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

Anton Trusov  
Moscow Institute of Physics and Technology  
Dolgoprudny, Russia  
Smart Engines Service LLC  
Moscow, Russia  
Email: trusov.av@smartengines.ru

Elena Limonova  
FRC CSC RAS  
Moscow, Russia  
Smart Engines Service LLC  
Moscow, Russia  
Email: limonova@smartengines.com

Dmitry Slugin  
FRC CSC RAS  
Moscow, Russia  
Smart Engines Service LLC  
Moscow, Russia  
Email: slugin@smartengines.com

Dmitry Nikolaev  
Institute for Information Transmission Problems RAS  
Moscow, Russia  
Smart Engines Service LLC  
Moscow, Russia  
Email: dimonstr@iitp.ru

Vladimir V. Arlazarov  
FRC CSC RAS  
Moscow, Russia  
Smart Engines Service LLC  
Moscow, Russia  
Email: vva@smartengines.com

**Abstract**—Quantized low-precision neural networks are very popular because they require less computational resources for inference and can provide high performance, which is vital for real-time and embedded recognition systems. However, their advantages are apparent for FPGA and ASIC devices, while general-purpose processor architectures are not always able to perform low-bit integer computations efficiently. The most frequently used low-precision neural network model for mobile central processors is an 8-bit quantized network. However, in a number of cases, it is possible to use fewer bits for weights and activations, and the only problem is the difficulty of efficient implementation. We introduce an efficient implementation of 4-bit matrix multiplication for quantized neural networks and perform time measurements on a mobile ARM processor. It shows 2.9 times speedup compared to standard floating-point multiplication and is 1.5 times faster than 8-bit quantized one. We also demonstrate a 4-bit quantized neural network for OCR recognition on the MIDV-500 dataset. 4-bit quantization gives 95.0% accuracy and 48% overall inference speedup, while an 8-bit quantized network gives 95.4% accuracy and 39% speedup. The results show that 4-bit quantization perfectly suits mobile devices, yielding good enough accuracy and low inference time.

**Index Terms**—Quantized neural networks, convolutional neural networks

**I. INTRODUCTION**

Nowadays, convolutional neural networks (CNNs) are widely used to solve pattern recognition, segmentation, object detection, and other problems [1], [2], [3], [4]. Often applications that solve these problems run on low-powered mobile devices like smartphones and IoT devices with limited computational and memory resources. It strongly restricts the use of deep convolutional neural networks, which generally require a vast amount of memory and perform billions of multiplications. Moreover, real-time recognition systems (for example, [5], [6], [7]) impose even more time constraints on CNNs.

There are several ways for CNNs to satisfy the efficiency constraints of mobile devices. One way is to simplify neural network: use pruning [8] to remove non-informative weights or knowledge distillation [9] to transfer knowledge to smaller network, train neural network with early exit [10]. Those methods modify or completely change the neural network, but we also can accelerate inference while preserving network architecture.

As was mentioned in [11] direct convolution algorithm is extremely ineffective, compared to GEMM-based algorithms, which convert the convolution problem into a GEMM (Matrix-Matrix Multiplication) problem. Therefore a significant part of computations during inference of CNNs is matrix multiplication. Of course, to perform fast matrix multiplication on specific devices, one can use optimized libraries and BLAS packages. They provide efficient linear algebra algorithms and are available for many computing architectures. They suit perfectly for a fast inference of standard floating-point neural networks (Eigen library [12], for example). The next speed up step is to approximate floating-point matrices with low-bit integer ones. This process is called quantization, and neural networks that benefit from such approximations are called quantized neural networks (QNNs). The critical difference between the multiplication of low-bit matrices and floating-point ones is that low-bit multiplication should be widening. Therefore, we can not use standard methods and need special libraries and frameworks. For example, Google’s gemmlowp library [13] provides low-precision matrix multiplication for 8-bit QNNs. They use 8-bit unsigned integers for inputs and accumulate the result into 32-bit integers. In Facebook’s QNNPACK [14], authors also use 8-bit unsigned integers for inputs but re-quantize 32-bit intermediates to get 8-bit outputs. Of course, there are many QNNs, which use 8-bit [15], 4-
II. QUANTIZED MATRIX MULTIPLICATION

A. Quantization scheme

In the paper we use the linear quantization method:

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

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 (in our case it is 4, in case of gemmlowp library – 8).

B. Quantized multiplication

Let us 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 \(\hat{x}_{kj}\) are their quantized approximations due to (1), \(s_w\) and \(s_x\) are scale factors, \(z_w\) and \(z_x\) are zero-points and \(D\) is a depth of multiplication (number of columns in left matrix (\(W\)) and rows in right matrix (\(X\))). The first term in (2) is a result of matrix multiplication of quantized matrices, the second is a sum over rows of quantized right matrix multiplied by zero-point of left matrix and can be effectively computed while unrolling activation matrix, the third term is the sum over columns of the left matrix multiplied by zero-point of the right matrix, and the last term is simply a constant. In QNN \(W\) is a matrix of weights, so its quantized approximation, scale factor, zero point and sums over columns can be computed only once to speed up inference.

For 8-bit quantization \(\hat{w}_{ik}\) and \(\hat{x}_{kj}\) are unsigned 8-bit integers and thus have values in range \([0, \ldots, 255]\), in process of multiplication they are zero-expanded to 16-bit integers and their sums (first term of (2)) are stored in 32-bit integer accumulators. For 4-bit quantization (in range \([0, \ldots, 15]\)) zero-expanded to 8-bit with 16-bit accumulators.

C. Quantized multiplication kernel

As mentioned above, to speed up neural network computations we need effective matrix multiplication. To achieve it we take advantage of SIMD-extension of ARM processor, which allows us to compute the same instruction on several values at once. ARM SIMD stores values in 128 or 64-bit registers. According to ARM architecture reference manual [25] on ARM-v7 there are 16 128-bit registers which can be viewed as 32 64-bit registers. Knowing that we construct a multiplication kernel (a function that simultaneously computes several values of result matrix). It takes blocks from left and right matrices and computes a block of the result values which are then added to the current value of accumulators. In Fig.2 kernel layout is shown. We use 2 kernels: bigger one for a major part of a matrix and smaller for a “tail” of a matrix that can not be processed by a big kernel. The rest of the matrix is processed without SIMD acceleration. We use 16-bit accumulators, so our kernel sizes are 24 × 4 and 8 × 4 for bigger and smaller kernels respectively.

For kernel to work efficiently it needs to minimize the number of loads from memory, so before multiplication itself values from the left and right matrices packed in temporal buffers in a specific order. In our implementation values of the right matrix are stored in temporal buffer \(\text{RHS}\) so the first 2 values from \(1^{st}\), \(2^{nd}\), \(3^{rd}\) and \(4^{th}\) column are followed by the second 2 values from the same columns, than \(3^{rd}\), etc. If the number of columns is not divided by 4, or the number of rows
is not divided by 2 the rest of the matrix is not packed into a temporal buffer and is processed without SIMD acceleration. The order of values in the temporal buffer of left matrix LHS depends on kernel size: for smaller kernel first 8 values from the 1st column are followed by the first 8 values from 2nd, 3rd, etc. For bigger kernel its values 1-8 from 1st column 1-8 from 2nd, 9-16 from 1st, 9-16 from 2nd, 17-24 from 1st, 17-24 from 2nd than go 3rd and 4th columns, etc. Again if the number of rows is not divided by the number of rows in kernel or number of columns is not divided by 2 is not packed into the temporal buffer and is processed without SIMD acceleration. Fig. 1 illustrates how values from right and left matrices are stored in RHS and LHS buffers, in case of smaller kernel.

![Matrix Multiplication](image)

Fig. 1. Order or values of right (1a) and left (1b) matrices in temporal buffers

One step of matrix multiplication is shown in Fig. 2 registers are denoted as res for 16-bit unsigned integer accumulators, rhs and lhs for blocks of 8 4-bit integers from right and left matrices, zero-extended to 8 bits. Registers rhs and lhs sequentially load values from buffers RHS and LHS, numbers denote order of elements in original matrix. This way we need only one memory call to load 64-bit rhs register and another one (or three for bigger kernel) to load 128-bit lhs registers. When all registers are loaded, each value of rhs register is duplicated and stored in another 64-bit register rhs registers. When all registers are loaded, each value of rhs register is duplicated and stored in another 64-bit register usingVDUP_LANE. The result is multiplied by corresponding half of lhs register and added to res register with a single VMAL instruction. After the summation over depth dimension is finished res registers are stored back into a memory.

For smaller kernel pseudocode of our method is given below:

```c
pack right matrix into RHS
pack left matrix into LHS
for j in 0, ..., cols / nr
    {res0 ... res3} ← next result block
for i in 0, ..., rows / mr
```

for k in 0, ..., depth / 2

```
lhs ← next block from LHS
rhs ← next block from RHS
VMAL(dst0, LOW(lhs), rhs[0]);
VMAL(dst0, HIGH(lhs), rhs[1]);
VMAL(dst3, LOW(lhs), rhs[6]);
VMAL(dst3, HIGH(lhs), rhs[7]);
result block ← {res0 ... res3}
```

Here notation LOW and HIGH stands for getting lower or higher half of a register (which does not generate any instructions on assembler level) and rhs[n] getting and duplicating n’th bit of rhs usingVDUP_LANE instruction.

cols, rows and depth represent dimensions on matrices, nr, mr are height and width of a kernel (in our case they are 8 and 4 and 24 and 4 for smaller and bigger kernels respectively). Algorithm for a bigger kernel is similar to this one but uses more registers.

The bigger kernel uses 15 of 16 128-bit ARM vector registers and one 64-bit, in this way we minimize calls to memory and a total number of instructions: in inner loop we upload only 3-128 bit registers and one 64 bit and use 24 VMAL and VDUP_LANE instructions to compute 384 multiplications and additions.

III. QNN MODEL

Based on proposed quantization scheme and algorithm of matrix multiplication, described above we propose QNN model.

A. Quantized convolution layer

Let us consider

```
\hat{r}_{ij} = \sum_{k=1}^{D} \hat{w}_{ik}\hat{x}_{kj} = z_w \sum_{k=1}^{D} \hat{x}_{kj} = z_w \sum_{k=1}^{D} \hat{w}_{ik} + D z_w z_s. \tag{3}
```

where definition are the same as in [2]. Then

```
r_{ij} = s_w s_x \hat{r}_{ij}. \tag{4}
```

So we can treat \( \hat{r}_{ij} \) as quantized matrix with the zero-point \( z_r = 0 \) and the scale factor \( s_r = s_w s_x \). That is why we do not convert the quantized result back to 32-bit floating-point values but preserve it as is in 16-bit integer form for 4-bit quantization and 32-bit integer for 8-bit quantization), then pass it through activation function to the next layer where it is quantized to 4-bit (8-bit) once again. If that quantization results in \( s^*_r \) scale factor, than total scale factor will be \( s = s^*_r s_r \).

As we can see all terms in [3] are signed integers. In our case they are signed 16-bit integers, which means that first term should be less than \( 2^{15} - 1 \) to avoid integer overflow. Due to [1] \( \hat{w}_{ik} \) and \( \hat{x}_{kj} \) are not greater than 15, so their product is not greater than 225. That gives us theoretical constraint on depth of matrix multiplication: \( D \leq \left\lfloor \frac{2^{15} - 1}{225} \right\rfloor = 145. \)

In case of convolution \( D = k_h k_w c \), where \( k_h \) and \( k_w \) are convolution kernel height and width, \( c \) is a number of channels
in input of current layer. So for $q$-bit quantization with $p$-bit accumulators we obtain a constraint on number of input channels:

$$c \leq \left\lfloor \frac{2^{p-1} - 1}{(2^q - 1)^2} \right\rfloor \frac{1}{k_h k_w}. \quad (5)$$

This gives us 145 channels for $1 \times 1$ convolution kernel, 16 channels for $3 \times 3$ kernel, and only 5 channels for $5 \times 5$ convolutions. To weaken this constraint one can compute the first term in unsigned 16-bit integers than zero expand it up to 32-bits and compute (3) in 32-bit signed integers. This way first term should be less than $2^{16} - 1$, and so $D \leq \left\lfloor \frac{2^{16} - 1}{225} \right\rfloor = 291$.

However, the situation, when all quantized values in one row of unrolled filter matrix equal to 15, is not likely to happen in practice, so one can try to use more channels.

### B. Quantized network

Although quantized integer computations are faster then floating-point, mobile CPUs do not constrain us to use only integer operations the way it is done in end-to-end QNNs [26]. To achieve better inference results, we can preserve some layers not quantized like in [27], where authors do not quantize the first and last layers of a network. In our experiment, we do not quantize only the last one. So our QNN model consists of quantized layers (Q) and not quantized layers (F) with floating-point weights and activations.

For each quantized layer, we convert weights matrix (un-rolled for convolution layers) according to (1) and save scale factor $s_w$, zero point $z_w$ and sums of rows of this matrix $\sum_{k=1}^{D} w_{ik}$.

During network inference, the quantized layer converts its input according to (1) and returns a 16-bit integer activation vector and a scale factor, that can be used to convert the data back to floating-point.

To be more specific let us consider inputs (with conversions if needed) and outputs of F and Q layers, depending on the previous layer (input of a network is treated as F).

- **F → F.** Input: floating-point activation. Output: floating-point activation.
- **F → 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 → 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 → F.** Input: 16-bit integer activation, scale factor $s^*$. Activation is converted back to floating-point by multiplication by $s^*$. Output: floating-point activation.

### C. Training and fine-tuning

To train our 4-bit QNN we first train CNN with only floating-point weights. We can not simply quantize all its layers and apply such a network, because quantization error for 4-bit approximation is significant and rises dramatically with increasing of the layer number. To deal with this problem we apply layer-by-layer fine-tuning as described in [28]: quantize the first layer, freeze its weights and retrain the rest of the network, then quantize and freeze the second layer and so one. The fully-connected layer remains not quantized.

### IV. Experiments

#### A. Matrix multiplication

First we compare the time required to compute matrix multiplication of 32-bit floating-point, 32-bit integers, 8-bit integers and 4-bit unsigned integers values. For 4-bit integers we use the algorithm described in section 1 for 8-bit we use an algorithm similar to one from gemmlowp, and for floating-point and 32-bit integers an algorithm from Eigen library. We randomly initialize matrices, compute matrix multiplication
several times, take average time and repeat the experiment with different initialization until a result is stable enough. We take different sizes of matrices that are close to those which appear in compact lightweight CNNs.

All tests are performed in a single thread of ODROID-XU4 device with Samsung Exynos5422 ARM processor.

Results are provided in a Table [I]. We can see that multiplication of 32-bit integers and floating point values takes almost the same time, multiplication of 4-bit unsigned integers works approximately 2.4 times faster than that of 32-bit floating-point and 1.4 times faster compared to 8-bit unsigned integers for the smaller kernel (upper half of the Table). For the bigger kernel it is 2.9 and 1.5 times respectively (lower half of the Table).

### TABLE I

| Model |
|-------|
| CNN   |
| QNN-8 |
| QNN-4 |
| QNN-32|

| Parameters |
|------------|
| 8-bit (QNN-8) and 4-bit (QNN-4) quantization. Than we compared their recognition accuracy on synthetic validation data from the training dataset and real-world data from MIDV-500 dataset, and also measured image recognition time and per computation of all convolutions. Moreover we measured time of naive implementation of quantized network, that is based on multiplication of 32-bit integer matrices. This experiment was performed on the same ODROID-XU4 device. Results are presented in a Table [III].

![MRZ from German driving licence in MIDV-500](image1)

![Synthetic images](image2)

![Real-world images](image3)

CNN architecture is provided in a Table [II] where Conv denotes a convolutional layer without bias or padding and FC a fully-connected layer. This network used fixed-size $25 \times 33$ grayscale images as an input. It contained 10892 trainable parameters. While converting to QNN all convolutional layers (8264 parameters) were quantized.

### TABLE II

| 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 | -                     | -                | -                    | -             |

According to the results of experiment naive implementation of quantized neural network is even less efficient than floating point CNN, because additional time is required for quantization, while matrix multiplication gains no additional speedup. We can also see that both QNN-8 and proposed QNN-4 maintain a good quality of recognition and speed
up computation by 39% and 48% respectively (QNN-4 is 1.17 times faster than QNN-8). This is lower than speedup in computation of convolutions (44% and 54%) because additional time is required for quantization of input of each layer.

V. CONCLUSION

In this paper, we provide an implementation of an efficient matrix multiplication algorithm for 4-bit quantized matrices for mobile devices with ARM-based architecture. We experimentally prove its efficiency on ODROID-XU4: it works about 2.9 times faster than floating-point multiplication from Eigen library and 1.5 times faster than 8-bit quantized multiplication from gemmlowp library.

Then we propose the 4-bit QNN model that benefits greatly from our algorithm of matrix multiplication. We demonstrate that this model is both viable and efficient. On the real-world problem of OCR recognition on the MIDV-500 dataset, it demonstrates 95.0% accuracy, while the floating-point network gives 95.6% accuracy. Our network works 1.93 times faster than traditional CNN and 1.17 times faster than 8-bit QNN of the same architecture.

Our work shows that the efficient implementation of highly quantized neural networks is possible for CPUs. It allows creating applications for complex real-world tasks and satisfy memory and computational constraints of modern smartphones and IoT devices. Thus, this greatly expands the applicability of modern recognition methods and brings many quantization ideas closer to practical use in edge computing.

ACKNOWLEDGMENTS

This work is partially supported by Russian Foundation for Basic Research (projects 17-29-03240, 18-07-01384).

REFERENCES

[1] P. V. Bezmaternykh, D. A. Ilin, and D. P. Nikolaev, “U-net-bn: hacking the document image binarization contest,” Computer optics, vol. 43, no. 5, pp. 825–832, 2019, doi: 10.18287/2412-6179-2019-43-5-825-832.

[2] R. Laroca, E. Severo, L. A. Zanoloresni, L. S. Oliveira, G. R. Gonçalves, W. R. Schwartz, and D. Menotti, “A robust real-time automatic license plate recognition based on the yolo detector,” in 2018 International Joint Conference on Neural Networks (IJCNN). IEEE, 2018, pp. 1–10.

[3] R. Lienhart and A. Wernick, “Localizing and segmenting text in images and videos,” IEEE Transactions on circuits and systems for video technology, vol. 12, no. 4, pp. 256–268, 2002.

[4] A. Robles-Kelly and R. Wei, “A convolutional neural network for pixelwise illuminant recovery in colour and spectral images,” in 2018 24th International Conference on Pattern Recognition (ICPR). IEEE, 2018, pp. 109–114.

[5] S. A. Usilin, P. V. Bezmaternykh, and V. V. Arlazarov, “Fast approach for qr code localization on images using viola-jones method,” in International Conference on Machine Vision (ICMV), vol. 11433, no. 11433.3G. SPIE. Jan. 2020, pp. 1–9, doi: 10.1117/12.2553986.

[6] K. Bulatov, N. Razumnyi, and V. V. Arlazarov, “On optimal stopping strategies for text recognition in a video stream as an application of a monotone sequential decision model,” International Journal on Document Analysis and Recognition (IJDAR), vol. 22, no. 3, pp. 303–314, 2019, doi: 10.1007/s10032-019-00333-0.

[7] M. A. Povolotskiy, D. V. Tropin, T. S. Chernov, and B. I. Savelev, “Dynamic programming approach to textual structured objects segmentation in images,” TIFS, vol. 69, no. 3, pp. 66–78, 2019, doi: 10.14357/20718632190306.

[8] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” in International Conference on Learning Representations (ICLR), 2016.

[9] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” arXiv preprint arXiv:1503.02531, 2015.

[10] L. Teplyakov, S. Gladlain, and E. Shvets, “Complexity-aware loss function for fast neural networks with early exits,” in International Conference on Machine Vision (ICMV), vol. 11433, no. 1143331. SPIE. Jan. 2020, doi: 10.1117/12.2557077.

[11] M. Dukhan, “The indirect convolution algorithm,” arXiv preprint arXiv:1907.02139, 2019.

[12] G. Guennebaud, B. Jacob et al., “Eigen: a c++ linear algebra library,” 2010. [Online]. Available: http://eigen.tuxfamily.org

[13] B. Jacob et al., “gemmlowp: a small self-contained low-precision gemm library,” 2017. [Online]. Available: https://github.com/google/gemmlowp

[14] M. Dukhan, Y. Wu, and H. Lu, “Qnnpack: open source library for optimized mobile deep learning,” 2018. [Online]. Available: https://code.fb.com/ml-applications/qnnpack

[15] P. Gysel, M. Møltømæddi, and S. Ghiasi, “Hardware-oriented approximation of convolutional neural networks,” arXiv preprint arXiv:1604.03168, 2016.

[16] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid, “Towards effective low-bitwidth convolutional neural networks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 7920–7928.

[17] H. Alendar, V. Leroy, A. Prost-Boucle, and F. Pétrot, “Ternary neural networks for resource-efficient ai applications,” in 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, 2017, pp. 2547–2554.

[18] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid, “Structured binary neural networks for accurate image classification and semantic segmentation,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 413–422.

[19] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua, “Quantization networks,” in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[20] S. Yu, Z. Li, P.-Y. Chen, H. Wu, B. Gao, D. Wang, W. Hu, and H. Qian, “Binary neural network with 16 mb ram macro chip for classification and online training,” in 2016 IEEE International Electron Devices Meeting (IEDM). IEEE, 2016, pp. 16–2.

[21] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, “Fp-bnn: Binarized neural network on fpga,” Neurocomputing, vol. 275, pp. 1072–1086, 2018.

[22] E. E. Limonova, M. I.-O. Neyman-Zade, and V. L. Arlazarov, “Special aspects of matrix operation implementations for low-precision neural network model on the elbrus platform,” Bulletin of the South Ural State University. Ser. Mathematical Modelling, Programming & Computer Software, vol. 13, no. 1, pp. 118–128, 2020, doi: 10.14529/mmp200109.

[23] B. Rudner, Y. Nahshan, and D. Soudry, “Post training 4-bit quantization of convolutional networks for rapid deployment,” in Advances in Neural Information Processing Systems, 2019, pp. 7950–7958.

[24] V. V. Arlazarov, K. Bulatov, T. Chernov, and V. L. Arlazarov, “Midv-500: A dataset for identity document analysis and recognition on mobile devices in video stream,” Computer images, vol. 43, no. 5, pp. 818–824, 2019, doi: 1827/2412-6179-2019-43-5-818-824.

[25] ARM architecture reference manual ARMv7-A and ARMv7-R edition, 2014. [Online]. Available: https://static.docs.arm.com/ddi0406c/c/ARM Architecture Reference Manual.pdf

[26] R. Li, Y. Wang, F. Liang, H. Qun, J. Yan, and R. Fan, “Fully quantized network for object detection,” in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

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

[28] D. A. Ilin, E. E. Limonova, V. V. Arlazarov, and D. P. Nikolaev, “Fast integer approximations in convolutional neural networks using layer-by-layer training,” in International Conference on Machine Vision (ICMV), A. Verkas, Ed., vol. 10341, no. 103410Q. Bellingham, Washington 98227-0010 USA: SPIE. July 2017, pp. 1–5, doi: 10.1117/12.2268722.

[29] Y. S. Chernyshova, A. V. Sheshkus, and V. V. Arlazarov, “Two-step cnn framework for text line recognition in camera-captured images,” IEEE Access, vol. 8, pp. 32587–32600, 2020, doi: 10.1109/ACCESS.2020.2974051.