An Experimental Analysis of the Influence of the Precision of Parameters in an Artificial Neural Network

Denise Katritschenko, Elias Reich, Sayed Abozar Sadat, Lukas Schwaiger
University of Salzburg
Department of Artificial Intelligence and Human Interfaces (AIHI)

July 16, 2025

Abstract

We study how the numerical precision of weights, activations, and gradients in neural networks can be reduced without sacrificing too much of the ~~classification~~ performance. Using MNIST as a dataset, we benchmark uniform ~~64-16-bit~~ quantisation, mixed-precision formats, and post-training decimal rounding. The results reveal that (i) 16-bit end-to-end training does not significantly decrease test accuracy for the MNIST dataset and the network used, (ii) three-decimal-place weight rounding while keeping 32-bit precision incurs minimal accuracy loss. These findings verify limits for memory- and energy-constrained usecases like inference in mobile devices.

1 Introduction

no contractions

Artificial Neural Networks (ANNs) are a key part of today's machine learning systems. They've made huge progress in areas like image recognition, speech processing, and self-driving cars. However, one of the biggest challenges with ANNs is that they use a lot of memory and computing power. This becomes a problem when you want to run them on devices with limited resources, such as smartphones, embedded systems, or edge devices.

in order

To solve this, researchers have looked into ways to make neural networks more efficient. A common approach is to reduce the size of the network or shrink the input batch size. But doing this can harm the network's ability to learn and generalize. A better strategy is to reduce the numerical precision of the network's parameters such as things like weights, activations, and gradients. This method, called quantization, makes the model smaller and faster while using less energy, without necessarily hurting accuracy.

An

advanced version of this method is called mixed-precision quantization. Instead of using the same number of bits for every layer in the network, mixed-precision assigns different bitwidths depending on how important each part is. For example, the early layers that handle low-level features might use higher precision, while deeper layers could work fine with fewer bits. This flexible approach often leads to better results than using a single fixed precision for the whole model. Researchers have used a range of techniques, like gradient descent, heuristics, and even evolutionary algorithms, to find the best bitwidth settings for each layer.

good

Another method that helps reduce resource usage is network pruning. This means cutting out parts of the network that don't contribute much to its performance. One interesting way to do this is with simulated annealing, a search algorithm that mimics the process of cooling metal to find a stable structure. It doesn't need gradient information and can still find good network shapes by exploring many possibilities. This makes it useful when you want to shrink a network without going through heavy retraining.

reference!

Previous studies have shown that reducing the precision of neural network parameters can lead to significant improvements in memory usage and processing speed. Even when precision is reduced to very low levels, models can still perform competitively on complex tasks. However, while these early results are promising, the overall effects of lowering precision, especially how it influences model accuracy, training behavior, and storage efficiency, are still not fully understood and deserve more detailed investigation.

This is where our experimental work begins. In this paper, we take a closer look at how different levels of parameter precision affect the behavior and performance of neural networks. Our goal is to understand what happens when we reduce precision in various ways, and how much we can lower it before performance starts to drop.

We start by testing how reducing precision during both training and inference affects model accuracy and runtime. Next, we look at what happens when we train the network using full precision, but lower the precision only during inference. This setup is common in real-world applications, where a powerful machine handles training, but the final model is deployed on a lightweight device. Then, we run a series of tests where we round the trained weights to a limited number of decimal places. This helps us to see how much we can simplify the stored weight values without hurting accuracy too much. This is especially useful for devices with very limited storage. We also try out mixed-precision training, using a mix of FP16 and FP32 formats. On top of that, we experiment with different types of gradient clipping, like global norm clipping and value clipping, to see how they affect training stability and final accuracy. Even though we use relatively simple network architectures and datasets, our results show that neural networks are quite robust to low-precision settings. In some cases, we were able to round weights to just three decimal digits and still get nearly the same accuracy as the full-precision version. These findings show that it's possible to significantly reduce the memory and compute needs of neural networks without losing much performance. This makes them more practical for use in low-resource settings. It also opens the door for more advanced techniques that combine quantization, pruning, and precision-aware training to build highly efficient models for real-world applications.

good, split the last paragraph a bit

2 ~~Background~~

Artificial Neural Networks

A typical neuron in a neural network performs a weighted sum of its inputs, followed by an activation function:

$$a = \omega^\top x + b, \quad \hat{y} = \sigma(a),$$

where $\omega \in \mathbb{R}^n$ is the weight vector, $b \in \mathbb{R}$ is the bias, $x \in \mathbb{R}^n$ is the input, and σ is a nonlinear activation function (e.g., sigmoid or ReLU).

Training is conducted via backpropagation, where gradients of a loss function $L(\hat{y}, y)$ with respect to the model parameters are computed. For example, using the mean squared

error:

$$
\begin{aligned}
L(f(x_1, x_2)) &= \left(\sigma(\omega^T x + b) - y\right)^2 \\
&= \left(\sigma(a) - y\right)^2 \\
&= \left(\hat{y} - y\right)^2
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial L}{\partial \omega_1} &= \frac{\partial L}{\partial \sigma} \quad \frac{\partial \sigma}{\partial a} \quad \frac{\partial a}{\partial \omega_1} \\
&= 2(\hat{y} - y) \quad \frac{\mathrm{d}}{\mathrm{d}a}\sigma(a) \quad x_1
\end{aligned}
$$

Memory and computational requirements during training are substantial, as gradients, activations, and parameters must be stored and updated iteratively. In contrast, inference requires only a forward pass, making it more amenable to optimization via precision reduction.

3 Memory Efficiency and Numerical Formats

To reduce memory usage and computational load, precision-aware techniques such as quantization and mixed-precision training are increasingly employed. Instead of using 32-bit or 64-bit floating-point formats, parameters can be stored in 16-bit or even 8-bit fixed-point representations. Figure 1 depicts the general floating point format and terminology of significand, base, and exponent. Tables 1 and 2 provide an overview of the different floating point types and their respective characteristics.

$$
2469/200 = 12.345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10^{\overbrace{-3}^{\text{exponent}}}}_{\text{base}}
$$

Figure 1: Illustration of standard floating-point format.

Type	Sign	Exponent	Significand	Total
Half (FP16)	1	5	10	16
Single (FP32)	1	8	23	32
Double (FP64)	1	11	52	64

Table 1: Field widths (in bits) of common IEEE754 formats (adapted from [1]).

Type	Exponent bias	Precision (bits)	Decimal digits
Half (FP16)	15	11	∼3.3
Single (FP32)	127	24	∼7.2
Double (FP64)	1023	53	∼15.9

Table 2: Range and precision characteristics of the same formats (adapted from [1]).

4 Related Work

Prior research in this domain can be broadly classified into two categories:

Quantization During Training

Several studies have explored training neural networks with reduced precision directly. Notable contributions include:

- **Mixed Precision Training**, which leverages float16 arithmetic for speed while maintaining float32 master weights [2].
- **DoReFa-Net**, which quantizes weights, activations, and gradients to arbitrary bitwidths [3].
- **Q-BERT**, a quantized BERT model that maintains accuracy using only 4-bit weights [4].

Post-Training Quantization

Other approaches focus on converting pretrained models into lower-precision equivalents:

- **Integer-Only Quantization** using symmetric quantization scales [5].
- **GPTQ**, a method for quantizing large transformer models without retraining [6].
- **SmoothQuant**, which scales activations to better match weight ranges during quantization [7].

5 Experimental Design

We trained a simple fully connected neural network on the MNIST dataset, consisting of 60,000 training and 10,000 test grayscale images of handwritten digits (28×28 pixels, see figure 2). Model training was performed under varying parameter bitwidths, and 5-fold cross-validation to evaluate the model performance. The structure of the MLP is depicted in figure 3.

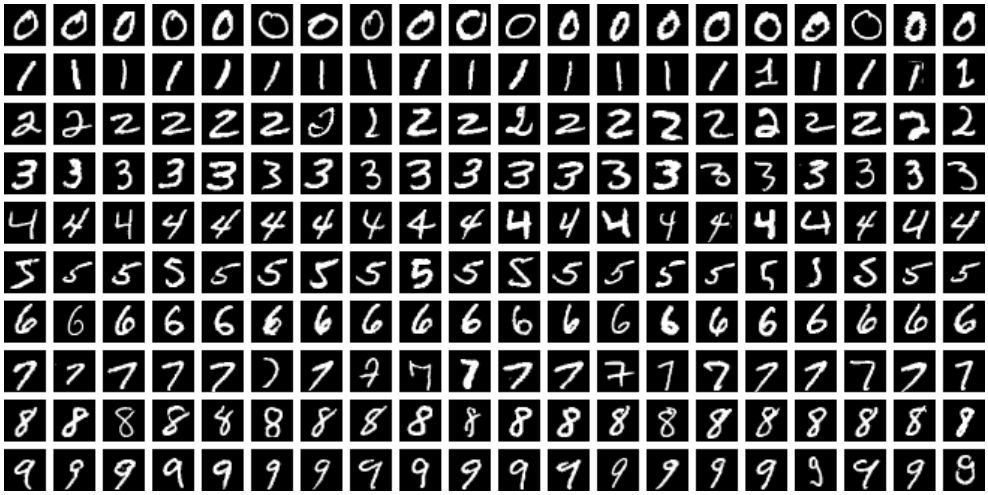

Figure 2: Sample images from the MNIST dataset.

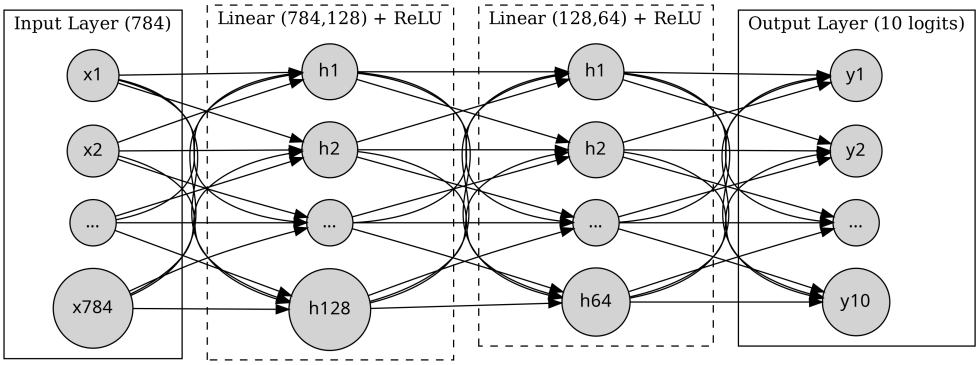

Figure 3: MLP structure used in the experiments.

6 Results

In the first experiment, every floating-point tensor that is used inside the training loop was converted to a lower precision data type. This therefore includes the weights and biases, activations, gradients, optimizer states, etc. The types tried were float64, float32, and float16.

Accuracy Across Bitwidths

Figure 4 shows how the accuracy changed after the models were trained on the different precision settings. Keep in mind that the y-axis is not starting from zero, so the absolute height differences are amplified visually, though the delta is only around .1 percent. Re-training the models and re-evaluating them has also lead to the accuracies being "reversed", meaning the float64 model had the least accuracy. It is safe to assume that the model accuracy has neither decreased, nor increased.

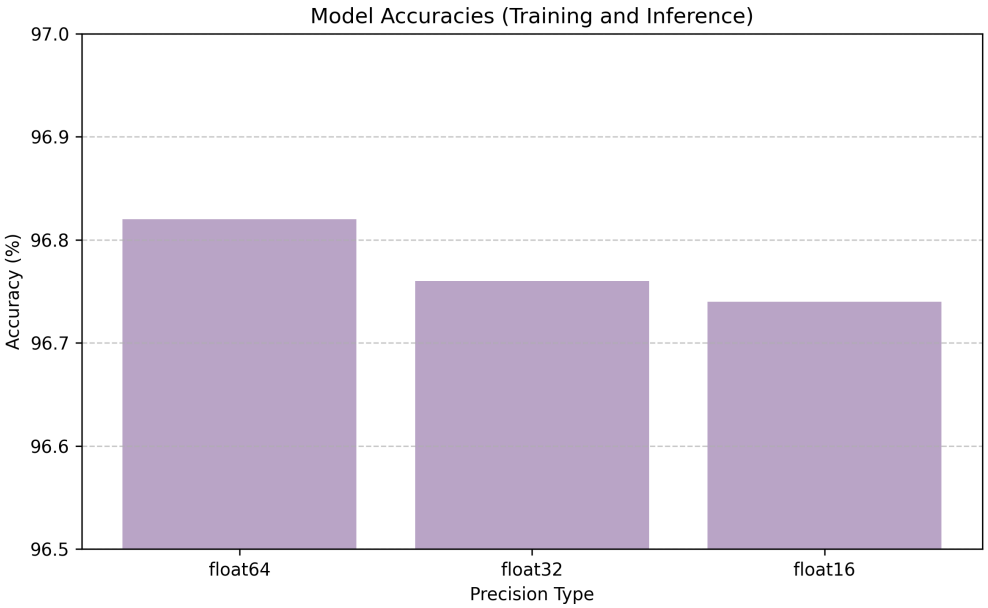

Figure 4: Model accuracy evaluated after training with ~~the~~ different precisions.

Training and Inference Time

we Also analyzed ~~were~~ the times needed for both ~~the~~ training and inference. Figure 5 depicts the times measured in the training loop and later on at the inference stage. Again, the times are almost identical in both regards, and neither a real speed-up nor a slow-down has been measured.

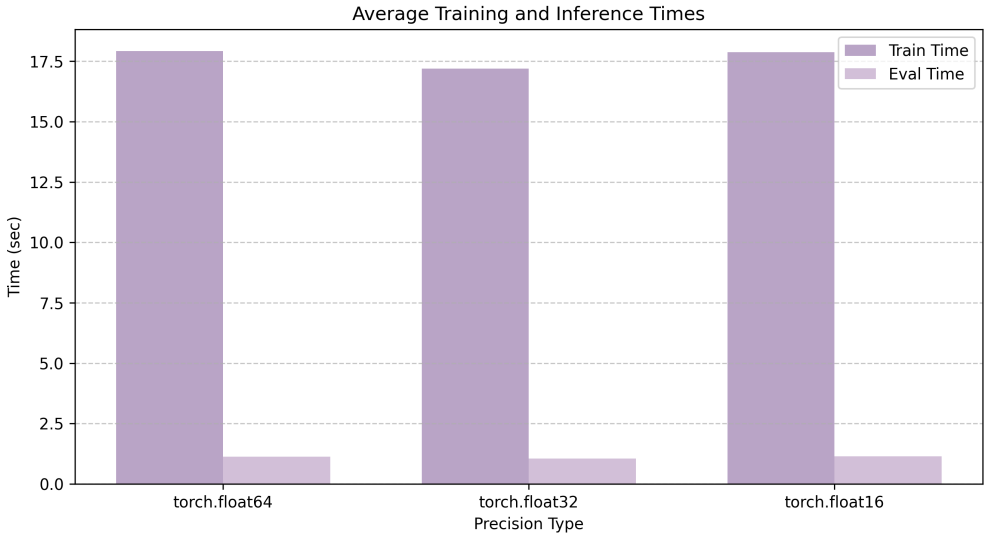

Figure 5: Training and evaluation times for the different precisions. cross val?

Inference-Only Testing

This third experiment was conducted to test ~~a possible~~ post-training precision reduction. For this, the model was trained on the highest precision (float64) and then quantized at the inference stage. Figure 6 shows that the accuracy of the baseline (float64 training AND inference) is maintained as the precision is decreased at the later stages. In contrast to the *Accuracy Across Bitwidths* experiment, there are no fluctuations at all because for the lower precisions, there was no new model trained.

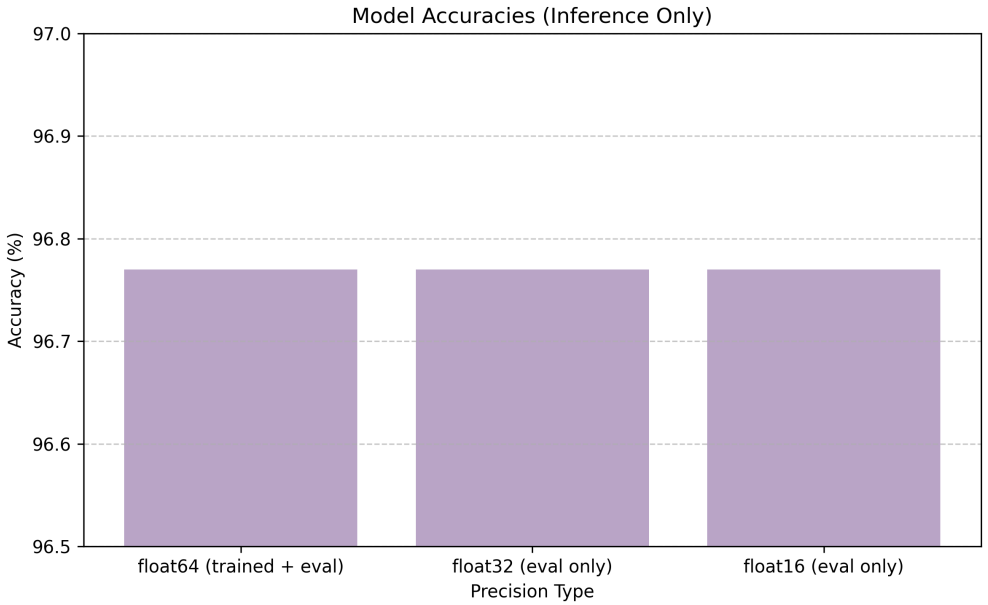

Figure 6: Inference accuracy using float64-trained weights followed by quantization.

Automatic Mixed Precision and Gradient Clipping

Automatic Mixed Precision (AMP) accelerates deep learning training by using both float16 and float32 data types reducing memory usage and increasing computational speed. However, float16's limited dynamic range can cause numerical instability, especially during backpropagation. To address this, Gradient Clipping is used to cap the magnitude of gradients (either globally or per element) preventing extreme updates that could hinder convergence. Together, AMP and gradient clipping enable efficient, stable training by balancing performance and numerical precision.

First step is using global norm clipping, all gradients are collectively scaled to ensure their combined norm does not exceed a specified threshold, promoting stable updates across the network. In second step per-element clipping constrains each individual gradient component within a fixed range, directly limiting extreme values. These methods are evaluated under Automatic Mixed Precision (AMP) to compare their impact on training stability, model accuracy, and computation time across different precision types.

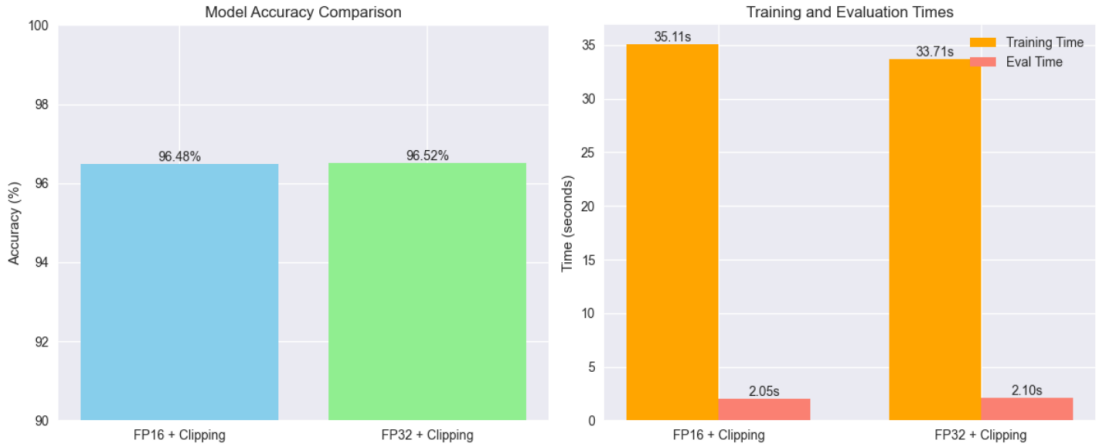

Figure 7: Affect of Gradient Norm on accuracy, training and testing timings. Accuracy is virtually the same between FP16 and FP32 when gradient clipping is used validating the stability of AMP and gradient scaling. Eval time is nearly identical for both formats, as evaluation involves fewer operations and less intense memory bandwidth usage.

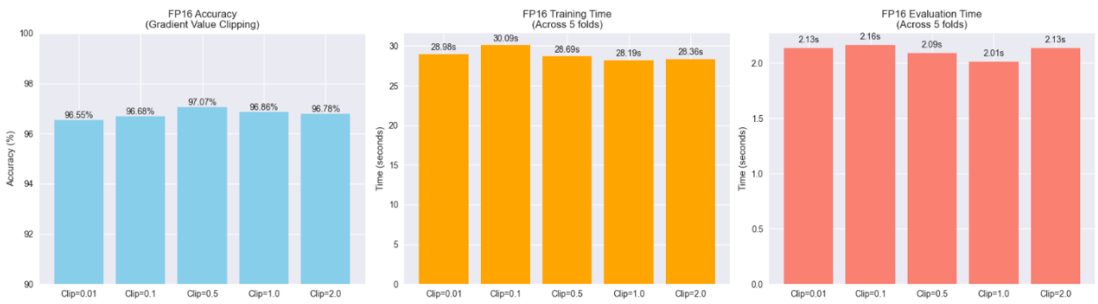

Figure 8: Effect of Per-Element Gradient Clipping on FP16. Accuracy vs Training & Eval Time

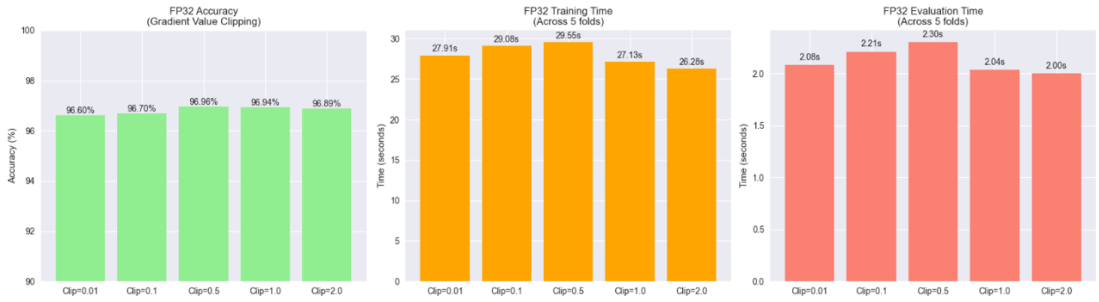

Figure 9: Effect of Per-Element Gradient Clipping on FP32. Accuracy vs Training & Eval Time

Figure 8 shows accuracy peaked at 97.07% with a clipping value of 0.5, while the lowest accuracy (96.55%) occurred at clip=0.01, suggesting that very small clip values may overly restrict gradient updates and hurt learning. Training time ranged indicating that moderate clipping supports efficient convergence. Evaluation time remained stable across all clip values (around 2.01s to 2.16s), showing that clipping primarily impacts training rather than inference performance. Also Figure 9 shows that accuracy was most stable across all clip values with the highest accuracy at clip=0.5, demonstrating its robustness to gradient clipping variations. Training time suggesting that larger clip values can speed up convergence without sacrificing stability. Evaluation time confirming that clipping had minimal effect on inference time.

Effect of Decimal Rounding

A non training related issue is the storage of the NN parameters. Usually, the parameters are stored in the format that was chosen to train with. This is best for reproducibility as well as accuracy when doing inference with the trained model weights. The datatype precision has a higher impact during training though, which begs the question if one could omit precision from the model weights. In this section, we explore a naive quantization method: Rounding to a fixed number of decimals. With significantly little decimals and a reasonable range for the integer part, one could store only the significant and provide base as well as exponent as metadata. When loading the model, the weights are converted back to floating point, with whatever precision is desired. Only the weights are quantized; the input data, as well are the values propagated through the network are at full precision.

The first step is to examine the distribution of parameter values. By the standard deviation around modes one can make an educated guess as to how many decimals are needed. The range will decide how large the integer part may get. Reading from the distribution is just a heuristic though, so we will also test the hypotheses.

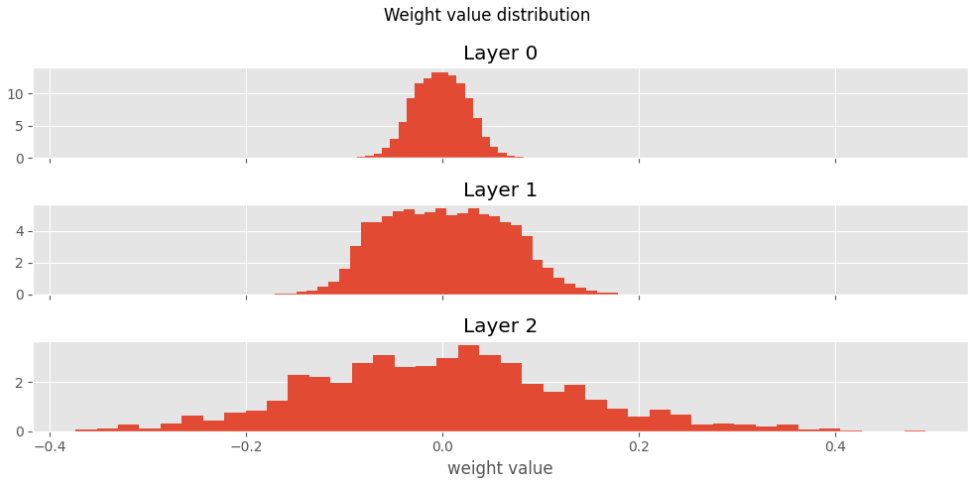

Figure 10: Distribution of model weights in each layer. The values are in $[-0.5, 0.5]$, this means no integer part is needed in this example. Generally, with normalized data and the quite popular L1-/L2-/Batch-Norm layers, weight values in $[-1, 1]$ is to be expected.

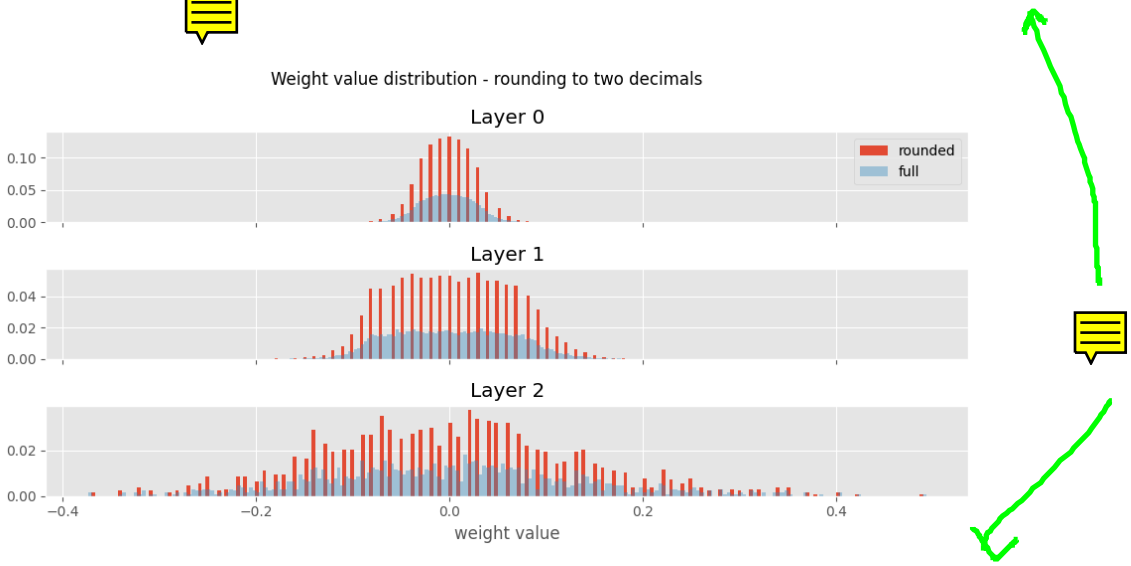

Figure 11: Distribution of model weights in each layer after rounding to two decimals. This appears to be a good approximation already.

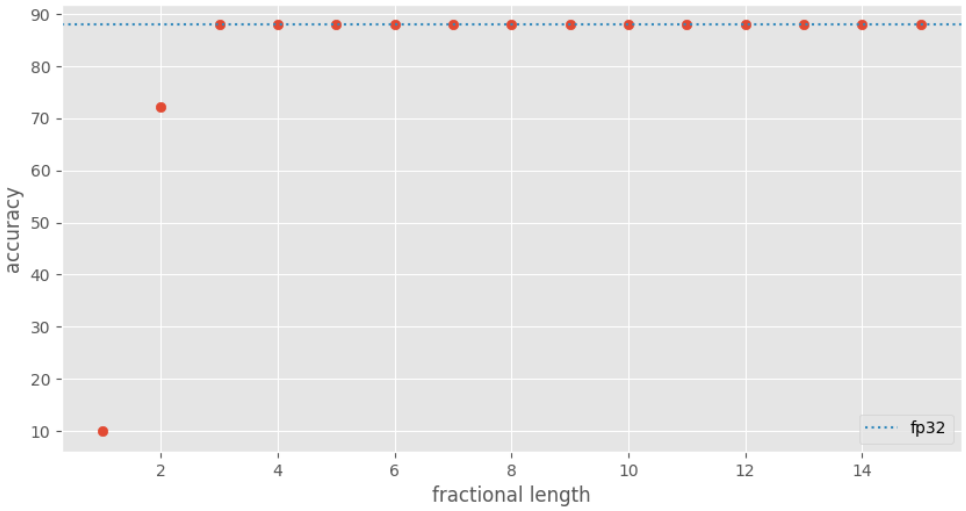

Figure 12: Test Accuracy vs. number of decimal digits retained in weights.

Figure 12 shows how rounding the decimal part affects the test accuracy. For just one decimal the model has a maximum of 19 different weights at its disposal, which makes the model have only 10% accuracy, random guessing with a 10 class problem. Two decimals perform significantly better. At 70% accuracy the model gets close to the full precision 88% accuracy with 32 bit floating point. Making and educated guess on distribution such as Figure 11 seems to be a good heuristic for this example. At just 3 decimals, the full 32 bit precision is reached. Thus we can conclude that with just 10 bit for representing 3 decimals, no bits for the integer part and one additional bit for the sign, only 11 bit must be stored to have the best performing model for inference.

7 Summary

The experimental results presented in this paper demonstrate the robustness of neural networks to reduced parameter precision, particularly for simpler tasks like MNIST classification. Key findings include:

- **Precision Reduction During Training and Inference**: Training and evaluating models with 16-bit floating-point (FP16) precision preserved accuracy comparable to higher-precision formats (FP32 and FP64). This suggests that for tasks of similar complexity, FP16 is sufficient, offering potential memory and computational savings without sacrificing performance.
- **Post-Training Quantization**: Reducing precision only during inference, after training with high-precision (FP64), maintained baseline accuracy. This is particularly valuable for deployment scenarios where training is done on powerful hardware, but inference occurs on resource-constrained devices.
- **Mixed-Precision:** Training with mixed precision and Gradient Clipping resulted in similar test accuracy scores as precision reduction during training as well as post-training quantization. No significant decrease in model performance accuracy was detected. The train and test times were similar, regardless of using FP32, FP16 or a mix of both.

- **Weight Rounding:** Rounding trained weights to just three decimal digits (approximately 11 bits) achieved full accuracy, highlighting the potential for aggressive quantization in storage-efficient model deployment. This finding is significant for edge applications where memory footprint is critical.

However, the simplicity of the MNIST dataset and the small network architecture used in the experiments may limit the generalizability of these findings. More complex tasks (e.g., CIFAR-10, ImageNet) and deeper architectures (e.g., CNNs, Transformers) could exhibit greater sensitivity to precision reduction. Additionally, the impact of precision on training dynamics, such as gradient flow and convergence, was not deeply explored and could be a focus for future work.

8 Conclusion

This study demonstrates that neural networks can maintain strong performance even under significant reductions in parameter precision, particularly for simpler tasks like MNIST classification. Key findings confirm that FP16 training and inference, post-training quantization, and aggressive weight rounding (down to three decimal digits) preserve model accuracy while improving memory and computational efficiency. These results are promising for deploying models in resource-constrained environments, such as edge devices and real-time applications.

To build on these insights and expand their applicability, future work should focus on the following directions:

- **Complex Architectures and Datasets:** Future research should validate these findings on more complex models (e.g., CNNs, Transformers) and datasets (e.g., CIFAR-10, ImageNet) to assess the scalability of low-precision techniques.
- **Quantization-Aware Training:** Investigating advanced methods like learned quantization, non-uniform bit allocation, and dynamic precision adjustment could further optimize the trade-off between efficiency and accuracy.
- **Hardware-Specific Optimizations:** Exploring how reduced precision interacts with specialized hardware (e.g., TPUs, neuromorphic chips) could unlock new efficiency gains in real-world deployments.
- **Theoretical Understanding:** A deeper analysis of how precision affects gradient dynamics, convergence, and generalization could provide foundational insights for designing robust low-precision training algorithms.

References

1. Wikipedia contributors. Floating-point arithmetic. Accessed 15 July 2025. 2025. URL: https://en.wikipedia.org/wiki/Floating-point_arithmetic.
2. Micikevicius P, Narang S, Alben J, et al. Mixed precision training. In: *International Conference on Learning Representations (ICLR)*. 2018.
3. Zhou S, Ni Z, Zhou X, Wen H, and Wu Y. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. arXiv preprint arXiv:1606.06160v3 2018.
4. Shen S, Dong Z, Ye Z, et al. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT. In: *AAAI Conference on Artificial Intelligence*. 2020.
5. Jacob B, Kligys S, Chen B, et al. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.
6. Frantar E, Hubis F, and Alistarh D. GPTQ: Accurate Post-training Quantization for Generative Pretrained Transformers. arXiv preprint arXiv:2210.17323v2 2023.
7. Xiao G, Lin J, Seznec M, Wu H, Demouth J, and Han S. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Krause A, Brunskill E, Cho K, Engelhardt B, Sabato S, and Scarlett J. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023:38087–99. URL: https://proceedings.mlr.press/v202/xiao23c.html.