Heidelberg University

Institute of Computer Engineering
Hardware and Artificial Intelligence (HAWAII) Lab
Master's Thesis

Mix-Precision Stochastic Quantization of Activations during Backpropagation

Name:	Jiufeng Li
Matriculation number:	3772108
Study program:	Master in Data and Computer Science
Supervisor:	Prof. Dr. Holger Fröning
Second examiner:	Prof. Dr. Robert Strzodka
Date of submission:	31/08/2025

DECLARATION

I hereby certify that I have written the work myself and that I have not used any sources or aids other than those specified and that I have marked what has been taken over from other people's works, either verbatim or in terms of content, as foreign. I also certify that the electronic version of my thesis transmitted completely corresponds in content and wording to the printed version. I agree that this electronic version is being checked for plagiarism at the university using plagiarism software.

Heidelberg, 31/08/2025

Jiufeng Li

ABSTRACT

Training deep networks is memory intensive because the backward pass reuses *saved forward activations*, which often dominate the GPU footprint. While quantization for inference and forward-path QAT are well studied, prior work has not treated *activation quantization specifically for backpropagation* as a first-class problem. This thesis formulates activation quantization for the backward pass, develops practical realizations, and evaluates their stability–accuracy trade-offs and memory benefits.

This thesis explores two complementary approaches. A deterministic baseline selectively quantizes saved activations while leaving the forward computation path unchanged. Implemented with Brevitas on EfficientNetV2, this method achieves substantial training-time memory reductions (e.g., $\sim 8\times$ at 4-bit) while preserving competitive accuracy. The second approach introduces *stochastic activation quantization* with per-bucket scaling to eliminate rounding bias and reduce quantization variance. The vectorized implementation enables efficient bucket-level scaling and unbiased rounding, supporting *arbitrary two-bit stochastic mixing* through probability-controlled precision selection. Experiments on CIFAR-10 and CIFAR-100 demonstrate that stochastic quantization improves both stability and accuracy in the 2–4 bit range while closely tracking 32-bit reference performance at moderate precisions. Results on Tiny ImageNet (ImageNet-200) confirm these trends, albeit with greater training variability due to baseline and optimization sensitivities.

Overall, activation quantization *for backpropagation* is feasible and effective: quantizing saved activations to 4-bit or even 2-bit (stochastic) reduces activation memory by $\sim 8\times$ and $\sim 16\times$, respectively, while maintaining competitive accuracy and enabling larger batch sizes or smaller GPUs. The results support stochastic, per-bucket activation quantization, with optional two-bit mixing, as a practical recipe for memory-efficient training, and motivate future work on stability at larger scales, broader model coverage (CNNs and Transformers), and automated multi-bit assignment.

ZUSAMMENFASSUNG

Das Training tiefer Netze ist speicherintensiv, weil der Rückwärtsdurchlauf *gespeicherte Vorwärtsaktivierungen* wiederverwendet, die häufig den größten Anteil am GPU-Speicher ausmachen. Während Quantisierung für Inferenz und QAT im Vorwärtspfad gut untersucht sind, wurde die *Aktivierungsquantisierung speziell für den Rückwärtsdurchlauf* bislang nicht als eigenständiges Problem behandelt. Diese Arbeit formuliert Aktivierungsquantisierung für die Backpropagation, entwickelt praxisnahe Realisierungen und bewertet deren Stabilitäts–Genauigkeits-Abwägungen sowie die Speichereffekte.

Diese Arbeit untersucht zwei sich ergänzende Ansätze. Eine deterministische Basisvariante quantisiert selektiv gespeicherte Aktivierungen, während der Vorwärtsberechnungspfad unverändert bleibt. Die Implementierung mit Brevitas auf EfficientNetV2 erzielt erhebliche Einsparungen des trainingsseitigen Aktivierungsspeichers (z. B. $\sim 8\times$ bei 4-Bit) unter Wahrung wettbewerbsfähiger Genauigkeit. Der zweite Ansatz führt *stochastische Aktivierungsquantisierung* mit *gruppenweiser Skalierung* ein, um Rundungsbias zu eliminieren und Quantisierungsvarianz zu reduzieren. Die vektorisierte Implementierung ermöglicht effiziente gruppenweise Skalierung und unverzerrtes Runden und unterstützt *beliebige Zwei-Bit-stochastische Mischungen* durch wahrscheinlichkeitsgesteuerte Präzisionswahl. Experimente auf CIFAR-10 und CIFAR-100 demonstrieren, dass stochastische Quantisierung sowohl Stabilität als auch Genauigkeit im 2–4-Bit-Bereich verbessert und die 32-Bit Referenzleistung bei moderaten Präzisionen eng verfolgt. Ergebnisse auf Tiny ImageNet (ImageNet-200) bestätigen diese Trends, wenngleich mit größerer Trainingsvariabilität aufgrund von Baseline- und Optimierungssensitivitäten.

Insgesamt ist Aktivierungsquantisierung *für die Backpropagation* praktikabel und wirksam: 4-Bit bzw. 2-Bit (stochastisch) reduzieren den Aktivierungsspeicher um etwa $\sim 8\times$ bzw. $\sim 16\times$, bei Erhalt konkurrierender Genauigkeit und mit der Möglichkeit größerer Batches oder kleinerer GPUs. Die Ergebnisse stützen stochastische, gruppenweise skalierte Aktivierungsquantisierung, mit optionaler Zwei-Bit-Mischung als praktikables Rezept für speichereffizientes Training und motivieren weitere Arbeiten zu Stabilität im größeren Maßstab, breiterer Modellabdeckung (CNNs und Transformer) sowie automatisierter Mehrbit-Zuweisung.

"The people who are crazy enough to think they can change the world are the ones who do."

— Steve Jobs

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to those who made this thesis possible. First and foremost, I thank my tutor, **Daniel Barley**, for insightful discussions, guidance, and inspiration throughout both the experimental work and the writing process. I am equally grateful to my supervisor, **Holger Fröning**, for his critical, patient, and attentive supervision at every key stage of the project.

This work greatly benefited from the computational resources provided by the **Hardware and Artificial Intelligence (HAWAII) Lab**. Access to the GPU cluster enabled me to conduct the experiments efficiently and to complete this thesis on time.

On a personal note, I owe profound thanks to my mother for her constant care, support, and love, which allowed me to pursue my studies with confidence. I also wish to dedicate this thesis to my dearest aunt, who tragically passed away in a car accident in August. May she find peace in heaven. I will strive for excellence and success, and I hope she will be proud of me.

CONTENTS

1 INTRODUCTION

1.1 Motivation

Deep neural networks have achieved remarkable success across various domains, from computer vision to natural language processing. However, the computational and memory requirements of these models have grown exponentially, making them increasingly challenging to deploy on resource-constrained devices. Quantization, the process of reducing the precision of neural network weights and activations from 32-bit floating-point to lower bit-width representations, has emerged as a crucial technique for model compression and acceleration. While weight quantization has been extensively studied and successfully deployed, activation quantization during training, particularly during backpropagation, remains a challenging and less explored area.

The backpropagation process is fundamental to training neural networks, where gradients flow backward through the network to update weights. During this process, activations are stored in memory for gradient computation, often constituting the majority of memory usage during training. Traditional approaches maintain full precision (32-bit) for activations during backpropagation to preserve training stability and convergence. However, this approach significantly limits the memory efficiency gains that could be achieved through quantization.

In modern convolutional neural networks (CNNs), the tensors that dominate the training-time GPU footprint are precisely these saved activations rather than model parameters. Empirically, across ResNet-like backbones, storing per-layer activations for the backward pass frequently accounts for a substantial fraction of the total memory budget, whereas weights and optimizer states are comparatively smaller. This asymmetry makes activations the primary target for memory optimization: quantizing activations that are kept for backpropagation can yield large savings with limited impact on convergence when done carefully. Recent systems demonstrate that randomly/stochastically quantized activations with per-group scaling can compress activations to as low as an average of 2 bits, achieving about an order-of-magnitude reduction in activation memory while maintaining accuracy and enabling significantly larger batch sizes [7]. Follow-up work further generalizes activation-compressed training beyond specific backbones to generic architectures, reinforcing that activation quantization during backpropagation is an effective and widely applicable lever for reducing training memory [22]. Complementary to quantization, compression by pooling for the backward pass has also been shown to reduce peak memory

while preserving accuracy, further underscoring that activations dominate training memory [2].

Despite progress on inference quantization and forward-path quantization-aware training, prior work has largely not treated *activation quantization specifically for the backpropagation stage* as a first-class problem. This thesis fills that gap: it formulates backpropagation activation quantization, implements it end-to-end, and *empirically validates* its feasibility and efficiency, achieving large memory reductions with competitive accuracy across CIFAR-10/100 and Tiny ImageNet (ImageNet-200). Beyond a deterministic baseline, the thesis further introduces a *stochastic activation quantization* strategy with per-group scaling that removes rounding bias and stabilizes low-bit training, enabling the 2–4 bit regime to approach the 32-bit reference while conserving substantially more memory. These constitute novel and practically useful contributions relative to prior art focused on weights, forward activations, or non-quantization compression.

1.2 Literature Review

Recent research has demonstrated the potential of activation quantization during forward propagation, with techniques such as post-training quantization and quantization-aware training showing promising results. However, the application of quantization to activations during backpropagation has received limited attention. Existing work primarily focuses on weight quantization or forward-pass activation quantization, leaving a significant gap in understanding how quantization affects the gradient flow and training dynamics during backpropagation.

Several studies have explored mixed-precision quantization strategies, where different layers or components use different bit-widths based on their sensitivity to quantization. These approaches have shown that not all parts of a neural network require the same precision level, suggesting that a more nuanced approach to activation quantization during backpropagation could be beneficial. However, the specific challenges and opportunities of applying mixed-precision quantization to backpropagation activations remain largely unexplored.

Beyond forward-path quantization, there is a line of work that directly targets training-time memory by approximating or compressing the activations stored for the backward pass. Early work stores approximate activations and uses them only in the backward pass, demonstrating that low-bit approximations can preserve accuracy while reducing memory [6]. Recent approaches quantize derivatives of activation functions to a few bits, yielding sizable savings with negligible accuracy loss in practice [15]. System-level methods compress or quantize the saved activations stochastically with per-group scaling, enabling average 2-bit activations during training [7, 22]. Complementary techniques reduce activation footprint without quantization, such as checkpointing/recomputation [8], CPU/GPU offloading [16], and mixed-precision arithmetic to shrink tensor sizes [14]. In parallel, hardware-aware and Hessian-

aware quantization tailor bit-widths to sensitivity, providing a basis for mixed-precision policies that can be adapted to backpropagation activations [9, 20]. Together with recent pooling-based compression for the backward pass [2], these works motivate a unified, mixed-precision framework that balances accuracy, memory, and compute overhead for activations during backpropagation.

1.3 The Need for the Present Work

Current approaches to activation quantization during backpropagation face several critical limitations. First, existing implementations often suffer from training instability, characterized by sudden spikes or drops in test accuracy, which can lead to poor convergence or complete training failure. Second, the computational overhead of quantization operations during backpropagation can significantly slow down training, making the approach impractical for real-world applications. Third, there is a lack of systematic understanding of how different quantization parameters, such as bucket sizes and bit-widths, affect training performance and memory efficiency.

Furthermore, existing work has not adequately addressed the trade-off between quantization precision and training stability. While lower bit-widths offer greater memory savings, they often lead to increased quantization noise that can destabilize training. Conversely, higher bit-widths provide better training stability but offer limited memory savings. This trade-off necessitates a more sophisticated approach that can dynamically balance precision and stability based on the specific requirements of different parts of the network.

1.4 Objective of the Present Work

This thesis addresses the challenges of activation quantization during backpropagation by developing and evaluating a comprehensive mixed-precision quantization framework. The primary objective is to enable efficient and stable training of quantized neural networks while maximizing memory savings through intelligent quantization of backpropagation activations.

The specific contributions of this work are:

(1) **Comprehensive Analysis and Implementation of Activation Quantization during Backpropagation**: This thesis develops a systematic approach to quantizing activations during backpropagation using the Brevitas framework, addressing the fundamental challenges of maintaining training stability while achieving significant memory savings. This includes the implementation of layer-wise quantization strategies and the development of optimization techniques to prevent training spikes and ensure stable convergence.

(2) **Advanced Stochastic Quantization Framework with Mixed-Precision Capabilities**: This thesis proposes and implements a sophisticated stochastic quantization approach that combines different bit-widths (2-bit and 4-bit) with configurable bucket sizes and probability distributions. This framework includes vectorized implementations for improved computational efficiency and flexible hyperparameter tuning capabilities to optimize the trade-off between accuracy and memory usage.

(3) **Extensive Experimental Validation and Generalization Analysis**: This thesis conducts comprehensive experiments on multiple datasets (CIFAR-10, CIFAR-100, Tiny ImageNet (ImageNet-200)) to validate the effectiveness of the proposed approach and demonstrate its generalization capabilities. This includes detailed analysis of different bucket sizes, mixed-precision configurations, and their impact on training performance, memory efficiency, and computational overhead.

1.5 Justification and Importance

The work presented in this thesis addresses a critical gap in the field of neural network quantization and has significant implications for both research and practical applications. From a research perspective, this work advances our understanding of how quantization affects the fundamental training process of neural networks, particularly during backpropagation. The insights gained from this research can inform future developments in quantization techniques and contribute to the broader field of efficient deep learning.

From a practical standpoint, the ability to quantize activations during backpropagation while maintaining training stability opens up new possibilities for training large neural networks on memory-constrained devices. This is particularly important for edge computing applications, where both training and inference need to be performed on resource-limited hardware. The mixed-precision approach developed in this work provides a flexible framework that can be adapted to different hardware constraints and application requirements.

Furthermore, the optimization strategies and implementation techniques developed in this work can serve as a foundation for future research in efficient neural network training. The vectorized implementations and hyperparameter tuning methodologies can be applied to other quantization scenarios, making this work valuable for the broader machine learning community.

1.6 Preview

The remainder of this thesis is organized as follows: Chapter 2 provides the necessary background on neural network quantization, backpropa-

gation, and related concepts. Chapter 3 reviews the state-of-the-art in activation quantization and identifies gaps in current research. Chapter 4 presents the first contribution, focusing on the implementation and optimization of activation quantization during backpropagation using Brevitas. Chapter 5 details the second contribution, introducing the mixed-precision stochastic activation quantization framework. Chapter 6 presents the third contribution: an extensive validation of the generalization of stochastic activation quantization by applying EfficientNetV2 to larger datasets such as CIFAR-100 and Tiny ImageNet (ImageNet-200), with additional experiments and analyses. Finally, the thesis concludes with a summary of the contributions and directions for future work.

2 BACKGROUND

This chapter introduces the key ideas and terminology used throughout the thesis: (i) quantization for backpropagation, (ii) activation quantization and how it differs from weight quantization and forward-only activation quantization, (iii) stochastic versus deterministic quantization and why stochastic quantization is often preferred in training, and (iv) how to realize activation quantization in practice with the Brevitas library.

2.1 Quantization for Backpropagation

During training, modern deep networks store intermediate activations for each layer so that the backward pass can compute gradients. These saved activations dominate GPU memory for common CNN and Transformer backbones. Thus, compressing the tensors retained for the backward pass is a direct lever to reduce training memory [2, 6, 7].

This thesis uses the term *backpropagation quantization* to denote techniques that compress the tensors needed only for the backward pass (e.g., saved activations, context) without changing the forward computation. Representative approaches include storing approximate activations for use only in backward [6], quantizing the saved activations with per-group scaling and unbiased rounding [7, 22], or compressing backward activations via pooling to reduce footprint and movement [2].

2.1.1 Backpropagation and Parameter Updates

Consider a feedforward layer l with pre-activation and activation

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}, \qquad a^{(l)} = \phi^{(l)}\left(z^{(l)}\right).$$

Given a loss $L\left(a^{(L)}, y\right)$, the output-layer error signal is

$$\delta^{(L)} = \nabla_{z^{(L)}} L = \nabla_{a^{(L)}} L \odot \phi'^{(L)}\left(z^{(L)}\right).$$

For hidden layers $l = L-1, \ldots, 1$, the chain rule yields

$$\delta^{(l)} = \left(W^{(l+1)}\right)^{\top} \delta^{(l+1)} \odot \phi'^{(l)}\left(z^{(l)}\right).$$

The gradients with respect to parameters are

$$\nabla_{W^{(l)}} L = \delta^{(l)} \left(a^{(l-1)}\right)^{\top}, \qquad \nabla_{b^{(l)}} L = \delta^{(l)}.$$

With stochastic gradient descent (SGD) and learning rate η, the updates read

$$W^{(l)} \leftarrow W^{(l)} - \eta \nabla_{W^{(l)}} L, \qquad b^{(l)} \leftarrow b^{(l)} - \eta \nabla_{b^{(l)}} L.$$

These derivations follow the standard backpropagation algorithm [10, 17].

In backpropagation quantization, when the saved forward activations are stored in quantized form $\tilde{a}^{(l-1)}$ for use only during the backward pass, the weight gradient uses the quantized tensor

$$\nabla_{W^{(l)}} L \approx \delta^{(l)} \left(\tilde{a}^{(l-1)}\right)^{\top},$$

and unbiased stochastic rounding helps avoid systematic gradient bias [7, 11].

2.2 Activation Quantization

Activation quantization maps floating-point activations to a low-bit representation. It can be applied in two distinct places:

- **Forward activations (inference/training)**: used to speed up compute or shrink feature maps during forward passes.
- **Saved activations (training-only)**: stored during forward and reused during backward; quantizing only these saves memory with minimal impact on forward compute [7, 15, 22].

This thesis focuses on the second case: low-bit storage of saved activations for the backward pass. This is complementary to other memory techniques such as gradient checkpointing [8] and offloading [16], and to arithmetic mixed precision [14].

2.3 Deterministic vs. Stochastic Quantization

2.3.1 Deterministic Quantization

A deterministic quantizer maps a real value to the nearest (or floor/ceil) quantization level. It is simple but introduces *bias*: in general, $\mathbb{E}[Q(x)] \neq x$ when the input distribution is taken into account, and the rounding error can accumulate through layers.

2.3.2 Stochastic Quantization

A stochastic quantizer selects neighboring quantization levels at random so that the dequantized value is *unbiased*. For uniform quantization with step size $\Delta > 0$ (quantization granularity), consider an input value x (real-valued activation). Let $t = x/\Delta$ (scaled position on quantization

grid), $k = \lfloor t \rfloor$ (lower quantization level index), and the fractional part $\alpha = t - k \in [0, 1)$ (relative position between quantization levels). The canonical rule is

$$Q(x) = \begin{cases} \lfloor x/\Delta \rfloor \, \Delta & \text{w.p. } 1 - \alpha, \\ \lceil x/\Delta \rceil \, \Delta & \text{w.p. } \alpha. \end{cases}$$

where $Q(x)$ (quantized output) probabilistically selects the lower bound with probability $1 - \alpha$ or the upper bound with probability α. Defining the quantization error $e = Q(x) - x$ (difference between quantized and original value), one obtains

$$\mathbb{E}[e \mid x] = 0, \qquad \mathrm{Var}[e \mid x] = \Delta^2 \, \alpha(1 - \alpha) \;\le\; \tfrac{\Delta^2}{4}.$$

Thus $\mathbb{E}[Q(x) \mid x] = x$ (unbiasedness), and the mean-squared error is bounded by a simple function of the step size Δ. Compared to deterministic rounding-to-nearest, the stochastic rule removes systematic bias and mitigates error accumulation across layers/iterations—properties that are especially beneficial when quantized values participate in gradient computations [1, 11].

In training, unbiasedness is most impactful when the quantized quantity enters gradient formulas (e.g., saved activations used to form weight gradients). With the straight-through estimator (STE) [3] for the non-differentiable rounding, using a quantized, dequantized activation $\tilde{a} = Q(a)$ in place of a yields weight-gradient estimates whose expectation tracks the full-precision counterpart under mild independence assumptions, while enjoying much lower memory cost [1, 7].

Practical systems further reduce error by adopting *per-group/per-bucket scaling*: Δ is computed locally from the data range within buckets, shrinking Δ and hence the variance bound $\mathrm{Var}[e] \le \Delta^2/4$. This also lowers the probability of representational *clipping* when values exceed the dynamic range, which would otherwise introduce bias. Such bucketed scaling and stochastic rounding together underpin low-bit training that remains stable at precisions as low as ∼2-bit for saved activations [7, 15].

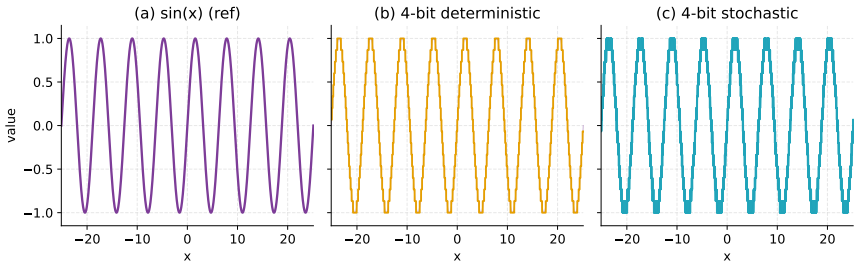

Figure 2.1: Tripanel illustration of 4-bit quantization of $\sin(x)$ over a wide range. (a) Reference signal. (b) Deterministic rounding shows a biased staircase. (c) Stochastic rounding is unbiased in expectation and better tracks the reference over many samples.

The illustration in Figure 2.1 contrasts the *bias* introduced by deterministic rounding with the unbiased behavior of stochastic rounding.

- **(a) Reference**: the continuous $\sin(x)$ signal without quantization.
- **(b) 4-bit deterministic**: rounding-to-nearest produces a staircase that systematically deviates from the reference ($\mathbb{E}[Q(x) \mid x] \neq x$). The plateaus and clipped peaks illustrate persistent bias and phase-like distortions at transitions.
- **(c) 4-bit stochastic**: randomizing between neighboring levels makes the dequantized value *unbiased* in expectation ($\mathbb{E}[Q(x) \mid x] = x$). Individual samples fluctuate around the reference, but over many samples the trace follows the true signal more faithfully.

Deterministic quantization is simple but biased; when such quantized activations are reused in backpropagation (e.g., to form weight gradients), the bias can accumulate across layers and iterations. Stochastic quantization, combined with per-group scaling, preserves the expected signal level and keeps mean-squared error bounded by a function of the step size, which helps maintain training stability and accuracy at low bit-widths.

2.4 Why Stochastic Quantization for Backpropagation

When activations are quantized for the backward pass, unbiasedness avoids systematic gradient bias. In practice, per-group scaling ("bucketed" min/max and scale per chunk) plus stochastic rounding reduces quantization variance at low bit-widths (e.g., average 2-bit) while maintaining accuracy and enabling larger batch sizes [7]. Compared to deterministic rounding, stochastic rounding prevents error accumulation across layers and iterations, making it preferable when quantized values directly influence gradient computations [11, 15].

2.5 Brevitas: Practical Activation Quantization

Brevitas is an open-source PyTorch library that enables quantization-aware training (QAT) of weights and activations via modular quantizers that can be attached to layers or inserted as standalone modules [5]. For activations, typical realizations include:

- **Standalone activation quantizer**: inserting a quantizer module (e.g., a quantized identity) to quantize the tensor produced by a layer's forward pass.
- **In-layer activation quantization**: using Brevitas' quantized layers (e.g., quantized convolution/linear) that expose activation quantization parameters.

Core configuration knobs include: bit-width (e.g., 2/4/8-bit), quantization scheme (uniform; per-tensor vs. per-channel/per-group), scale/zero-point computation (static vs. dynamic), and rounding mode (deterministic vs. stochastic). For training-time memory reduction, one can structure models such that activations to be saved for backward are quantized by an activation quantizer in the forward pass while leaving the mathematical forward compute effectively unchanged at full precision downstream, so only the saved tensors are low-bit. This mirrors the backpropagation quantization goal described above.

In summary, Brevitas provides the building blocks to prototype and evaluate activation quantization policies (bit-width, grouping/buckets, stochastic rounding) within PyTorch, aligning with the stochastic, per-group quantization strategies shown to be effective in prior work [7, 15].

3 STATE OF THE ART AND RELATED WORK

This chapter reviews the literature relevant to memory-efficient training via quantization, with a particular focus on activation quantization during backpropagation (training-time compression of saved activations). The discussion is organized along four axes: (i) CNN quantization for inference and classic QAT, (ii) training-time memory reduction techniques, (iii) activation compression for the backward pass, and (iv) the gaps motivating this thesis.

3.1 CNN Quantization: From Inference to QAT

Quantization for efficient inference is now mainstream in CNNs, with integer-arithmetic-only inference and 8-bit (or lower) activations/weights widely adopted [12, 13]. Quantization-aware training (QAT) introduces quantizers in the forward path to preserve accuracy at low bit-widths by simulating quantization effects during training. Hardware-aware [20] and Hessian-aware [9] methods further tailor per-layer bit-widths by sensitivity, motivating mixed precision policies.

While highly successful for inference, these methods do not directly address the *training memory* bottleneck: they often still store full-precision activations in the forward pass for use by the backward pass, leaving peak memory largely dominated by saved activations.

3.1.1 QAT Formulation: Uniform Affine Quantization and STE

Let x be a real-valued tensor (activation or weight). A uniform affine quantizer with scale $s > 0$, integer zero-point z, and integer range $[q_{\min}, q_{\max}]$ defines the integer code as

$$q(x) = \mathrm{clip}\left(\mathrm{round}(x/s) + z,\ q_{\min},\ q_{\max}\right).$$

The corresponding dequantized (fake-quantized) value used in QAT is

$$\tilde{x} = s\left(q(x) - z\right).$$

This *fake quantization* inserts quantization effects in the forward pass while keeping training in floating point [12, 13]. During backpropagation, QAT uses a straight-through estimator (STE) for the non-differentiable rounding and clipping:

$$\frac{\partial \tilde{x}}{\partial x} \approx \mathbb{1}\Big\{q_{\min} \le \mathrm{round}(x/s) + z \le q_{\max}\Big\},$$

that is, unit gradient through the quantizer within the representable range and zero outside [3]. For per-tensor (or per-channel) scale, gradients with respect to parameters flow as usual through the fake-quantized tensor, e.g. for a linear layer with weights W and input activations a, one uses

$$y = (\tilde{W})\, a, \qquad \nabla_W L \approx \nabla_{\tilde{W}} L,$$

where $\tilde{W}$ is the fake-quantized weight. Similarly, activation fake quantization uses $\tilde{a}$ in place of a in the forward, and STE passes gradients to a within range. These definitions extend to per-channel/per-group scales and mixed-precision policies common in modern QAT.

3.2 Training-Time Memory Reduction

Three complementary families aim to reduce training memory:

- **Recomputation (Checkpointing)**: Drop some activations during forward and recompute them in backward [8]. Reduces memory but increases compute.
- **Offloading/Swapping**: Move tensors between GPU/CPU to expand available memory [16]. Reduces GPU usage but adds PCIe/NVLink traffic.
- **Arithmetic Mixed Precision**: Use FP16/BF16 to shrink tensor sizes [14]. Reduces storage/computation, but still stores tensors unless combined with other methods.

These approaches are orthogonal to activation quantization during backpropagation and can be combined.

3.2.1 Stochastic Quantization in Gradient Descent vs. Activation Quantization

Stochastic quantization has a long history in optimization and distributed training. A large body of work compresses *gradients* or *model updates* with unbiased (or nearly unbiased) quantizers to reduce communication or enable low-precision arithmetic, e.g., limited-precision deep learning with stochastic rounding [11], QSGD [1], TernGrad [21], 1-bit SGD [18], and signSGD variants [4]. These methods provide theoretical and empirical evidence that unbiased (or appropriately corrected) quantization can preserve convergence in non-convex deep learning.

By contrast, this thesis focuses on *activation quantization during backpropagation*: quantizing the *saved forward activations* that enter gradient formulas. This targets the *training-memory* bottleneck rather than communication. While recent systems (e.g., ActNN/GACT) compress saved activations using per-group scaling and stochastic rounding

[7, 22], gaps remain in formalizing bucketed stochastic activation quantization, analyzing its interaction with STE and clipping, and evaluating generalization on larger datasets and modern CNNs. This thesis addresses those gaps with a clear mathematical formulation, vectorized implementations, and systematic experiments across CIFAR-10/100 and ImageNet-100.

3.3 Activation Compression for the Backward Pass

A direct way to reduce peak training memory is to compress the emphsaved activations that are only required for gradient computation:

- **Approximate activations for backward**: BLPA stores approximate per-layer copies used only in backward, enabling low-bit storage while controlling error propagation [6].
- **Few-bit backward derivatives**: Quantize the derivatives of activation functions (e.g., GELU/Swish) to a few bits with optimized piecewise-constant approximations, achieving up to 40% memory reduction with negligible loss [15].
- **Stochastic per-group activation quantization**: ActNN/GACT store activations with per-group min/max scaling and unbiased stochastic rounding, realizing average ∼2-bit activations and enabling much larger batch sizes with accuracy retention [7, 22]. Unbiased stochastic rounding is supported theoretically and empirically for preserving convergence under low precision [1, 11].
- **Pooling-based compression for backward**: Compress saved activations via pooling to reduce both memory footprint and data movement while maintaining accuracy [2].

3.4 Limitations of Prior Work

Despite clear progress, gaps remain:

- **Limited analysis of mixed-precision across layers/buckets**: Many works fix a global bit-width; layer-/bucket-wise policies tuned to gradient variance and activation heterogeneity are less explored, especially for CNNs where per-channel/per-group distributions vary [7, 22].
- **Trade-off characterization**: Systematic comparisons of bucket size, stochastic vs. deterministic rounding, and bit-width mixing (e.g., 2-bit with 4-bit) on training stability/accuracy are scarce.
- **Activation quantization vs. gradient quantization**: The unbiased quantization literature in SGD largely targets gradients/updates for communication efficiency [1, 4, 18, 21], whereas fewer

studies provide a rigorous, bucketed stochastic *activation* quantization treatment specific to backpropagation with modern CNNs and large-scale datasets.

- **Implementation friction**: End-to-end recipes to integrate activation quantization for backward into standard training stacks (e.g., with libraries like Brevitas) are under-documented.

3.5 Motivation and Objectives

This thesis targets the activation memory bottleneck during training by focusing on *activation quantization for the backward pass*. The objectives of this thesis are:

- **Reduce peak memory with low-bit activations**: Evaluate 2-bit and 4-bit settings and their mixtures, aiming for order-of-magnitude activation compression with minimal accuracy loss [7].
- **Use unbiased stochastic rounding**: Prefer stochastic quantization to avoid gradient bias and stabilize training at low bits [11, 15].
- **Understand bucket size effects**: Study per-group scaling and bucket sizes to balance quantization variance and metadata overhead, and to adapt to activation heterogeneity [7].
- **Mixed-precision policies**: Explore mixing 2-bit and 4-bit across layers or buckets, motivated by sensitivity-aware quantization literature [9, 20].
- **Practical realization**: Implement unbiased *stochastic* activation quantization within PyTorch using Brevitas-like quantizers and integrate with standard training code paths [5].

3.6 Positioning and Expected Outcomes

Compared to prior work, this thesis emphasizes:

- **Fine-grained comparisons**: bucket sizes, rounding modes, and 2b/4b mixes on CNN training stability and accuracy.
- **Implementation patterns**: simple, reproducible hooks to quantize only the saved activations for backward, complementary to checkpointing/offloading/AMP.
- **Generalization on larger datasets (CIFAR-100, ImageNet-100)**: evaluate how *stochastic* low-bit activation quantization generalizes beyond small-scale settings using EfficientNetV2 across CIFAR-10/100 and ImageNet-100.

- **Stability and accuracy via stochastic activation quantization**: relative to traditional deterministic activation quantization, the proposed per-group, unbiased *stochastic* activation quantization mitigates rounding bias and reduces error accumulation, yielding more stable training and equal-or-better accuracy at low bit-widths [7, 11, 15].

This thesis demonstrates that carefully designed stochastic, mixed-precision activation quantization during backpropagation can substantially cut training memory while retaining competitive accuracy, thereby enabling larger batch sizes or training on smaller GPUs [2, 7].

4 DETERMINISTIC QUANTIZATION OF ACTIVATIONS FOR BACKPROPAGATION

4.1 Goal

This chapter introduces low-bit *activation quantization for the backward pass* to reduce peak training memory while preserving training stability and accuracy relative to traditional deterministic activation quantization [7, 11, 15].

4.1.1 Motivation: Activation Memory Dominance and Principle

In typical CNN training, the peak memory during backpropagation is dominated by *saved forward activations* rather than parameters or optimizer states. If the total number of saved-activation elements across layers is N_{act}, then under FP32 the footprint is approximately $M_{\text{FP32}} \approx 4\,N_{\text{act}}$ bytes. By storing only the saved activations in a low-bit representation while keeping forward computation unchanged, the footprint becomes

$$M_{\text{quant}} \approx \frac{b}{8} N_{\text{act}} \quad + \quad M_{\text{meta}},$$

where b is the activation bit-width and M_{meta} is small per-bucket metadata for scale/zero-point. Ignoring metadata, the theoretical reduction factor is about $\frac{32}{b}$ (e.g., $\sim 8\times$ at 4-bit), which directly translates into larger feasible batch sizes or the ability to train on smaller GPUs.

Principle: quantize the *saved* activations for use in gradient formulas, but do *not* alter the mathematical forward path. Concretely, gradients use the dequantized activations $\tilde{a} = Q(a)$ in place of a, e.g., $\nabla_W L \approx \delta\,\tilde{a}^\top$. This chapter establishes a deterministic (nearest/round-to-even) baseline for the backprop-only activation quantization; subsequent chapters extend to stochastic rounding, which removes rounding bias and further stabilizes low-bit training while retaining the same memory benefits.

4.2 Method: Activation Quantization for Backpropagation

4.2.1 Design

The approach follows the backpropagation quantization principle (Chapter 2): keep forward compute unchanged, but *store/consume* quantized activations for gradient computation. Concretely:

- Quantizer: uniform, per-group scaling (bucketed min/max) to reduce quantization error; default rounding is nearest/round-to-even unless otherwise specified.
- Bit-widths: evaluate 1–32 bit; focus on 2-bit, 4-bit, and 2-bit + 4-bit mixes.
- Where applied: to saved activations needed by backward in linear, convolution, batch normalization, and custom affine/scale ops.

Design rationale: this chapter establishes a deterministic baseline for activation quantization during backpropagation. Per-bucket scaling improves local dynamic-range utilization while keeping metadata overhead bounded (one scale/zero-point per bucket). Two storage policies are compared (save-quantized vs. quantize-on-backward) to separate pure memory savings from ablation-centric measurements.

4.2.2 Mathematical Formulation for Backpropagation

Let $a \in \mathbb{R}^n$ denote the saved forward activations of a layer. Partition the elements into disjoint groups $\{\mathcal{G}_g\}_{g=1}^{G}$ ("buckets"). For group g, compute an affine uniform quantizer with bit-width b, integer range $[q_{\min}, q_{\max}] = [0, 2^b - 1]$, scale $s_g > 0$, and zero-point $z_g \in \mathbb{Z}$:

$$s_g = \frac{\max_{i \in \mathcal{G}_g} a_i - \min_{i \in \mathcal{G}_g} a_i}{q_{\max} - q_{\min}}, \qquad z_g = \operatorname{round}\left(q_{\min} - \frac{\min_{i \in \mathcal{G}_g} a_i}{s_g} \right).$$

For any $i \in \mathcal{G}_g$, define the code preimage $u_i = a_i / s_g + z_g$. In this chapter we use *deterministic* nearest (round-to-even) rounding:

$$q_i = \operatorname{clip}\big(\operatorname{round}(u_i),\ q_{\min},\ q_{\max}\big), \qquad \tilde{a}_i = s_g\,(q_i - z_g).$$

In the backward pass for layer l with error signal $\delta^{(l)}$ and previous-layer activations $a^{(l-1)}$, the weight gradient uses the *quantized* saved activations $\tilde{a}^{(l-1)}$:

$$\nabla_{W^{(l)}} L \;\approx\; \delta^{(l)} \left(\tilde{a}^{(l-1)}\right)^{\top}, \qquad \nabla_{b^{(l)}} L \;=\; \delta^{(l)}.$$

Per-group scaling reduces quantization variance within each bucket, enabling low average bit-widths while preserving training stability in this deterministic setting.

4.2.3 Implementation with Brevitas

The implementation integrates Brevitas' activation quantizer into custom autograd `Functions`. Two complementary patterns are used:

1. **Save-quantized**: quantize activation in forward solely for saving (reduces memory), while propagating the full-precision output forward.

2. **Quantize-on-backward**: save FP activation but quantize it at the start of backward (no forward-time overhead; useful for ablations and accuracy studies).

   Both patterns are compatible with standard PyTorch training loops and can be selected per layer depending on the memory/overhead trade-off being studied.

Below is concise core usage from the custom layers (QuantIdentity with low-bit activations):

```
from brevitas.nn import QuantIdentity

quant_input = QuantIdentity(bit_width=2, return_quant_tensor=False).
    to(grad_output.device)
quantized_input = quant_input(input)

input = quantized_input
grad_weight = torch.matmul(grad_output.mT, input)
```

Listing 4.1: Linear: injecting a low-bit activation quantizer for backward

```
from brevitas.nn import QuantIdentity

quant_input = QuantIdentity(bit_width=2, return_quant_tensor=False).
    to(grad_out.device)
quantized_input = quant_input(input)

grad_input = G.conv2d_input(
    input_size=quantized_input.shape,
    weight=weight,
    grad_output=grad_out,
    stride=stride,
    padding=padding,
    dilation=dilation,
    groups=groups,
)
grad_weight = G.conv2d_weight(
    input=quantized_input,
    weight_size=weight.shape,
    grad_output=grad_out,
    stride=stride,
    padding=padding,
    dilation=dilation,
    groups=groups,
)
```

Listing 4.2: Convolution: using quantized activations in gradient computation

```
quant_input = QuantIdentity(bit_width=2, return_quant_tensor=False).
    to(grad_output.device)
quantized_input = quant_input(input)
input = quantized_input
```

Listing 4.3: BatchNorm: feeding quantized activations to the backward formulas

```
quant_input = QuantIdentity(bit_width=2, return_quant_tensor=False).
    to(grad_output.device)
quantized_input = quant_input(input)
```

Listing 4.4: Affine/scale: backward uses quantized activations

4.2.4 Forward Activation Storage

Two storage modes are supported:

- **Save-quantized (memory-optimized)**: in forward, compute a quantized copy $a_q = Q(a)$ for saving (`ctx.save_for_backward(a_q, ...)`) and free the FP a. Backward consumes a_q. This minimizes activation footprint and follows activation-compressed training [7, 22].
- **Quantize-on-backward (ablation)**: save FP activation and quantize at the start of backward (code above). Useful for controlled comparisons; can be combined with checkpointing to reduce saved FP windows.

4.3 Stability: Eliminating Test-Accuracy Spikes

Occasional test accuracy drops ("spikes") during training at low bit-widths are observed. These are addressed with five orthogonal optimization strategies (warmup included):

1. **Learning-rate scheduling**: cosine decay with restarts or step decay.
2. **Warmup phases**: linear warmup over initial epochs stabilizes early dynamics.
3. **Weight decay**: decoupled weight decay (AdamW) proved most effective under strong quantization.
4. **Gradient clipping**: per-parameter norm clipping to limit outliers.
5. **Label smoothing**: small smoothing (e.g., 0.05) to regularize targets.

Across ablations, **weight decay** is consistently the most effective lever, approaching **90%** test accuracy under the same setup while significantly suppressing spikes.

4.4 Experiments on EfficientNetV2 / CIFAR-10

This section reports empirical results that instantiate the design above on EfficientNetV2 and CIFAR-10, covering global bit-width sweeps, the 7-bit stability case, and comparative optimization strategies.

4.4.1 Bit-width Sweep (1 to 32 Bits)

EfficientNetV2 is trained on CIFAR-10 with activation bit-widths {1,2,3,4,5,6,7,8,16,32}. The plot summarizes accuracy vs. epochs and final accuracy vs. bit-width. **Finding.** Accuracy degrades and becomes unstable at **very low**

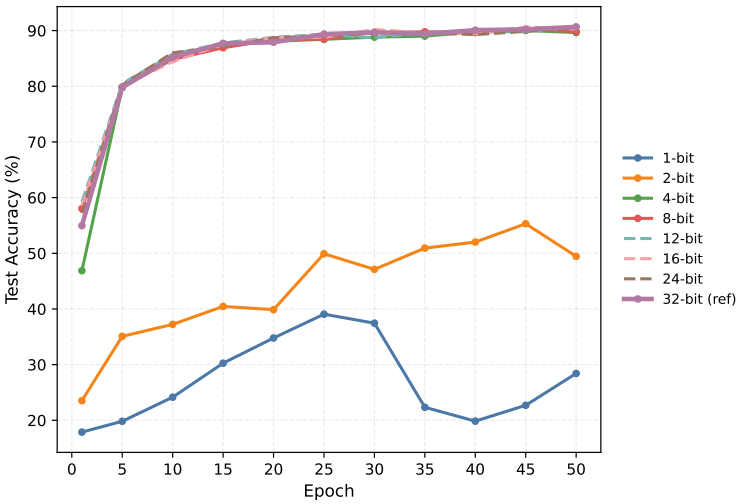

Figure 4.1: EfficientNetV2 on CIFAR-10 with activation bit-widths from 1 to 32. Accuracy across bit-widths under activation quantization.

precision (1–2 bit) but stabilizes at 4-bit and above. This establishes a baseline trade-off between memory savings (monotone with lower bits) and accuracy. Combined with the memory model above, 4-bit typically yields $\sim 8\times$ activation-memory reduction with competitive accuracy, making it a practical default for backprop-only activation quantization on CNNs like EfficientNetV2.

4.4.2 Fixing the 7-bit Spike

At 7 bits, a repeatable mid-training test-accuracy drop is observed. With the four strategies above, spikes disappear and final accuracy improves. **Finding.** Training-time optimization (especially weight decay and warmup) removes the transient instability at 7-bit and restores accuracy close to the FP32 reference.

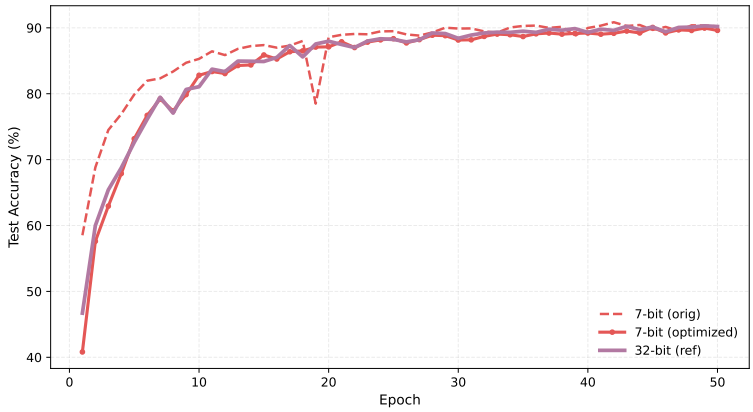

Figure 4.2: 7-bit case on EfficientNetV2/CIFAR-10: test-accuracy spikes can occur at certain bit-widths during training; applying the training-time optimizations (learning-rate scheduling, warmup, weight decay, gradient clipping/label smoothing) eliminates the spikes and recovers accuracy, approaching the 32-bit reference.

4.4.3 Optimization Strategies: Comparative Analysis (7-bit, with Warmup)

Building on the stability observations, this thesis compares four major optimization strategies under the 7-bit setting with *warmup enabled for all configurations*: learning-rate scheduling, weight decay (AdamW), gradient clipping, and label smoothing. The comparison uses a consistent training schedule and reports full-epoch trajectories alongside the 32-bit reference and the unoptimized 7-bit baseline.

Results indicate that **weight decay** provides the best stabilization of training dynamics and mitigates accuracy degradation associated with low-bit activation quantization, while the other individual strategies (scheduler, label smoothing, gradient clipping) underperform in this setting.

Finding. Among individual interventions, decoupled weight decay (AdamW) most consistently improves stability and final accuracy at low bits; the other single levers yield smaller gains.

4.4.4 Layer-wise Activation Quantization

Layer-wise perturbation studies on EfficientNetV2 are performed to measure global accuracy sensitivity to each layer's activation bit-width, informing mixed-precision policies and aligning with the network organization in [19].

Concretely, this thesis considers the EfficientNetV2 backbone as a sequence of stages (stem + MBConv/FCU blocks + head). For each stage (and representative internal layers), only that layer's activations are quantized (others kept at reference precision), training/finetuning is conducted under the same schedule, and the test accuracy trajectory

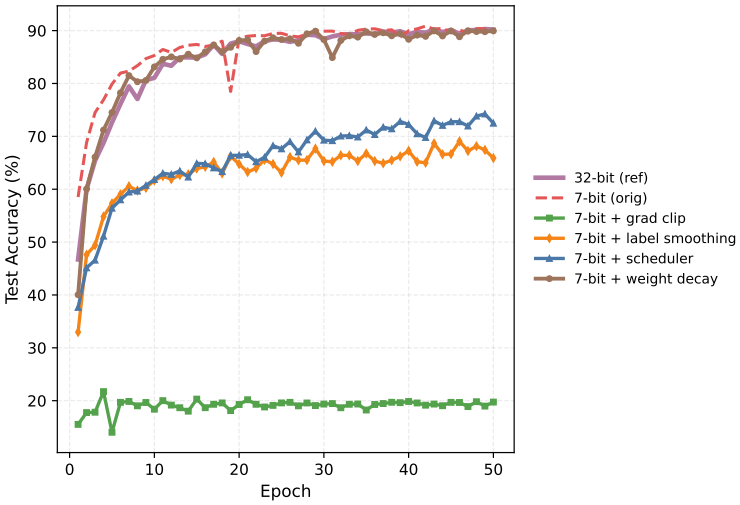

Figure 4.3: Comparison of four optimization strategies at 7-bit (all with warmup), against 7-bit baseline and 32-bit reference. Weight decay yields the most stable training and the highest final accuracy among the single-strategy variants. Legend is placed outside on the right to avoid overlap.

and final accuracy are recorded. This isolates the marginal impact of per-layer activation quantization on end-to-end accuracy.

Method highlights:

- Per-layer activation quantizer injected via the custom autograd Functions (Section 4) using Brevitas' `QuantIdentity`.
- Bit-width primarily set to 4-bit for the perturbation runs to mirror the practical deployment candidate.
- Training schedule aligned across layers to ensure comparability; metrics aggregated over the last epochs.

Empirical result: For EfficientNetV2 on CIFAR-10, the results indicate **low layer-wise sensitivity** of test accuracy to activation quantization: quantizing individual layers (at 4-bit) does not yield consistent accuracy improvements nor cause significant degradations compared to the reference. This suggests that, for this network/dataset setting, **no single layer dominates** the activation quantization sensitivity, and simple uniform or coarse mixed-precision schemes are sufficient.

The layer/block organization of EfficientNetV2 used throughout follows the official description [19].

4.4.5 Activation Memory Savings (FP32 vs 4-bit)

Using the EfficientNetV2-M configuration on CIFAR-10 (batch size 128), the total number of saved-activation elements across main blocks sums to 12,976,128. Under FP32 (4 bytes/element), this corresponds to approximately 49.5 MB per batch, whereas under 4-bit storage (0.5

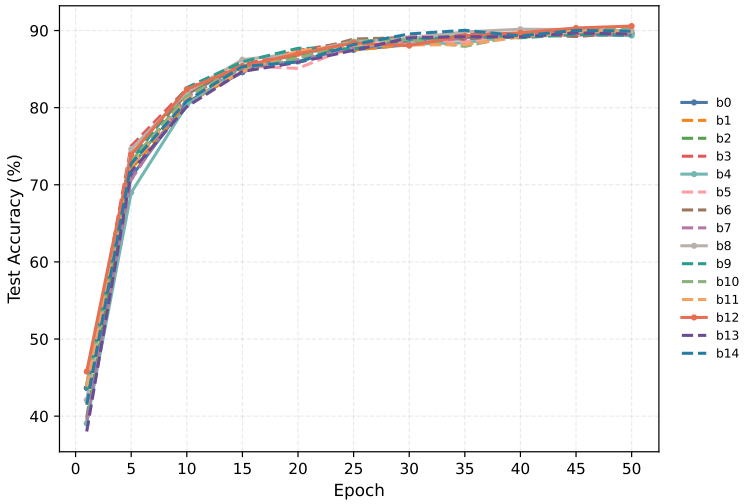

Figure 4.4: Per-layer (b0–b14) 4-bit activation quantization on EfficientNetV2/CIFAR-10. Curves are sampled every 5 epochs; legend is placed outside on the right for readability.

bytes/element), it is approximately 6.2 MB. The difference indicates an **87.5%** reduction in saved-activation memory when moving from FP32 to 4-bit, aligning with the motivation that activations dominate training-time memory.

Table 4.1: EfficientNetV2 activation memory by block (per batch). FP32 uses 4 bytes per element; 4-bit uses 0.5 bytes per element. MB computed with 1 MB=1024^2 bytes.

Block	Elements	FP32 (MB)	4-bit (MB)	Saved (MB)
Stem	3,145,728	12.00	1.50	10.50
MB1	3,145,728	12.00	1.50	10.50
MB2	1,572,864	6.00	0.75	5.25
MB3	655,360	2.50	0.31	2.19
MB4	327,680	1.25	0.16	1.09
MB5	557,056	2.12	0.27	1.86
MB6	786,432	3.00	0.38	2.62
Head	2,621,440	10.00	1.25	8.75
Classifier	163,840	0.62	0.08	0.55
Total	12,976,128	49.50	6.19	43.31

This block-level view is intended to guide mixed-precision design: larger activation tensors (e.g., early high-resolution stages) offer greater absolute savings when quantized, while later narrower stages contribute less to the peak footprint. Subsequent experiments therefore focus on low-bit activations for the backward pass where these savings materialize during training.

Finding. Moving from FP32 to 4-bit reduces saved-activation mem-

Table 4.2: Activation memory summary (EfficientNetV2, CIFAR-10, per batch).

Quantity	Value (MB)	Note
FP32 total	49.5	32-bit (4 bytes/elem)
4-bit total	6.2	0.5 bytes/elem
Saved	43.3	Reduction 87.5%

ory by roughly $8\times$ (here, about 87.5%), enabling larger batch sizes or operation on smaller GPUs without altering forward compute.

4.5 Summary

This thesis implements low-bit activation quantization on activations saved for backward via Brevitas, introduces storage modes to reduce training memory, and provides a deterministic baseline demonstrating that CNN training can be conducted with *much lower activation memory* while retaining competitive accuracy (e.g., $\sim 8\times$ reduction at 4-bit). We further present optimization techniques that stabilize low-bit training and a layer-wise study confirming that sensitivity is broadly distributed across layers rather than concentrated in a few. These results support mixed-precision activation quantization during backpropagation as a practical path to training on smaller GPUs [7, 22].

5 STOCHASTIC QUANTIZATION OF ACTIVATIONS FOR BACKPROPAGATION

5.1 Motivation and Concept

Activation quantization for the backward pass (Chapter 4) reduces training memory, but *deterministic* nearest rounding at very low bit-widths is **biased**: the dequantized value $\tilde{a} = Q(a)$ generally satisfies $\mathbb{E}[\tilde{a} \mid a] \neq a$. When such $\tilde{a}$ enters gradient formulas (e.g., $\nabla_W L \approx \delta\, \tilde{a}^\top$), rounding bias can propagate and accumulate across layers/steps, degrading stability and final accuracy.

This motivates adopting **stochastic activation quantization** with *unbiased* rounding: within the representable range, $\mathbb{E}[\tilde{a} \mid a] = a$, removing systematic bias while keeping variance controlled by the step size and local scaling. With per-bucket affine scaling, the quantization step shrinks to match local dynamic range, reducing variance and clip probability. The *hypothesis* is that, at **equal bit-width**, stochastic activation quantization yields more stable optimization and **higher final accuracy** than deterministic rounding, especially in the 2–4 bit regime, while retaining the same memory savings.

This chapter formalizes bucketed stochastic activation quantization, details straight-through gradient propagation, and provides a *vectorized* implementation suitable for custom autograd Functions. The hypothesis is evaluated by (i) sweeping bit-widths (1–32) with stochastic quantization, (ii) directly comparing **stochastic vs. deterministic** at fixed low bits, and (iii) studying bucket size and simple mixed-precision variants. Empirical results support the unbiased-stochastic design as a superior choice over deterministic rounding under low precision [1, 7, 11].

5.2 Deterministic vs. Stochastic Quantization

Deterministic quantization maps a value to the nearest level and is generally biased: $\mathbb{E}[Q(x)] \neq x$ for distributed inputs. Stochastic quantization instead selects neighboring levels with probabilities to ensure $\mathbb{E}[Q(x)] = x$. For uniform quantization with step Δ, a canonical rule is

$$Q(x) = \begin{cases} \lfloor x/\Delta \rfloor\, \Delta & \text{w.p. } 1-\alpha, \\ \lceil x/\Delta \rceil\, \Delta & \text{w.p. } \alpha, \end{cases} \qquad \alpha = x/\Delta - \lfloor x/\Delta \rfloor,$$

which yields unbiasedness and better stability at low precision when quantized values participate in gradient computations [11]. Stochastic

gradient quantization is also well studied in communication-efficient training (e.g., QSGD) [1].

5.3 Stochastic Activation Quantization: Design

This thesis adopts stochastic activation quantization with per-bucket scaling to reduce local dynamic range. The design exposes the following hyperparameters (names match the implementation):

- **base_levels**: number of quantization levels for the dominant precision (e.g., 4 for 2-bit; $\log_2$(base_levels) is the bit-width).
- **bucket**: bucket size for local scale estimation; activations are partitioned into contiguous chunks of this length for per-bucket scaling. Smaller buckets adapt better to heterogeneity but increase metadata/overhead.
- **use_max**: scale rule for each bucket; *True* uses ℓ_∞ (max) norm; *False* uses ℓ_2 norm. The ℓ_∞ rule is simple and robust; ℓ_2 can be smoother but may under-scale outliers.
- **mix_levels**: optional higher-precision levels (e.g., 16 for 4-bit, 64 for 6-bit) for mixed-precision stochastic quantization.
- **mix_levels_prob**: probability in $[0, 1]$ of using `mix_levels` instead of `base_levels` per bucket (or per tensor in a global variant). This enables coarse stochastic mixing between two bit-widths.

This stochastic, per-bucket scheme is complementary to prior activation-compression systems that use per-group scaling [7, 22] and to few-bit derivative proxying [15].

5.3.1 Mathematical Formulation of Stochastic Quantization

Let $a \in \mathbb{R}^n$ denote the saved forward activations of some layer. Partition the index set into disjoint buckets $\{\mathcal{G}_g\}_{g=1}^{G}$. For bucket g, define a uniform affine quantizer with bit-width b and integer range $[q_{\min}, q_{\max}] = [0, 2^b - 1]$. The per-bucket scale and zero-point are

$$s_g = \frac{\max_{i \in \mathcal{G}_g} a_i - \min_{i \in \mathcal{G}_g} a_i}{q_{\max} - q_{\min}}, \qquad z_g = \operatorname{round}\left(q_{\min} - \frac{\min_{i \in \mathcal{G}_g} a_i}{s_g} \right).$$

For element $i \in \mathcal{G}_g$, let $u_i = a_i / s_g + z_g$, $k_i = \lfloor u_i \rfloor$, and $\alpha_i = u_i - k_i \in [0, 1)$. The *unbiased stochastic rounding* selects

$$q_i = \operatorname{clip}\left(k_i + \operatorname{Bernoulli}(\alpha_i),\ q_{\min},\ q_{\max}\right), \qquad \tilde{a}_i = s_g \left(q_i - z_g\right).$$

When clipping does not activate, $\mathbb{E}[\tilde{a}_i \mid a_i] = a_i$; hence the dequantized value is an unbiased estimator of the original activation within range [1, 11]. In practice, choosing buckets that match local statistics reduces the probability of clipping and the variance of the quantization noise.

During backpropagation, gradients with respect to layer parameters use the quantized saved activations $\tilde{a}$. For a linear layer $y = Wa + b$, the gradients are computed as

$$\nabla_W L \approx (\nabla_y L)\,(\tilde{a})^\top, \qquad \nabla_b L = \nabla_y L.$$

The gradient through the quantizer adopts a straight-through estimator (STE), passing unit gradient within the representable range and zero outside [3]:

$$\frac{\partial \tilde{a}_i}{\partial a_i} \approx \mathbb{1}\Big\{q_{\min} \le \mathrm{round}(u_i) \le q_{\max}\Big\}.$$

This combination—per-bucket affine scaling, unbiased stochastic rounding, and STE—matches the implementation studied in this contribution and aligns with prior systems using per-group activation compression [7, 15, 22].

5.4 Implementation

A custom autograd `Function` realises vectorized stochastic activation quantization, and a thin `Module` wrapper eases integration. The forward path quantizes activations using per-bucket scales and stochastic rounding; the backward of the quantizer is the straight-through estimator.

Stochastic Quantization Kernel (Vectorized) The core kernel eliminates Python loops and performs bucket-level stochastic decisions and rounding in a batched, vectorized way:

```
# Vectorized decision for each bucket's levels
bucket_rand = torch.rand(n_buckets, device=x.device)
levels_per_bucket = torch.where(bucket_rand < mix_levels_prob,
    mix_levels, base_levels)

# Vectorized scale computation
if use_max:
    scales = reshaped.abs().max(dim=1, keepdim=True)[0]
else:
    scales = reshaped.norm(p=2, dim=1, keepdim=True)
scales = scales.clamp(min=1e-8)

# Vectorized stochastic quantization
ax = (reshaped.abs() / scales).clamp(0, 1 - 1e-8)
levels_expanded = levels_per_bucket.unsqueeze(1).float()
idx = torch.floor(ax * levels_expanded)
frac = ax * levels_expanded - idx
go_up = (torch.rand_like(frac) < frac).float()
q = (idx + go_up) / levels_expanded
quantized = q * scales * reshaped.sign()
```

Listing 5.1: Vectorized bucketed stochastic quantization kernel

A module wrapper exposes the hyperparameters with sensible defaults for downstream layers:

```
class OptimizedStochQuantIdentity(torch.nn.Module):
    def __init__(self,
                 base_levels: int = 4,
                 bucket: int = 512,
                 use_max: bool = True,
                 mix_levels: int = 16,
                 mix_levels_prob: float = 0.3):
        super().__init__()
        self.base_levels = base_levels
        self.bucket = bucket
        self.use_max = use_max
        self.mix_levels = mix_levels
        self.mix_levels_prob = mix_levels_prob
    def forward(self, x: Tensor) -> Tensor:
        return OptimizedStochActQuant.apply(
            x, self.base_levels, self.bucket, self.use_max, self.
                mix_levels, self.mix_levels_prob
        )
```

Listing 5.2: Module wrapper for stochastic activation quantization

Integration into Layers The stochastic quantizer is injected in the backward-sensitive pathways of common layers (linear, convolution, batch normalization, and simple affine/scale ops). For example, in batch normalization the saved activation is quantized before gradient computation:

```
stoch_quant = OptimizedStochQuantIdentity(
    base_levels=ctx.base_levels,
    bucket=ctx.bucket,
    use_max=ctx.use_max,
    mix_levels=ctx.mix_levels,
    mix_levels_prob=ctx.mix_levels_prob
).to(grad_output.device)

quantized_input = stoch_quant(input)
# --- End Stochastic Quantization ---
input = quantized_input # Use quantized input for backward
    calculation
```

Listing 5.3: BatchNorm: feeding quantized activations to the backward formulas

Similar hooks appear in convolution and linear functions to ensure gradients use quantized activations:

```
stoch_quant = OptimizedStochQuantIdentity(
    base_levels=ctx.base_levels,
    bucket=ctx.bucket,
    use_max=ctx.use_max,
    mix_levels=ctx.mix_levels,
    mix_levels_prob=ctx.mix_levels_prob
```

```
).to(grad_out.device)

quantized_input = stoch_quant(input)
```

Listing 5.4: Hooking the quantizer inside convolution backward paths

5.5 Hyperparameter Study

Experiments are conducted on EfficientNetV2/CIFAR-10 under the same training schedule as Chapter 4 unless stated otherwise. The study varies one factor at a time:

- **Bit-width (base_levels)**: 2-bit, 4-bit. Lower bits yield larger memory savings but require stronger regularization (weight decay) to maintain stability.
- **Bucket size**: {64, 128, 256, 512}. Smaller buckets adapt to local statistics and slightly improve accuracy at the cost of more metadata; larger buckets are cheaper but may under-scale heterogeneous activations.
- **Scale rule (use_max vs ℓ_2)**: Max-norm scaling is robust to outliers and showed consistent stability; ℓ_2 can be marginally better when activations are well-behaved.
- **Mixed precision (mix_levels, mix_levels_prob)**: Occasional promotion (e.g., 2-bit base with 4-bit mix at small probabilities) improves stability during difficult phases without sacrificing most of the memory gains.

Across settings, the combination of unbiased stochastic rounding [11] and weight decay remains the most reliable recipe for stable convergence at low precision. These findings align with prior observations that unbiased quantization preserves optimization properties [1].

5.6 Runtime Optimization: Vectorization and Global Variant

To reduce overhead, the stochastic quantizer is fully vectorized and avoids Python loops. Bucket-level scale computation and rounding decisions are computed in parallel, yielding substantial speedups (10–100 × as observed empirically in kernel microbenchmarks). In scenarios where per-bucket scaling is unnecessary, a global variant removes bucketing entirely and quantizes using a single scale for the tensor:

```
class GlobalStochActQuant(Function):
    @staticmethod
    def forward(ctx, x: Tensor, base_levels: int, mix_levels: int,
        mix_levels_prob: float) -> Tensor:
```

```
        scale = x.abs().max().clamp(min=1e-8)
        use_mix = torch.rand(1, device=x.device) < mix_levels_prob
        levels = mix_levels if use_mix else base_levels
        ax = (x.abs() / scale).clamp(0, 1 - 1e-8)
        idx = torch.floor(ax * levels)
        frac = ax * levels - idx
        go_up = (torch.rand_like(frac) < frac).float()
        q = (idx + go_up) / levels
        return q * scale * x.sign()
```

Listing 5.5: Global variant for stochastic quantization

5.7 Arbitrary Two-bit Mixing for Stochastic Quantization

To enable mixed-precision research without redesigning kernels, the implementation supports *arbitrary mixing of two bit-widths* through the pair `(base_levels, mix_levels)` and a probability knob `mix_levels_prob`. The decision can be made per bucket (local scaling variant) or per tensor (global variant), preserving flexibility for different granularity choices.

Key aspects:

- **Bit-to-levels mapping**: a utility converts bit-widths to levels; configuring a 2/4-bit mix becomes a one-line change.
- **Granularity**: per-bucket mixing adapts locally; global mixing applies one choice to the full tensor for minimal overhead.
- **Scheduling**: `mix_levels_prob` may be constant or scheduled (e.g., higher in early epochs to ease optimization, decayed later to favor the lower bit).
- **Research utility**: by isolating the mixing policy from the kernel, the design accelerates exploration of mixed-precision strategies across layers and training phases.

Examples of ready-made configurations include 2/4, 2/6, 3/6, and 4/8-bit mixes:

```
QUANT_CONFIGS = {
    '2bit_4bit_mix': {'base_levels': 4, 'mix_levels': 16, '
        mix_levels_prob': 0.5},
    '2bit_6bit_mix': {'base_levels': 4, 'mix_levels': 64, '
        mix_levels_prob': 0.3},
    '3bit_6bit_mix': {'base_levels': 8, 'mix_levels': 64, '
        mix_levels_prob': 0.4},
    '4bit_8bit_mix': {'base_levels': 16, 'mix_levels': 256, '
        mix_levels_prob': 0.2},
}
```

Listing 5.6: Example quantization configuration presets

Custom mixes can be specified using the helper to convert bit-widths to levels:

```python
def levels_to_bits(levels: int) -> int:
    return int(torch.log2(torch.tensor(levels)).item())

def bits_to_levels(bits: int) -> int:
    return 2 ** bits
```

Listing 5.7: Helpers for mapping between levels and bits

In practice, a 2/4-bit mix with bucket-level decisions is obtained by setting `base_levels=bits_to_levels(2)`, `mix_levels=bits_to_levels(4)`, and the desired `mix_levels_prob`. This modular control facilitates later chapters and follow-up work on layer-wise mixed-precision policies.

5.8 Experiments

This section outlines the experimental evaluation based on stochastic activation quantization. Unless otherwise noted, experiments use EfficientNetV2 on CIFAR-10 with the same training schedule as Chapter 4. Plots are organized to progressively answer: (i) how stochastic quantization behaves across 1–32 bits, (ii) how much 2-bit benefits from stochastic rounding, (iii) how bucket size impacts 2-bit performance and how it compares with a non-stochastic (deterministic) Brevitas baseline, and (iv) how mixing 4-bit into a 2-bit base (with different probabilities) trades stability/accuracy vs. memory.

5.8.1 Bit-Width Sweep with Stochastic Quantization (1 to 32 Bits)

A full sweep using stochastic quantization across the restricted set {1,2,4,8,12,16,24,32} bits to visualize convergence and final accuracy as a function of precision.

5.8.2 2-bit: Stochastic vs. Deterministic

At 2-bit precision, unbiased stochastic rounding substantially improves stability and final accuracy relative to a deterministic (nearest) quantizer. **Finding.** At 2-bit precision, unbiased stochastic rounding consistently outperforms deterministic rounding in final test accuracy and training stability, narrowing the gap to the 32-bit reference.

5.8.3 2-bit/4-bit: Bucket Size Study and Comparison with a Non-stochastic Baseline

The impact of bucket size $\in \{64, 128, 256, 512\}$ is evaluated for 2-bit stochastic quantization and compared with a non-stochastic Brevitas

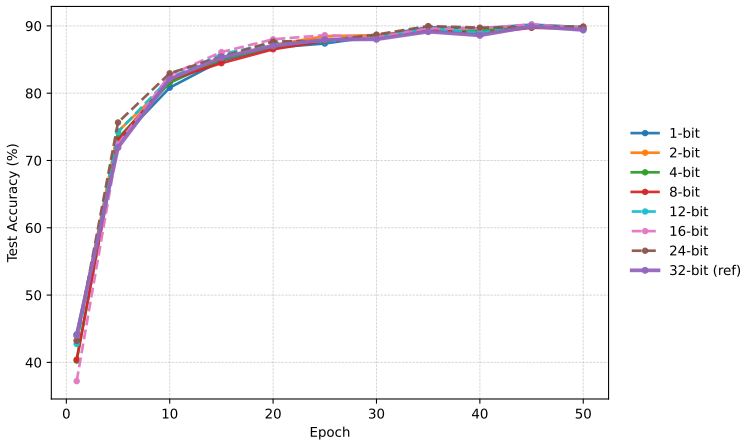

Figure 5.1: Stochastic activation quantization from 1 to 32 bits (restricted to {1,2,4,8,12,16,24,32}) on EfficientNetV2/CIFAR-10. Legend placed to the right to avoid overlap.

realization at the same nominal precision. **Finding.** Decreasing bucket size improves scale accuracy and yields higher test accuracy at 2-bit, with a mild overhead trade-off for smaller buckets.
Finding. The same trend holds at 4-bit: smaller buckets (finer local scaling) translate into better accuracy, while larger buckets reduce overhead at a small accuracy cost.

5.8.4 2/4-bit Mixed Stochastic Quantization (Fixed Bucket=512)

Using 2-bit as the base, mix 4-bit at probabilities $p \in \{0.2, 0.5, 0.8\}$ with bucket size fixed to 512. Curves are compared against a 32-bit reference and a 2-bit deterministic baseline to show the stability/accuracy trade-off introduced by occasional promotion to 4-bit. **Finding.** Mixed precision improves accuracy over pure 2-bit: increasing the 4-bit mixing probability steadily raises accuracy toward the 4-bit curve, at the cost of higher average precision (memory).

5.9 Summary

This contribution introduces stochastic activation quantization with unbiased rounding for the backward pass, details a per-bucket design with practical hyperparameters, integrates the method into common layers, and optimizes the implementation via vectorization. The approach builds on established theory for unbiased quantization [1, 11] and aligns with recent systems leveraging per-group activation compression [5, 7, 15].

Empirical findings from this chapter can be summarized as follows:

- **Stochastic vs. deterministic at low bits**: At 2-bit precision, unbiased stochastic rounding improves training stability and final

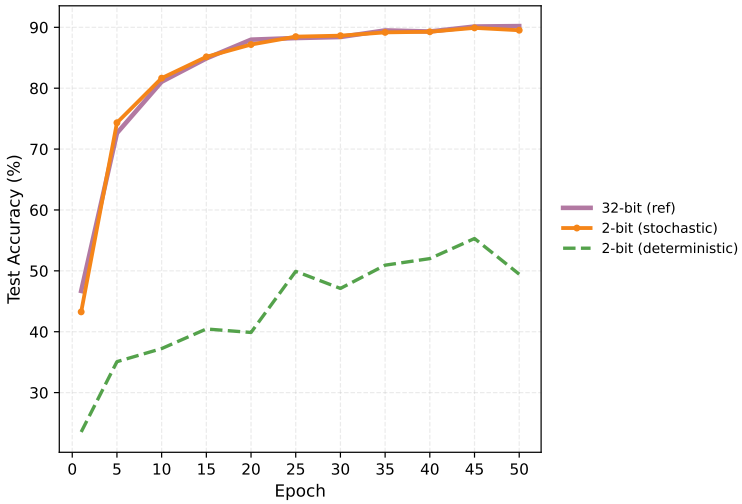

Figure 5.2: 2-bit comparison: stochastic vs. deterministic activation quantization with 32-bit reference (legend on the right). Stochastic rounding improves training stability and accuracy at very low precision.

accuracy relative to deterministic rounding, reducing the gap to the 32-bit reference.

- **Bucket-size trade-off**: Smaller buckets provide more accurate per-bucket scaling and higher accuracy (both at 2-bit and 4-bit), at the cost of increased metadata and compute overhead; larger buckets reduce overhead with a small accuracy penalty.
- **Mixed precision benefits**: Mixing a higher bit-width (e.g., 4-bit) into a low-bit base (2-bit) improves accuracy in proportion to the mixing probability, offering a tunable balance between accuracy and memory/precision.

Overall, stochastic quantization of backward activations is effective for stabilizing low-bit training and, when combined with weight decay, delivers competitive accuracy while retaining substantial memory savings demonstrated in Chapter 4. Mixed-precision and bucket-size selection offer practical knobs to trade accuracy against overhead, enabling flexible policies for different deployment constraints.

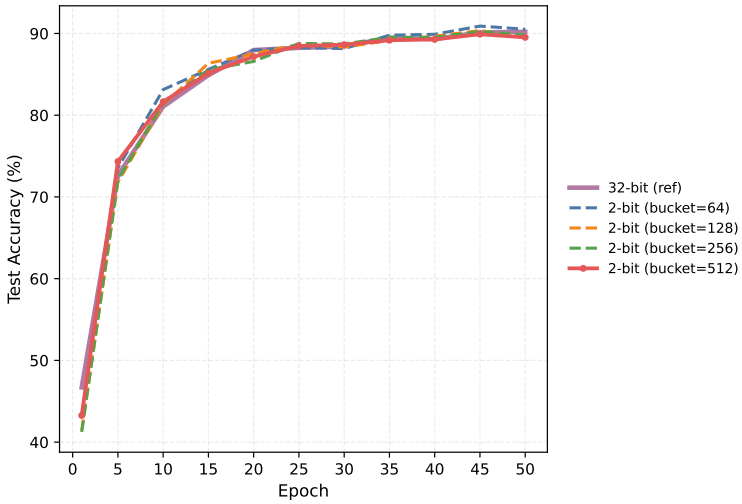

Figure 5.3: 2-bit bucket size study (64, 128, 256, 512) with stochastic quantization vs. 32-bit reference. Smaller buckets adapt better (at added overhead); larger buckets are cheaper but less adaptive.

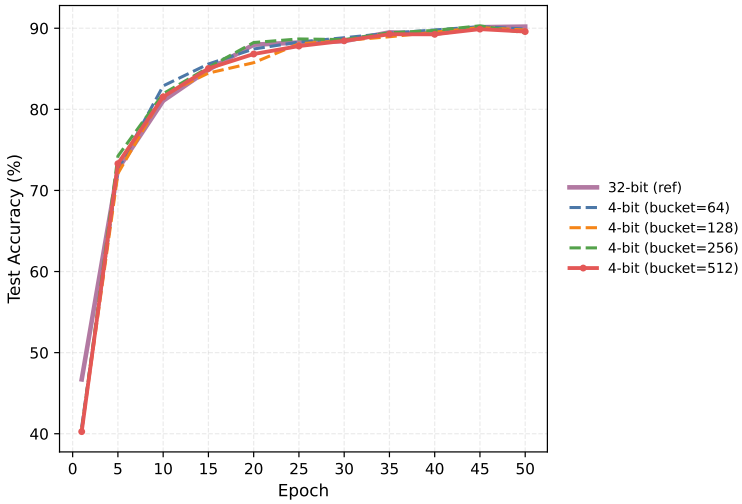

Figure 5.4: 4-bit bucket size study (64, 128, 256, 512) with stochastic quantization vs. 32-bit reference, as a companion to the 2-bit analysis.

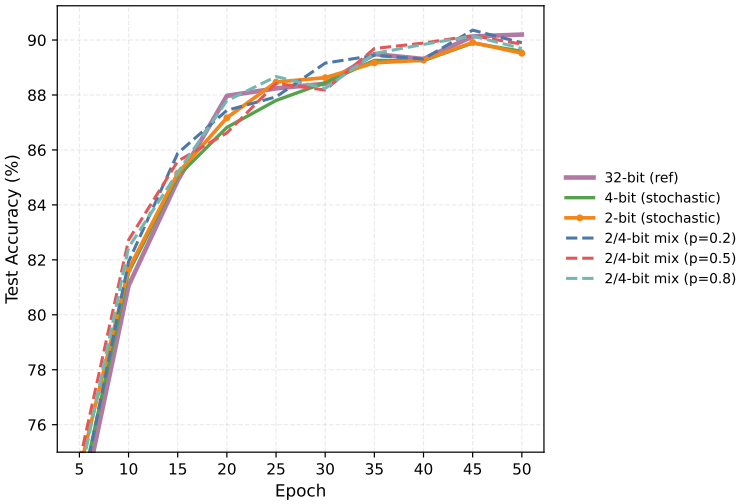

Figure 5.5: 2/4-bit mixed stochastic quantization at mixing probabilities 0.2/0.5/0.8 (bucket=512), compared against 2-bit, 4-bit stochastic baselines and 32-bit reference. Curves are sampled every 5 epochs (from epoch 5) with y-axis clipped at 75% for clarity.

6 GENERALIZATION OF STOCHASTIC ACTIVATION QUANTIZATION

6.1 Motivation and Concept

The previous chapters established that stochastic activation quantization stabilizes low-bit training and offers strong accuracy–memory trade-offs on CIFAR-10. This chapter investigates *generalization* to larger-scale settings by applying the same methodology to CIFAR-100 and ImageNet-200. The objective is to rigorously assess whether unbiased stochastic rounding, per-bucket scaling, and simple mixed-precision policies preserve their advantages as data complexity, input resolution, and class cardinality increase.

From a learning-theoretic perspective, an *unbiased* activation quantizer is expected to induce gradients whose expectation matches the full-precision counterpart, thereby preventing systematic drift that would otherwise accumulate across layers and epochs. The working hypothesis tested here is that this property is *dataset-agnostic*: as the label space grows from 10 to 100 and 200 classes and visual variability increases (Tiny ImageNet), unbiased stochastic activation quantization should maintain optimization stability and keep the accuracy gap to the 32-bit reference small at moderate bit-widths (2–4 bits).

Beyond accuracy, the experiments highlight the practical importance of stochastic activation quantization for training-time memory reduction. Because activations dominate the memory footprint during back-propagation, replacing full-precision activations with their stochastic low-bit surrogates can reduce memory by multiples while avoiding bias in the backward pass. Demonstrating robust behavior on CIFAR-100 and Tiny ImageNet (ImageNet-200) underscores that the approach is not dataset-specific but a general mechanism for memory-efficient training.

Concretely, this chapter evaluates:

- **Robustness at low precision** (2–4 bits) relative to deterministic rounding, focusing on both final accuracy and the smoothness of optimization trajectories.
- **Sensitivity to scaling policy** through fixed bucketed scaling and its metadata trade-offs, checking that benefits persist under realistic implementation constraints.
- **Closeness to the 32-bit reference** across datasets, quantified by the mean and maximum (best) test accuracy per bit-width.

The following sections report results on CIFAR-100 and Tiny ImageNet (ImageNet-200), providing direct deterministic vs. stochastic comparisons and summaries across bit-widths.

6.2 Datasets and Models

Experiments use CIFAR-100 (100 classes, 32×32 images) and Tiny ImageNet (ImageNet-200; 200 classes, 64×64 resolution). EfficientNetV2 is adopted as the backbone, with minor adjustments for image size and width/depth where appropriate (consistent with the configurations used in prior chapters).

6.3 Training Settings

Unless stated otherwise, training follows the schedules validated on CIFAR-10: AdamW with weight decay, a learning-rate schedule (cosine or OneCycle for larger datasets), label smoothing, and gradient clipping for stability. For stochastic quantization, the following configuration is used: bucket size of 512, max scaling rule, 50 epochs training, learning rate of 0.001, and batch size of 128. Base bit-widths range from 1 to 32 bits. All comparisons include a 32-bit reference.

6.4 Baselines and Metrics

Baselines include: (i) 32-bit reference, (ii) deterministic (nearest) quantization at matched bit-widths, and (iii) stochastic quantization without mixing. Metrics are top-1 test accuracy and training stability (absence of spikes), with memory savings reported from nominal bit-widths and bucket metadata.

6.5 Results on CIFAR-100

6.5.1 Bit-Width Sweep (Deterministic vs. Stochastic, 1–32 Bits)

Deterministic (Brevitas) vs. Stochastic on CIFAR-100. The two plots report, respectively, the deterministic (Brevitas) and the stochastic (unbiased rounding) bit-width sweeps on CIFAR-100:

- **Deterministic (Brevitas)**: At 1–2 bits, test accuracy is markedly low and the training trajectory exhibits instability. From 4-bit upward, accuracy improves but remains below the 32-bit reference both along the trajectory and at convergence.
- **Stochastic (unbiased rounding)**: Curves are uniformly elevated. In the 2–4-bit regime, accuracy is substantially higher than

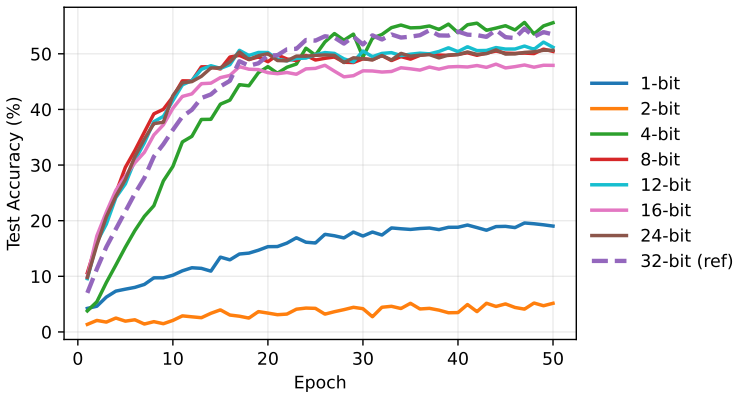

Figure 6.1: CIFAR-100 (Brevitas, deterministic rounding): bit-width sweep sampled every 5 epochs; legend on the right with 32-bit marked as reference.

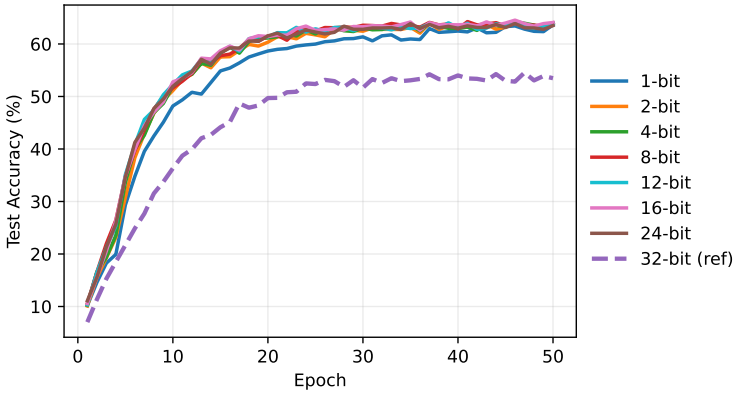

Figure 6.2: CIFAR-100 (stochastic, unbiased rounding): bit-width sweep across 1–32 bits; accuracy vs. epochs and final accuracy summary.

the deterministic counterpart. Beyond the ultra-low-bit corner, **most bit-widths closely track and are above the 32-bit (ref)**, which shows the stochastic approach is effective on larger datasets and training is noticeably smoother.

Overall, **stochastic activation quantization performs strongly on CIFAR-100**: at matched bit-widths it delivers higher final accuracy and more stable optimization than deterministic rounding. These results support the hypothesis that unbiased stochastic rounding, coupled with per-bucket scaling, generalizes beyond CIFAR-10 and remains effective on larger, more challenging datasets.

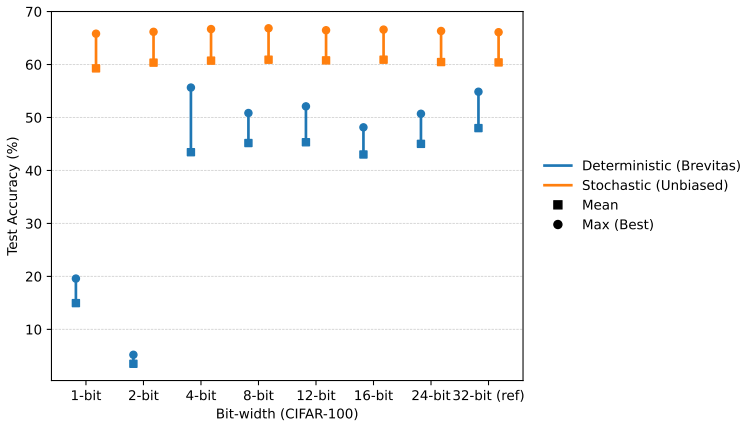

Figure 6.3: CIFAR-100 deterministic (Brevitas) vs. stochastic (unbiased) comparison per bit-width. For each method and bit, the *vertical segment* spans from the epoch-wise mean accuracy to the maximum (best) accuracy; the square marks the mean and the circle marks the maximum. Legend is placed on the right.

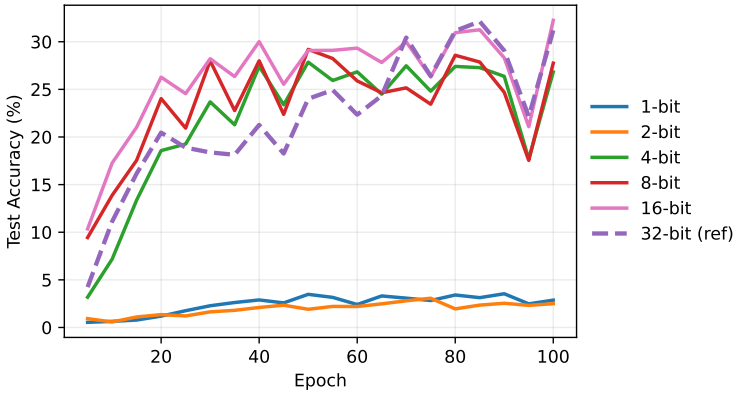

Figure 6.4: Tiny ImageNet (ImageNet-200) with deterministic activation quantization across 1–32 bits.

6.6 Results on Tiny ImageNet (ImageNet-200)

6.6.1 Bit-Width Sweep and 2-bit Comparison

Finding. Tiny ImageNet on EfficientNetV2 exhibits noisy training dynamics: the test-accuracy trajectories fluctuate noticeably across epochs and bit-widths. This reflects that the FP32 baseline on this dataset/architecture is itself not fully stabilized, so absolute gaps between adjacent bit-widths are less reliable than on CIFAR-100. Even with the instability, higher bit-widths (e.g., 16/32) generally peak above the lowest-bit settings, consistent with expectations.

Finding. The stochastic variant shows similar oscillatory behavior, driven largely by the same baseline/optimization sensitivity on Tiny ImageNet. While the curves are less stable than on CIFAR-100, stochastic activation quantization remains competitive with the deterministic

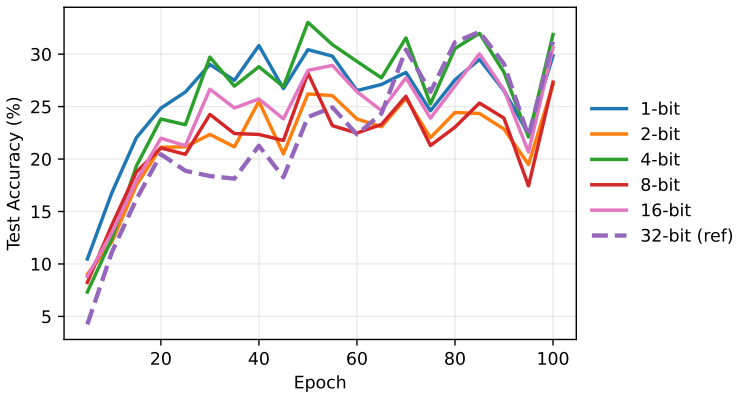

Figure 6.5: Tiny ImageNet (ImageNet-200) with stochastic activation quantization across 1–32 bits.

counterpart at moderate/high bits and avoids collapse at low bits; the 32-bit reference shares the same fluctuations, pointing to dataset/architecture-level instability rather than a quantization-specific artifact.

6.7 Discussion

Across datasets, the quantitative picture is consistent with the thesis goals:

- **Deterministic (Brevitas) at 4–16 bits** already performs strongly: on CIFAR-100 it is often close to the 32-bit baseline while offering large *training-time memory savings* (activations dominate backprop memory). This shows that conventional quantization is a practical drop-in for moderate precisions.
- **Stochastic activation quantization** delivers remarkable performance improvements: not only does it close the gap to the 32-bit reference at matched bit-widths, but strikingly, *low-bit configurations (2–4 bit) even exceed the unquantized 32-bit baseline accuracy on CIFAR-100*. This surprising result demonstrates that stochastic quantization can achieve superior accuracy while dramatically reducing memory usage during training—a compelling demonstration of the method's effectiveness on larger, more challenging datasets.
- **Generalization.** The above effects persist beyond CIFAR-10: they hold on CIFAR-100 and are observable on Tiny ImageNet (ImageNet-200). The Tiny ImageNet curves are less stable, likely due to optimization sensitivity of EfficientNetV2 on this dataset and a not-fully-stable FP32 baseline, so absolute gaps are noisier; nevertheless the overall trend (stochastic $\geq$ deterministic, and moderate bits $\approx$ 32-bit) remains evident.

Overall, the results support stochastic activation quantization as a *general* recipe for memory-efficient training with minimal accuracy loss,

and often with accuracy gains in the ultra-low-bit regime. Importantly, this thesis *pioneers the application of stochastic quantization to saved activations during the backward pass*, a previously unexplored direction that demonstrates substantial training accuracy improvements. The significant performance gains, particularly the ability of low-bit quantized models to exceed unquantized baselines, underscore the profound research value and potential of this novel approach for future deep learning optimization.

7 DISCUSSION AND OUTLOOK

7.1 Summary and Outlook

This thesis advances activation quantization *for the backpropagation stage* along two complementary directions and validates the resulting methods empirically:

- **Deterministic activation quantization for backpropagation** (Chapter 4): formulates the problem, implements a practical Brevitas-based pipeline, and shows substantial training-time memory reductions while retaining competitive accuracy on EfficientNetV2.
- **Stochastic activation quantization with per-bucket scaling** (Chapter 5): removes rounding bias, stabilizes low-bit training (notably at 2–4 bits), and improves accuracy relative to deterministic rounding at matched precision.

The third contribution (Chapter 6) evaluates *generalization* on CIFAR-100 and Tiny ImageNet (ImageNet-200), confirming that the stochastic approach remains beneficial beyond CIFAR-10, while Tiny ImageNet exhibits higher trajectory noise attributable to baseline and optimization sensitivity.

7.1.1 Quantitative Highlights

Across chapters, several patterns recur:

- **Low-bit stability and accuracy.** On CIFAR-10 and CIFAR-100, 2–4 bit *stochastic* activation quantization consistently raises accuracy and smooths optimization relative to deterministic rounding at the same bit-width (see Chapters 4 and 5). At moderate bits (e.g., 4/8/12/16), trajectories closely track the 32-bit reference.
- **Backprop-only memory savings.** Quantizing only the saved activations reduces training-time activation memory approximately in proportion to bit-width ($\sim 8\times$ at 4-bit; $\sim 16\times$ at 2-bit), without altering forward computation. This enables larger batch sizes or training on smaller GPUs (Chapter 4).
- **Per-bucket scaling.** Bucketed min/max scaling lowers quantization variance and clipping; smaller buckets benefit low-bit regimes at a modest metadata cost (Chapter 5).

- **Mixed precision (arbitrary two-bit).** The implementation supports *arbitrary two-bit stochastic mixing* via `(base_levels, mix_levels)` with a probability parameter `mix_levels_prob`, enabling tunable accuracy–memory trade-offs without kernel changes (Chapter 5).
- **Engineering efficiency.** A vectorized kernel removes Python loops and performs bucket-level decisions in parallel, substantially reducing the overhead of the quantization step.

7.1.2 Activation Memory Savings: FP32 vs 4-bit vs 2-bit (Stochastic)

Table 7.1 summarizes the activation-memory reduction on EfficientNetV2 when replacing FP32 saved activations with 4-bit deterministic and 2-bit stochastic variants for the backward pass. Results align with the theoretical ratios ($\sim 8\times$ at 4-bit; $\sim 16\times$ at 2-bit) and the empirical measurements presented in Chapter 4.

Table 7.1: Activation memory comparison on EfficientNetV2 (per batch). Lower is better.

Precision	**Bits**	**Rel. mem.**	**Example (MB)**
FP32 reference	32	$1.0\times$	~ 49.5
Deterministic (Brevitas)	4	$\approx 0.125\times$	~ 6.2
Stochastic (unbiased)	2	$\approx 0.0625\times$	~ 3.1

7.1.3 Limitations

On Tiny ImageNet (ImageNet-200) we observe noisier trajectories even at the 32-bit reference. This points to sensitivity to optimizer/schedule and backbone choice (EfficientNetV2), rather than a quantization-specific artifact. While stochastic quantization remains competitive, absolute gaps across adjacent bit-widths should be interpreted with caution in this setting.

7.1.4 Future Work

Building on these contributions, three directions are promising:

1. **Stability on larger datasets**: strengthen stochastic quantization on larger or noisier datasets through (i) schedule search (e.g., warmup length; cosine vs. OneCycle), (ii) weight-decay tuning and gradient clipping, (iii) adaptive bucket sizing or EMA-based scale updates, and (iv) backbone-specific adjustments.
2. **Broader model coverage**: evaluate generalization across additional CNN families (e.g., ResNet/ConvNeXt/MobileNet) and

extend to ViT/Transformer backbones by quantizing backward-sensitive tensors in attention and MLP blocks; report accuracy–memory trade-offs across tasks.

3. **Richer framework capabilities**: extend the current design (arbitrary two-bit mixing) to *multi-bit* mixing with automatic per-layer bit selection under memory/accuracy constraints. An AutoML-style selector could assign (2/3/4/6/8)-bit per block to maximize memory savings while preserving accuracy.

a | APPENDIX

BIBLIOGRAPHY

[1] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. "QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017. URL: https://arxiv.org/abs/1610.02132.

[2] Daniel Barley and Holger Fr"oning. "Less Memory Means Smaller GPUs: Backpropagation with Compressed Activations." In: *arXiv preprint arXiv:2409.11902* (2024). URL: https://arxiv.org/abs/2409.11902.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation." In: *arXiv preprint arXiv:1308.3432*. 2013. URL: https://arxiv.org/abs/1308.3432.

[4] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. "signSGD: Compressed Optimisation for Non-Convex Problems." In: *International Conference on Machine Learning (ICML)*. 2018.

[5] Davide Cai and contributors. *Brevitas: Quantization-Aware Training in PyTorch*. GitHub repository. 2019. URL: https://github.com/Xilinx/brevitas.

[6] Ayan Chakrabarti and Benjamin Moseley. "Backprop with Approximate Activations for Memory-Efficient Network Training." In: *arXiv preprint arXiv:1901.07988* (2019). URL: https://arxiv.org/abs/1901.07988.

[7] Jianfei Chen, Lianmin Zheng, Zhewei Yao, Dequan Wang, Ion Stoica, Michael W. Mahoney, and Joseph E. Gonzalez. "ActNN: Reducing Training Memory Footprint via 2-Bit Activation Compressed Training." In: *International Conference on Machine Learning*. 2021. URL: https://arxiv.org/abs/2104.14129.

[8] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. "Training Deep Nets with Sublinear Memory Cost." In: *arXiv preprint arXiv:1604.06174* (2016). URL: https://arxiv.org/abs/1604.06174.

[9] Xiaohan Dong, Sheng Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. "HAWQ: Hessian Aware Quantization of Neural Networks with Mixed-Precision." In: *ICCV*. 2019.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: https://www.deeplearningbook.org/.

[11] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and P. Narayanan. "Deep Learning with Limited Numerical Precision." In: *International Conference on Machine Learning (ICML)*. 2015. URL: https://arxiv.org/abs/1502.02551.

[12] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference." In: *arXiv preprint arXiv:1712.05877* (2018). URL: https://arxiv.org/abs/1712.05877.

[13] Raghuraman Krishnamoorthi. "Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper." In: *arXiv preprint arXiv:1806.08342* (2018). URL: https://arxiv.org/abs/1806.08342.

[14] Paulius Micikevicius et al. "Mixed Precision Training." In: *arXiv preprint arXiv:1710.03740* (2017). URL: https://arxiv.org/abs/1710.03740.

[15] Georgii Novikov, Daniel Bershatsky, Julia Gusak, Alex Shonenkov, Denis Dimitrov, and Ivan Oseledets. "Few-Bit Backward: Quantized Gradients of Activation Functions for Memory Footprint Reduction." In: *arXiv preprint arXiv:2202.00441* (2022). URL: https://arxiv.org/abs/2202.00441.

[16] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W. Keckler. "vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design." In: *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2016. DOI: 10.1109/MICRO.2016.7783721.

[17] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0.

[18] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. "1-bit Stochastic Gradient Descent and its Application to Data-parallel Distributed Training of Speech DNNs." In: *INTERSPEECH*. 2014.

[19] Mingxing Tan and Quoc V. Le. "EfficientNetV2: Smaller Models and Faster Training." In: *arXiv preprint arXiv:2104.00298* (2021).

[20] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. "HAQ: Hardware-Aware Automated Quantization." In: *CVPR*. 2019.

[21] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. "TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[22] Lianmin Zheng, Jianfei Chen, Zhewei Yao, Dequan Wang, Joseph E. Gonzalez, Michael W. Mahoney, and Ion Stoica. "GACT: Activation Compressed Training for Generic Network Architectures." In: *arXiv preprint arXiv:2206.11357* (2022). URL: https://arxiv.org/abs/2206.11357.