Low-Power Approximations of Convolutional Neural Networks

Stefan Duffner
University of Lyon, INSA-Lyon, CNRS, LIRIS, France

Christophe Garcia, Renato J. Cintra, André Leite

28/03/2023

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

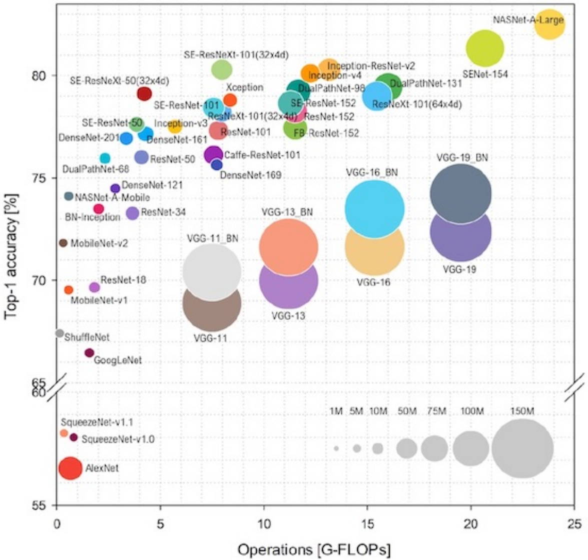
Outline

Outline

Context

- Machine Learning has become a powerful tool for various applications
- Modelling of complex functions for processing/analysing/interpreting data
- State-of-the-art models: Deep Neural Network (DNN)
- Requires huge computational and memory resources
- Especially for training but also for the deployed systems
- Common solution: cloud computing

DNN models for image classification (ImageNet)



Context

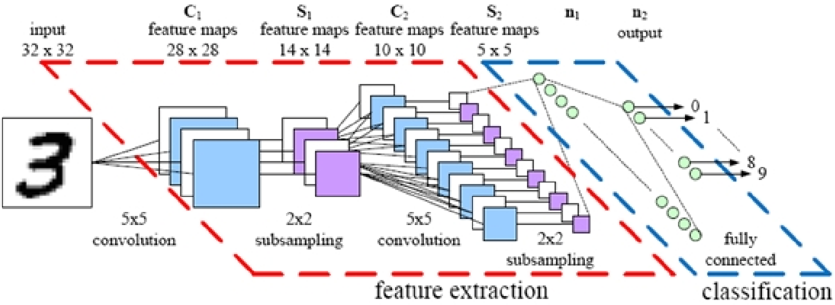
- Can DNNs be integrated in embedded systems, mobile and <u>low-power</u> devices?

Motivation

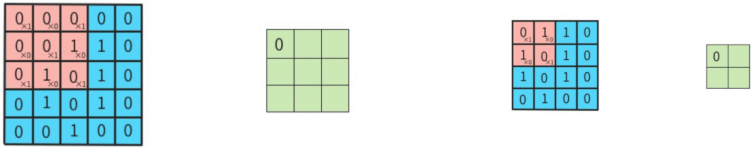
- Reducing server maintenance costs,
- Scalability,
- Reducing network transfer and lag,
- Privacy
- Reducing environmental impact

Outline

Model overview



Basic operations:



Model overview

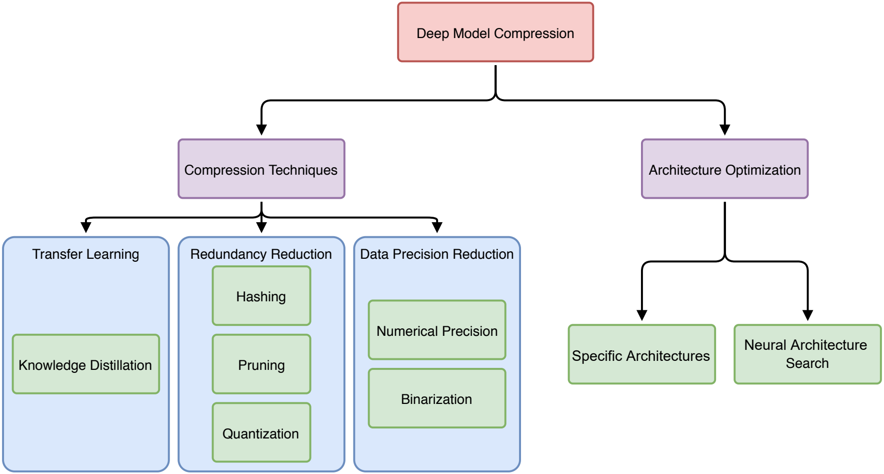
- Convolutions:
  - high computational complexity
  - low memory requirements
- Pooling: no memory and very fast
- Fully connected layers:
  - low computational complexity
  - high memory requirements
- Implementations heavily rely on parallelisation (using GPU)
  $\rightarrow$ high power consumption

Outline

Neural Network Compression

- Very large redundancy in trained DNN models (i.e. the weights)
- Some approaches try to avoid this during the design or training of a model (AutoML)
- Most current approaches:
    1. Train a (highly redundant) DNN
    2. Compress the model
    3. Retrain/refine the model (to compensate for errors)
- Recent DNN models: $\sim 10^6 - 10^7$ parameters, $\sim 10^2$ MB
- Compression rates: $\sim 1:10 - 1:200$
- Very little or no loss in classification performance!

Compression Approaches



Existing code and solutions

- Many implementations on-line (github etc.)
- Tensorflow Lite
- Core ML (Apple)
- CNTK netopt module (Microsoft)
- MXNet quantisation API (Apache) (BMXnet)
- PyTorch pruning/quantization package
- Deep Learning framework N2D2 (CEA LIST) (https://github.com/CEA-LIST/N2D2)
- Apache TVM
- Alibaba MNN
- Hardware-oriented: NVIDIA, Xilinx VITIS

Outline

Our approach

- Post-training quantisation method
- We focus on low-power approximations
- Approximate weights by dyadic rationals
  $\rightarrow$all multiplications replaced by bit-shifts and additions
- We approximate each convolution matrix $\mathbf{M}$ individually by:

$$\hat{\mathbf{M}} = \alpha^* \mathbf{T}^* \, . \tag{1}$$

- Each element of $\mathbf{T}^*$ is a dyadic rational $m/2^n$ from a set $\mathcal{D}$

Examples of $\mathcal{D}$

$$
\begin{aligned}
\mathcal{D}_1 &= \{-1, 0, 1\}\,, \\
\mathcal{D}_2 &= \{-2, -1, 0, 1, 2\}\,, \\
\mathcal{D}_3 &= \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}\,, \\
\mathcal{D}_4 &= \left\{-4, -3, -2, -1, -\frac{3}{4}, -\frac{1}{2}, -\frac{1}{4}, 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1, 2, 3, 4\right\}, \\
\mathcal{D}_5 &= \left\{-7, -6, -5, -4, -3, -2, -1, -\frac{3}{4}, -\frac{1}{2}, -\frac{1}{4}, 0,\right. \\
&\qquad \left.\frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1, 2, 3, 4, 5, 6, 7\right\}, \\
\mathcal{D}_6 &= \left\{-4, -\frac{15}{4}, -\frac{7}{2}, -\frac{13}{4}, \dots, \frac{13}{4}, \frac{7}{2}, \frac{15}{4}, 4\right\}, \\
\mathcal{D}_7 &= \left\{-5, -\frac{19}{4}, -\frac{9}{2}, -\frac{17}{4}, \dots, \frac{17}{4}, \frac{9}{2}, \frac{19}{4}, 5\right\}
\end{aligned}
$$

CSD representation

- $\alpha^* > 0 \in \mathbb{R}$ is a expansion factor approximated by the closest CSD representation (Canonical Signed Digit encoding)
- CSD:
  - CSD presentation of a number consists of numbers 0, 1 and -1.
  - The CSD presentation of a number is unique.
  - The number of nonzero digits is minimal.
  - There cannot be two consecutive non-zero digits.
- Example: 1 0010 000-1 = 256 + 32 -1 = 287

Optimisation

- We formulate this as an optimisation problem:

$$(\alpha^*, \mathbf{T}^*) = \arg\min_{\alpha, \mathbf{T}} \|\mathbf{M} - \alpha \cdot \mathbf{T}\|^2, \tag{2}$$

- Frobenius norm $\|\mathbf{M}\| = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |m_{i,j}|}$
- Mixed integer non-linear programming (INLP)

Optimisation

- Linearisation of problem
- $\mathbf{M} = [m_{i,j}] \quad i,j = 1,2,\ldots,N$ and $r \in \mathcal{D}$
- binary decision variables:

$$x_{i,j}(r) = \begin{cases} 1, & \text{if } m_{i,j} = r, \\ 0, & \text{otherwise.} \end{cases}$$

- (2) can be re-written according to the following binary linear programming problem:

$$\min_{x_{i,j}(r)} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{r \in \mathcal{D}} (r - \alpha \cdot m_{i,j})^2 \cdot x_{i,j}(r), \tag{3}$$

subject to

$$\sum_{r \in \mathcal{D}} x_{i,j}(r) = 1, \qquad i,j = 1,2,\ldots,N.$$

Optimisation

- Each entry of $\mathbf{T}_\alpha$ can be computed as:

$$t_{i,j}^{(\alpha)} = \sum_{r \in \mathcal{D}} r \cdot x_{i,j}^{(\alpha)}(r). \tag{4}$$

- Resulting approximation error:

$$\mathrm{Error}(\alpha) = \|\mathbf{M} - \alpha \cdot \mathbf{T}_\alpha\|^2.$$

- Can be solved in $\mathcal{O}(N)$
- Global optimum value α^*:

$$\alpha^* = \arg\min_\alpha \mathrm{Error}(\alpha), \tag{5}$$

- Solved by simple minimization over a vector of values.

Example

$$\mathbf{M}_0 = \begin{bmatrix} 1.5200701 & 1.0317051 & 0.7906240 & -0.2153791 & -0.2340538 \\ 1.3982610 & 2.1860176 & 2.0152923 & 1.5620477 & 0.8270900 \\ -0.6848867 & 0.7470516 & 1.6923728 & 1.2537112 & 1.1946758 \\ -1.2387477 & -0.5483563 & 0.1261987 & 0.8677799 & 0.7742613 \\ -1.4691808 & -1.2178997 & -0.2924347 & 0.2172496 & 0.1325074 \end{bmatrix}.$$

Example

Solving (2) for the above matrix using $\mathcal{D}_8$, we obtain:

$$
\begin{aligned}
\alpha^* &= 0.30931, \\
\mathbf{T}^* &= \begin{bmatrix} 5 & 3.25 & 2.5 & -0.75 & -0.75 \\ 4.5 & 7 & 6.5 & 5 & 2.75 \\ -2.25 & 2.5 & 5.5 & 4 & 3.75 \\ -4 & -1.75 & 0.5 & 2.75 & 2.5 \\ -4.75 & -4 & -1 & 0.75 & 0.5 \end{bmatrix} \\
&= \frac{1}{4} \cdot \begin{bmatrix} 20 & 13 & 10 & -3 & -3 \\ 18 & 28 & 26 & 20 & 11 \\ -9 & 10 & 22 & 16 & 15 \\ -16 & -7 & 2 & 11 & 10 \\ -19 & -16 & -4 & 3 & 2 \end{bmatrix}.
\end{aligned}
$$

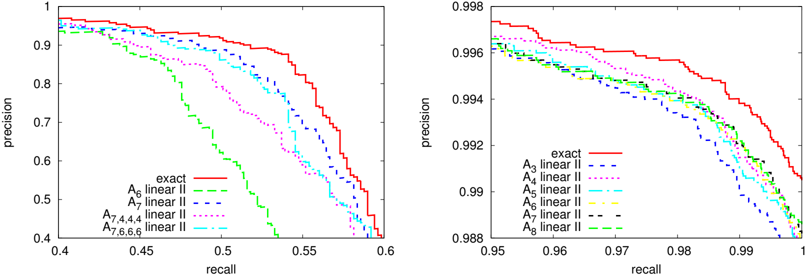
Example

- CSD approximation of α^*:
  $\alpha^* = 0.30931 \approx 2^{-2} + 2^{-4} - 2^{-8} = 0.30859375$.
- Fully multiplierless approximation:

$$\hat{\mathbf{M}} = (2^{-4} + 2^{-6} - 2^{-10}) \cdot \begin{bmatrix} 20 & 13 & 10 & -3 & -3 \\ 18 & 28 & 26 & 20 & 11 \\ -9 & 10 & 22 & 16 & 15 \\ -16 & -7 & 2 & 11 & 10 \\ -19 & -16 & -4 & 3 & 2 \end{bmatrix}.$$

Results

- Tested on three different models of different complexity:
  - Face detection CNN: 1k parameters, 97% of exact model
  - MNIST: 180k parameters, 99% of exact model
  - AlexNet/ImageNet: 1.2M convolution matrices, 96% of exact model
- Different approximations for different layers



Low-complexity face detection with our approach

approximate vs. exact

Results

Mean classification rates for the MNIST test set and different approximations relative to the exact model.

	Exact	ASG	PLAN	Linear I	Linear II	Quadratic I	Quadratic II
Exact	1.0000	1.0000	0.9847	0.9680	0.9978	1.0000	1.0000
A_1	0.9684	0.9684	0.9588	0.9260	0.9615	0.9684	0.9684
A_2	0.9643	0.9643	0.9627	0.8805	0.9573	0.9643	0.9643
A_3	0.9961	**0.9961**	0.9848	0.9655	**0.9944**	**0.9961**	**0.9961**
A_4	0.9973	**0.9973**	0.9863	0.9700	**0.9969**	**0.9973**	**0.9973**
A_5	0.9976	**0.9976**	0.9866	0.9666	**0.9969**	**0.9976**	**0.9976**
A_6	0.9991	**0.9991**	0.9868	0.9701	**0.9973**	**0.9991**	**0.9991**
A_7	0.9992	**0.9992**	0.9846	0.9680	**0.9977**	**0.9992**	**0.9992**
A_8	0.9994	**0.9994**	0.9848	0.9675	**0.9981**	**0.9994**	**0.9994**
$A_{3,3,1,1}$	0.9931	**0.9931**	0.9749	0.9625	**0.9924**	**0.9931**	**0.9931**
$A_{3,1,1,1}$	0.9891	0.9891	0.9684	0.9580	0.9866	0.9891	0.9891
$A_{4,4,1,1}$	0.9937	**0.9937**	0.9780	0.9618	**0.9943**	**0.9937**	**0.9937**
$A_{4,1,1,1}$	0.9885	0.9885	0.9655	0.9572	0.9872	0.9885	0.9885

Further information

- R. J. Cintra, S. Duffner, C. Garcia, A. Leite, "Low-complexity Approximate Convolutional Neural Networks", IEEE Transactions on Neural Networks and Learning Systems, 2018
- Y. Cheng, D. Wang, P. Zhou, T. Zhang, and S. Member, “A Survey of Model Compression and Acceleration for Deep Neural Networks,” IEEE Signal Processing Magazine, 2017
- J. Cheng, P. Wang, G. Li, Q. Hu, and H. Lu, “Recent Advances in Efficient Computation of Deep Convolutional Neural Networks,” Frontiers of Information Technology Electronic Engineering, 2018
- T. Elsken, J. H. Metzen, F. Hutter, "Neural Architecture Search: A Survey", JMLR, 2019
- http://www.automl.org