# ActNN: Reducing Training Memory Footprint via 2-Bit Activation Compressed Training

**Jianfei Chen** [*1] **Lianmin Zheng** [*1] **Zhewei Yao** [1] **Dequan Wang** [1]
**Ion Stoica** [1] **Michael W. Mahoney** [1] **Joseph E. Gonzalez** [1]

## Abstract

The increasing size of neural network models has been critical for improvements in their accuracy, but device memory is not growing at the same rate. This creates fundamental challenges for training neural networks within limited memory environments. In this work, we propose ActNN, a memory-efficient training framework that stores randomly quantized activations for back propagation. We prove the convergence of ActNN for general network architectures, and we characterize the impact of quantization on the convergence via an exact expression for the gradient variance. Using our theory, we propose novel mixed-precision quantization strategies that exploit the activation's heterogeneity across feature dimensions, samples, and layers. These techniques can be readily applied to existing dynamic graph frameworks, such as PyTorch, simply by substituting the layers. We evaluate ActNN on mainstream computer vision models for classification, detection, and segmentation tasks. On all these tasks, ActNN compresses the activation to 2 bits on average, with negligible accuracy loss. ActNN reduces the memory footprint of the activation by $12\times$, and it enables training with a $6.6\times$ to $14\times$ larger batch size. We implement ActNN as a PyTorch library at https://github.com/ucbrise/actnn.

## 1. Introduction

Within the last three years, state-of-the-art machine learning models have become over 4,000 times larger (Devlin et al., 2018; Fedus et al., 2021). On the other hand, the memory capacity of GPUs has increased relatively slowly, remaining on the order of tens of gigabytes. This creates a fundamental barrier to the development and training of neural networks.

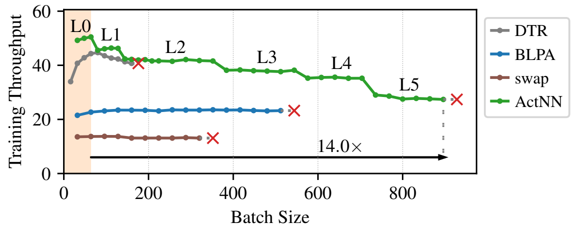


*Figure 1.* Batch size vs. training throughput on ResNet-152. Red cross mark means out-of-memory. The shaded yellow region denotes the possible batch sizes with full precision training. ActNN achieves significantly larger maximum batch size over other state-of-the-art systems and displays a nontrivial trade-off curve.

Activation compressed training (ACT) is a promising approach to reduce the training memory footprint (Chakrabarti & Moseley, 2019; Fu et al., 2020). During training, all layers' activations need to be kept in the memory for computing the gradients. ACT saves memory by compressing activations to lower numerical precision via quantization. It was proposed in BLPA (Chakrabarti & Moseley, 2019), and it was later extended by TinyScript (Fu et al., 2020) with non-uniform quantization strategies. These prior works succeeded in training ResNet-50 with 4-bit activations.

However, applications of ACT are hindered by several drawbacks. First, the convergence behavior of ACT methods is not well understood (except for an analysis for multi-layer perceptrons (Fu et al., 2020), under strong mean-field assumptions (Yang & Schoenholz, 2017)). Second, prior works mostly focus on dedicated architectures, e.g., a customized version of the pre-activation ResNet (He et al., 2016b), limiting their generality. Third, existing quantization strategies are not specifically designed for ACT, making their compression ratio suboptimal.

In this work, we propose ActNN, a framework for ACT that overcomes all the challenges. ActNN stores randomly quantized activations to compute the gradients. Theoretically, we view the gradient computed by ActNN ("ActNN gradient") as a stochastic approximation of the gradient computed with the full-precision activation ("FP gradient"). We show that the ActNN gradient is an unbiased estimator of the FP gradi-

[*]Equal contribution [1]UC Berkeley. Correspondence to: Jianfei Chen <jianfeic@berkeley.edu>, Lianmin Zheng <lmzheng@berkeley.edu>.

*Proceedings of the 38th International Conference on Machine Learning*, PMLR 139, 2021. Copyright 2021 by the author(s).

ent, and we prove ActNN's convergence *for general model architectures*. This enables one to apply ACT to general problems with theoretical guarantees.

We characterize the impact of quantization on the convergence via an exact expression for the gradient variance. Better quantization strategies reduce the gradient variance, and can achieve satisfactory convergence with fewer bits. Inspired by the theory, we design novel quantization strategies to exploit activations' heterogeneity across feature dimensions, samples, and layers. This includes a per-group quantizer and a fine-grained mixed precision algorithm, which approximately minimizes the gradient variance under a given memory budget. ActNN tunes the quantization strategy on-the-fly. On a wide range of tasks, including image classification, semantic segmentation, and object detection, ActNN compresses activations to 2 bits, with negligible ($< 0.5\%$) accuracy loss. ActNN even converges and produces reasonable results with only 1.25-bit activations. This improves significantly from prior work (Chakrabarti & Moseley, 2019; Fu et al., 2020), which only converges with 4 bits.

We implement our method as a library based on PyTorch. The library consists of a collection of activation compressed layers. Memory saving training can be achieved with simply layer substitution, e.g., replace `torch.nn.Conv2d` with `actnn.Conv2d`. The library also provides several optimization levels to exploit the trade-off between memory saving and training speed. In practice, ActNN reduces the activation memory by $12\times$, enabling training with a $6.6\times$ to $14\times$ larger batch size on the same GPU. We compare ActNN with existing systems, where ActNN achieves a much larger batch size (Fig. 1). ActNN also enables training larger models without additional computational resources. With a fixed amount of memory, ActNN scales the training of ResNet to $6.4\times$ deeper, or $3.7\times$ wider, or $3.1\times$ higher resolution.

To summarize, our contributions are in three folds:

1. A general convergence theory for ACT;
2. An heterogeneity-aware quantization strategy that achieves 2-bit compression;
3. An efficient implementation of activation compressed layers in PyTorch.

## 2. Related Works

**Quantized Training (QT)** Quantization-aware training (Zhou et al., 2016; Choi et al., 2018; Zhang et al., 2018; Jacob et al., 2018; Dong et al., 2019) or fully-quantized training (Micikevicius et al., 2018; Wang et al., 2018b; Chen et al., 2020a; Sun et al., 2020) aim to reduce the computational cost with quantization at the inference or training time. As a side effect, the training memory footprint can also be reduced. However, QT is a more challenging task to solve, as *computational* kernels must directly support quantized tensors. In contrast, ACT only considers the *storage*, and it can utilize more flexible quantization strategies for better compression. Furthermore, QT and ACT are complementary. One can utilize QT to accelerate the training, and apply ACT to further reduce the memory footprint.

**Model / Gradient Compression** Model compression (Han et al., 2016) and gradient compression (Lin et al., 2018) compress the weight and gradient to reduce the storage and communication overhead. However, activations have different properties with the weight and gradient, e.g., it has a "sample" axis. Moreover, ACT is more sensitive to the compression speed, as activations need to be compressed on-the-fly, and they are typically much larger than weights and gradients. ActNN's compression strategy is designed specifically for these unique properties.

**Memory-Efficient Training Systems** Gradient checkpointing (Chen et al., 2016; Jain et al., 2019; Shah et al., 2020; Kirisame et al., 2020) trades computation for memory by dropping some of the activations in the forward pass and recomputing them in the backward pass. Swapping (Meng et al., 2017; Huang et al., 2020; Wang et al., 2018a; Peng et al., 2020; Ren et al., 2021) utilizes the huge amount of available CPU memory by swapping tensors between CPU and GPU. Model-parallel training (Shoeybi et al., 2019; Lepikhin et al., 2020; Wang et al., 2019) partitions the model across GPUs, so each GPU only stores a fraction of layers. All these methods save memory by storing fewer tensors in GPU. In contrast, ActNN compresses saved tensors, and is complementary to these approaches.

## 3. Formulation and Theory

In this section, we present a mathematical formulation of ActNN. Then, we establish its convergence by viewing it as a special case of stochastic gradient descent (SGD). The proofs of all theorems as well as a table of notations can be found in Appendix A.

### 3.1. Problem Formulation

Consider training an $L$-layer neural network on a dataset $\mathcal{D}$. In each training iteration, we sample a minibatch $(\mathbf{X}, \mathbf{Y})$ from the dataset. Given the input $\mathbf{H}^{(0)} = \mathbf{X}$, the $l$-th layer of the network is defined in a general form

$$\mathbf{H}^{(l)} = \mathbf{F}^{(l)}\left(\mathbf{H}^{(l-1)}; \mathbf{\Theta}^{(l)}\right), \tag{1}$$

where $\mathbf{H}^{(l)}$ is a $N \times D^{(l)}$-dimensional feature map, $N$ is the batch size, $D^{(l)}$ is the number of features, and $\mathbf{\Theta}^{(l)}$ is a vector of parameters. Given the minibatch loss $\mathcal{L} = l(\mathbf{H}^{(L)}, \mathbf{Y})$, we compute the gradient $\nabla_{\mathbf{\Theta}^{(l)}}\mathcal{L}$, and update the parameter with SGD (Bottou, 2010). Since the gradient is always taken with the loss $\mathcal{L}$, we simply denote the activation / parameter gradient as $\nabla_{\mathbf{H}^{(l)}}$ and $\nabla_{\mathbf{\Theta}^{(l)}}$. To compute the gradient, the

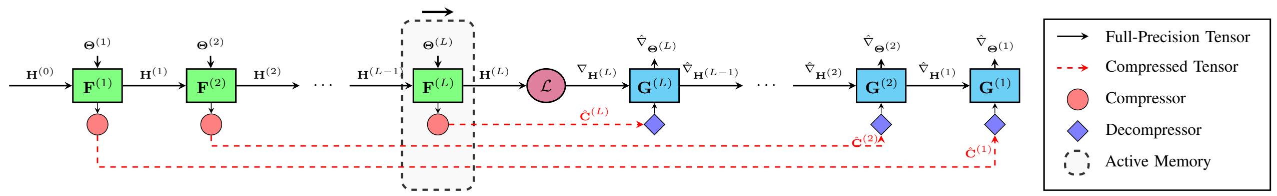


*Figure 2.* ActNN's computational graph. Nodes: operations; Edges: tensors. Edges that intersect with the dashed box are kept in memory.

back-propagation can be expressed as

$$\nabla_{\mathbf{H}^{(l-1)}}, \nabla_{\mathbf{\Theta}^{(l)}} = \mathbf{G}^{(l)}\left(\nabla_{\mathbf{H}^{(l)}}, \mathbf{C}(\mathbf{H}^{(l-1)}, \mathbf{\Theta}^{(l)})\right), \quad (2)$$

where $\mathbf{C}(\cdot)$ is the *context*, i.e., the information that needs to be kept in memory for back propagation. Essentially, the function $\mathbf{G}^{(l)}(\cdot)$ takes the gradient of the output $\nabla_{\mathbf{H}^{(l)}}$ and the context, and computes the gradient of the input. We refer this approach as full-precision (FP) training, and $\nabla_{\mathbf{H}^{(l)}}$ and $\nabla_{\mathbf{\Theta}^{(l)}}$ as the FP gradient. As a special case, consider a linear layer $\mathbf{H}^{(l)} = \mathbf{H}^{(l-1)}\mathbf{\Theta}^{(l)}$ and its gradient

$$\nabla_{\mathbf{H}^{(l-1)}} = \nabla_{\mathbf{H}^{(l)}}\mathbf{\Theta}^{(l)\top}, \quad \nabla_{\mathbf{\Theta}^{(l)}} = \mathbf{H}^{(l-1)\top}\nabla_{\mathbf{H}^{(l)}}. \quad (3)$$

In this case, we have $\mathbf{C}(\mathbf{H}^{(l-1)}, \mathbf{\Theta}^{(l)}) = (\mathbf{H}^{(l-1)}, \mathbf{\Theta}^{(l)})$.

## 3.2. Activation Compressed Training

The context, in particular the activation, dominants the memory overhead for training neural networks on many tasks. To address this, instead of saving the full-precision context, ActNN saves a compressed version $\hat{\mathbf{C}}(\mathbf{H}^{(l-1)}, \mathbf{\Theta}^{(l)})$. In principle, any compression algorithm, either lossy or lossless, can be used here. In this paper, however, we solely focus on compressing by quantizing the context to lower numerical precision, since its overhead is relatively small. In this way, the compressed context $\hat{\mathbf{C}}$ is a lower precision version of the context $\mathbf{C}$. With the compressed context, we define the activation-compressed (AC) gradient as:

$$\hat{\nabla}_{\mathbf{H}^{(l-1)}}, \hat{\nabla}_{\mathbf{\Theta}^{(l)}} = \mathbf{G}^{(l)}\left(\hat{\nabla}_{\mathbf{H}^{(l)}}, \hat{\mathbf{C}}(\mathbf{H}^{(l-1)}, \mathbf{\Theta}^{(l)})\right), \quad (4)$$

where $\hat{\nabla}_{\mathbf{H}^{(L)}} = \nabla_{\mathbf{H}^{(L)}}$. ActNN uses the AC gradient to update parameters. See Fig. 2 for an illustration. Notice that, FP training and ActNN share the same forward propagation Eq. (1), so their behavior is identical at inference time.

## 3.3. Convergence Theory

Now we study the convergence of ActNN. Assume that $\hat{\mathbf{C}}$ is quantized randomly, such that $\hat{\mathbf{C}}$ can be viewed as a stochastic estimator of $\mathbf{C}$. In this way, both FP and ActNN can be considered as SGD algorithms, with different stochastic gradients. Formally, let $\mathbf{\Theta}_t = \{\mathbf{\Theta}^{(l)}\}_{l=1}^L$ be a flattened vector of parameters at the $t$-th iteration, and $\nabla_{\mathbf{\Theta}_t} = \{\nabla_{\mathbf{\Theta}_t^{(l)}}\}_{l=1}^L$ / $\hat{\nabla}_{\mathbf{\Theta}_t} = \{\hat{\nabla}_{\mathbf{\Theta}_t^{(l)}}\}_{l=1}^L$ be the corresponding FP / AC gradient, defined as Eq. (2) / Eq. (4). Furthermore, let $\mathcal{L}_\mathcal{D}(\mathbf{\Theta})$ be the batch loss on the entire dataset. Then, both $\nabla_{\mathbf{\Theta}}$ and $\hat{\nabla}_{\mathbf{\Theta}}$ are stochastic estimators of the batch gradient $\nabla_{\mathbf{\Theta}}\mathcal{L}_\mathcal{D}(\mathbf{\Theta})$. The stochasticity of the FP gradient $\nabla_{\mathbf{\Theta}}$ comes solely from random sampling of the minibatch, which we assume to be unbiased, i.e., $\mathbb{E}[\nabla_{\mathbf{\Theta}}] = \nabla_{\mathbf{\Theta}}\mathcal{L}_\mathcal{D}(\mathbf{\Theta})$. On the other hand, the stochasticity of the AC gradient $\hat{\nabla}_{\mathbf{\Theta}}$ further comes from the random quantization of the context. The question is whether the AC gradient can be made unbiased as well, and for this, the answer is positive.

**Theorem 1.** *(Unbiased Gradient) There exists random quantization strategies for* $\hat{\mathbf{C}}$, *such that*

$$\mathbb{E}\left[\hat{\nabla}_{\mathbf{\Theta}}\right] = \nabla_{\mathbf{\Theta}}\mathcal{L}_\mathcal{D}(\mathbf{\Theta}).$$

Intuitively, according to the chain rule, the back-propagation Eq. (2) can be rewritten as

$$\nabla_{H_{ij}^{(l-1)}} = \sum_{kl}\frac{\partial H_{kl}^{(l)}}{\partial H_{ij}^{(l-1)}}\nabla_{H_{kl}^{(l)}}, \nabla_{\Theta_i^{(l)}} = \sum_{kl}\frac{\partial H_{kl}^{(l)}}{\partial \Theta_i^{(l)}}\nabla_{H_{kl}^{(l)}}. \quad (5)$$

Take $\hat{\mathbf{C}}(\mathbf{H}^{(l-1)}, \mathbf{\Theta}^{(l)}) = Q(\{\partial H_{kl}^{(l)}/\partial H_{ij}^{(l-1)}, \partial H_{kl}^{(l)}/\partial\Theta_i^{(l)}\})$, where $Q(\cdot)$ is an unbiased quantizer. As Eq. (5) is just a linear operation, we can show that $\hat{\nabla}_{\mathbf{H}^{(l-1)}}$ and $\hat{\nabla}_{\mathbf{\Theta}^{(l)}}$ are unbiased as long as $\hat{\mathbf{C}}(\mathbf{H}^{(l-1)}, \mathbf{\Theta}^{(l)})$ and $\hat{\nabla}_{\mathbf{H}^{(l)}}$ are unbiased, which can be proven by induction. The linear layer $\hat{\nabla}_{\mathbf{\Theta}^{(l)}} = Q(\mathbf{H}^{(l-1)})^\top\hat{\nabla}_{\mathbf{H}^{(l)}}$ is especially simple, where we can just use $Q(\mathbf{H}^{(l-1)})$ as the compressed context. General layers are more complicated as directly storing the Jacobian matrices $\{\partial H_{kl}^{(l)}/\partial H_{ij}^{(l-1)}, \partial H_{kl}^{(l)}/\partial\Theta_i^{(l)}\}$ might be prohibitive. However, we show in Appendix B that for most frequently used layers, including convolution, pointwise, normalization, and up/down sampling, can be approximated in an unbiased way with a practical cost.

Given an unbiased gradient, we now establish the convergence of ActNN. Assume the SGD iteration takes the form $\mathbf{\Theta}_{t+1} \leftarrow \mathbf{\Theta}_t - \alpha\hat{\nabla}_{\mathbf{\Theta}_t}$, starting from an initial model $\mathbf{\Theta}_1$, and

**A1.** The loss $\mathcal{L}_\mathcal{D}(\mathbf{\Theta})$ is continuous differentiable and $\nabla\mathcal{L}_\mathcal{D}(\mathbf{\Theta})$ is $\beta$-Lipschitz continuous.
**A2.** $\mathcal{L}_\mathcal{D}(\mathbf{\Theta})$ is bounded below by $\mathcal{L}_{inf}$.
**A3.** There exists $\sigma^2 > 0$, such that $\forall\mathbf{\Theta}$, $\mathrm{Var}\left[\hat{\nabla}_{\mathbf{\Theta}}\right] \le \sigma^2$, where for any vector $\mathbf{x}$, $\mathrm{Var}[\mathbf{x}] := \mathbb{E}\|\mathbf{x}\|^2 - \|\mathbb{E}[\mathbf{x}]\|^2$.

The following convergence theorem is a standard result for SGD, taken from Theorem 4.8 in Bottou et al. (2018).

**Theorem 2.** *(Convergence) If A1-A3 holds, and* $0 < \alpha \le \frac{1}{\beta}$, *take the number of iterations t uniformly from* $\{1, \ldots, T\}$, *where* $T$ *is a maximum number of iterations. Then*

$$\mathbb{E}\|\nabla\mathcal{L}_\mathcal{D}(\mathbf{\Theta}_t)\|^2 \le \frac{2(\mathcal{L}(\Theta_1) - \mathcal{L}_{inf})}{\alpha T} + \alpha\beta\sigma^2. \quad (6)$$

Eq. (6) is composed of two terms. The first term converges to zero as the number of iterations $T$ goes to infinity, while the second term does not. Intuitively, the algorithm converges to the neighborhood of a stationary point, where the radius is controlled by the gradient variance. Note that unlike the previous work (Fu et al., 2020), the convergence of ActNN is established for general network architectures, not just for multi-layer perceptrons.

### 3.4. Gradient Variance

According to Thm. 2, gradient variance plays a critical role to the quality of the converged solution. We investigate how does the quantization affect the variance, so we can design quantization strategies accordingly. Let $\mathbf{G}_{\mathbf{H}}(\cdot)$ and $\mathbf{G}_{\boldsymbol{\Theta}}(\cdot)$ be components of $\mathbf{G}(\cdot)$, corresponding to $\nabla_{\mathbf{H}}$ and $\nabla_{\boldsymbol{\Theta}}$. For simplicity, let $\mathbf{C}^{(l)}$ and $\hat{\mathbf{C}}^{(l)}$ be the full-precision and compressed context. Further, define $\mathbf{G}_{\boldsymbol{\Theta}}^{(l\sim m)}\left(\hat{\nabla}_{\mathbf{H}^{(m)}}, \hat{\mathbf{C}}^{(m)}\right)$

$$= \mathbf{G}_{\boldsymbol{\Theta}}^{(l)}\left(\mathbf{G}_{\mathbf{H}}^{(l+1)}\left(\cdots \mathbf{G}_{\mathbf{H}}^{(m)}\left(\hat{\nabla}_{\mathbf{H}^{(m)}}, \hat{\mathbf{C}}^{(m)}\right)\cdots, \mathbf{C}^{(l+1)}\right), \mathbf{C}^{(l)}\right),$$

which is the gradient $\hat{\nabla}_{\boldsymbol{\Theta}^{(l)}}$ computed from $\hat{\nabla}_{\mathbf{H}^{(m)}}$, using the compressed context only at the $m$-th layer, and the full-precision context for all the other layers. Then, the gradient variance is specified by the following theorem:

**Theorem 3.** *(Gradient Variance)*

$$\mathrm{Var}\left[\hat{\nabla}_{\boldsymbol{\Theta}^{(l)}}\right] = \mathrm{Var}\left[\nabla_{\boldsymbol{\Theta}^{(l)}}\right] + \sum_{m=l}^{L} \mathbb{E}\left[\mathrm{Var}\left[\mathbf{G}_{\boldsymbol{\Theta}}^{(l\sim m)}\left(\hat{\nabla}_{\mathbf{H}^{(m)}}, \hat{\mathbf{C}}^{(m)}\right) \middle| \hat{\nabla}_{\mathbf{H}^{(m)}}\right]\right]. \quad (7)$$

Thm. 3 disentangles all stochasticities in the gradient. The first term in Eq. (7) is just the FP gradient variance, and it accounts for the minibatch sampling. All the rest terms account for the noise of utilizing compressed context. Specifically, the term with $\mathbf{G}_{\boldsymbol{\Theta}}^{(l\sim m)}\left(\cdot, \hat{\mathbf{C}}^{(m)}\right)$ is the variance introduced by utilizing the compressed context $\hat{\mathbf{C}}^{(m)}$. See Appendix C.2 for a visualization of these terms.

The significance of Thm. 3 is in two folds. Firstly, it tells how much extra variance does activation compression introduce. If the activation compression variance is much smaller than the origin minibatch sampling variance, according to Thm. 2, we are confident that ActNN will converge similarly with FP training. In this case, we may reduce the numerical precision for free, as the quantization variance is negligible. Secondly, having an exact measurement of the variance, we can design quantization strategies to explicitly minimize it, as we shall see soon.

## 4. Compression Strategy

As mentioned earlier, ActNN compresses the activation by quantizing them to lower precision. As the number of bits goes down, the compression ratio gets better, but the gradient variance also grows.

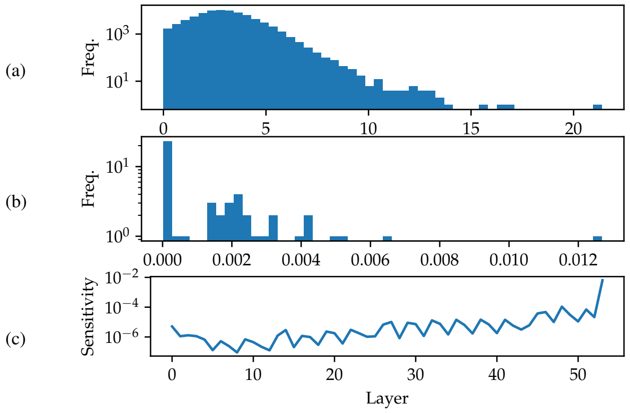


*Figure 3.* Heterogeneity in a ResNet50's activations. (a) Histogram of the per-group range at the conv_2_2_1 layer; (b) Histogram of the per-sample sensitivity at the same layer; (c) The sensitivity per dimension for each layer.

The activation is highly heterogeneous. As illustrated in Fig. 3, the activation's magnitude, sensitivity, and dimensionality vary across different feature dimensions, samples in the batch, and network layers. Therefore, it is suboptimal to use the same quantization scheme for all the elements, as done in prior works (Chakrabarti & Moseley, 2019; Fu et al., 2020). We design ActNN's quantization strategy to be aware of these heterogeneities. Based on our theory, ActNN tunes its quantization strategy on-the-fly to approximately minimize the variance defined as Eq. (7). As we mentioned in Sec. 2, these techniques exploit unique characteristics of the activation compression problem, and differ from existing methods for quantized training and model compression.

### 4.1. Per-group Quantization

First, we propose a per-group quantization strategy to tackle the distinct numerical range across feature dimensions. Given an activation tensor $\mathbf{H} \in \mathbb{R}^{N\times D}$, we partition its dimensions into groups $\mathbf{h}_{ni}$, where each group has $G$ elements. The numbers are quantized to $b$-bit unsigned integers, or $B = 2^b - 1$ quantization bins. For each element, we compute the minimum and maximum, and scale the activation:

$$\bar{\mathbf{u}}_{ni} \leftarrow B\left(\mathbf{h}_{ni} - Z_{ni}\right)/R_{ni},$$

where $R_{ni} = \max\{\mathbf{h}_{ni}\} - \min\{\mathbf{h}_{ni}\}$ is the range, $Z_{ni} = \min\{\mathbf{h}_{ni}\}$ is the zero point, and $\bar{\mathbf{u}}_{ni}$ is the activation scaled to $[0, B]$. Convert $\bar{\mathbf{u}}_{ni}$ to integers with stochastic rounding (Courbariaux et al., 2015) and store the result in memory as

$$\hat{\mathbf{u}}_{ni} = \lceil \bar{\mathbf{u}}_{ni} \rceil \text{ w.prob. } \bar{\mathbf{u}}_{ni} - \lfloor \bar{\mathbf{u}}_{ni} \rfloor \text{ otherwise } \lfloor \bar{\mathbf{u}}_{ni} \rfloor.$$

During back-propagation, the activation is dequantized as

$$\hat{\mathbf{h}}_{ni} = \hat{\mathbf{u}}_{ni} R_{ni}/B + Z_{ni}.$$

Due to the unbiased nature of stochastic rounding, it is clear that $\mathbb{E}\left[\hat{\mathbf{u}}_{ni}\right] = \bar{\mathbf{u}}_{ni}$ and $\mathbb{E}\left[\hat{\mathbf{h}}_{ni}\right] = \mathbf{h}_{ni}$.

Assuming that $\bar{\mathbf{u}}_{ni} - \lfloor \bar{\mathbf{u}}_{ni} \rfloor \sim \mathcal{U}(0,1)$, the quantization variance is $\mathrm{Var}\left[\hat{\mathbf{h}}_{ni}\right] = \frac{R_{ni}^2}{B^2}\mathrm{Var}\left[\hat{\mathbf{u}}_{ni}\right] = \frac{R_{ni}^2 G}{6B^2}$. The advantage of per-group quantization (PG) can be seen through the variance. Existing quantization strategies (Chakrabarti & Moseley, 2019; Fu et al., 2020) use a single range and zero-point per tensor, which can be viewed as a single group with the range $R = \max_{ni} R_{ni}$. However, as illustrated in Fig. 3(a), the range for most groups is far smaller than $R$. Therefore, this strategy uses unnecessarily large range for most groups, significantly enlarging the variance. In practice, we set $G = 256$ and store the per-group range and zero points in `bfloat16`, so each group costs extra 32 bits, which is 0.125 bits on average.

## 4.2. Fine-Grained Mixed-Precision

To further reduce the variance, ActNN uses mixed-precision quantization strategies, that choose the numerical precision adaptively for each sample and each layer. Let $b_n^{(l)}$ be the number of bits for sample $n$'s activation at layer $l$, $\mathbf{H}_n^{(l)}$. Let $B_n^{(l)}$ be the corresponding number of quantization bins. Theoretically, $b_n^{(l)}$ should be chosen to minimize the quantization variance specified as the second term in Eq. (7). However, the full gradient variance is too complicated to be tractable. Instead, ActNN minimizes the following objective

$$\mathrm{Var} = \sum_{l=1}^{L} \mathbb{E}\left[\mathrm{Var}\left[\mathbf{G}_{\mathbf{\Theta}}^{(l)}\left(\hat{\nabla}_{\mathbf{H}^{(l)}}, \hat{\mathbf{C}}^{(l)}\right) \,\middle|\, \hat{\nabla}_{\mathbf{H}^{(l)}}\right]\right], \quad (8)$$

which omits some terms from Eq. (7). Firstly, it omits the minibatch sampling term, which is not affected by the quantization scheme. Secondly, it only keeps the impact to $\hat{\nabla}_{\mathbf{H}^{(l)}}$ from $\hat{\mathbf{C}}^{(l)}$, omitting all the more distant contexts $\hat{\mathbf{C}}^{(m)} (m > l)$. As studied in Appendix C.2, the impact diminishes as the parameter and context become more distant. We find optimizing with this approximate objective already significantly reduces the true gradient variance.

Regarding the specific form of variance, we take linear layers as an example, where

$$\mathrm{Var}\left[\mathbf{G}_{\mathbf{\Theta}}^{(l)}\left(\hat{\nabla}_{\mathbf{H}^{(l)}}, \hat{\mathbf{C}}^{(l)}\right) \,\middle|\, \hat{\nabla}_{\mathbf{H}^{(l)}}\right] = \mathrm{Var}\left[\hat{\mathbf{H}}^{(l-1)\top}\hat{\nabla}_{\mathbf{H}^{(l)}}\right].$$

Simplifying the notations by omitting the conditional variance, layer indices, and let $\nabla := \hat{\nabla}_{\mathbf{H}^{(l)}}$, we have

$$\mathrm{Var}\left[\hat{\mathbf{H}}^{\top}\nabla\right] = \sum_{ij}\mathrm{Var}[\sum_{n}\hat{h}_{ni}\nabla_{nj}] = \sum_{ijn}\nabla_{nj}^2\mathrm{Var}\left[\hat{h}_{ni}\right]$$
$$= \frac{G}{6}\sum_{ijn}\nabla_{nj}^2 R_{ni}^2/B_n^2 = \frac{G}{6}\sum_{n}\left\|\nabla_n\right\|^2\left\|\mathbf{R}_n\right\|^2/B_n^2, \quad (9)$$

where $G$ and $R_{ni}$ are the group size and per-group range defined in Sec. 4.1, and $\mathbf{R}_n = \{R_{ni}\}$. For each sample, the variance depends on the gradient magnitude $\left\|\nabla_n\right\|^2$ and the range $\left\|\mathbf{R}_n\right\|^2$.

In general, we can minimize the overall variance under a bits budget $b_{total}$ by allocating more bits to sensitive layers and samples, described as the following optimization problem:

$$\min_{b_n^{(l)}}\sum_{l=1}^{L}\sum_{n=1}^{N} w_n^{(l)}/{B_n^{(l)}}^2 \quad \text{s.t.} \sum_{l=1}^{L} D^{(l)}\sum_{n=1}^{N} b_n^{(l)} \le b_{total}, \quad (10)$$

where $B_n^{(l)} = 2^{b_n^{(l)}} - 1$ as defined earlier, $D^{(l)}$ is the feature dimensionality, and $w_n^{(l)}$ is the sensitivity for sample $n$ at layer $l$. For linear layers, we have $w_n^{(l)} = \frac{G}{6}\|\hat{\nabla}_{\mathbf{h}_n^{(l)}}\|^2\|\mathbf{R}_n^{(l)}\|^2$ by Eq. (9). We derive the sensitivity for other layers in Appendix B.

Mixed-precision can reduce the gradient variance significantly. According to Fig. 3(b, c), the sensitivity is diverse across samples, and the per-dimension sensitivity varies by several orders of magnitudes across layers. Mixed-precision considers these heterogeneities. Furthermore, with mixed-precision, the average number of bits is no longer limited to integers, enabling a more fine-grained tradeoff between compression ratio and gradient variance.

## 4.3. Run-time Adaptation

To best utilize the data characteristics, ActNN tunes the mixed-precision quantization strategy at run time. In each SGD iteration, the tuning happens in two stages:

1. *(per-sample allocation)* During the forward propagation for each layer $l$, ActNN computes and stores the sensitivity $w_n^{(l)}$ for each sample. Then, it computes the optimal $b_n^{(l)}$ for each sample under a fixed bits budget $b^{(l)}$ *for this layer*, by solving Prob. (10). $b_n^{(l)}$ is used to compress the activation.

2. *(per-layer allocation)* After finishing the back propagation, ActNN solves Prob. (10) again for all the layers together, and sets $b^{(l)} \leftarrow \sum_n b_n^{(l)}$.

Prob. (10) is a discrete optimization problem, and can be solved exactly by dynamic programming (DP). However, DP is too costly to be computed in each SGD iteration. Instead, ActNN adopts a simple greedy algorithm. It starts with high numerical precision, e.g., $b_n^{(l)} = 8$ for all layers and samples, and progressively reduces the precision until it fits in the total bits budget. In each move, it chooses a $b_n^{(l)}$ to reduce by one, such that the increment of variance is minimal. With a binary heap for picking up the optimal move, this greedy algorithm runs in $O(NL\log_2(NL))$, where $N$ is the batch size, and $L$ is the model depth.

Finally, the sensitivity $w_n^{(l)}$ might depend on the gradient magnitude for each sample, which is unknown by the time we compress. ActNN provides two options for estimating the gradient magnitude. The first option uses the stale gradient magnitude in the last epoch. The second option uses the moving average of gradient magnitude across samples. Both strategies work well in practice.

## 5. System Implementation

We implement ActNN as a library based on PyTorch (Paszke et al., 2019). The system includes a collection of activation compressed layers. Using the system only requires substi-

```
class RegularLayer:
  def forward(context, input):
    context.save_for_backward(input)
    return compute_output(input)

  def backward(context, grad_output):
    input = context.saved_tensors
    return compute_gradient(grad_output, input)

class ActivationCompressedLayer:
  def forward(context, input):
    context.save_for_backward(compress(input))
    return compute_output(input)

  def backward(context, grad_output):
    input = decompress(context.saved_tensors))
    return compute_gradient(grad_output, input)
```

*Figure 4.* Pseudo code for activation compressed layers.

*Table 1.* Usability Comparison of Memory Saving Systems. The systems include Checkmate (Jain et al., 2019), DTR (Kirisame et al., 2020), MONet (Shah et al., 2020), SwapAdvisor (Huang et al., 2020), Capuchin (Peng et al., 2020), and BLPA (Chakrabarti & Moseley, 2019). Remat.=Rematerialization, Comp.=Compression, AOT=ahead-of-training.

| System | Method | Arbitrary Graph | Dynamic Exec. | Zero AOT Overhead | Standalone Package |
|---|---|---|---|---|---|
| Checkmate | Remat. | yes | no | no | yes |
| MONet | Remat. | yes | no | no | yes |
| DTR | Remat. | yes | yes | yes | no |
| SwapAdvisor | Swap | yes | no | no | no |
| Capuchin | Swap | yes | yes | yes | no |
| BLPA | Comp. | no | no | yes | yes |
| ActNN | Comp. | yes | yes | yes | yes |

tuting all PyTorch's default layers with ActNN's layers (e.g., replace all `torch.nn.Conv2d` with `actnn.Conv2d`). This substitution can be done automatically with a model converter. The system also provides different optimization levels to control the trade-off between memory and speed.

### 5.1. Activation Compressed Layers

Fig. 4 shows the pseudo-code for regular layers and activation compressed layers. A regular layer saves full-precision activations into its context, while an activation compressed layer compresses the activations before saving them. Therefore, the memory required for saving context is reduced. We implement highly-optimized CUDA kernels for compression and decompression, which quantize floating-point numbers into integers and compress them into bit streams.

Tab. 1 compares the usability of ActNN against other memory saving systems. ActNN's layers can be easily plugged into existing deep learning frameworks without modifying the frameworks themselves. The layers are fully compatible with existing features of PyTorch, such as dynamic execution and auto-differentiation. In contrast, advanced swapping and tensor rematerialization methods require heavy modification of the frameworks. Another benefit of ActNN is not introducing any ahead-of-training overhead. In contrast, some systems (e.g., Checkmate, MONeT, SwapAdvisor) require non-trivial ahead-of-training overhead to solve expensive optimization problems, which can take up to hours.

*Table 2.* Optimization levels for ActNN.

| Level | Compression Strategy | Bits |
|---|---|---|
| L0 | Do not compress | 32 |
| L1 | per-group quantization for conv. layers | 4, 32 |
| L2 | per-group quantization | 4 |
| L3 | L2 + fine-grained mixed-precision | 2 |
| L4 | L3 + swapping | 2 |
| L5 | L4 + defragmentation | 2 |

### 5.2. Optimization Levels

There is a trade-off between memory saving and training speed. More overhead will be introduced for saving more memory. To exploit this trade-off, ActNN provides 6 optimization levels. Higher levels can save more memory but with more overhead.

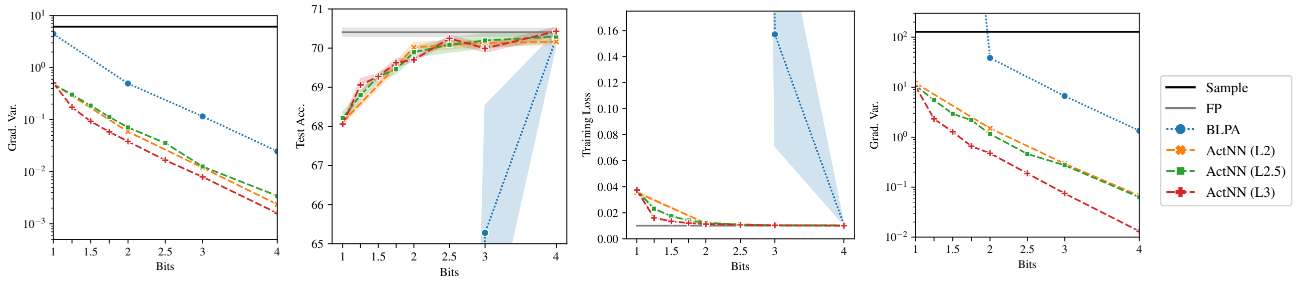
Tab. 2 lists these optimization levels. L1 uses per-group quantization to compress convolutional layers and leaves all other layers unchanged, while L2 uses per-group quantization for all the layers. L3 further adds fine-grained mixed-precision, which achieves a better compression ratio with some additional computational overhead. L4 combines compression with swapping. We swap out all compressed activations to CPU memory during the forward pass and swap them in during the backward pass. At L5, we improve the memory allocator to reduce fragmentation. PyTorch uses a caching allocator to reuse GPU buffers. This makes allocation faster but introduces a serious memory fragmentation issue. At this level, we disable this caching allocator for large tensors to reduce fragmentation.

## 6. Experiments

In this section, we evaluate ActNN on a wide range of tasks and compare it with other memory-saving systems. We use open-source model implementations and recipes for all tasks. Detailed experimental setup is in Appendix C.1. The training logs for all the models we used are available at `https://wandb.ai/actnn`.

### 6.1. Quantization Strategy

We study the impact of activation compression on the accuracy and demonstrate our heterogeneity-aware quantization strategies. The baselines are full-precision (FP) training and BLPA (Chakrabarti & Moseley, 2019), a per-tensor quantization strategy with fixed numerical precision. To compare



(a) Gradient variance on CIFAR-100 (b) Testing accuracy on CIFAR-100 (c) Testing loss on CIFAR-100 (d) Gradient variance on ImageNet

*Figure 5.* Ablation study on the quantization strategy. BLPA diverges with 1 and 2 bits. The gradient variance is calculated at the 10th epoch for CIFAR-100 and the 50th epoch for ImageNet. Sample=minibatch sampling, FP=full precision.

quantization strategies, we use ActNN (L2, L3) listed in Tab. 2. ActNN (L4, L5) do not further compress the activation, so they have identical behavior with ActNN (L3). We also add an ActNN (L2.5) for comparison, which only allocates bits between samples while keeping all the layers with the same bits per dimension.

We perform the ablation studies on ResNet-56 (He et al., 2016b) on CIFAR-100 (Krizhevsky & Hinton, 2009), and ResNet-50 (He et al., 2016a) on ImageNet (Deng et al., 2009). We also provide results on CIFAR-10 in Appendix C.3 for reference. The average number of bits is varied between $\{1, 1.25, 1.5, 1.75, 2, 2.5, 3, 4\}$. Non-integer bits is only supported by mixed-precision approaches, namely, ActNN (L2.5, L3). Each configuration is repeated by 5 times on CIFAR-100, and by once on ImageNet. Fig. 5(a,d) shows the gradient variance from both minibatch sampling ("Sample") and activation compression. The activation compression variance increases exponentially as the number of bits decreases, as analyzed in Eq. (9). However, better algorithms achieve lower variance under a given bits budget. On ImageNet, the required bits to reach a variance below 2 is 4-bit for BLPA, 2-bit for ActNN (L2), 1.75-bit for ActNN (L2.5), and 1.5-bit for ActNN (L3). At the extreme 1-bit case, mixed-precision ActNN (L2.5, L3) fall back to ActNN (L2), since each number needs at least 1 bit to encode. This can be potentially improved by allowing the bits to go below 1 bit, e.g., with product quantization (Stock et al., 2020), which we leave as future work.

The validation accuracy (Fig. 5(b)) and the training loss (Fig. 5(c)) align with the gradient variance results, where BLPA needs 4-bit to achieve near-lossless accuracy, and ActNN only needs 2-bit. These results support our theoretical analysis (Thm. 2), where the gradient variance can be used as a surrogate of the approximation quality. Tab. 3 shows the results on ImageNet. ActNN significantly outperforms BLPA. BLPA achieves near-lossless result at 4-bit, and diverges at 3-bit, while ActNN (L2.5, L3) are still lossless at 2-bit. The result of BLPA at 4-bit is on par with ActNN (L3) with only 1.5 bits. Remarkably, ActNN converges and gives reasonable results even at the extreme 1.25-bit setting. This shows the robustness of our quantization strategy. Another relevant work, TinyScript (Fu et al., 2020), utilizes non-uniform quantization intervals for activation compression. TinyScript has no public code, and it reports 7.74% top-5 error with 3-bit, which is on par with ActNN with 1.25-bit.

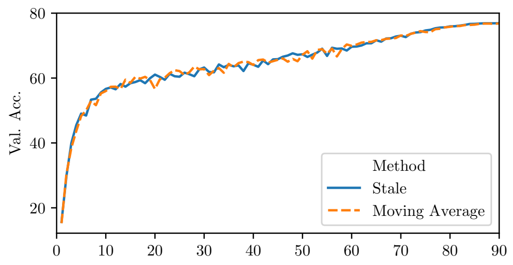


*Figure 6.* Ablation study on the gradient estimation strategy.

As we mentioned in Sec. 2, ActNN can be combined with other quantized training approaches. Here, we demonstrate this idea by combining ActNN with a mixed-precision training library, AMP (Nvidia, 2019). In this setting, AMP accelerates the training by computing the convolutions using 16-bit floating point numbers. We further apply ActNN upon AMP to compress the saved activation, reducing the memory footprint. The result is shown in Table 4. ActNN works well with AMP, and the accuracy is not affected.

We also conduct an ablation study of the gradient estimation strategy discussed in Sec. 4.3, which is used for ActNN (L3). The result is shown in Fig. 6, where "Stale" estimates the gradient magnitude with stale gradients, and "Moving Average" averages the gradient across training samples. Both strategies work well in practice, and there is not perceivable differences. We use the "Moving Average" strategy in all our experiments for its simplicity.

## 6.2. Memory Saving and Computational Overhead

Next, we measure the memory saving and the overhead introduced by ActNN. The experiments are done with PyTorch

*Table 3.* ResNet-50 on ImageNet. N/A: not available; Div.: diverge; "-": skipped since lower precision achieves lossless results.

| Bits | 32 | 4 | 3 | 2 | 1.5 | 1.25 |
|---|---|---|---|---|---|---|
| FP | **77.1** | N/A | N/A | N/A | N/A | N/A |
| BLPA | N/A | **76.6** | Div. | Div. | N/A | N/A |
| ActNN (L2) | N/A | - | **77.4** | 0.1 | N/A | N/A |
| ActNN (L2.5) | N/A | - | - | **77.1** | 75.9 | 75.1 |
| ActNN (L3) | N/A | - | - | **76.9** | 76.4 | 75.9 |

*Table 4.* ActNN with mixed precision training. Act. Bits indicates the average number of bits for the saved activation.

| Method | Act. Bits | Val. Acc. |
|---|---|---|
| FP | 32 | 77.1 |
| AMP | 16 | 77.3 |
| ActNN (L2) + AMP | 4 | 77.1 |
| ActNN (L3) + AMP | 2 | 76.9 |

v1.7 and an AWS `g4dn.4xlarge` instance, which has a 16GB NVIDIA T4 GPU and 64GB CPU memory.

Tab. 5 shows the memory usage right before backward pass. The activation memory reaches its peak at this point. For both models, activations consume over 90% of the total memory. This justifies the potential of activation compressed training. ActNN compresses activation memory by $12\times$. This matches the theoretical value. Take a Conv-BN-ReLU block as example, FP takes 32 bits (Conv) + 32 bits (BN) = 64 bits, while ActNN takes 2.125 bits (Conv) + 2.125 bits (BN) + 1 bit (ReLU) = 5.25 bit. The extra 0.125 bit is used to store the zero point and scale for each group. The compression ratio is $64/5.25 \approx 12$.

We compare the training throughput of ActNN against other memory saving systems in Fig. 1 and Fig. 7. Each curve shows the trade-off between memory saving and training speed for one system. "DTR" is dynamic tensor rematerialization (Kirisame et al., 2020), a state-of-the-art rematerialization method for dynamic graphs. DTR runs out-of-memory very soon when trying to increase the batch size. "BLPA" is the system implemented in Chakrabarti & Moseley (2019). It only supports a dedicated ResNet architecture, so we cannot run it on DenseNet. Further, its dedicated design is not compatible with fused high-performance kernel libraries, so its training speed is very slow. "CAP" is the Capuchin system based on swapping and recomputation (Peng et al., 2020). It is not open-source, so we get relative overhead numbers from its paper and compute the scaled training throughput. Because our system runs in dynamic graph mode, we use Capuchin's numbers in eager mode as a fair comparison. "swap" is a simple swapping strategy that swaps all activations to the CPU. However, the CPU memory is finite, so it cannot increase the batch size unlimitedly. "ActNN" is our system with 6 different levels of optimization, so the curve of ActNN shows roughly 6 segments. As shown in the figures, ActNN achieves a much larger maximum batch size and extends the Pareto frontier significantly over state-of-the-art systems. ActNN enables training with a $6.6\times - 14.0\times$ larger batch size under the same memory budget.

*Table 5.* Memory usage before backward pass. "Total Mem." includes model parameters, optimizer states, input data and activation. "Act. Mem." includes activation for all layers except the final loss layer. R=memory saving ratio; OOM=out of memory.

| Network | Batch | Total Mem. (GB) FP | Total Mem. (GB) ActNN (L3) | Total Mem. (GB) R | Act. Mem. (GB) FP | Act. Mem. (GB) ActNN (L3) | Act. Mem. (GB) R |
|---|---|---|---|---|---|---|---|
| ResNet-152 | 32 | 6.01 | 1.18 | 5× | 5.28 | 0.44 | 12× |
| | 64 | 11.32 | 1.64 | 7× | 10.57 | 0.88 | 12× |
| | 96 | OOM | 2.11 | / | OOM | 1.32 | / |
| | 512 | OOM | 8.27 | / | OOM | 7.01 | / |
| FCN-HR-48 | 2 | 5.76 | 1.39 | 4× | 4.76 | 0.39 | 12× |
| | 4 | 10.52 | 1.79 | 6× | 9.52 | 0.79 | 12× |
| | 6 | OOM | 2.17 | / | OOM | 1.18 | / |
| | 20 | OOM | 4.91 | / | OOM | 3.91 | / |

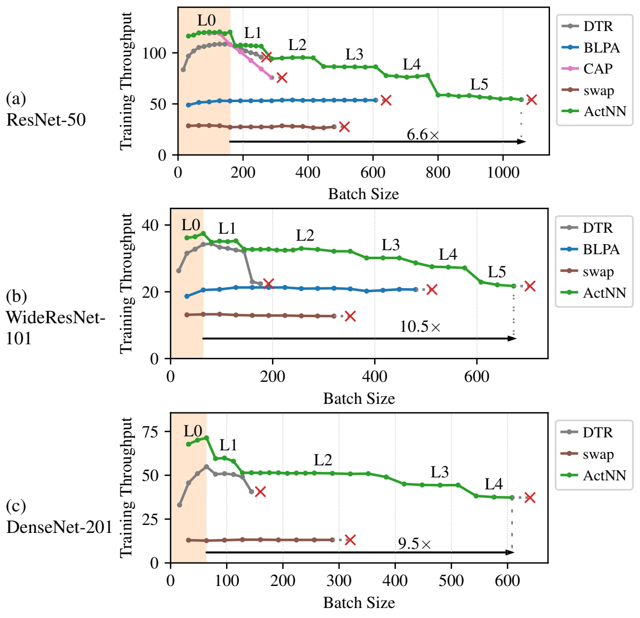


*Figure 7.* Training throughput vs batch size. Red cross mark means out-of-memory. The shaded yellow region denotes the possible batch sizes with full precision training given the memory budget.

To show the potential of training larger models, we scale a ResNet-152 to deeper, wider, or higher resolution. Table 6 compares the largest model we can train against full precision. With the same memory budget and batch size (64), ActNN can scale a ResNet-152 to $6.4\times$ deeper, $3.7\times$ wider or $3.1\times$ higher resolution, while maintaining $64\% - 155\%$ original training throughput.

*Table 6.* Comparison of the largest models ActNN can train before out-of-memory with the same batch size (64). D = depth = the number of layers, W = width = the base width of the bottleneck block, R = resolution = width and height of input images.

| Dim. | Maximum Value | | | Training Throughput (TFLOPS) | | |
|---|---|---|---|---|---|---|
| | FP | ActNN (L3) | ActNN (L4) | FP | ActNN (L3) | ActNN (L4) |
| D | 160 | 660 | 1016 | 0.59 | 0.46 | 0.38 |
| W | 92 | 332 | 340 | 0.70 | 1.07 | 1.09 |
| R | 240 | 636 | 740 | 0.59 | 0.46 | 0.42 |

*Table 7.* Semantic segmentation on Cityscapes and object detection on Coco. Models: HRNetV2W48 (HRNet) (Wang et al., 2020), ResNet-50 dilation 8 (Dilation8), FPN (Kirillov et al., 2019), RetinaNet (Lin et al., 2017).

| Task | Model | Method | Bits | Batch | mIoU / AP |
|---|---|---|---|---|---|
| Seg. | HRNet | FP | 32 | 8 | 80.65 |
| | HRNet | ActNN (L3) | 2 | 8 | 81.02 |
| | HRNet | ActNN (L3) | 2 | 160 | 80.31 |
| | Dilation8 | FP | 32 | 8 | 73.61 |
| | Dilation8 | ActNN (L3) | 2 | 8 | 72.92 |
| | Dilation8 | ActNN (L3) | 2 | 160 | 75.85 |
| | FPN | FP | 32 | 8 | 74.52 |
| | FPN | ActNN (L3) | 2 | 8 | 74.18 |
| | FPN | ActNN (L3) | 2 | 160 | 75.98 |
| Det. | RetinaNet | FP | 32 | 16 | 36.5 |
| | RetinaNet | ActNN (L3) | 2 | 16 | 36.2 |
| | RetinaNet | ActNN (L3) | 2 | 80 | 36.0 |

### 6.3. Segmentation and Detection

Here, we report results for high-resolution segmentation and detection in Tab. 7. Activation memory is a more severe problem for these tasks, as the activation memory scales quadratically with the image resolution. For all the models, ActNN converges within $0.5\%$ mIoU / AP comparing with the full-precision baseline. It worth noticing that ActNN could finish the default training recipe, batch size of 16/8 for detection/segmentation, within only one GPU. Remarkably, by training with a larger batch size of 160, ActNN gains $1.8\%$/$1.4\%$ mIoU for Dilation8 (dilated FCN) (Shelhamer et al., 2017) / FPN (Kirillov et al., 2019). This gain comes from the more reliable estimation of normalization parameters (Peng et al., 2018) with large batch size. Without memory saving techniques, such a large batch size is only achievable with a cluster of machines.

### 6.4. Self-supervised Learning

Here, we test ActNN for two self-supervised learning methods, MoCov2 (Chen et al., 2020c) and BYOL (Grill et al., 2020). As the contrastive loss used in these methods involves comparing pairs of examples in a batch, larger batch size gives more accurate estimation of the contrastive loss, and has positive impact to the quality of the learned representation (Chen et al., 2020b).

*Table 8.* Self-supervised learning on ImageNet.

| Model | Method | Bits | Batch | GPUs | Val. Acc. |
|---|---|---|---|---|---|
| MoCov2 | FP | 32 | 256 | 8 | 67.69 |
| MoCov2 | ActNN (L3) | 2 | 512 | 2 | 67.25 |
| BYOL | FP | 32 | 256 | 8 | 72.35 |
| BYOL | ActNN (L3) | 2 | 1024 | 8 | 72.65 |

We directly apply ActNN (L3) to MoCov2 and BYOL. Both methods use ResNet-50 as the backbone. MoCov2 (Chen et al., 2020c) is trained for 200 epochs and it uses the last layer's feature after global pooling for evaluation; BYOL (Grill et al., 2020) is trained for 300 epochs and it combines multiple layer's features for evaluation. As shown in Table 8, ActNN can train the models with significantly larger batch size per GPU, and achieve good validation accuracy using only 2-bit activations.

## 7. Conclusions

We have presented ActNN, a framework for training neural networks with randomly quantized activations. ActNN is grounded by the convergence guarantee for general network architectures that we provide. Quantization affects the convergence through the gradient variance. We propose per-group quantization and fine-grained mixed-precision quantization strategies, which approximately minimizes the gradient variance during training. On a wide range of tasks, ActNN achieves negligible accuracy loss with 2-bit activation, improving significantly over prior state-of-the-arts. ActNN can be readily applied as a collection of layers in PyTorch, and it enables up to $14\times$ batch size, $6.4\times$ deeper, $3.7\times$ wider, or $3.1\times$ higher-resolution models.

## Acknowledgements

This work was supported by a gracious fund from Intel corporation, Berkeley Deep Drive (BDD), and Berkeley AI Research (BAIR) sponsors. In addition to NSF CISE Expeditions Award CCF-1730628, this research is supported by gifts from Amazon Web Services, Ant Group, CapitalOne, Ericsson, Facebook, Futurewei, Google, Intel, Microsoft, Nvidia, Scotiabank, Splunk and VMware. We would like to thank the Intel VLAB team for providing us with access to their computing cluster. We also gratefully acknowledge the support of NVIDIA Corporation for their donation of two Titan Xp GPU used for this research. We would also like to acknowledge the UC Berkeley CLTC, ARO, DARPA, IARPA, NSF, and ONR for providing partial support of this work. Our conclusions do not necessarily reflect the position or the policy of our sponsors, and no official endorsement should be inferred.

## References

Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

Chakrabarti, A. and Moseley, B. Backprop with approximate activations for memory-efficient network training. *arXiv preprint arXiv:1901.07988*, 2019.

Chen, J., Gai, Y., Yao, Z., Mahoney, M. W., and Gonzalez, J. E. A statistical framework for low-bitwidth training of deep neural networks. In *Advances in neural information processing systems*, 2020a.

Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Xu, J., Zhang, Z., Cheng, D., Zhu, C., Cheng, T., Zhao, Q., Li, B., Lu, X., Zhu, R., Wu, Y., Dai, J., Wang, J., Shi, J., Ouyang, W., Loy, C. C., and Lin, D. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

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

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020b.

Chen, X., Fan, H., Girshick, R., and He, K. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020c.

Choi, J., Wang, Z., Venkataramani, S., Chuang, P. I.-J., Srinivasan, V., and Gopalakrishnan, K. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

Contributors, M. Mmsegmentation, an open source semantic segmentation toolbox. `https://github.com/open-mmlab/mmsegmentation`, 2020.

Courbariaux, M., Bengio, Y., and David, J.-P. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pp. 3123–3131, 2015.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. Ieee, 2009.

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

Dong, Z., Yao, Z., Cai, Y., Arfeen, D., Gholami, A., Mahoney, M. W., and Keutzer, K. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *arXiv preprint arXiv:1911.03852*, 2019.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

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

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., et al. Bootstrap your own latent: A new approach to self-supervised learning. In *Advances in neural information processing systems*, 2020.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

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

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

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

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

Jain, P., Jain, A., Nrusimha, A., Gholami, A., Abbeel, P., Keutzer, K., Stoica, I., and Gonzalez, J. E. Checkmate: Breaking the memory wall with optimal tensor rematerialization. *arXiv preprint arXiv:1910.02653*, 2019.

Kirillov, A., Girshick, R., He, K., and Dollár, P. Panoptic feature pyramid networks. In *CVPR*, 2019.

Kirisame, M., Lyubomirsky, S., Haan, A., Brennan, J., He, M., Roesch, J., Chen, T., and Tatlock, Z. Dynamic tensor rematerialization. *arXiv preprint arXiv:2006.09616*, 2020.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *ICCV*, 2017.

Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations (ICLR)*, 2018.

Meng, C., Sun, M., Yang, J., Qiu, M., and Gu, Y. Training deeper models by gpu memory optimization on tensorflow. In *Proc. of ML Systems Workshop in NIPS*, volume 7, 2017.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al. Mixed precision training. In *International Conference on Learning Representations*, 2018.

Nvidia. apex.amp. https://nvidia.github.io/apex/amp.html, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Peng, C., Xiao, T., Li, Z., Jiang, Y., Zhang, X., Jia, K., Yu, G., and Sun, J. Megdet: A large mini-batch object detector. In *CVPR*, 2018.

Peng, X., Shi, X., Dai, H., Jin, H., Ma, W., Xiong, Q., Yang, F., and Qian, X. Capuchin: Tensor-based gpu memory management for deep learning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 891–905, 2020.

Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., and He, Y. Zero-offload: Democratizing billion-scale model training. *arXiv preprint arXiv:2101.06840*, 2021.

Shah, A., Wu, C.-Y., Mohan, J., Chidambaram, V., and Krähenbühl, P. Memory optimization for deep networks. *arXiv preprint arXiv:2010.14501*, 2020.

Shelhamer, E., Long, J., and Darrell, T. Fully convolutional networks for semantic segmentation. *TPAMI*, 2017.

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

Stock, P., Joulin, A., Gribonval, R., Graham, B., and Jégou, H. And the bit goes down: Revisiting the quantization of neural networks. In *International Conference on Learning Representations*, pp. 1–11, 2020.

Sun, X., Wang, N., Chen, C.-Y., Ni, J., Agrawal, A., Cui, X., Venkataramani, S., El Maghraoui, K., Srinivasan, V. V., and Gopalakrishnan, K. Ultra-low precision 4-bit training of deep neural networks. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., et al. Deep high-resolution representation learning for visual recognition. *TPAMI*, 2020.

Wang, L., Ye, J., Zhao, Y., Wu, W., Li, A., Song, S. L., Xu, Z., and Kraska, T. Superneurons: Dynamic gpu memory management for training deep neural networks. In *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*, pp. 41–53, 2018a.

Wang, M., Huang, C.-c., and Li, J. Supporting very large models using automatic dataflow graph partitioning. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pp. 1–17, 2019.

Wang, N., Choi, J., Brand, D., Chen, C.-Y., and Gopalakrishnan, K. Training deep neural networks with 8-bit floating point numbers. In *Advances in Neural Information Processing Systems*, pp. 7675–7684, 2018b.

Yang, G. and Schoenholz, S. S. Mean field residual networks: on the edge of chaos. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 2865–2873, 2017.

Zhang, D., Yang, J., Ye, D., and Hua, G. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In *The European Conference on Computer Vision (ECCV)*, September 2018.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.