Deep neural networks performance optimization in image recognition

A.G. Rassadin[a], A.V. Savchenko[a]

[a] National Research University Higher School Of Economics, 603155, 25/12 Bolshaya Pecherskaya Street, Nizhny Novgorod, Russia

Abstract

In this paper, we consider the problem of insufficient runtime and memory-space complexities of contemporary deep convolutional neural networks in the problem of image recognition. A survey of recent compression methods and efficient neural networks architectures is provided. The experimental study is focused on the visual emotion recognition problem. We compare the computational speed and memory consumption during the training and the inference stages of such methods as the weights matrix decomposition, binarization and hashing in the visual emotion recognition problem. It is experimentally shown that the most efficient recognition is achieved with the full network binarization and matrices decomposition.

Keywords: deep neural networks; image recognition; visual emotion recognition; deep compression; binarized neural networks; tensor train; tensor decomposition; XNOR-Net

1. Introduction

Deep neural networks have recently become everyday tool for such tasks as image recognition [1], [2], speech recognition [1], signal processing and text analysis [3], [4]. The neural network study, started in the middle of the last century, made an incredible leap in the beginning of the XXI century. The success of deep convolutional neural networks (CNN) has started from the paper of Alex Krizhevsky et al. [5], which introduced a novel approach known as the AlexNet architecture, for solving visual recognition tasks with the very-large ImageNET dataset. Contemporary CNN architectures [6], [7], [8] are much more accurate when compared with original AlexNet. However, their runtime complexity becomes insufficient for application in several practical tasks, especially with implementation on mobile platforms. Hence, the performance optimization of deep CNN is now considered as one of the most important studies in deep learning. For instance, the work on deep compression [9] received the Best Paper Award in very prestigious International Conference on Learning Representation (ICLR'16).

The CNN optimization starts even before the training or testing the model. The proper choice of hardware can drastically improve the training performance. The leader in the deep learning hardware development, NVIDIA, nowadays release the wide range of GPUs, different by the form-factor (from tiny development boards to server-side Tesla) and the purpose. However, NVIDIA is not alone on this field. Recent year Google has lifted the veil about its Tensor Processing Unit (TPU) – a special purpose chip, highly optimized for the tensor (matrix) operations, which is the core routine in neural networks. Such acquisitions of Intel as Nervana and Movidius are also not accidental. Nervana introduced not only deeply optimized (and, probably, most efficient) framework but also a special-purpose hardware platform, which includes not only optimized chip but also an optimized memory and data channels. Movidius, in its turn, provides a small device capable for inferencing deep neural networks in real time. Qualcomm announced its SnapDragon 820 – probably the most optimized mobile chip: more than 5 times faster in comparison with nearest generations [10].

The next step in the performance optimization pipeline is the choice of an appropriate driver. The last trend in this field is the lower-precision operations. Using lower precision, e.g., FP16 (2 bytes per float weight) or INT8 (one byte per integer) at the training stage can significantly limit the generalization ability of the CNN model. However, using such precision at the inference stage saves both memory space and processing time with permissible losses in accuracy. It is important that nowadays there are several frameworks, which contain very efficient implementations of the neural network training procedures, e.g. convolutional operation, back propagation or stochastic gradient descent. However, none of frameworks is absolutely optimal but every one tends to be. Moreover, most of them are general-purpose: they cannot be used on usual mobile device, they inherits the limitations of the high-level programming languages in which they are written etc. The progress both in mobile industry and in deep learning area marked the emergence of special-purpose frameworks, usually written in C/C++ capable or even intended to use on mobile platforms.

However, the most remarkable research direction in this field is the optimization of algorithms and neural network architectures. To compare the various methods, we will use the following goals:

- Recognition accuracy;
- GPU total training time;
- GPU average inference time;
- Space (memory) complexity of the training and inference procedures.

In this paper we tried to select the most promising and effective optimization possibilities introduced in the papers from last year. We compared author implementations in such acute task as the task of emotion recognition from facial expressions. The rest of the paper is organized as follows. In Section 2, we provide a survey of recent literature devoted to the performance

improvements of deep neural networks. Section 3 contains an experimental study of performance optimization methods in visual emotion recognition. Finally, concluding comments are given in Section 4.

2. CNN performance optimization methods overview

There are several types of classification of deep neural networks performance optimization techniques. The methods can differ:

1) by the *accuracy loss*: lossless, optimization with accuracy loss, optimization-accuracy trade-off);
2) by the *applicability level*: architectural, operational (by model / framework modification), computational (exactly while training or inference), hardware;
3) by the *limitations*: architecture-dependent, and architecture-independent;
4) by the *implementation*: runtime implementation, two-step (training -> optimization), sequential (training -> optimization -> re-training);
5) by the *optimization building block*: all blocks, convolutional layers, fully connected layers.

Perhaps the most fundamental approach and in the same time one of the most efficient and universal is the *pruning* [9], [11]. It is known that in huge amount of weights (connections) in the trained network even with the superior generalization ability the contribution to the every neuron (connection) is different. One can alternately remove connections with low weight and, in turn, minimal impact to the prediction results, and fine-tuning after every pruning, until achieving the allowable loss in the accuracy. It is important to note that the pruning can be applied to any neural network architecture and both before and after every another performance optimization technique being the most general approach.

The *distilling the knowledge* technique has been initially suggested by Hinton et al. in 2014 [12]. The idea of this approach is to train a cumbersome neural model or an ensemble of models with the superior generalization ability ("teacher") and then transfer its predictive power to another, thinner but usually deeper model ("student") by training the latter to predict the same labels as the original one. The main disadvantage of this approach is that the time cost for the optimization is on the same level with the training from scratch. Another obvious drawback is that the "student" model is not protected from the mistakes of the "teacher" model. In fact, the resulted model is even weaker, because it can tends to unexpected behavior in predictions. Moreover, it is not an absolute performance optimization but rather relative to the original teacher network. This technique have not become widely used. We can only mention the work of Romero et al. [13] in which some limitations of the initial approach were overcome.

The idea of *weights hashing (quantization)* [14] is based on that close values of the CNN weights may be considered equal (with some precision), which makes it possible to share the same memory unit, and, in turn, drastically reduce the memory costs. This approach continues to exploit the idea of lower precision computations. It is exactly the key part of the famous Deep Compression method [9], in which a very effective pipeline to optimize the performance and the size of the network is described. Unfortunately, it is hard to distill from the paper the real influence of quantization to overall compression quality, because it also includes pruning, which can be the most important factor, which allowed the authors to achieve their outstanding results in compression of AlexNet architecture.

The *tensor decomposition* exploits a very intuitive idea: since that deep neural network contains high order matrices (tensor) of weights in each layer, they can be decomposed to the sequence of lower order matrices and vectors. The most popular techniques nowadays are CP (CANDECOMP/PARAFAC or Canonical Polyadic Decomposition) [15], Tucker [16] and the most recent one – Tensor Train [17], [18]. Such approaches allows to explicitly choose between the amount of memory consumption and the accuracy loss by setting the rank of the decomposition.

The group of *binarization* methods is based on the observation that it is to enough for weights to be stored in FP32 and continues the trend of lower precision computations. These techniques differs from the hashing (quantization) by going deeper into performance optimization problem caring out not only about the storage and native (because of lower precision) computational efficient. Original idea is followed by the observation that only 1 bit ({0, 1} or {-1, +1}) is enough for weights values. Thus, it is possible to store only the sign of values instead of usiage of full FP32 precision. Hence, the arithmetic operations can be replaced to much faster logical operations. However, the binarization of the network right after traditional training leads to the complete loss of the predictive power of the network. It is important to apply binarization iteratively, epoch-by-epoch. The procedure of binary weights backpropogation was suggested in [19] to implement this approach. Initial idea to binarize only weights outgrew to binarizing the whole network including the input vector. Such an approach [20] leads to the complete replacement of the arithmetic operations by XNOR. It has recently been shown [21] that the applied binary mapping does not matter, hence, the sign of the variable is usually the simplest and fastest techinque.

There exist other methods, which optimize a fixed architecture or even already learned model by using several *architectural tricks*. Among these methods, it is important to mention the SqueezeNet [22] and the Tiny Darknet [23], which achieve the accuracy compared to the AlexNet [5], but are much smaller and even faster. The PVANet [24] is the architecture for the object

detection task with minimal computational cost obtained by adapting and combining recent technical innovations. The BranchyNet [25] introduces early exits (classifiers) along the architecture by which the researcher can explicitly balance between the inference speed and the accuracy.

To summarize our brief survey, we present in Table 1 the potential of the most important discussed methods to achieve four optimization goals, which we mentioned in introduction. As we can see here, despite of the large number of reviewed papers, there are no "silver-bullet" methods, which guarantee the training speedup or memory consumption while training. Most of these techniques dedicated on reduction the memory consumption while inference. The pruning can be very common approach, e.g. integrated in every modern framework but unfortunately, it is still not common.

Table 1. Reported results of deep neural networks performance optimization

	Memory reduction while training	Memory reduction while inference	Inference speedup	Baseline model	Dataset
Deep Compression [9]	no	49	unknown	VGG-16	ImageNet
FitNets [13]	no	36	13.36	Maxout	CIFAR-10
HashedNets [14]	no	64	unknown	same-size	MNIST
CP-Decomposition [15]	no	12	4.5	AlexNet	ImageNet
TensorNet [18]	unknown	80	unknown	simple	CIFAR-10
BinaryNet [19]	~32 (theoretical)	~32 (theoretical)	3.4~23	Maxout	CIFAR-10
Binary-Weight-Network and XNOR-Net [20]	no	67	58 (CPU)	ResNet-18	ImageNet
SqueezeNet [22]	unknown	50	1.	AlexNet	ImageNet
Tiny Darknet [23]	unknown	60	2.9		
BranchyNet [25]	no	no	1.9	ResNet-110	CIFAR-10

3. Experimental Results

In this section we will discuss the computational experiments dedicated on performance optimization power of the following techniques: HashedNets, BWN, XNOR-Net and CP-decomposition. We have used the author's code that guarantees us the exact implementation and results reproducibility. These methods were evaluated on the real task of emotion recognition from frontal and 45° rotated facial images detected in the widely used Radboud Faces Database (RaFD) [26]. The neural networks were trained from scratch using identical training samples and learning procedures. Inspired by the well-known facial expression recognition CNN [27], we choose VGG-S architecture for the HashedNet, BWN and XNOR-Net as a baseline. All the neural network models are freely available at (https://mega.nz/#F!2FVz1SAT!dRdzpfc7UEwHC-jI9jEkIQ). In fact, in [27] authors trained an ensemble of neural networks using RGB and different kind LBP (Local Binary Patterns) visual features, but we decided to use a single RGB input for simplicity. All the experiments were done on the same machin**e** using Tesla M2090 with 6 GB of memory under Ubuntu 14.04 with CUDA Toolkit 8.0.

The testing of the CP-decomposition [15] was performed using the SqueezeNet-1.1 [22] architecture instead of VGG-S. Indeed, convolutional layers take a small portion of weights in such architectures with massive fully-connected layers as VGG-S. Hence, the CP-decomposition is appropriate only for such convolutional architectures without fully connected layers as SqueezeNet. The baseline model was trained with Caffe framework using stochastic gradient descent (SGD) with momentum 0.9, fixed learning rate 0.001 and 32 images in a mini-batch. To compare the neural networks computing efficiency we measured: 1) epoch time for single forward pass and subsequent gradient update on GPU for mini-batch in one random sample, averaged over 1000 runs; and 2) GPU inference time for single random sample, averaged over 1000 runs. We additionally estimated the accuracy loss and the reduction in number of weights.

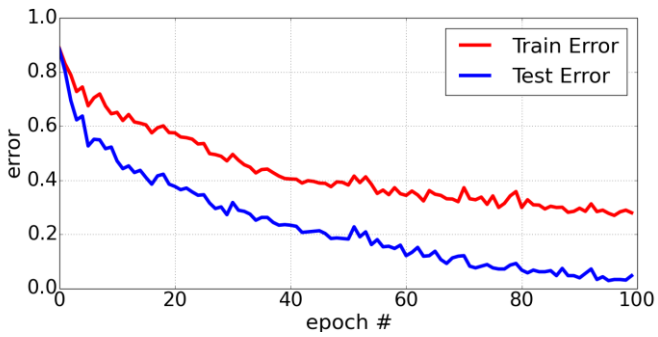

Fig. 1. The training/testing error rates for the baseline VGG-S neural network model [27].

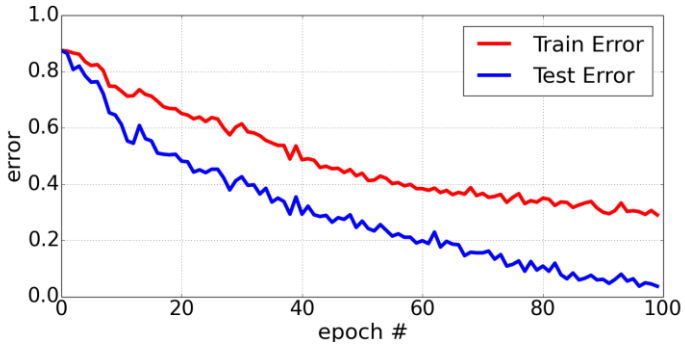

Fig. 2. The training/testing error rates for the HashedNet.

Original version of the neural network architecture SqueezeNet-1.1 has relatively small number of filters in every convolutional layer. Hence, the decomposition of every layer to a lower rank, e.g., 16, tends to the complete loss in accuracy. However, when only two last convolutional layers were decomposed with the rank equal to 192, the number of parameters reduced at 23.5% with 1.65% of the accuracy loss (from 89.14% to 87.5%). Unfortunately, the inference in the resulted network became even 1.5 time slower. It seems that replacement of the single large convolutional layer to four sequentially connected small layers causes higher computing complexity in parallel environment.

The HashedNets, BWN and XNOR-Net have been trained using RGB images from the same distinct and balanced training / testing subsets of the RaFD dataset using the Torch framework. We used SGD with momentum equal to 0.9, learning rate fixed at 0.001 and mini-batch of 20 sample. The common baseline model [27] was trained with the same settings. This baseline CNN converged to accuracy 97.13% after 100 epochs (Fig. 1). Here and bellow, the testing error rate is in practically all cases less than the training error rate. Though such behavior seems to be not obvious, it is reasonable due to the usage of dropout regularization layer, which is activated while training phase and deactivated when evaluating on the validation set. Moreover, the training error rate is computed as the mean error rate for all mini-batches in one epoch. On the contrary, the testing error rate is computed only after each epoch with more optimal weights, which were learned during this epoch. Let us compare this result with the performance optimization techniques.

We used default compression settings, provided by the authors of the HashedNet technique [14]: compression rate is equal to 0.125 and the bias hashing was set. The latter option leads to the 81.64% reduction in the weights count. Despite this reduction, the training process (Fig. 2) is practically identical to the baseline (Fig. 1): the network converged to 96.31% accuracy after 100 epochs, which is 0.8% lower when compared to the baseline CNN (Fig. 1). However, the training procedure is 6.7 times slower when compared to the baseline. The inference procedure of the HashedNet is also 4.7 times slower. We believe that such slowdown can be drastically reduced by replacing the current third-party implementation of hashing, which does not allow us saving trained model and measure memory consumption while inference accurately.

In next experiments, the BWN and the XNOR-Net are implemented according to the paper [20]. Every *conv-bn-activation* block excluding the first was replaced with the *bn-activation-conv* block. The dependences of the testing and training error rates of the BWN on the epoch number are shown in Fig. 3. Here the BWN converged to the very low error rate 1.43% after forty epochs. After that time both training and testing error rate started to grow. We cannot precisely explain this behavior but probably, advance learning rate policy can suppress this binarization shortcoming. In fact, all our experiments demonstrate that BWN model always converges 2-4 times faster, when compared to the baseline CNN, which can be explained by very strong regularization effect introduced by the BWN architecture. We have not observed the inference memory reduction or inference speedup. The number of parameters also remains unchanged.

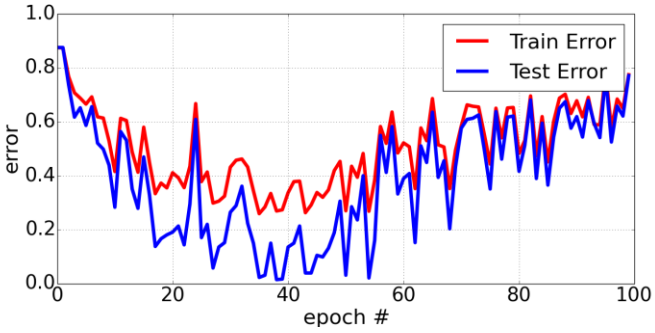

Fig. 3. The training/testing error rates for the BWN.

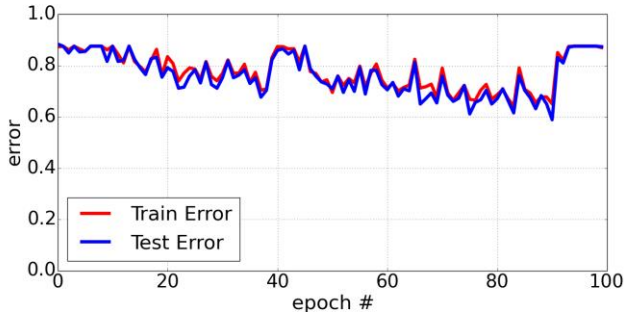

Fig. 4. The training/testing error rates for the XNOR-Net.

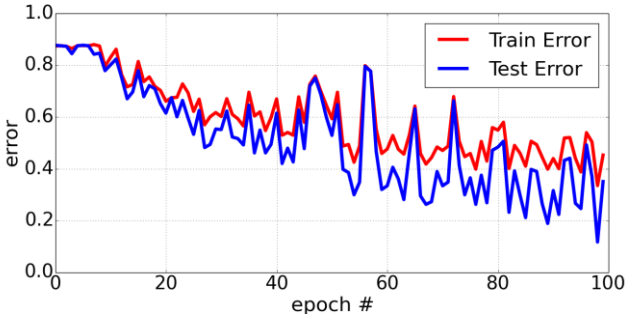

Fig. 5. The training/testing error rates for the modified XNOR-Net.

The XNOR-Net [20] was not converged in our experiments (Fig. 4). The lowest error rate for the testing set was equal to 41.19%. The only advantage of this method is the slight (2.4%) reduction in the memory consumption while inference, which is the benefit of the modified binarized activation layer. It is interesting to note that using only binarized activation layer without weights binarization leads to the same parameters reduction and even slight epoch time speedup. What is more important, such modification is capable to converge much closer to the accuracy of the baseline model – 88.32% – within the same learning procedure (Fig. 5).

4. Conclusion

In this paper, we have reviewed several modern approaches to optimize performance of deep neural networks. We emphasized the obvious trends in this field, namely, efficient tensor decomposition techniques, lower precision calculations and more accurate network binarization. We performed an evaluation of the state-of-the-art techniques in application to visual emotion recognition based on facial expressions. Our experimental results are briefly summarized in Table 2. The best value in each column is marked by bold.

The main direction for further research of the will be concentrated on combining of the most successful reviewed techniques. It is important to test these methods with other datasets, e.g., in EmotiW challenge. Another research direction is the implementation of the complete pipeline to *video*-based emotion recognition. Finally, it is necessary to examine the possibility to implement discussed methods in image recognition on mobile platforms.

Table 2. Methods evaluation results

	Training time per one epoch, ms	Inference time, ms	Model size, MB	Accuracy, %
VGG-S (baseline)	43.7	33.4	372.2	97.13
SqueezeNet-1.1 (baseline)	**22.94**	**4.94**	2.8	89.14
SqueezeNet-1.1, CP-Decomposition	**22.94**	7.74	**2.1**	87.5
HashedNets	294.8	158.2	68.3	96.31
Binary-Weight-Network	83.8	33.5	11.6	**98.57**
XNOR-Net	84.3	34.2	11.6	58.81
XNOR-Net w/o weights activation	43.4	34.1	11.6	88.32

Acknowledgments

The article was prepared within the framework of the Academic Fund Program at the National Research University Higher School of Economics (HSE) in 2017 (grant №17-05-0007) and by the Russian Academic Excellence Project "5-100". A.V. Savchenko is supported by Russian Federation President grant no. МД-306.2017.9.

References

[1] Savchenko, A.V. Search Techniques in Intelligent Classification Systems. / A.V. Savchenko // Springer International Publishing – 2016

[2] Savchenko, A.V. Maximum-Likelihood Approximate Nearest Neighbor Method in Real-time Image Recognition / A.V. Savchenko // Pattern Recognition – 2017 – Vol. 61 – p. 459-469

[3] Choi, K. Automatic Tagging using Deep Convolutional Neural Networks / K. Choi, G. Fazekas, M. Sandler // arXiv preprint arXiv:1606.00298 - 2016

[4] Choi, K. Convolutional Recurrent Neural Networks for Music Classification / K. Choi, G. Fazekas, M. Sandler, K. Cho // arXiv preprint arXiv:1609.04243 - 2016

[5] Krizhevsky A. ImageNet Classification with Deep Convolutional Neural Networks / A. Krizhevsky, I. Sutskever, G. E. Hinton // Advances in Neural Information Processing Systems – 2012 – Vol. 25 – p. 1106-1114

[6] Lin, M. Network In Network / M. Lin, Q. Chen, S. Yan // arXiv preprint arXiv:1312.4400 - 2013

[7] Szegedy, C. Going Deeper with Convolutions / C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich // arXiv preprint arXiv:1409.4842 - 2014

[8] He, K. Deep Residual Learning for Image Recognition / K. He, X. Zhang, S. Ren, J. Sun // arXiv preprint arXiv:1512.03385 - 2015

[9] Han, S. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding / S. Han, H. Mao, W. J. Dally // arXiv preprint arXiv:1510.00149 - 2015

[10] ARM Community [Electronic resource]: https://community.arm.com/iot/embedded/f/discussions/166/fast-deep-learning-inference-on-qualcomm-snapdragon-achieving-super-fast-inference-times-by-tweaking-models-and-codes

[11] Molchanov, P. Pruning Convolutional Neural Networks for Resource Efficient Transfer Learning / P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz // arXiv preprint arXiv:1611.06440 - 2016

[12] Hinton, G. Distilling the Knowledge in a Neural Network / G. Hinton, O. Vinyals, J. Dean // arXiv preprint arXiv:1503.02531 - 2015

[13] Romero, A. FitNets: Hints for Thin Deep Nets / A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, Y. Bengio // arXiv preprint arXiv:1412.6550 - 2014

[14] Chen, W. Compressing Neural Networks with the Hashing Trick / W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, Y. Chen // arXiv preprint arXiv: 1504.04788 - 2015

[15] Lebedev, V. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition / V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, V. Lempitsky // arXiv preprint arXiv:1412.6553 - 2014

[16] Kim, Y.-D. Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications / Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, D. Shin // arXiv preprint arXiv:1511.06530 - 2015

[17] Novikov, A. Tensorizing Neural Networks / A. Novikov, D. Podoprikhin, A. Osokin, D. Vetrov // arXiv preprint arXiv:1509.06569 - 2015

[18] Garipov, T. Ultimate tensorization: compressing convolutional and FC layers alike / T. Garipov, D. Podoprikhin, A. Novikov, D. Vetrov // arXiv preprint arXiv:1611.03214 – 2016

[19] Courbariaux, M. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1 / M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio // arXiv preprint arXiv:1602.02830 - 2016

[20] Rastegari, M. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks / M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi // arXiv preprint arXiv:1603.05279 - 2016

[21] Merolla, P. Deep neural networks are robust to weight binarization and other non-linear distortions / P. Merolla, R. Appuswamy, J. Arthur, S. K. Esser, D. Modha // arXiv preprint arXiv:1606.01981 - 2016

[22] Iandola, F. N. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size / F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, K. Keutzer // arXiv preprint arXiv:1602.07360 - 2016

[23] The Darknet project web site [Electronic resource]: http://pjreddie.com/darknet/tiny-darknet/

[24] Hong, S. PVANet: Lightweight Deep Neural Networks for Real-time Object Detection / S. Hong, B. Roh, K.-H. Kim, Y. Cheon, M. Park // arXiv preprint arXiv:1608.08021 - 2016

[25] Teerapittayanon, S. BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks / S. Teerapittayanon, B. McDanel, H. T. Kung // ICPR – 2016

[26] Langner, O. Presentation and validation of the Radboud Faces Database / O. Langner, R. Dotsch, G. Bijlstra, D. H. J. Wigboldus, S.T. Hawk, A. van Knippenberg // Cognition & Emotion - 2010 - 24(8), 1377—1388. DOI: 10.1080/02699930903485076

[27] Levi, G. Emotion Recognition in the Wild via Convolutional Neural Networks and Mapped Binary Patterns / G. Levi, T. Hassner // ICMI '15 Proceedings of the 2015 ACM on International Conference on Multimodal Interaction – 2015 – p. 503-510