Quantized convolutional neural networks through the lens of partial differential equations

Anonymous Author(s)
Affiliation
Address
email

Abstract

Quantization of Convolutional Neural Networks (CNNs) is a common approach to ease the computational burden involved in the deployment of CNNs. However, fixed-point arithmetic is not natural to the type of computations involved in neural networks. In our work, we consider symmetric and stable variants of common CNNs for image classification, and Graph Convolutional Networks (GCNs) for graph node-classification. We demonstrate through several experiments that the property of forward stability preserves the action of a network under different quantization rates, allowing stable quantized networks to behave similarly to their non-quantized counterparts while using fewer parameters. We also find that at times, stability aids in improving accuracy. These properties are of particular interest for sensitive, resource-constrained or real-time applications.

1 Introduction

Deep neural networks have demonstrated superiority in solving many real world problems. In particular, CNNs are an effective approach for processing structured high-dimensional data. Consequently, the demand for the deployment of CNNs on resource-constrained devices is constantly increasing, where at the same time, CNNs grow larger than ever. CNNs still face critical challenges despite this success and their predictions can be highly sensitive to perturbations of the input [32, 18]. Because they often require extremely deep and wide architectures they often impose a high computational cost and make deployment on resource-constrained devices prohibitive, especially for mission-critical applications such as autonomous driving.

Several authors have established a direct link between CNNs and partial differential equations (PDEs) [12, 8, 21]. The connection is two-fold. First, CNNs filter input features with multiple layers, employing both elementwise non-linearities and affine transformations, which can be seen as linear combinations of the finite difference discretizations of spatial derivative operators [35, 15, 14]. Furthermore, the popular residual network architectures (ResNets) [23] can be interpreted as a discrete time integration of a non-linear ODE using the forward Euler scheme. Another possibility is to view the network as a continuous function [10, 35, 20, 17]. This conceptual framework gives rise to many interesting questions such as: what is the importance of the forward stability of CNNs, and can we create analogues to stable time integration through Courant-Friedrichs-Lewy conditions, to make CNNs more robust to errors? Specifically, in this work we consider round-off and quantization errors.

Quantization methods enable neural networks to carry out computations with fixed-point operations rather than floating-point arithmetic. This contributes to the efficiency of the networks and reduces their memory footprint, often at the cost of accuracy [11]. Accordingly we consider a per-layer, uniform and static (same number of bits for every layer) quantization-aware training method.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

In our work we wish to examine the concept of quantized CNNs by considering that forward time integration in PDEs inevitably generates error at every step. To satisfy forward stability, this error must decay from time step to time step, such that the integration error is bounded, resulting in a discrete approximation close to the real continuous solution of the PDE. Therefore, a significant contribution of our work is the promotion of stability in quantized CNNs via small changes to common architectures. We examine the behaviour of quantization under symmetric and stable, heat equation-like CNNs [21, 1, 2]. We show that the quantization process produces significantly lighter-weight networks, with minimal loss of accuracy, and study the *consistency* of quantization methods in CNNs, a key attribute in the construction of quantized variants of CNNs. To this end, we measure the aforementioned similarity using symmetric (potentially stable) and non-symmetric (potentially unstable) networks. Our experiments indicate that symmetric and stable networks achieve better consistency and that *quantized networks are expected to perform better and more in line with their full-precision counterparts when stable architectures are used, rather than unstable architectures*.

Neural networks are known to be susceptible to noise in their inputs [19]. Similarly, the quantization of the activation maps inevitably adds noise to the input of *every* layer. While the quantization error is not chosen specifically for a given input, it does appear in every layer. The introduction of such errors throughout the layers of the network resembles the approximation and round-off errors in time integration of PDEs which require forward stability to converge. To relieve this phenomenon, we follow the approach of constructing stable architectures that are more robust against such errors [21].

We call a discrete forward propagation *stable* if it prevents any perturbation from growing as it propagates through time steps. More explicitly, an N-layered network is stable if there exists some $M > 0$ such that

$$\|\mathbf{x}_N - \tilde{\mathbf{x}}_N\| \leq M\|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|, \quad i = 1, ..., N-1 \tag{1}$$

where $\mathbf{x}_i$ and $\tilde{\mathbf{x}}_i$ are the true and perturbed feature maps of the i-th layer, respectively. The Jacobian of $\mathbf{x}_N$ with respect to $\mathbf{x}_i$ can then be bounded proportionally to M, and accordingly, any stable architecture needs to have a bounded Jacobian [35]. See Appendix A for a more precise formulation.

2 Stable and quantized residual networks

To promote stability in CNNs and to prevent the amplification of quantization errors, we would like the quantized network to behave similarly (in terms of its activations) to a non-quantized instance of the same network. The symmetric variant of ResNet, together with the activation quantization operator is given as follows:

$$\mathbf{x}_{j+1} = Q_b(\mathbf{x}_j - h\mathbf{K}_j^\top \, Q_b(\sigma(\mathbf{K}_j\mathbf{x}_j))), \tag{2}$$

where Q_b denotes the quantization operation. We consider the same weights $\mathbf{K}$ between the two architectures, whether quantized or not, and also assume that the absolute quantization error for any scalar is bounded by some δ. We analyze the propagation of the quantization error and show that it is the Jacobian matrix of the ResNet block that multiplies the error at every iteration and propagates the previous error into the next block. Assuming that σ is non-decreasing, it means that $\mathbf{K}_j^\top \sigma'()\mathbf{K}_j$ is positive semi-definite, and with a proper choice of $\mathbf{K}_j$ and h, we can force $\rho(\mathbf{J}_j) < 1$, so that the error decays. To ensure this forward stability we must set $h < 2(L\|K_j\|_2^2)^{-1}$ for every layer j in the network, where L is the upper bound for $\sigma'()$ [1]. This is generally possible in ResNets only if we use the symmetric variant in of ResNet. We measure the quantization error by

$$\text{MSE}(\mathbf{x}, \mathbf{x}_b) = \frac{1}{n}\sum_{i=1}^{n}(x_i - (x_b)_i)^2, \tag{3}$$

and reduce this MSE to maximize the accuracy of the quantized network, as shown in [4].

3 Stable and quantized GCNs

We also study the importance of stability for quantized Graph Convolution Networks (GCNs), which can be thought of as a PDE discretized on unstructured grids [14]. Specifically, we use the diffusive PDE-GCN architecture formulated in [13], which utilizes symmetric operators, only on unstructured graphs:

$$\mathbf{x}_{j+1} = \mathbf{x}_j - h\mathbf{S}_j^\top\mathbf{K}_j^\top\sigma(\mathbf{K}_j\mathbf{S}_j\mathbf{x}_j). \tag{4}$$

Table 1: CIFAR-10 image classification on quantized stable and unstable networks, and their consistency with their non-quantized counterparts.

Architecture	**Params** (M)	**Accuracy** (%) FP/FP	4W/8A	4W/4A	**MSE (Acc.)** 4A $\rightarrow$ 32A
ResNet56 (orig.)	0.85	93.8	93.0	92.6	0.076 (92.9)
Stable ResNet56 (ours)	0.41	92.1	92.3	91.6	0.024 (92.0)
MobileNetV2 (orig.)	2.20	94.0	92.9	92.4	0.067 (92.5)
Stable MobileNetV2 (ours)	1.70	93.1	92.2	91.6	0.034 (91.7)

Table 2: CIFAR-100 image classification on quantized stable and unstable networks, and their consistency with their non-quantized counterparts.

Architecture	**Params** (M)	**Accuracy** (%) FP/FP	4W/8A	4W/4A	**MSE (Acc.)** 4A $\rightarrow$ 32A
ResNet34 (orig.)	21.3	78.5	75.1	74.7	0.019 (74.7)
Stable ResNet34 (ours)	9.60	76.8	75.4	75.0	0.0083 (75.2)
ResNet56 (orig.)	0.86	72.0	69.0	69.6	0.14 (70.0)
Stable ResNet56 (ours)	0.41	68.4	66.7	66.1	0.031 (67.3)
MobileNetV2 (orig.)	2.30	74.2	71.1	69.8	0.083 (70.6)
Stable MobileNetV2 (ours)	1.80	73.1	71.6	70.9	0.058 (71.0)

Here, $\mathbf{x}_j$ are the features defined over the nodes of a graph, $\mathbf{K}_j$ is a learnt 1×1 convolution operator and $\mathbf{S}_j$ is either learnt (e.g., as in [14]) or pre-defined spatial operation (e.g., the graph Laplacian), both for the j-th layer of the network. Here, the forward stability is guaranteed in the continuous case by the symmetry of the operator [13]. Refer to Appendix A for a more precise formulation.

We refer to a network governed by the dynamics in Eq. (4) as PDE-GCN$_{\text{D}}$(sym.). Analogously, we define the non-symmetric residual layer:

$$\mathbf{x}_{j+1} = \mathbf{x}_j - h\mathbf{S}_j^\top \mathbf{K}_{j_2} \sigma(\mathbf{K}_{j_1} \mathbf{S}_j \mathbf{x}_j) \tag{5}$$

Where $\mathbf{K}_{j_1}$ and $\mathbf{K}_{j_2}$ are distinct learnt 1×1 convolution operators. This formulation does not necessarily yield a symmetric operator, thus, we denote by such a network by PDE-GCN$_{\text{D}}$(non-sym.). The quantization for Eq. (4) and (5) is applied to the weights and before each convolution operator.

4 Image classification using stable and quantized networks

We evaluate our symmetric architectures under quantization by testing them using ResNet-34, ResNet-56 [23] and MobileNetV2 [36]. We conduct several experiments using CIFAR-10/100 and present their results in Tab. 1 and 2. Symmetric networks achieve similar accuracy as non-symmetric networks, while using approximately half of the parameters, a significant saving obtained as a by-product of promoting stability.

Furthermore, we quantify the notion of stability by comparing the behaviour of symmetric and non-symmetric networks under 4-bit quantization. In addition to the 4-bit quantized network, we relax the quantization of the activation maps, allowing them to use the full 32-bit precision, and we measure the divergence between corresponding activations in both runs of the same network - one at 4 bits and the other at 32 bits. This divergence is summarized as the per-entry MSE between the activation maps throughout the two networks. Tables 1 and 2 show that the divergence of the two runs is greater in the non-symmetric (original) network variants, as predicted by the theoretical analysis. This is illustrated in Fig. 1, where we plot the same MSE difference along the layers of each network.

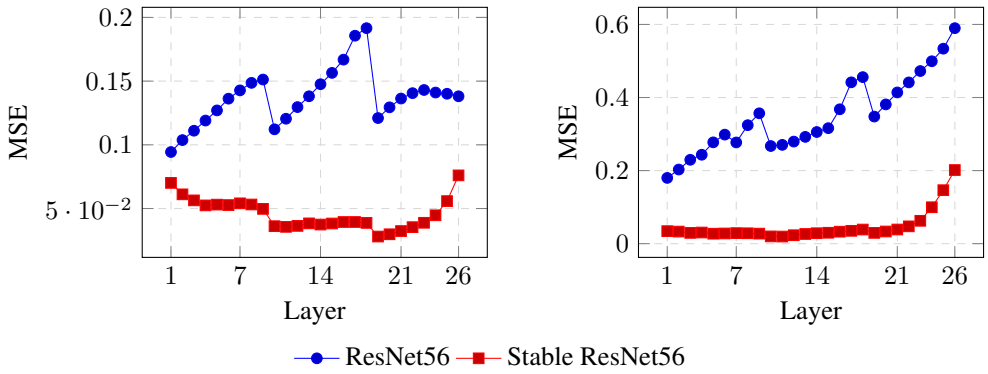


Figure 1: Per-layer MSE between the activation maps of symmetric and non-symmetric network pairs - in each pair one network has quantized activation maps and the other does not. The symmetric variants (red) exhibit a bounded divergence, while the non-symmetric networks diverge as the information propagates through the layers (blue), and hence are unstable. Both networks in each pair achieve similar classification accuracy. ResNet56 on CIFAR-10 (left) and CIFAR-100 (right).

Table 3: PDE-GCN$_{\mathrm{D}}$ symmetric vs non-symmetric. All networks are of 32 layers and and float-precision weights. Activation quantization of 8 and 4 bits is applied.

Data-set	**Architecture**	**Params** (M)	**Accuracy** (%) FP	4W/8A	4W/4A	**MSE** 4A $\rightarrow$ 32A
Cora	PDE-GCN$_{\mathrm{D}}$ (non-sym.)	0.35	82.7	82.2	75.7	6.11
	PDE-GCN$_{\mathrm{D}}$ (sym.)	0.22	84.3	84.0	79.4	2.03
CiteSeer	PDE-GCN$_{\mathrm{D}}$ (non-sym.)	5.14	73.9	72.6	71.1	20.48
	PDE-GCN$_{\mathrm{D}}$ (sym.)	3.04	75.6	74.1	72.2	12.44
PubMed	PDE-GCN$_{\mathrm{D}}$ (non-sym.)	4.32	79.0	79.3	75.1	5.14
	PDE-GCN$_{\mathrm{D}}$ (sym.)	2.22	80.2	80.1	77.6	2.52

5 Semi-supervised node-classification with quantized GCNs

We employ PDE-GCN$_{\mathrm{D}}$(sym.) and PDE-GCN$_{\mathrm{D}}$(non-sym.) from Eq. (4)-(5) for semi-supervised node-classification on the Cora, CiteSeer and PubMed data-sets. Namely, we measure the impact of symmetry on the network's accuracy, as well as its ability to maintain similar behaviour with respect to activation similarity under activation quantization of 8 and 4 bits. In all experiments, we use 32 layers, with 64 hidden channels for Cora, and 256 hidden channels for CiteSeer and PubMed. The training and evaluation procedure we used is identical to [28]. Details on initialization and hyper-parameter tuning, are identical to [13]. The results provided in Tab. 3 reveal two benefits of a symmetric (stable) formulation over a non-symmetric (unstable) one. First, it is apparent that that the former results in better accuracy, often by over 2%, while using almost half the number of parameters. In addition, the action of the network is better preserved under quantization using the symmetric formulation. We note that the symmetric formulation in Eq. (4) is a sub-set of the non-symmetric counterpart in Eq. (5). Therefore, theoretically, both networks can achieve identical expressiveness. However, as demonstrated in Tab. 3, this was not observed in our experiments. We attribute this gap to the smoother optimization process of the stable PDE-GCN$_{\mathrm{D}}$(sym.).

6 Summary

In this work, we explored quantized neural networks from the perspective of PDEs and demonstrated that stability preserves the action of a network under different quantization rates. We find that at times, stability aids to improve accuracy. These properties are of particular interest for resource-constrained, low-power or real-time applications like autonomous driving.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [No]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

References

[1] Tobias Alt, Pascal Peter, Joachim Weickert, and Karl Schrader. Translating numerical concepts for pdes into neural architectures. *arXiv preprint arXiv:2103.15419*, 2021.

[2] Tobias Alt, Karl Schrader, Matthias Augustin, Pascal Peter, and Joachim Weickert. Connections between numerical algorithms for pdes and neural networks. *arXiv preprint arXiv:2107.14742*, 2021.

[3] L. Ambrosio and V. M. Tortorelli. Approximation of functional depending on jumps by elliptic functional via t-convergence. *Communications on Pure and Applied Mathematics*, 43:999–1036, 1990.

[4] R. Banner, Y. Nahshan, and D. Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. *NeurIPS*, pages 7948–7956, 2019.

[5] Yoshua Bengio. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv*, 2013.

[6] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[7] T.F. Chan and L.A. Vese. Active contours without edges. *IEEE Transactions on Image Processing*, 10(2):266–277, 2001. doi:10.1109/83.902291.

[8] Pratik Chaudhari, Adam Oberman, Stanley Osher, Stefano Soatto, and Guillaume Carlier. Deep relaxation: partial differential equations for optimizing deep neural networks. *Research in the Mathematical Sciences*, 5(3):30, 2018.

[9] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *37th International Conference on Machine Learning (ICML)*, volume 119, pages 1725–1735, 2020.

[10] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

[11] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136, 2018.

[12] Weinan E. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, March 2017.

[13] Moshe Eliasof, Eldad Haber, and Eran Treister. Pde-gcn: Novel architectures for graph neural networks motivated by partial differential equations. *arXiv preprint arXiv:2108.01938*, 2021.

[14] Moshe Eliasof and Eran Treister. DiffGCN: Graph convolutional networks via differential operators and algebraic multigrid pooling. *34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.*, 2020.

[15] Jonathan Ephrath, Moshe Eliasof, Lars Ruthotto, Eldad Haber, and Eran Treister. LeanConvNets: Low-cost yet effective convolutional neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):894–904, 2020.

[16] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[17] A. Gholami, K. Keutzer, and G. Biros. ANODE: Unconditionally accurate memory-efficient gradients for neural odes. In *IJCAI*, 2019.

[18] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. Making machine learning robust against adversarial inputs. *Communications of the ACM*, 61(7):56–66, 2018.

[19] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[20] Eldad Haber, Keegan Lensink, Eran Triester, and Lars Ruthotto. Imexnet: A forward stable deep neural network. *arXiv preprint arXiv:1903.02639*, 2019.

[21] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, pages 1–21, 2017.

[22] Song Han. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv*, 2016.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[24] Berthold K. P. Horn and B. G. Schunck. Determining optical flow. *Artif. Intell.*, 17:185–203, 1981.

[25] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[26] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18:187:1–187:30, 2017.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *The International Conference on Learning Representations (ICLR)*, 2017.

[29] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[30] Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. *International Conference on Learning Representations (ICLR)*, 2019.

[31] Google LLC. gemmlowp: a small self-contained low-precision gemm library, 1999. URL: `https://github.com/google/gemmlowp`.

[32] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2016.

[33] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12:629–639, 1990.

[34] L. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60:259–268, 1992.

[35] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pages 1–13, 2019.

[36] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[37] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[38] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[39] Penghang Yin, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin. Blended coarse gradient descent for full quantization of deep neural networks. *Research in the Mathematical Sciences*, 6(1):1–23, 2019.

[40] Linan Zhang and Hayden Schaeffer. Forward stability of resnet and its variants. *Journal of Mathematical Imaging and Vision*, 62(3):328–351, 2020.

A Continuous neural networks and stable ResNet architectures

The general goal of supervised machine learning is to model a function $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$ and train its parameters $\boldsymbol{\theta} \in \mathbb{R}^p$ such that

$$f(\mathbf{y}, \boldsymbol{\theta}) \approx \mathbf{c} \tag{6}$$

for input-output pairs $\{(\mathbf{y}^i, \mathbf{c}^i)\}_{i=1}^s$ from a certain data set $\mathcal{Y} \times \mathcal{C}$. Typical tasks include regression or classification, and the function f can be viewed as an interpolation of the true function that we want the machine to learn.

In learning tasks involving images and videos, the learnt function f commonly includes a CNN that filters the input data. Among the many architectures we focus on feed-forward networks of a common type called by ResNets [23]. In the simplest case, the forward propagation of a given example $\mathbf{y}$ through an N-layer ResNet can be written as

$$\mathbf{x}_{j+1} = \mathbf{x}_j + hF(\mathbf{x}_j, \boldsymbol{\theta}_j), \quad j = 0, \ldots, N-1, \quad \mathbf{x}_0 = F_{\text{open}}(\mathbf{y}, \boldsymbol{\theta}_{\text{open}}). \tag{7}$$

The layer function F consists spatial convolution operators parameterized by the weights $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N$, and non-linear element-wise activation functions. The parameter $h > 0$ in this notation serves as a time step, arising from a discretization of a continuous network, as we show in the next section. The classical ResNet layer reads

$$F_{\text{ResNet}}(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{K}_1 \sigma\left(\mathbf{K}_2 \mathbf{x}\right). \tag{8}$$

where $\mathbf{K}_1, \mathbf{K}_2$ are two different convolution operators parameterized by θ, and σ is a non-linear activation function, like $\sigma(x) = \max\{x, 0\}$, known as the Rectified Linear Unit (ReLU) function. The term F_{open} denotes an opening layer that typically outputs a n_c-channel image $\mathbf{x}_0$, where n_c is greater than the number of input channels, which is typically 3 for RGB images, while possibly also reducing the size of the image. Usually, the opening layer reads

$$\mathbf{x}_0 = \sigma(\mathbf{K}_{\text{open}}\mathbf{y}) \tag{9}$$

where $\mathbf{K}_{\text{open}}$ is a convolution operator parameterized by $\boldsymbol{\theta}_{\text{open}}$ that widens the number of channels from this point onward in the network.

Continuous neural networks have recently been suggested as an abstract generalization of the more common discrete network, where the network is viewed as a discretized instance of a continuous ODE or PDE. As shown by [8, 12, 21, 10], Eq. (7) (or a ResNet) is essentially a forward Euler discretization of a continuous non-linear ODE

$$\partial_t \mathbf{x}(t) = F(\mathbf{x}(t), \boldsymbol{\theta}(t)), \quad t \in [0, T], \quad \mathbf{x}(0) = \mathbf{x}_0 = F_{\text{open}}(\mathbf{y}, \boldsymbol{\theta}_{\text{open}}), \tag{10}$$

where $[0, T]$ is an artificial time interval related to the depth of the network. Relating spatial convolution filters with differential operators, the authors of [35] propose a layer function representation F that renders Eq. (10) similar to a parabolic diffusion-like PDE

$$F_{\text{sym}}(\mathbf{x}, \boldsymbol{\theta}) = -\mathbf{K}^\top \sigma\left(\mathbf{K}\mathbf{x}\right). \tag{11}$$

For example, when $\mathbf{K}$ represents a discrete gradient operator and $\sigma(x) = x$, we obtain the heat equation under this treatment. The approach of Eq. (11) is natural, as similar developments have led to several breakthroughs in image processing, including optical flow models for motion estimation [24], non-linear anisotropic diffusion models for image denoising [33, 34], and variational methods for image segmentation [3, 7]. To best balance the network's representational abilities with its computational cost, a "bottleneck" structure is used commonly, exploiting a different number of input and output channels in $\mathbf{K}$, as in Eq. (11).

Generally speaking, the training of the neural network model consists of finding parameters $\boldsymbol{\theta}$ such that Eq. (6) is approximately satisfied for examples from a training data set. The same should also hold for examples from a validation data set, which is not used to optimize the parameters. The training objective is commonly modeled as an expected loss to be minimized, denoted as

$$\min_{\boldsymbol{\theta}} \Phi(\boldsymbol{\theta}), \quad \text{where} \quad \Phi(\boldsymbol{\theta}) = \frac{1}{s}\sum_{k=1}^{s} \text{loss}(f(\mathbf{y}^k, \boldsymbol{\theta}), \mathbf{c}^k) + R(\boldsymbol{\theta}). \tag{12}$$

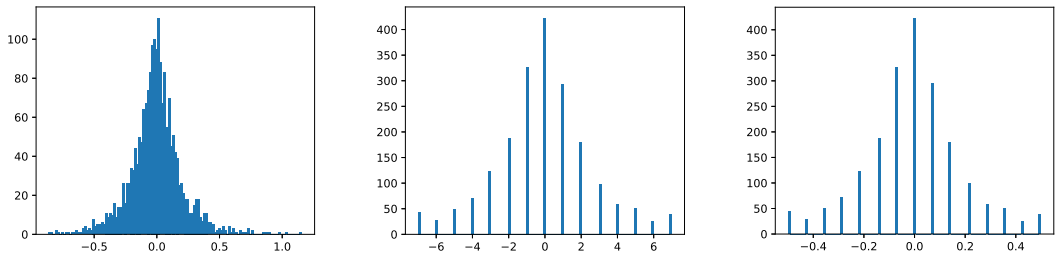
Here, $(\mathbf{y}^1, \mathbf{c}^1), \ldots, (\mathbf{y}^s, \mathbf{c}^s) \in \mathcal{Y} \times \mathcal{C}$ are the training data, $f(\mathbf{y}, \boldsymbol{\theta})$ includes the action of the neural network, but may also contain other layers, e.g., fully-connected layers, opening and connective layers, and softmax transformations in classification. R is a regularization term. The optimization problem in Eq. (12) is typically solved with gradient-based non-linear optimization techniques [6]. Due to the large-scale and stochastic nature of the learning problem, it is common to use stochastic approximation schemes such as variants of stochastic gradient descent (SGD) like Adam [27]. These methods perform iterations using gradient information from randomly chosen subsets of the data.

B Stable and quantized residual networks

As discussed earlier, our goal is to promote stability in CNNs, to prevent the amplification of quantization errors. To obtain that, we would like the quantized network to behave similarly (in terms of its activations) to a non-quantized instance of the same network. The symmetric variant of ResNet in Eq. (8) and (11), together with the activation quantization operator in Eq. (18) is given as follows:

$$\mathbf{x}_{j+1} = Q_b(\mathbf{x}_j - h\mathbf{K}_j^\top \, Q_b(\sigma(\mathbf{K}_j\mathbf{x}_j))), \tag{13}$$

where Q_b denotes the quantization operation in Eq. (18). Let us denote the feature maps of the non-quantized network by $\hat{\mathbf{x}}_j$ (i.e., assuming $q_b(x) = x$ in Eq. (18)), and let the error between corresponding activations be denoted by $\eta_j = \mathbf{x}_j - \hat{\mathbf{x}}_j$. We consider the same weights $\mathbf{K}$ between the two architectures, whether quantized or not, and also assume that the absolute quantization error for any scalar is bounded by some δ. We analyze the propagation of the quantization error η_{j+1} as a function of η_j.



(a) Original distribution (b) Integer representation (c) Real quantized values

Figure 2: An example of a uniform signed 4-bit quantized weight tensor. The original distribution is given in (a). The original values are then clipped to the range $[-\alpha, \alpha]$ (in this example $\alpha = 0.5$), divided by α and multiplied by 7 ($= 2^{4-1} - 1$, for 4 bits). Then, the values are quantized to integer values in (b), and scaled back (multiplied by $\alpha/7$) to their original values in (c).

We start unwrapping the block in Eq. (2). First, we subtract $\hat{\mathbf{x}}_{j+1}$ from both sides, and replace the outer quantization with the error term η_{j_1}.

$$\eta_{j+1} = \eta_{j_1} + \mathbf{x}_j - h\mathbf{K}_j^\top Q_b(\sigma(\mathbf{K}_j\mathbf{x}_j)) - \hat{\mathbf{x}}_{j+1} \tag{14}$$

Next, we remove the other instance of Q_b and add another error term η_{j_2}:

$$\eta_{j+1} = \eta_{j_1} - h\mathbf{K}_j^\top \eta_{j_2} + \mathbf{x}_j - h\mathbf{K}_j^\top \sigma(\mathbf{K}_j\mathbf{x}_j) - \hat{\mathbf{x}}_{j+1}. \tag{15}$$

Remembering that $\mathbf{x}_j = \hat{\mathbf{x}}_j + \eta_j$, and using the first-order approximation of σ, we note that $\sigma(\mathbf{K}_j\mathbf{x}_j) \approx \sigma(\mathbf{K}_j\hat{\mathbf{x}}_j) + \sigma'(\mathbf{K}_j\hat{\mathbf{x}}_j)\mathbf{K}_j\eta_j$. Therefore, we have

$$\begin{aligned}
\eta_{j+1} &= \eta_{j_1} - h\mathbf{K}_j^\top \eta_{j_2} + \hat{\mathbf{x}}_j + \eta_j - h\mathbf{K}_j^\top(\sigma(\mathbf{K}_j\hat{\mathbf{x}}_j) + \sigma'(\mathbf{K}_j\hat{\mathbf{x}}_j)\mathbf{K}_j\eta_j) - \hat{\mathbf{x}}_{j+1} \\
&= \eta_{j_1} - h\mathbf{K}_j^\top \eta_{j_2} + \eta_j - h\mathbf{K}_j^\top \sigma'(\mathbf{K}_j\hat{\mathbf{x}}_j)\mathbf{K}_j\eta_j \\
&= (\mathbf{I} - h\mathbf{K}_j^\top \sigma'(\mathbf{K}_j\hat{\mathbf{x}}_j)\mathbf{K}_j)\eta_j + \eta_{j_1} - h\mathbf{K}_j^\top \eta_{j_2}.
\end{aligned} \tag{16}$$

The key ingredient of the analysis above is that η_{j_1} and η_{j_2} are fixed and bounded for every layer. On the other hand, it is the Jacobian matrix of the block $\mathbf{J}_j = \mathbf{I} - h\mathbf{K}_j^\top \sigma'(\mathbf{K}_j\hat{\mathbf{x}}_j)\mathbf{K}_j$ that multiplies η_j at every iteration and propagates the previous error into the next block. Assuming that σ is non-decreasing, it means that $\mathbf{K}_j^\top \sigma'()\mathbf{K}_j$ is positive semi-definite, and with a proper choice of $\mathbf{K}_j$ and h, we can force $\rho(\mathbf{J}_j) < 1$, so that the error decays. To ensure this forward stability we must set $h < 2(L\|K_j\|_2^2)^{-1}$ for every layer j in the network, where L is the upper bound for $\sigma'()$ [1]. This is generally possible in ResNets only if we use the symmetric variant in Eq. (11). [35, 40] also achieved a similar stability result, but not in the context of quantization.

C Quantization-aware training

As mentioned before, in this paper we focus on quantized neural networks, and in particular, on the scenario of quantization-aware training. We restrict the values of the weights to a smaller set, so that after training, the calculation of a prediction by the network can be carried out in fixed-point integer arithmetic. The intermediate hidden layers are quantized (rounded) as well. Even though the quantization involves a discontinuous rounding function, the discrete weights can be optimized using gradient-based methods, also known as quantization-aware training schemes [22, 39]. During the forward pass, both the weights and hidden activation maps are quantized, while during gradient calculation in the backward pass, derivative information is passed through the rounding function, whose exact derivative is zero. This method is known as the Straight Through Estimator (STE) [5]. We note that the gradient-based optimization for the weights is applied in floating-point arithmetic as we describe next, but during inference, the weights and activations are quantized, and all operations are performed using integers only.

We now present the details of the quantization scheme that we use, based on [30]. First, we define the pointwise quantization operator:

$$q_b(t) = \frac{\text{round}((2^b - 1) \cdot t)}{2^b - 1}, \tag{17}$$

where t is a real-valued scalar in one of the ranges [-1, 1] or [0, 1] for signed or unsigned quantization[1], respectively. b is the number of bits that are used to represent t as integer at inference time (during training, t and $q_b(t)$ are real-valued). During the forward pass, we force each quantized value to one of the ranges above by applying a clipping function to the quantized weights and activations before applying Eq. (17):

$$\begin{aligned} w_b &= Q_b(w) = \alpha_w q_{b-1}(\text{clip}(\frac{w}{\alpha_w}, -1, 1)) \\ x_b &= Q_b(x) = \alpha_x q_b(\text{clip}(\frac{x}{\alpha_x}, 0, 1)). \end{aligned} \tag{18}$$

Here, w, w_b are the real-valued and quantized weight tensors, x, x_b are the real-valued and quantized input tensors, and α_w, α_x are their associated clipping parameters (also called scales), respectively. An example of Eq. (18) applied to a weight tensor using 4-bit signed quantization, is given in Fig. 2. During training, we iterate on the floating-point values of the weights w, while both the weights and activation maps are quantized in the forward pass (i.e., x_b and w_b are passed through the network). The STE [5] is used to compute the gradient in the backward pass, where the derivative of q_b is ignored and we use the derivatives w.r.t w_b in the SGD optimization to update w iteratively.

The quantization scheme in Eq. (18) involves the clipping parameters α_w and α_x, also called scales, that are used to translate the true value of each integer in the network to its floating-point value. This way, at inference time, two integers from different layers can be multiplied in a meaningful way that is faithful to the original floating-point values [31, 26]. These scales also control the quantization error, and need to be chosen according to the values propagating through the network and the bit allocation b, whether the weights or the activations.

Alternatively, the works [30, 16], which we follow here, introduced an effective gradient-based optimization to find the clipping values α_x, α_w for each layer. Given Eq. (18) the gradients w.r.t α_w, and α_x can be approximated using the STE [30, 16]. For the activation maps, for example, this resolves to:

$$\frac{\partial x_b}{\partial \alpha_x} = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x \geq \alpha_x \\ \frac{x_b}{\alpha_x} - \frac{x}{\alpha_x} & \text{if } 0 < x < \alpha_x. \end{cases} \tag{19}$$

This enables the quantized network to be trained end-to-end manner with backpropagation. To further improve the optimization, [30] normalize the weights before quantization:

$$\hat{w} = \frac{w - \mu}{\sigma + \epsilon}. \tag{20}$$

Here, μ and σ are the mean and standard deviation of the weight tensor, respectively, and $\epsilon = 10^{-6}$.

D Stable channel and resolution changing layers

The classical CNN architecture typically applies several layers like Eq. (7). However, to obtain better representational capabilities for the network, the number of channels typically increases every few layers. In CNNs, this is typically accompanied by a down-sampling operation performed by a pooling layer or a strided convolution. In such cases, the residual equation Eq. (7) cannot be used, since the update $F(\mathbf{x}_j, \boldsymbol{\theta}_j)$ does not match $\mathbf{x}_j$ in terms of dimensions. For this reason, ResNets typically include 3-4 steps like

$$\mathbf{x}_{j+1} = \mathbf{K}_j \mathbf{x}_j + F(\mathbf{x}_j, \boldsymbol{\theta}_j), \quad \text{or} \quad \mathbf{x}_{j+1} = F(\mathbf{x}_j, \boldsymbol{\theta}_j), \tag{21}$$

throughout the network to match the changing resolution and number of channels. These layers are harder to control than Eq. (7) and (11) if one wishes to ensure stability.

In this work we are interested in testing and demonstrating the theoretical property described in Sec. 2 empirically, using fully stable networks (except for the very first and last layers). To this end, when

[1] We assume that the ReLU activation function is used in between any convolution operator, resulting in non-negative activation maps, and can be quantized using an unsigned scheme. If a different activation function is used that is not non-negative, like $\tanh()$, signed quantization should be used instead.

increasing the channel space from $n_{c_{in}}$ to $n_{c_{out}}$ for $\mathbf{x}_{j+1}$ and $\mathbf{x}_j$ respectively, we simply concatenate the output of the step with the necessary channels from the current iteration to maintain the same dimensions. That is, we apply the following:

$$\mathbf{x}_{j+1} = \begin{bmatrix} \mathbf{x}_j + hF(\mathbf{x}_j, \boldsymbol{\theta}_j) \\ (\mathbf{x}_j)_{1:n_{c_{out}} - n_{c_{in}}} \end{bmatrix}, \tag{22}$$

where $(\mathbf{x}_j)_{1:n_{c_{out}} - n_{c_{in}}}$ are the first $n_{c_{out}} - n_{c_{in}}$ channels of $\mathbf{x}_j$. This assumes that each of the channel changing steps satisfies $n_{c_{in}} \leq n_{c_{out}} \leq 2n_{c_{in}}$, which is quite common in CNNs. This way, we only use the symmetric dynamics as in Eq. (11) throughout the network, which is guaranteed to be stable for a proper choice of parameters. Finally, we do not apply strides as part of Eq. (22), and to reduce the channel resolution we simply apply average pooling following Eq. (22), which is a stable, parameter-less, operation.

E A stable variant of MobileNetV2

In addition to standard residual networks, we also consider the popular MobileNet family of light-weight CNN architectures [36, 25], which are the most common architectures for edge devices, achieving very nice results while requiring modest computational resources to deploy. These architectures utilize the "inverse bottleneck" structure, where the channel space throughout the network is relatively small, but in every step it is expanded and reduced by 1×1 convolutions, with "depthwise" 3×3 convolutions in the middle expanded channel space. The depthwise convolutions apply 3×3 kernels on each channel without mixing between the channels, and hence they are less expensive than 1×1 convolutions. The general structure of MobileNetV2 [36] reads:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{K}^3_{1\times1}\sigma(\mathbf{K}^2_{\mathsf{dw}}\sigma(\mathbf{K}^1_{1\times1}\mathbf{x}^{(k)})) \tag{23}$$

where $\mathbf{K}^3_{1\times1}, \mathbf{K}^1_{1\times1}$ are two different learnable 1×1 convolution filters, and $\mathbf{K}^2_{\mathsf{dw}}$ is a learnable depthwise 3×3 convolution matrix. Due to the relatively small channel space used throughout the architecture, the quantization of this network to low bit-rates is challenging.

The MobileNetV2 architecture in Eq. (23) can also be seen as a residual network, and is generally unstable. Because of its inverse bottleneck structure, it does not fit the structure of Eq. (7) and Eq. (11). To test the importance of stability under quantization for such networks as well, we define the stable MobileNetV2 variant:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (\mathbf{K}^1_{1\times1})^\top((\mathbf{K}^2_{\mathsf{dw}})^\top\sigma(\mathbf{K}^2_{\mathsf{dw}}\mathbf{K}^1_{1\times1}\mathbf{x}^{(k)})), \tag{24}$$

where now, the separable convolution operator $\mathbf{K}^2_{\mathsf{dw}}\mathbf{K}^1_{1\times1}$ takes the role of the single operator $\mathbf{K}$ in Eq. (11). Going between Eq. (23) and (24) we lost one non-linearity, but as we show later, this has marginal influence on the accuracy of the network. We also apply the depthwise operation twice, but this is inexpensive compared to 1×1 convolutions. Changing the resolutions and channel space is done the same way as described in section D.

F Settings and data-sets

Our code is implemented in PyTorch, and all experiments are conducted on an Nvidia RTX-3090 with 24GB of memory. Below, we elaborate on the different data-sets explored throughout the numerical experiments.

CIFAR-10/100. The CIFAR-10/100 image classification data-sets [29] each consist of 60k natural images of size 32×32 where each image is assigned to one of ten categories (for CIFAR-10) or one hundred categories (for CIFAR-100). The data-set includes 50K training examples and 10K test examples. We derive a validation set for training by holding out 10% of the training data and report accuracy metrics on the test data.

Node-classification data-sets. Lastly, we use graph neural networks on three citation network node classification data-sets: Cora, CiteSeer and PubMed [37]. For each data-set, we use the standard train/validation/test split as in [38], with 20 nodes per class for training, 500 validation nodes and 1,000 testing nodes. We use the same training scheme as in [9].

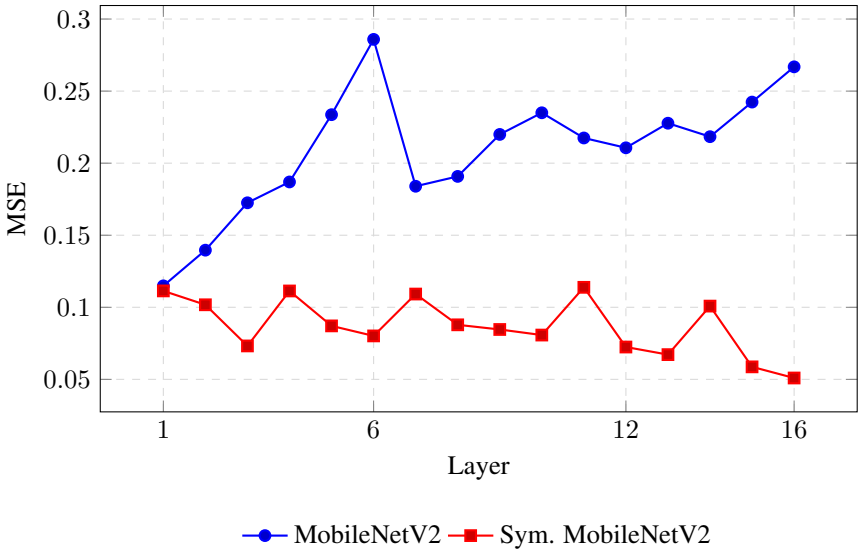


Figure 3: Per-layer MSE between the activation maps of symmetric and non-symmetric MobileNetV2 network pairs - in each pair one network has quantized activation maps and the other does not. Values are normalized per-layer to account for the different dimensions of each layer. In all the cases, the symmetric variants (in red) exhibit a bounded divergence, while the non-symmetric networks diverge as the information propagates through the layers (in blue), and hence they are unstable. Both networks in each pair achieve similar classification accuracy.