-V2 (top-1 accuracy 91.9%) on the CIFAR-10 dataset; ResNet-34 (top-1 accuracy 77.1%), DenseNet-121 (top-1 accuracy 79.75%), ResNet-164 (top-1 accuracy 77.3%), and MobileNet-V2 (top-1 accuracy 71.0%) on the CIFAR-100 dataset; and ResNet-50 (top-1 accuracy 76.15%), DenseNet-161 (top-1 accuracy 77.65%) and Swin Transformer-T (top-1 accuracy 81.2%) on the ImageNet dataset. Our reproduced validating accuracy on the ImageNet dataset is consistent with the official results of torchvision[11]. Moreover, for benchmarking the memory cost and training throughput, we consider the large models ResNet-50 and WRN-50-2 on the ImageNet dataset. To study the performance of DIVISION on depthwise and pointwise convolutional Layers, we consider MobileNet-V2. The comprehensive comparison in Figure 1 considers ResNet-50 and ResNet-34 on the ImageNet and CIFAR-100 datasets, respectively.

F. Implementation details about DIVISION and Baseline Methods

DIVISION: DIVISION adopts block-size 8 ($B = 8$) and 2-bit quantization ($Q = 2$) to compress the activation maps of linear, convolutional and BatchNorm layers, where the theoretical compression rate is not less than $10.35\times$. For the operators without quantization error during backward propagation such as pooling layers, ReLu activation, and Dropout, DIVISION follows the algorithms in Appendix G to compress the activation maps. Other hyper-parameter settings are given in Table 10. **BLPA:** Existing work (Chakrabarti & Moseley, 2019) has shown that BLPA requires at least 4-bit ACT for loss-less DNN training. We follow this setting for BLPA, where the compression rate of activation maps is not more than $8\times$. **AC-GC:** AC-GC follows existing work (Evans & Aamodt, 2021) to take the multiplicative error $(1 + e^2_{\text{AC-GC}}) = 1.5$, where the searched bit-width enables AC-GC to satisfy this loss bound (training loss not more than 150% of normal training). In this setting, AC-GC finalizes the bit-with as 7.01 after the searching, which has a nearly $3.5\times$ compression rate of activation maps. **ActNN:** ActNN adopts 2-bit ACT and dynamic programming for searching the optimal bit-width specific for each layer, and uses per-group quantization for compressing the activation map, which has approximately $10.5\times$ compression rate of activation maps. Such experimentally setting is denoted as L3 strategy in the original work (Chen et al., 2021), and we follow this setting in this section. **Mesa:** We follow the default setting in the original work of Mesa (Pan et al., 2021). **Checkpoint:** Checkpoint relies on a manually design of the checkpointing layers. We follow the checkpoint strategy of Megatron-LM in our experiment (Shoeybi et al., 2019). Specifically, Megatron-LM checkpoints the activation map after each transformer block. We follow this strategy to checkpoint the activation map after each transformer block in the Swin Transformer, and after each Bottleneck block in the ResNet-50. For all methods, the memory cost is measured in a single mini-batch updating; and the throughput is estimated by averaging that of 20 mini-batch updating to achieve a stable result.

G. Compression of Pooling layers, Relu activations, and Dropout

DIVISION follows Algorithms 2, 3, 4 and 5 to compresse the activation map of a Max-Pooling layer, Average-Pooling layer, Relu activation and Dropout operator, respectively. For the pooling layers, we consider a simple case $\mathrm{kernelsize} = \mathrm{movingstride} = k$. General cases with different $\mathrm{kernelsize}$ and $\mathrm{movingstride}$ can be designed in analogous ways.

Algorithm 2 Max-Pooling layer.

```
Function Forward ($\mathbf{H}_{l-1}$, $k$, **kwargs)
    $\mathbf{H}_l, \mathbf{V}_{l-1}{}^{11}$ =Max-Pooling($\mathbf{H}_{l-1}$, $k$, kwargs)
    Pack & Cache $\mathbf{V}_{l-1}$ using Int8.
    return $\mathbf{H}_l$

Function Backward($\nabla_{\mathbf{H}_l}$)
    Load $\mathbf{V}_{l-1}$ and $k$.
    $\nabla'_{\mathbf{H}_l} = \mathbf{1}_{k\times k} \otimes \nabla_{\mathbf{H}_l}$
    $\nabla_{\mathbf{H}_{l-1}} = \mathbf{V}_{l-1} \odot \nabla'_{\mathbf{H}_l}$
    return $\nabla_{\mathbf{H}_{l-1}}$
```

Algorithm 3 Average-Pooling layer.

```
Function Forward ($\mathbf{H}_{l-1}$, $k$, **kwargs)
    $\mathbf{H}_l$ =Avg-Pooling($\mathbf{H}_{l-1}$, $k$, kwargs)
    return $\mathbf{H}_l$

Function Backward($\nabla_{\mathbf{H}_l}$)
    $\nabla'_{\mathbf{H}_l} = \mathbf{1}_{k\times k} \otimes \nabla_{\mathbf{H}_l}$
    $\nabla_{\mathbf{H}_{l-1}} = k^{-2}\nabla_{\mathbf{H}_l}$
    return $\nabla_{\mathbf{H}_{l-1}}$
```

H. Evaluation of DIVISION on Multi-layer Perceptrons (MLPs)

We conduct experiments on the GAS dataset (Dua & Graff, 2017) (128-dimensional features, 13910 instances, 6 classification task). The classification model is a 4-layer MLP (128 neuros in the input layer, 6 neuros in the output layer, and 64 neuros in the hidden layer); The setting of DIVISION is $B = 16$ and $Q = 2$. The model accuracy and memory cost of activation map are given in Table 6. It is observed that DIVISION has $7.3\times$ compression rate with

[11] https://paperswithcode.com/lib/torchvision

Table 6: Model Accuracy on the GAS dataset.

Training	Testing Accuracy (%)	Memory (KB)	Compression
Normal Training	98.92	250	N/A
DIVISION	98.85	34.2	7.3×

Table 7: Evaluation results on GLUE tasks.

Dataset	Standard Evaluation Metric	Normal	DIVISION
CoLA	Matthew's Correlation	60.3	60.6
SST2	Accuracy	93.9	94.6
MRPC	F1 score	91.4	91.9
STS-B	Pearson-Spearman correlation	90.7	90.4

only 0.07% degradation of model accuracy. This indicates the effectiveness of DIVISION on the MLP models.

I. Evaluation of DIVISION on NLP tasks

To evaluate DIVISION on the tasks of natural language processing, we conducted experiments of deploying DIVISION (with $B = 8$ and $Q = 4$) to the T5-Base (Raffel et al., 2020) language model on the CoLA, SST2, MRPC, and STS-B datasets. The input text is padded to a maximum length of 128 during the training. The results of evaluation metrics on different datasets are shown in the Table 7. It is observed that DIVISION achieves nearly loss-less model accuracy compared with normal training. The memory cost considers that of model weight and activation maps, and DIVISION achieves over 4.2× compression rate on all of the datasets, further emphasizing its effectiveness on language models.

Table 8: Comparison of DIVISION with TinyScript.

Top-1	Error	Top-5 Error	Compression Rate
TinyScript n=8	N/A	7.74	9.8×
DIVISION	24.1	7.33	10.4×

J. Comparison with TinyScript

In this section, we conducted a comparison between DIVISION and TinyScript (Fu et al., 2020) for training ResNet-50 on the ImageNet dataset. The model accuracy and compression rate are given in Table 8. It is observed that DIVISION outperforms TinyScript, achieving a higher compression rate with lower Top-5 error. These results demonstrate the superiority of DIVISION over TinyScript.

K. Computation Infrastructure

The details about our physical computing infrastructure for testing the training memory cost and throughput are given in Table 9.

Algorithm 4 Relu operator.

```
Function Forward ($\mathbf{H}_{l-1}$)
    $\mathbf{V}_{l-1} = \mathrm{sgn}(\mathbf{H}_{l-1})$
    $\mathbf{H}_l = \mathbf{V}_{l-1} \odot \mathbf{H}_{l-1}$
    Pack & Cache $\mathbf{V}_{l-1}$ using Int8
    return $\mathbf{H}_l$


Function Backward($\nabla_{\mathbf{H}_l}$)
    Load $\mathbf{V}_{l-1}$.
    $\nabla_{\mathbf{H}_{l-1}} = \mathbf{V}_{l-1} \odot \nabla_{\mathbf{H}_l}$
    return $\nabla_{\mathbf{H}_{l-1}}$
```

Algorithm 5 Dropout operator.

```
Function Forward ($\mathbf{H}_{l-1}$)
    Generate a $\mathrm{Minibatch} \times \mathrm{Channel} \times N \times N$ binary matrix
    $\mathbf{V}_{l-1}$ following the Bernoulli distribution with dropout
    probability $p$.
    $\mathbf{H}_l = \mathbf{V}_{l-1} \odot \mathbf{H}_{l-1}$
    Pack & Cache $\mathbf{V}_{l-1}$ using Int8.
    return $\mathbf{H}_l$

Function Backward($\nabla_{\mathbf{H}_l}$)
    Load $\mathbf{V}_{l-1}$.
    $\nabla_{\mathbf{H}_{l-1}} = \mathbf{V}_{l-1} \odot \nabla_{\mathbf{H}_l}$
    return $\nabla_{\mathbf{H}_{l-1}}$
```

L. Implementation Details of the Experiment in Section 5.7

We give the implementation details of the experiment in Section 5.7. The model of image classification on the CIFAR-100 and ImageNet datasets are ResNet-34 and ResNet-50, respectively. Regarding to the training methods, *DIVISION w/o HFC* takes block-size $B = 4$ for estimating LFC; *DIVISION w/o LFC* takes the bit-width $Q = 2$ for the quantization of HFC; DIVISION combines these settings for the training; and *Fixed Quant* adopts 4-bit and 2-bit per-group quantization to compress the activation maps during the training, where the group size of quantization follows existing work (Chen et al., 2021) to be 256. Other training hyper-parameters are given in Table 10.

Table 9: Computing infrastructure for the experiments.

Device Attribute	Value
Computing infrastructure	GPU
GPU model	Nvidia-RTX3090
GPU number	1
CUDA Version	12.0

[11] $\mathbf{V}_{l-1}$ reserves the locations of each kernel-wise max-values in $\mathbf{H}_{l-1}$.

Table 10: Hyper-parameter setting.

Dataset	CIFAR-10		CIFAR-100			ImageNet				
Architecture	ResNet-18	ResNet-164	MobileNet-V2	ResNet-34	ResNet-164	DenseNet-121	MobileNet-V2	ResNet-50	DenseNet-161	Swin-T
Epoch	100	100	100	100	200	100	100	120	120	300
Batch-size	256	256	256	256	256	256	256	256	256	128
Initial LR	0.1	0.1	0.1	0.1	0.15	0.1	0.1	0.1	0.1	5e-4
LR scheduler	Cos LR	Cos LR	Cos LR	Cos LR	Cos LR	Cos LR	Cos LR	Cos LR	Cos LR	Cos LR
Weight-decay	0.0005	0.0005	0.00004	0.0005	0.0005	0.0005	0.00004	0.0001	0.0001	0.05
Optimizer	SGD	SGD	SGD	SGD	SGD	SGD	SGD	SGD	SGD	SGD
SGD Momentum	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
Block-size B	8	8	8	8	8	8	8	8	8	64
Bit-width Q	2	2	2	2	2	2	2	2	2	2

M. Proof of Theorem 3.1

We prove Theorem 3.1 in this section.

Theorem 1 *During the backward pass of a convolutional layer l, $\mathrm{GEB}_l^{\mathsf{L}}$ and $\mathrm{GEB}_l^{\mathsf{H}}$ satisfy*

$$\mathrm{GEB}_l^{\mathsf{L}} - \mathrm{GEB}_l^{\mathsf{H}} = \Big(\alpha_{l,l}||\mathbf{H}_{l-1}^{\mathsf{T}}||_F + \beta_l\Big)(\lambda_l^{\mathsf{H}} - \lambda_l^{\mathsf{L}}) + ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F \sum_{i=l+1}^{L} \alpha_{l,i}(\lambda_i^{\mathsf{H}} - \lambda_i^{\mathsf{L}}) \prod_{j=l}^{i-1} \gamma_j, \tag{19}$$

where $\alpha_{l,i}, \beta_l, \gamma_l > 0$ for $1 \leq l, i \leq L$ are given by Equation (28); $\lambda_l^{\mathsf{L}} = ||\widetilde{\mathbf{H}}_l \odot \mathbf{M}||_F$; $\lambda_l^{\mathsf{H}} = ||\widetilde{\mathbf{H}}_l \odot (\mathbf{1} – \mathbf{M})||_F$; $\widetilde{\mathbf{H}}_l = \mathrm{DCT}(\mathbf{H}_l)$; and $\mathbf{M}$ denotes the loss-pass mask given by Equation (4).

Proof. For simplicity of derivation, we study the case with a single input channel and output channel number. In this case, $\mathbf{H}_l$ and $\mathbf{W}_l$ are 2-D matrix for each layer l, where $1 \leq l \leq L$. The backward propagation of a convolutional layer is given by

$$\begin{aligned} \hat{\nabla}_{\mathbf{Z}_l} &= \hat{\nabla}_{\mathbf{Z}_{l+1}} * \mathbf{W}_{l+1}^{\mathsf{rot}} \odot \sigma'(\hat{\mathbf{Z}}_l), \\ \hat{\nabla}_{\mathbf{W}_l} &= \hat{\nabla}_{\mathbf{Z}_l} * \hat{\mathbf{H}}_{l-1}^{\mathsf{T}}, \end{aligned} \tag{20}$$

where $*$ denotes a convolutional operation; $\hat{\mathbf{Z}}_l = \mathbf{W}_l * \hat{\mathbf{H}}_{l-1} + b_l$; b_l denotes the bias of layer l; and $\mathbf{W}_l^{\mathsf{rot}}$ denotes to rotate $\mathbf{W}_l$ by 180°. The case of multiple input and output channels can be proved in an analogous way, which is omitted in this work.

According to Equation (20), we have the gradient of $\mathbf{Z}_l$ given by

$$\begin{aligned} &\hat{\nabla}_{\mathbf{Z}_l} - \nabla_{\mathbf{Z}_l} \\ &= \hat{\nabla}_{\mathbf{Z}_{l+1}} * \mathbf{W}_{l+1}^{\mathsf{rot}} \odot \sigma'(\hat{\mathbf{Z}}_l) - \nabla_{\mathbf{Z}_{l+1}} * \mathbf{W}_{l+1}^{\mathsf{rot}} \odot \sigma'(\mathbf{Z}_l), \\ &= \hat{\nabla}_{\mathbf{Z}_{l+1}} * \mathbf{W}_{l+1}^{\mathsf{rot}} \odot \sigma'(\hat{\mathbf{Z}}_l) - \hat{\nabla}_{\mathbf{Z}_{l+1}} * \mathbf{W}_{l+1}^{\mathsf{rot}} \odot \sigma'(\mathbf{Z}_l) + \hat{\nabla}_{\mathbf{Z}_{l+1}} * \mathbf{W}_{l+1}^{\mathsf{rot}} \odot \sigma'(\mathbf{Z}_l) - \nabla_{\mathbf{Z}_{l+1}} * \mathbf{W}_{l+1}^{\mathsf{rot}} \odot \sigma'(\mathbf{Z}_l), \\ &= \hat{\nabla}_{\mathbf{Z}_{l+1}} * \mathbf{W}_{l+1}^{\mathsf{rot}} \odot [\sigma'(\hat{\mathbf{Z}}_l) - \sigma'(\mathbf{Z}_l)] + (\hat{\nabla}_{\mathbf{Z}_{l+1}} - \nabla_{\mathbf{Z}_{l+1}}) * \mathbf{W}_{l+1}^{\mathsf{rot}} \odot \sigma'(\mathbf{Z}_l). \end{aligned} \tag{21}$$

For the activation functions $\mathrm{ReLu}(\cdot)$, $\mathrm{LeakyReLu}(\cdot)$, $\mathrm{Sigmoid}(\cdot)$, $\mathrm{Tanh}(\cdot)$ and $\mathrm{SoftPlus}(\cdot)$, the gradient $\sigma'(\cdot)$ satisfies $|\sigma''(\cdot)| \leq 1$ in the differentable domains. Note that we have $||\mathbf{W}_l * \mathbf{H}_{l-1}||_F \leq (K_l + N_l - 1)||\mathbf{W}_l||_F||\mathbf{H}_{l-1}||_F$ according to Corollary M.1. $||\sigma'(\hat{\mathbf{Z}}_l) - \sigma'(\mathbf{Z}_l)||_F$ satisfies

$$||\sigma'(\hat{\mathbf{Z}}_l) - \sigma'(\mathbf{Z}_l)||_F \leq ||\hat{\mathbf{Z}}_l - \mathbf{Z}_l||_F \leq (K_l + N_l - 1)||\hat{\mathbf{H}}_{l-1} - \mathbf{H}_{l-1}||_F||\mathbf{W}_l'||_F, \tag{22}$$

where K_l and N_l denote the size of convolutional kernel $\mathbf{W}_l$ and activation map $\mathbf{H}_l$ in layer l, respectively. After taking Equation (22) into Equation (21), we have

$$\begin{aligned} ||\hat{\nabla}_{\mathbf{Z}_l} - \nabla_{\mathbf{Z}_l}||_F \leq\ & (K_l + N_l - 1)||\hat{\nabla}_{\mathbf{Z}_{l+1}}||_F||\mathbf{W}_{l+1}^{\mathsf{rot}}||_F||\sigma'(\hat{\mathbf{Z}}_l) - \sigma'(\mathbf{Z}_l)||_F \\ &+ (K_l + N_l - 1)||\hat{\nabla}_{\mathbf{Z}_{l+1}} - \nabla_{\mathbf{Z}_{l+1}}||_F||\mathbf{W}_{l+1}^{\mathsf{rot}}||_F||\sigma'(\mathbf{Z}_l)||_F, \\ &= (K_l + N_l - 1)^2||\hat{\nabla}_{\mathbf{Z}_{l+1}}||_F||\mathbf{W}_{l+1}^{\mathsf{rot}}||_F||\hat{\mathbf{H}}_{l-1} - \mathbf{H}_{l-1}||_F||\mathbf{W}_l'||_F \\ &+ (K_l + N_l - 1)||\hat{\nabla}_{\mathbf{Z}_{l+1}} - \nabla_{\mathbf{Z}_{l+1}}||_F||\mathbf{W}_{l+1}^{\mathsf{rot}}||_F||\sigma'(\mathbf{Z}_l)||_F, \\ &= \eta_l||\hat{\mathbf{H}}_{l-1} - \mathbf{H}_{l-1}||_F + \gamma_l||\hat{\nabla}_{\mathbf{Z}_{l+1}} - \nabla_{\mathbf{Z}_{l+1}}||_F, \end{aligned} \tag{23}$$

where η_l and γ_l are given by

$$\begin{aligned} \eta_l &= (K_l + N_l - 1)^2 ||\hat{\nabla}_{\mathbf{Z}_{l+1}}||_F ||\mathbf{W}_{l+1}||_F ||\mathbf{W}'_l||_F; \\ \gamma_l &= (K_l + N_l - 1) ||\mathbf{W}_{l+1}||_F ||\sigma'(\mathbf{Z}_l)||_F; \end{aligned} \tag{24}$$

the value η_l and γ_l depend on the model weight before backward propagation, which is constant with respect to the gradient. Iterate Equation (23) until $l = L$ where $||\hat{\nabla}_{\mathbf{Z}_L} - \nabla_{\mathbf{Z}_L}||_F \leq \eta_L ||\hat{\mathbf{H}}_{L-1} - \mathbf{H}_{L-1}||_F$. In this way, we have

$$||\hat{\nabla}_{\mathbf{Z}_l} - \nabla_{\mathbf{Z}_l}||_F \leq \eta_l ||\hat{\mathbf{H}}_{l-1} - \mathbf{H}_{l-1}||_F + \sum_{i=l+1}^{L} \eta_i ||\hat{\mathbf{H}}_{i-1} - \mathbf{H}_{i-1}||_F \prod_{j=l}^{i-1} \gamma_j. \tag{25}$$

According to Equation (20), we have the gradient of $\mathbf{W}_l$ given by

$$\begin{aligned} \hat{\nabla}_{\mathbf{W}_l} - \nabla_{\mathbf{W}_l} &= \hat{\nabla}_{\mathbf{Z}_l} * \hat{\mathbf{H}}_{l-1}^{\mathsf{T}} - \nabla_{\mathbf{Z}_l} * \mathbf{H}_{l-1}^{\mathsf{T}}, \\ &= \hat{\nabla}_{\mathbf{Z}_l} * \hat{\mathbf{H}}_{l-1}^{\mathsf{T}} - \hat{\nabla}_{\mathbf{Z}_l} * \mathbf{H}_{l-1}^{\mathsf{T}} + \hat{\nabla}_{\mathbf{Z}_l} * \mathbf{H}_{l-1}^{\mathsf{T}} - \nabla_{\mathbf{Z}_l} * \mathbf{H}_{l-1}^{\mathsf{T}}, \\ &= \hat{\nabla}_{\mathbf{Z}_l} * (\hat{\mathbf{H}}_{l-1}^{\mathsf{T}} - \mathbf{H}_{l-1}^{\mathsf{T}}) + (\hat{\nabla}_{\mathbf{Z}_l} - \nabla_{\mathbf{Z}_l}) * \mathbf{H}_{l-1}^{\mathsf{T}}. \end{aligned} \tag{26}$$

Taking Equation (25) into Equation (26), we have

$$\begin{aligned} &||\hat{\nabla}_{\mathbf{W}_l} - \nabla_{\mathbf{W}_l}||_F \\ &\leq (K_l + N_l - 1) ||\hat{\nabla}_{\mathbf{Z}_l}||_F ||\hat{\mathbf{H}}_{l-1}^{\mathsf{T}} - \mathbf{H}_{l-1}^{\mathsf{T}}||_F + (K_l + N_l - 1) ||\hat{\nabla}_{\mathbf{Z}_l} - \nabla_{\mathbf{Z}_l}||_F ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F, \\ &\leq (K_l + N_l - 1) \Bigg(||\hat{\nabla}_{\mathbf{Z}_l}||_F ||\hat{\mathbf{H}}_{l-1}^{\mathsf{T}} - \mathbf{H}_{l-1}^{\mathsf{T}}||_F + ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F \Big[\eta_l ||\hat{\mathbf{H}}_{l-1} - \mathbf{H}_{l-1}||_F + \sum_{i=l+1}^{L} \eta_i ||\hat{\mathbf{H}}_{i-1} - \mathbf{H}_{i-1}||_F \prod_{j=l}^{i-1} \gamma_j \Big] \Bigg) \\ &= (K_l + N_l - 1) \Big[\big(||\hat{\nabla}_{\mathbf{Z}_l}||_F + \eta_l ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F \big) ||\hat{\mathbf{H}}_{l-1}^{\mathsf{T}} - \mathbf{H}_{l-1}^{\mathsf{T}}||_F + ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F \sum_{i=l+1}^{L} \eta_i ||\hat{\mathbf{H}}_{i-1} - \mathbf{H}_{i-1}||_F \prod_{j=l}^{i-1} \gamma_j \Big], \\ &= \Big(\beta_l + \alpha_{l,l} ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F \Big) ||\hat{\mathbf{H}}_{l-1}^{\mathsf{T}} - \mathbf{H}_{l-1}^{\mathsf{T}}||_F + ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F \sum_{i=l+1}^{L} \alpha_{l,i} ||\hat{\mathbf{H}}_{i-1}^{\mathsf{T}} - \mathbf{H}_{i-1}^{\mathsf{T}}||_F \prod_{j=l}^{i-1} \gamma_j, \end{aligned} \tag{27}$$

where

$$\begin{aligned} \alpha_{l,i} &= (K_l + N_l - 1)(K_i + N_i - 1)^2 ||\hat{\nabla}_{\mathbf{Z}_{i+1}}||_F ||\mathbf{W}_{i+1}||_F ||\mathbf{W}'_i||_F; \\ \beta_l &= (K_l + N_l - 1) ||\hat{\nabla}_{\mathbf{Z}_l}||_F; \\ \gamma_l &= (K_l + N_l - 1) ||\mathbf{W}_{l+1}||_F ||\sigma'(\mathbf{Z}_l)||_F; \end{aligned} \tag{28}$$

K_l and N_l denote the size of convolutional kernel $\mathbf{W}_l$ and activation map $\mathbf{H}_l$ in layer l, respectively.

During the LFC-ACT and HFC-ACT trainings, the activation map of a convolutional layer satisfies

$$||\mathbf{H}_l - \mathbf{H}_l^{\mathsf{L}}||_F = ||\widetilde{\mathbf{H}}_l - \widetilde{\mathbf{H}}_l^{\mathsf{L}}||_F = ||\widetilde{\mathbf{H}}_l \odot (\mathbf{1} - \mathbf{M})||_F \triangleq \lambda_l^{\mathsf{H}}, \tag{29}$$

$$||\mathbf{H}_l - \mathbf{H}_l^{\mathsf{H}}||_F = ||\widetilde{\mathbf{H}}_l - \widetilde{\mathbf{H}}_l^{\mathsf{H}}||_F = ||\widetilde{\mathbf{H}}_l \odot \mathbf{M}||_F \triangleq \lambda_l^{\mathsf{L}}. \tag{30}$$

Taking Equations (29) and (30) into (27), we have $\mathrm{GEB}_l^{\mathsf{L}}$ and $\mathrm{GEB}_l^{\mathsf{H}}$ of a convolutional layer by

$$||\hat{\nabla}_{\mathbf{W}_l} - \nabla_{\mathbf{W}_l}^{\mathsf{L}}||_F \leq \Big(\alpha_{l,l} ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F + \beta_l \Big) \lambda_l^{\mathsf{H}} + ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F \sum_{i=l+1}^{L} \alpha_{l,i} \lambda_i^{\mathsf{H}} \prod_{j=l}^{i-1} \gamma_j \triangleq \mathrm{GEB}_l^{\mathsf{L}}, \tag{31}$$

$$||\hat{\nabla}_{\mathbf{W}_l} - \nabla_{\mathbf{W}_l}^{\mathsf{H}}||_F \leq \Big(\alpha_{l,l} ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F + \beta_l \Big) \lambda_l^{\mathsf{L}} + ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F \sum_{i=l+1}^{L} \alpha_{l,i} \lambda_i^{\mathsf{L}} \prod_{j=l}^{i-1} \gamma_j \triangleq \mathrm{GEB}_l^{\mathsf{H}}. \tag{32}$$

Given the expression of $\mathrm{GEB}_l^{\mathsf{L}}$ and $\mathrm{GEB}_l^{\mathsf{H}}$ by Equations (31) and (32), respectively, we have the GEB for a convolutional layer given by

$$\mathrm{GEB}_l^{\mathsf{L}} - \mathrm{GEB}_l^{\mathsf{H}} = \Big(\alpha_{l,l} ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F + \beta_l \Big) (\lambda_l^{\mathsf{H}} - \lambda_l^{\mathsf{L}}) + ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F \sum_{i=l+1}^{L} \alpha_{l,i} (\lambda_i^{\mathsf{H}} - \lambda_i^{\mathsf{L}}) \prod_{j=l}^{i-1} \gamma_j.$$

□

Corollary M.1. *For a $K \times K$ convolutional kernel and a $N \times N$ square matrix $\mathbf{H}$, we have the*

$$||\mathbf{W} * \mathbf{H}||_F \leq (K+N-1)||\mathbf{W}||_F||\mathbf{H}||_F \tag{33}$$

Proof. According to the relations between convolutional operation and Discrete Fourier Transformation (Sundararajan, 2001), $\mathbf{W} * \mathbf{H}$ satisfies

$$\mathrm{FFT}(\mathbf{W} * \mathbf{H}) = \mathrm{FFT}(\mathrm{ZP}(\mathbf{W})) \odot \mathrm{FFT}(\mathrm{ZP}(\mathbf{H})), \tag{34}$$

where $\mathrm{FFT}(\cdot)$ denotes the discrete Fourier transformation; $\mathrm{ZP}(\mathbf{W})$ denotes zero-padding $\mathbf{W}$ into a $(K{+}N{-}1){\times}(K{+}N{-}1)$ matrix. According to the Parseval's theorem (Diniz et al., 2010), $\mathrm{FFT}(\mathrm{ZP}(\mathbf{W}))$ and $\mathrm{FFT}(\mathrm{ZP}(\mathbf{H}))$ and $\mathrm{FFT}(\mathbf{W} * \mathbf{H})$ satisfy

$$\begin{aligned} ||\mathrm{FFT}(\mathrm{ZP}(\mathbf{W}))||_F &= (K+N-1)||\mathbf{W}||_F, \\ ||\mathrm{FFT}(\mathrm{ZP}(\mathbf{H}))||_F &= (K+N-1)||\mathbf{H}||_F, \\ ||\mathrm{FFT}(\mathbf{W} * \mathbf{H})||_F &= (K+N-1)||\mathbf{W} * \mathbf{H}||_F. \end{aligned} \tag{35}$$

Taking $||\mathbf{A}_1 \odot \mathbf{A}_2||_F \leq ||\mathbf{A}_1||_F||\mathbf{A}_2||_F$ into Equation (35), we have

$$\mathrm{FFT}(\mathrm{ZP}(\mathbf{W})) \odot \mathrm{FFT}(\mathrm{ZP}(\mathbf{H})) \leq ||\mathrm{FFT}(\mathbf{W})||_F||\mathrm{FFT}(\mathbf{H})||_F \tag{36}$$

Taking Equation (35) into Equation (36), we have

$$\begin{aligned} (K+N-1)||\mathbf{W} * \mathbf{H}||_F &= ||\mathrm{FFT}(\mathbf{W}) \odot \mathrm{FFT}(\mathbf{H})||_F \\ &\leq ||\mathrm{FFT}(\mathbf{W})||_F||\mathrm{FFT}(\mathbf{H})||_F \\ &= (K+N-1)||\mathbf{W}||_F(K+N-1)||\mathbf{H}||_F \end{aligned}$$

□

N. Gradient Error Bound (GEB) of a Linear Layer

We give the Gradient Error upper Bound (GEB) of a linear layer and proof in this section.

Theorem 1B. *During the backward pass of a linear layer l, $\mathrm{GEB}_l^{\mathsf{L}}$ and $\mathrm{GEB}_l^{\mathsf{H}}$ satisfy*

$$\mathrm{GEB}_l^{\mathsf{L}} - \mathrm{GEB}_l^{\mathsf{H}} = \big(\alpha_l||\mathbf{H}_{l-1}^{\mathsf{T}}||_F + \beta_l\big)(\lambda_l^{\mathsf{H}} - \lambda_l^{\mathsf{L}}) + ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F \sum_{i=l+1}^{L} \alpha_i(\lambda_i^{\mathsf{H}} - \lambda_i^{\mathsf{L}}) \prod_{j=l}^{i-1} \gamma_j, \tag{37}$$

where $\alpha_l, \beta_l, \gamma_l > 0$ for $1 \leq l \leq L$ are given by Equation (46); $\lambda_l^{\mathsf{L}} = ||\widetilde{\mathbf{H}}_l \odot \mathbf{M}||_F$; $\lambda_l^{\mathsf{H}} = ||\widetilde{\mathbf{H}}_l \odot (\mathbf{1} - \mathbf{M})||_F$; $\widetilde{\mathbf{H}}_l = \mathrm{DCT}(\mathbf{H}_l)$; and $\mathbf{M}$ denotes the 1-D loss-pass mask.

Proof. For simplicity of derivation, we consider the case $\mathrm{MiniBatch}{=}1$. In this case, $\mathbf{H}_l$ is a vector; and $\mathbf{W}_l$ is a 2-D matrix, for $1 \leq l \leq L$. The backward propagation of a linear layer is given by

$$\begin{aligned} \hat{\nabla}_{\mathbf{Z}_l} &= (\mathbf{W}_{l+1}\hat{\nabla}_{\mathbf{Z}_{l+1}}) \odot \sigma'(\hat{\mathbf{Z}}_l), \\ \hat{\nabla}_{\mathbf{W}_l} &= \hat{\nabla}_{\mathbf{Z}_l}\hat{\mathbf{H}}_{l-1}^{\mathsf{T}}, \end{aligned} \tag{38}$$

where $\hat{\mathbf{Z}}_l = \mathbf{W}_l^{\mathsf{T}}\hat{\mathbf{H}}_{l-1} + b_l$; and b_l denotes the bias of layer l. The case of $\mathrm{MiniBatch} \geq 2$ can be proved in an analogous way, which is omitted in this work.

According to Equation (38), we have the gradient of $\mathbf{Z}_l$ given by

$$\begin{aligned} &\hat{\nabla}_{\mathbf{Z}_l} - \nabla_{\mathbf{Z}_l} \\ &= \mathbf{W}_{l+1}\hat{\nabla}_{\mathbf{Z}_{l+1}} \odot \sigma'(\hat{\mathbf{Z}}_l) - \mathbf{W}_{l+1}\nabla_{\mathbf{Z}_{l+1}} \odot \sigma'(\mathbf{Z}_l), \\ &= \mathbf{W}_{l+1}\hat{\nabla}_{\mathbf{Z}_{l+1}} \odot \sigma'(\hat{\mathbf{Z}}_l) - \mathbf{W}_{l+1}\hat{\nabla}_{\mathbf{Z}_{l+1}} \odot \sigma'(\mathbf{Z}_l) + \mathbf{W}_{l+1}\hat{\nabla}_{\mathbf{Z}_{l+1}} \odot \sigma'(\mathbf{Z}_l) - \mathbf{W}_{l+1}\nabla_{\mathbf{Z}_{l+1}} \odot \sigma'(\mathbf{Z}_l), \\ &= \mathbf{W}_{l+1}\hat{\nabla}_{\mathbf{Z}_{l+1}} \odot [\sigma'(\hat{\mathbf{Z}}_l) - \sigma'(\mathbf{Z}_l)] + (\hat{\nabla}_{\mathbf{Z}_{l+1}} - \mathbf{W}_{l+1}\nabla_{\mathbf{Z}_{l+1}}) \odot \sigma'(\mathbf{Z}_l). \end{aligned} \tag{39}$$

For activation functions $\mathrm{ReLu}(\cdot)$, $\mathrm{LeakyReLu}(\cdot)$, $\mathrm{Sigmoid}(\cdot)$, $\mathrm{Tanh}(\cdot)$ and $\mathrm{SoftPlus}(\cdot)$, the gradient $\sigma'(\cdot)$ satisfies $|\sigma''(\cdot)| \leq 1$ in each differentiable domain. Combined with Cauchy–Schwarz inequality $||\mathbf{A}_1\mathbf{A}_2||_F \leq ||\mathbf{A}_1||_F||\mathbf{A}_2||_F$ (Horn & Johnson, 2012), we have

$$||\sigma'(\hat{\mathbf{Z}}_l) - \sigma'(\mathbf{Z}_l)||_F \leq ||\hat{\mathbf{Z}}_l - \mathbf{Z}_l||_F \leq ||\mathbf{W}_l||_F||\hat{\mathbf{H}}_{l-1} - \mathbf{H}_{l-1}||_F. \tag{40}$$

According to inequality $||\mathbf{A}_1 \odot \mathbf{A}_2||_F \leq ||\mathbf{A}_1||_F||\mathbf{A}_2||_F$ (Horn & Johnson, 2012), we have the upper bound of $||\hat{\nabla}_{\mathbf{Z}_l} - \nabla_{\mathbf{Z}_l}||_F$ given by

$$\begin{aligned}
&||\hat{\nabla}_{\mathbf{Z}_l} - \nabla_{\mathbf{Z}_l}||_F \\
&\leq ||\mathbf{W}_{l+1}||_F||\hat{\nabla}_{\mathbf{Z}_{l+1}}||_F||\sigma'(\hat{\mathbf{Z}}_l) - \sigma'(\mathbf{Z}_l)||_F + ||\mathbf{W}_{l+1}||_F||\hat{\nabla}_{\mathbf{Z}_{l+1}} - \nabla_{\mathbf{Z}_{l+1}}||_F||\sigma'(\mathbf{Z}_l)||_F, \\
&= ||\mathbf{W}_{l+1}||_F||\hat{\nabla}_{\mathbf{Z}_{l+1}}||_F||\hat{\mathbf{H}}_{l-1} - \mathbf{H}_{l-1}||_F||\mathbf{W}'_l||_F + ||\mathbf{W}_{l+1}||_F||\hat{\nabla}_{\mathbf{Z}_{l+1}} - \nabla_{\mathbf{Z}_{l+1}}||_F||\sigma'(\mathbf{Z}_l)||_F, \\
&= \alpha_l||\hat{\mathbf{H}}_{l-1} - \mathbf{H}_{l-1}||_F + \gamma_l||\hat{\nabla}_{\mathbf{Z}_{l+1}} - \nabla_{\mathbf{Z}_{l+1}}||_F,
\end{aligned} \tag{41}$$

where α_l and γ_l are given by

$$\begin{aligned}
\alpha_l &= ||\mathbf{W}_{l+1}||_F||\hat{\nabla}_{\mathbf{Z}_{l+1}}||_F||\mathbf{W}'_l||_F; \\
\gamma_l &= ||\mathbf{W}_{l+1}||_F||\sigma'(\mathbf{Z}_l)||_F;
\end{aligned} \tag{42}$$

the value α_l and γ_l depend on the model weight before backward propagation, which are constant with respect to the gradient. Iterate Equation (41) until $l = L$ where $||\hat{\nabla}_{\mathbf{Z}_L} - \nabla_{\mathbf{Z}_L}||_F \leq \alpha_l||\hat{\mathbf{H}}_{L-1} - \mathbf{H}_{L-1}||_F$. In such a manner, we have

$$||\hat{\nabla}_{\mathbf{Z}_l} - \nabla_{\mathbf{Z}_l}||_F \leq \alpha_l||\hat{\mathbf{H}}_{l-1} - \mathbf{H}_{l-1}||_F + \sum_{i=l+1}^{L}\alpha_i||\hat{\mathbf{H}}_{i-1} - \mathbf{H}_{i-1}||_F\prod_{j=l}^{i-1}\gamma_j. \tag{43}$$

According to Equation (38), we have the gradient of $\mathbf{W}_l$ given by

$$\begin{aligned}
\hat{\nabla}_{\mathbf{W}_l} - \nabla_{\mathbf{W}_l} &= \hat{\nabla}_{\mathbf{Z}_l}\hat{\mathbf{H}}_{l-1}^{\mathsf{T}} - \nabla_{\mathbf{Z}_l}\mathbf{H}_{l-1}^{\mathsf{T}}, \\
&= \hat{\nabla}_{\mathbf{Z}_l}\hat{\mathbf{H}}_{l-1}^{\mathsf{T}} - \hat{\nabla}_{\mathbf{Z}_l}\mathbf{H}_{l-1}^{\mathsf{T}} + \hat{\nabla}_{\mathbf{Z}_l}\mathbf{H}_{l-1}^{\mathsf{T}} - \nabla_{\mathbf{Z}_l}\mathbf{H}_{l-1}^{\mathsf{T}}, \\
&= \hat{\nabla}_{\mathbf{Z}_l}(\hat{\mathbf{H}}_{l-1}^{\mathsf{T}} - \mathbf{H}_{l-1}^{\mathsf{T}}) + (\hat{\nabla}_{\mathbf{Z}_l} - \nabla_{\mathbf{Z}_l})\mathbf{H}_{l-1}^{\mathsf{T}}.
\end{aligned} \tag{44}$$

Taking Equation (43) into Equation (44), we have

$$\begin{aligned}
&||\hat{\nabla}_{\mathbf{W}_l} - \nabla_{\mathbf{W}_l}||_F \\
&\leq ||\hat{\nabla}_{\mathbf{Z}_l}||_F||\hat{\mathbf{H}}_{l-1}^{\mathsf{T}} - \mathbf{H}_{l-1}^{\mathsf{T}}||_F + ||\hat{\nabla}_{\mathbf{Z}_l} - \nabla_{\mathbf{Z}_l}||_F||\mathbf{H}_{l-1}^{\mathsf{T}}||_F, \\
&\leq ||\hat{\nabla}_{\mathbf{Z}_l}||_F||\hat{\mathbf{H}}_{l-1}^{\mathsf{T}} - \mathbf{H}_{l-1}^{\mathsf{T}}||_F + ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F\Big[\alpha_l||\hat{\mathbf{H}}_{l-1} - \mathbf{H}_{l-1}||_F + \sum_{i=l+1}^{L}\alpha_i||\hat{\mathbf{H}}_{i-1} - \mathbf{H}_{i-1}||_F\prod_{j=l}^{i-1}\gamma_j\Big], \\
&= \Big(\beta_l + \alpha_l||\mathbf{H}_{l-1}^{\mathsf{T}}||_F\Big)||\hat{\mathbf{H}}_{l-1}^{\mathsf{T}} - \mathbf{H}_{l-1}^{\mathsf{T}}||_F + ||\mathbf{H}_{l-1}^{\mathsf{T}}||_F\sum_{i=l+1}^{L}\alpha_i||\hat{\mathbf{H}}_{i-1}^{\mathsf{T}} - \mathbf{H}_{i-1}^{\mathsf{T}}||_F\prod_{j=l}^{i-1}\gamma_j,
\end{aligned} \tag{45}$$

where β_l is given by

$$\begin{aligned}
\alpha_l &= ||\mathbf{W}_{l+1}||_F||\hat{\nabla}_{\mathbf{Z}_{l+1}}||_F||\mathbf{W}'_l||_F; \\
\beta_l &= ||\hat{\nabla}_{\mathbf{Z}_l}||_F; \\
\gamma_l &= ||\mathbf{W}_{l+1}||_F||\sigma'(\mathbf{Z}_l)||_F.
\end{aligned} \tag{46}$$

During the LFC-ACT and HFC-ACT trainings, the activation map of a linear layer satisfies

$$||\mathbf{H}_l - \mathbf{H}_l^{\mathsf{L}}||_F = ||\widetilde{\mathbf{H}}_l - \widetilde{\mathbf{H}}_l^{\mathsf{L}}||_F = ||\widetilde{\mathbf{H}}_l \odot (\mathbf{1} - \mathbf{M})||_F \triangleq \lambda_l^{\mathsf{H}}, \tag{47}$$

$$||\mathbf{H}_l - \mathbf{H}_l^{\mathsf{H}}||_F = ||\widetilde{\mathbf{H}}_l - \widetilde{\mathbf{H}}_l^{\mathsf{H}}||_F = ||\widetilde{\mathbf{H}}_l \odot \mathbf{M}||_F \triangleq \lambda_l^{\mathsf{L}}. \tag{48}$$

Taking Equations (47) and (48) into (45), we have the $\mathrm{GEB}_l^{\mathsf{L}}$ and $\mathrm{GEB}_l^{\mathsf{H}}$ of a linear layer given by

$$||\hat{\nabla}_{\mathbf{W}_l} - \nabla^{\mathsf{L}}_{\mathbf{W}_l}||_F \leq \Big(\alpha_l||\mathbf{H}^{\mathsf{T}}_{l-1}||_F + \beta_l\Big)\lambda^{\mathsf{H}}_l + ||\mathbf{H}^{\mathsf{T}}_{l-1}||_F \sum_{i=l+1}^{L} \alpha_i \lambda^{\mathsf{H}}_i \prod_{j=l}^{i-1} \gamma_j \triangleq \mathrm{GEB}^{\mathsf{L}}_l, \tag{49}$$

$$||\hat{\nabla}_{\mathbf{W}_l} - \nabla^{\mathsf{H}}_{\mathbf{W}_l}||_F \leq \Big(||\alpha_l||\mathbf{H}^{\mathsf{T}}_{l-1}||_F + \beta_l\Big)\lambda^{\mathsf{L}}_l + ||\mathbf{H}^{\mathsf{T}}_{l-1}||_F \sum_{i=l+1}^{L} \alpha_i \lambda^{\mathsf{L}}_i \prod_{j=l}^{i-1} \gamma_j \triangleq \mathrm{GEB}^{\mathsf{H}}_l. \tag{50}$$

Given the expression of $\mathrm{GEB}_l^{\mathsf{L}}$ and $\mathrm{GEB}_l^{\mathsf{H}}$ by Equations (49) and (50), we have the GEB difference for a linear layer given by

$$\mathrm{GEB}^{\mathsf{L}}_l - \mathrm{GEB}^{\mathsf{H}}_l = \Big(||\alpha_l||\mathbf{H}^{\mathsf{T}}_{l-1}||_F + \beta_l\Big)(\lambda^{\mathsf{H}}_l - \lambda^{\mathsf{L}}_l) + ||\mathbf{H}^{\mathsf{T}}_{l-1}||_F \sum_{i=l+1}^{L} \alpha_i(\lambda^{\mathsf{H}}_i - \lambda^{\mathsf{L}}_i) \prod_{j=l}^{i-1} \gamma_j.$$

□

O. Proof of Theorem 4.1

We give the proof of Theorem 4.1 in this section.

Theorem 2. *For any real-valued function $f(x)$ and its moving average $\bar{f}(x) = \frac{1}{2B}\int_x^{x+2B} f(t)\mathrm{d}t$, let $F(\omega)$ and $\overline{F}(\omega)$ denote the Fourier transformation (Madisetti, 1997) of $f(x)$ and $\bar{f}(x)$, respectively. Generally, we have $\overline{F}(\omega) = H(\omega)F(\omega)$, where $|H(\omega)| = \big|\frac{\sin \omega B}{\omega B}\big|$.*

Proof. We adopt the limit operator to reformulate $\bar{f}(x)$ into

$$\bar{f}(x) = \frac{1}{2B}\int_x^{x+2B} f(t)\mathrm{d}t = \frac{1}{2B}\lim_{N\to\infty}\sum_{n=0}^{N-1}\frac{2B}{N}f(x+\frac{2Bn}{N}) = \lim_{N\to\infty}\sum_{n=0}^{N-1}\frac{1}{N}f(x+\frac{2Bn}{N}) \tag{51}$$

Taking Equation (51) into the Fourier Transform of $\bar{f}(x)$, we have

$$\begin{aligned}
F'(\omega) &= \int_{-\infty}^{\infty} \bar{f}(x)e^{-i\omega x}\mathrm{d}x = \int_{-\infty}^{\infty} \frac{1}{N}\lim_{N\to\infty}\sum_{n=0}^{N-1} f(x+\frac{2Bn}{N})e^{-i\omega x}\mathrm{d}x \\
&= \lim_{N\to\infty}\frac{1}{N}\sum_{n=0}^{N-1}\int_{-\infty}^{\infty} f(x+\frac{2Bn}{N})e^{-i\omega x}\mathrm{d}x \\
&= \lim_{N\to\infty}\frac{1}{N}\sum_{n=0}^{N-1} e^{i\omega\frac{2Bn}{N}}\int_{-\infty}^{\infty} f(x)e^{-i\omega x}\mathrm{d}x = F(\omega)\lim_{N\to\infty}\frac{1}{N}\sum_{n=0}^{N-1}e^{i\omega\frac{2Bn}{N}} \\
&= F(\omega)(1-e^{i\omega 2B})\lim_{N\to\infty}\frac{1}{N(1-e^{i\omega\frac{2B}{N}})} = F(\omega)\frac{1-e^{i\omega 2B}}{-i\omega 2B},
\end{aligned} \tag{52}$$

where i denotes the imaginary unit.

Let $H(\omega) = \frac{1-e^{i\omega 2B}}{-i\omega 2B}$. The magnitude of $H(\omega)$ is given by

$$\begin{aligned}
\Big|H(\omega)\Big| &= \frac{|1-\cos\omega 2B + i\sin\omega 2B|}{|\omega 2B|} = \frac{\sqrt{(1-\cos\omega 2B)^2 + \sin^2\omega 2B|}}{|\omega 2B} \\
&= \frac{\sqrt{4\sin^4\omega B + 4\sin^2\omega B\cos^2\omega B}}{|\omega 2B|} = \frac{\sqrt{4\sin^2\omega B(\sin^2\omega B + \cos^2\omega B)}}{|\omega 2B|} \\
&= \Big|\frac{\sin\omega B}{\omega B}\Big|
\end{aligned} \tag{53}$$

□

P. Theoretical Compression Rate of DIVISION

The compression rate of DIVISION is estimated in this section. A general case of convolutional neural networks (CNN) and multi-layer perceptron (MLP) are considered for the estimation.

P.1. Compression Rate of CNN training

Without loss of generality, we estimate the compression rate for a block of convolutional layer (`conv`), batch normalization layer (`BN`) and Relu activation. Most of existing backbones purely stacks `conv−BN−Relu` blocks (He et al., 2016; Huang et al., 2017; Szegedy et al., 2015; Tan & Le, 2019; Simonyan & Zisserman, 2014), which makes our estimated compression rate hold in practice. Generally, the compression rate is defined as the memory reduction ratio after the compression. To be concrete, let $\text{Minibatch}\times\text{Channel}\times N\times N$ denote the shape of activaition maps for a `conv−BN−Relu` block; given the block-size B and bit-width Q, DIVISION has the compression rate of activation maps given by Theorem 3A.

Theorem 3A. *DIVISION has average activation map compression rate for a `conv-BN-Relu` block given by*

$$R = \frac{\textit{Mem of}\,\mathbf{H}}{\textit{Mem of}\,(\mathbf{H}^{\mathsf{L}}, \mathbf{V}^{\mathsf{H}}, \Delta, \delta)} = \frac{9}{\frac{4}{\min\{B^2,N^2\}} + \frac{Q}{4} + \frac{8}{N^2} + \frac{1}{8}}, \tag{54}$$

where $\text{Minibatch}\times\text{Channel}\times N\times N$ *is the shape of activation map* $\boldsymbol{H}_l$*;* B *denotes the block-size of LFC average pooling; and* Q *denotes the bit-width of HFC quantization.*

Proof. For each mini-batch updating of normal training, a `conv`-layer or `BN`-layer caches $N^2\texttt{float32} \times 4\text{byte}/\texttt{float32} = 4N^2\text{byte}$ activation map; a Relu operator caches $N^2\texttt{int8} \times 1\text{byte}/\texttt{int8} = N^2\text{byte}$ activation map. For each mini-batch updating of DIVISION, a `conv`-layer or `BN`-layer caches $\frac{N^2}{\min\{B^2,N^2\}}\texttt{bfloat16} \times 2\text{byte}/\texttt{bfloat16} = \frac{2N^2}{\min\{B^2,N^2\}}\text{byte}$ LFC; and $QN^2\text{bit} \times \frac{1}{8}\text{bit/byte} = \frac{Q}{8}N^2\text{byte}$ HFC; and spends $2\texttt{bfloat16} \times 2\text{byte}/\texttt{bfloat16} = 4\text{byte}$ for Δ_l and δ_l. Moreover, a Relu operator caches $N^2\texttt{bit} \times \frac{1}{8}\text{byte}/\texttt{bit} = \frac{N^2}{8}\text{byte}$ activation map. Therefore, the average activation map compression rate of a `conv-BN-Relu` block is given by

$$R = \frac{4N^2 \times 2 + N^2}{\left(\frac{2N^2}{\min\{B^2,N^2\}} + \frac{Q}{8}N^2 + 4\right) \times 2 + \frac{1}{8}N^2} = \frac{9}{\frac{4}{\min\{B^2,N^2\}} + \frac{Q}{4} + \frac{8}{N^2} + \frac{1}{8}}. \tag{55}$$

□

A higher compression rate indicates more effective compression. It is observed that the compression rate grows with B and N, and decreases with Q. In our experiments, we found $B = 8$ and $Q = 2$ can provide loss-less model accuracy. In this condition, the shape of activation maps satisfies $N \geq 7$ for ResNet-50 and WRN-50-2 on the ImageNet dataset (He et al., 2016). According to Equation (54), we have $R_{\text{ResNet-50}}, R_{\text{WRN-50-2}} \geq 10.35$.

P.2. Compression Rate of MLP Training

We estimate the compression rate for a `linear-Relu` block in Theorem 3B. An MLP simply stacks multiple `linear-Relu` blocks, such that our estimated compression rate holds for MLP models.

Theorem 3B. *DIVISION has average activation map compression rate for a `linear-Relu` block given by*

$$R = \frac{\textit{Mem of}\,\mathbf{H}}{\textit{Mem of}\,(\mathbf{H}^{\mathsf{L}}, \mathbf{V}^{\mathsf{H}}, \Delta, \delta)} = \frac{5}{\frac{2}{\min\{B,N\}} + \frac{Q}{8} + \frac{4}{N} + \frac{1}{8}}, \tag{56}$$

where $\text{Minibatch}\times N$ *is the shape of activation map* $\boldsymbol{H}_l$*;* B *denotes the block-size of LFC average pooling; and* Q *denotes the bit-width of HFC quantization.*

Proof. For each mini-batch updating of normal training, a `linear`-layer caches $N\texttt{float32} \times 4\text{byte}/\texttt{float32} = 4N\text{byte}$ activation map; a Relu operator caches $N\texttt{int8} \times 1\text{byte}/\texttt{int8} = N\text{byte}$ activation map. For each mini-batch updating of DIVISION, a `linear`-layer caches $\frac{N}{\min\{B,N\}}\texttt{bfloat16} \times 2\text{byte}\texttt{bfloat16} = \frac{2N}{\min\{B,N\}}\text{byte}$ LFC; and

$QN\text{bit} \times \frac{1}{8}\text{bit}/\text{byte} = \frac{Q}{8}N\text{byte}$ HFC; and spends $2\texttt{bfloat16} \times 2\text{byte}/\texttt{bfloat16} = 4\text{byte}$ for Δ_l and δ_l. Moreover, a `Relu` operator caches $N\texttt{bit} \times \frac{1}{8}\text{byte}/\texttt{bit} = \frac{N}{8}\text{byte}$ activation map. Therefore, the average activation map compression rate of a `linear-Relu` block given by

$$R = \frac{4N + N}{\frac{2N}{\min\{B,N\}} + \frac{Q}{8}N + 4 + \frac{1}{8}N} = \frac{5}{\frac{2}{\min\{B,N\}} + \frac{Q}{8} + \frac{4}{N} + \frac{1}{8}}. \tag{57}$$

□

Q. Related Work

We discuss more related work about memory efficient training in this section. The discussion of existing work is from the perspectives of model pruning, quantization, model distributed training, randomized approximate method, embedding table sharding, and local sensitive harshing.

Model Pruning. Model pruning aims to reduce the memory footprint of DNNs by eliminating unnecessary connections or weights. These techniques leverage the observation that many weights in a trained network are redundant or have minimal impact on the model's performance. By removing these parameters, memory usage can be significantly reduced without compromising accuracy. Notable approaches include magnitude pruning, structured pruning, and iterative pruning, which selectively prune weights based on their importance or magnitude (Zhong et al., 2022; Duan et al., 2022a;b).

Quantization. Quantization focuses on reducing the precision of network weights and activations. Instead of using full-precision (32-bit) floating-point numbers, quantization methods represent weights and activations with lower bit-widths, such as 8-bit integers or even binary values. By quantizing parameters, memory requirements can be significantly reduced, allowing for efficient storage and computation. Recent advancements in quantization techniques, such as learned quantization and differentiable quantization (Wang et al., 2022a), have shown promising results in preserving model accuracy.

Model Distributed Training. Model Distributed Training divides the neural network across multiple devices or machines, allowing each device to handle a specific portion of the model (Yuan et al., 2021; 2022). This approach is particularly useful for training extremely large models that cannot fit into the memory of a single device. By partitioning the model and carefully orchestrating data and computation flows, memory requirements can be distributed across multiple devices, enabling the training of models with higher capacity.

Randomized Approximate Method. Randomized algorithms often employ sampling and sketching techniques to reduce memory requirements while providing useful insights about the data. These techniques involve selecting a representative subset of the data or summarizing the data using compact data structures. By utilizing random sampling, the algorithm can approximate characteristics of the entire dataset with a smaller memory footprint, making it more memory-efficient (Wang et al., 2022b; Xu et al., 2021b; Liu et al., 2022b;a).

Embedding Table Sharding. Embedding table sharding refers to the practice of partitioning or dividing the embedding table into multiple smaller tables, known as shards. Each shard contains a subset of the entire embedding table's entries. Specifically, sharding provides flexibility in allocating memory resources. Instead of allocating a single large block of memory for the entire embedding table, we can allocate memory separately for each shard. This flexibility enables more efficient memory management, as memory can be allocated dynamically based on the specific needs of each shard (Zha et al., 2022a;b; 2023; Ferber et al., 2023). It also allows for more effective utilization of memory hierarchies, such as caching mechanisms, by optimizing the storage and retrieval of shard-specific embeddings.

Local sensitive Harshing (LSH). LSH is a technique used in machine learning to approximate nearest neighbor search efficiently. It is known for its scalability in handling large-scale datasets. By employing LSH, it becomes possible to partition the data into smaller subsets and store them separately. This distributed representation of the data reduces the memory requirements for each subset, making it easier to handle and process large amounts of data within the available memory resources (Xu et al., 2021a; 2023; Desai et al., 2021).