Fast Implementation of 4-bit Convolutional Neural Networks for Mobile Devices

Anton Trusov[1, 3], Elena Limonova[1, 2, 3], Dmitry Slugin[2, 3], Dmitry Nikolaev[2, 3, 4], Vladimir V. Arlazarov[2, 3]

[1]Moscow Institute of Physics and Technology

[2]Institute for Systems Analysis, FRC CSC RAS

[3]Smart Engines Service LLC

[4]Institute for Information Transmission Problems RAS

Introduction

Introduction

Low-bit quantized neural networks (QNNs) allow us to:

- accelerate inference;
- decrease model size;
- perform real-time computation on low-powered devices;
- follow paradigm of edge intelligence.

Problems

Low-bit QNNs do not suit end devices of general architecture well:

- one can not directly use BLAS libraries for matrix multiplication;
- CPUs allow only 8-bit (or multiple) access and computations;
- no known efficient CPU implementations for lower than 8-bit quantization.

Our work

- We provide a novel algorithm for fast inference of 4-bit quantized neural network on CPU, based on a fast multiplication (used in convolution and fully-connected layers).
- We experimentally prove its efficiency for ARM architecture.

4-bit QNN

Quantization scheme

Linear quantization method:

$$\hat{w}_i = \left\lfloor \frac{w_i}{s} \right\rfloor - z$$
$$s = \frac{\max(\max_i w_i, 0) - \min(\min_i w_i, 0)}{2^p - 1}$$
$$z = \min(\min_i w_i, 0),$$

where $\hat{w}_i$ denotes quantized values, w_i are floating-point values, s is scale factor, z is a zero-point (offset), p is a number of bits used in quantized values

Quantized multiplication

Let's consider the quantized approximation of matrix multiplication $R = WX$:

$$r_{ij} = \sum_{k=1}^{D} w_{ik} x_{kj} \approx \sum_{k=1}^{D} s_w(\hat{w}_{ik} - z_w) s_x(\hat{x}_{kj} - z_x)$$

$$= s_w s_x \left(\sum_{k=1}^{D} \hat{w}_{ik} \hat{x}_{kj} - z_w \sum_{k=1}^{D} \hat{x}_{kj} - z_x \sum_{k=1}^{D} \hat{w}_{ik} + D z_x z_w \right)$$

where r_{ij} denotes values of R matrix, w_{ik} and x_{kj} are values of W and X matrices, $\hat{w}_{ik}$ and $\hat{w}_{ik}$ are their quantized approximations, s_w and s_x are scale factors, z_w and z_x are zero-points and D is a depth of multiplication.

Quantized multiplication micro-kernel

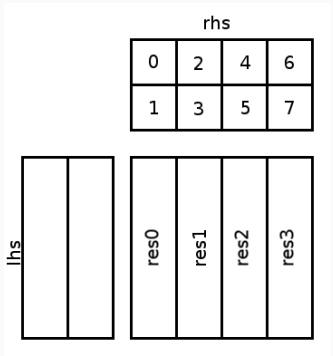
1	3	5	7	...
2	4	6	8	...
9	11	13	15	...
10	12	14	16	...
...	...	...	...	...

(a) RHS

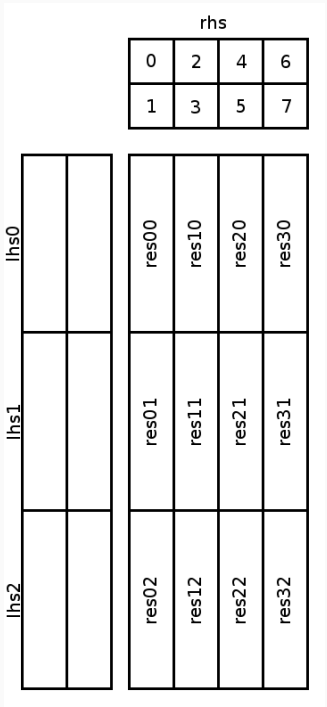
1	9	17	25	...
2	10	18	26	...
3	11	19	27	...
4	12	20	28	...
5	13	21	29	...
6	14	22	30	...
7	15	23	31	...
8	16	24	32	...

(b) LHS

Figure 1: The order or values of right (1a) and left (1b) matrices in temporal buffers



(a) The smaller kernel



(b) The bigger kernel

Figure 2: The kernel layout.

Quantized convolutional layer

- Perform Im2col transformation to turn convolution into matrix multiplication.
- Compute matrix multiplication (4-bit to 16-bit):

$$\hat{r}_{ij} = \sum_{k=1}^{D} \hat{w}_{ik}\hat{x}_{kj} - z_w \sum_{k=1}^{D} \hat{x}_{kj} - z_x \sum_{k=1}^{D} \hat{w}_{ik} + Dz_x z_w,$$

- Save floating-point scale factor:

$$s_r = s_w s_x$$

Quantized neural network

We can combine quantized (Q) and not quantized (F) layers in neural network:

- **F** $\rightarrow$ **F**. **Input**: floating-point activation. **Output**: floating-point activation.
- **F** $\rightarrow$ **Q**. **Input**: floating-point activation. It is quantized to 4-bit integer with scale factor s_x. **Output**: 16-bit integer activation, scale factor $s = s_x s_w$.
- **Q** $\rightarrow$ **Q**. **Input**: 16-bit integer activation, scale factor s^*. Activation is quantized to 4-bit integer with scale factor s_x. **Output**: 16-bit integer activation, scale factor $s = s_x s_w s^*$.
- **Q** $\rightarrow$ **F**. **Input**: 16-bit integer activation, scale factor s^*. Activation is converted back to floating-point by multiplication by s^*. **Output**: floating-point activation.

Experiments

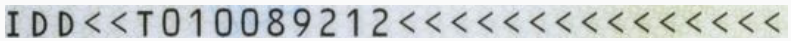
Matrix multiplication test

Height	Width	Depth	Floating point 32-bit time	Int 32-bit time	Unsigned int 8-bit time	Unsigned int 4-bit time	Unit
8	100	10	8.3	8.3	4.6	3.9	μs
		40	21	21	12	8.4	μs
		100	53	52	28	19	μs
	400	10	32	33	17	14	μs
		40	90	90	50	33	μs
		100	0.21	0.21	0.15	0.11	*ms*
	1600	10	0.13	0.14	0.070	0.058	*ms*
		40	0.35	0.36	0.25	0.19	*ms*
		100	2.2	2.4	0.78	0.55	*ms*
24	100	10	15	15	9.1	6.2	μs
		40	44	43	24	15	μs
		100	0.11	0.10	0.054	0.032	*ms*
	400	10	62	61	35	24	μs
		40	0.18	0.17	0.096	0.058	*ms*
		100	0.44	0.42	0.24	0.15	*ms*
	1600	10	0.24	0.24	0.14	0.097	*ms*
		40	0.72	0.71	0.43	0.28	*ms*
		100	3.2	3.1	1.2	0.76	*ms*

Measured on ODROID-XU4 single-board computer with Samsung Exynos5422 ARM processor

QNN test

- 36 MRZ character recognition from MIDV-500 dataset
- ODROID-XU4 single-board computer with Samsung Exynos5422 ARM processor



#	Layer type	Activation function	Parameters
1	Conv	ReLU	8 filters 5×5, stride 1×1
2	Conv	ReLU	8 filters 3×3, stride 1×1
3	Conv	ReLU	8 filters 3×3, stride 2×2
4	Conv	ReLU	16 filters 3×3, stride 1×1
5	Conv	ReLU	16 filters 3×3, stride 2×2
6	Conv	ReLU	24 filters 3×3, stride 1×1
7	FC	SoftMax	36 neurons

Model	**Accuracy synthetic**, %	**Accuracy MIDV**, %	**Convolution time**, *ms*	**Total time**, *ms*
CNN	99.8	95.6	0.99	1.22
QNN-8	99.7	95.4	0.55	0.74
QNN-4	99.2	95.0	0.45	0.63
QNN-32	-	-	1.16	1.47

Results

- Our 4-bit quantized matrix multiplication works about 3 times faster than floating-point multiplication from Eigen library and 1.5 times faster than 8-bit quantized multiplication similar to gemmlowp library
- Our 4-bit QNN works about 2 times faster than traditional CNN and 1.2 times faster than 8-bit QNN of the same architecture.
- The real-world problem of OCR recognition on the MIDV-500 dataset demonstrates 95.0% accuracy, while the floating-point network gives 95.6% accuracy.

Thank you
for your attention