23^{rd} *Computer Vision Winter Workshop*
Zuzana Kúkelová and Júlia Škovierová (eds.)
Český Krumlov, Czech Republic, February 5–7, 2018

Compact ConvNets with Ternary Weights and Binary Activations

Ondřej Holešovský [1,2] *
ondrej.holesovsky@cvut.cz

Atsuto Maki [2]
atsuto@kth.se

[1] Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University
166 36 Prague 6, Czech Republic

[2] School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology
Stockholm, SE-10044 Sweden

Abstract. *Compact convolutional neural network (CNN) architectures with ternary weights and binary activations is a combination of methods suitable for making neural networks more efficient. We show that the combination of ternary weights and depthwise separable convolutions on the CIFAR-10 benchmark can yield a small neural network of size* $32kB$ *and* 83.70% *test accuracy. We present a novel dithering binary activation which we expected to improve accuracy of networks with binary activations by randomizing quantization error. This work presents the outcome of our experiments which show that it brings only mild improvements. A compact SqueezeNet network with ternary weights and binary activations is more accurate than the same network with binary weights. Nevertheless, the accuracy gap to its full precision variant remains large.*

1. Introduction

Deep CNNs achieve great results in several pattern recognition/machine learning tasks. However, CNNs are usually computationally intensive, because of hundreds of thousands to millions trainable parameters. In addition, very many convolutional operations in several convolutional layers need to be performed. When deploying CNNs in a constrained environment like autonomous cars, drones or smartphones, a computationally efficient neural network is desirable.

We evaluated several methods, namely compact CNN architectures, one-bit or two-bit weight quantization and activation binarization on the CIFAR-10 image classification benchmark [1]. We aimed at finding out the current efficiency limits. Quantized weights reduce neural network size [2, 3]. When combined with binary quantized activations [3, 4], both runtime memory and energy footprint are further reduced and inference can be faster. Compact architectures [5, 6] aim at low computational demands with full precision (32 bit floating point) weights while fully preserving the good image classification accuracy of less compact models.

However, less work has be done in examining the complementarity of these methods. The contributions of this paper are:

1. We evaluate the utility of ternary weights ($-W^n, 0, +W^p$) for compact networks, by applying trained ternary quantization TTQ [2] to networks with depthwise separable convolutions. One of the smallest networks trained on CIFAR-10 has only 32 kB model size while achieving 83.7% test accuracy.

2. We present a combination of ternary weights and binary activations for making neural networks both smaller and faster.

3. We introduce a binary *dithering activation* based on ordered dithering. Ordered dithering is a method which yields a quantized image perceptibly as similar to the original image as possible. The dithering activation was motivated by a better information preservation in quantized feature maps. However, our experiments show that it brings only a mild improvement.

This paper extends the master thesis by Holesovsky [7], interested readers may find

*Former master student at KTH, currently employed as a researcher at CTU, CIIRC.

there more background information and experiments. However, many experiments in the thesis were not optimally performed, mostly resulting in inferior image classification accuracy. Therefore this paper presents both corrected and new results. The experiments in the thesis were implemented in TensorFlow framework [8], this paper implements them in PyTorch framework [9].

2. Related Work

2.1. Binary Weights and Activations

One way of reducing the memory footprint of a convolutional neural network is to quantize its weights or activations. When both weights and activations have their resolution reduced, for example to one bit instead of 32 bits, a significant speed-up can be also achieved [4].

Courbariaux et al. [3] introduced a method to train binarized neural networks (BNNs), with activations and weights constrained to -1 and $+1$ at runtime. The quantization of both weights and activations is performed by the signum function. Authors state that full precision weight representations are required during training in order to enable proper gradient updates and averaging. The binary weights are used only in the forward pass. They also presented two gradient approximations of the signum function. The gradient for weight updates involves clipping full precision weight representations to range $[-1, +1]$ after each update. The binary activation gradient is a straight-through (or identity) estimator but with the gradient set to zero for inputs $|x| > 1$, thus taking into account the saturation effect of the signum function. Binarized neural networks can be implemented using XOR and bitcount binary operations instead of floating point multiplications at runtime [3].

A binary weight architecture by Rastegari et al. [4] counts only with weights -1 or $+1$, with a per filter floating point scaling factor in the CNN architecture. The input data are still represented by floating point variables. While the model size is approximately $32\times$ reduced when comparing it to a model with floating point weights, the computational performance can be improved only on computer architectures without a multiply-accumulate (MAC) unit or on architectures without a hardware multiplier. MAC unit is able to compute a combination of two operations, multiplication and addition, in one clock cycle. However, most personal computer or smartphone CPUs like x86 Intel Haswell, ARM Cortex-A7 or GPUs like Nvidia Kepler and others do contain MAC units [10]. Thus the gain in speed will not be significant. The accuracy of their model with binarized weights is the same as AlexNet accuracy on ImageNet [4].

The other architecture proposed by [4], XNOR-Net, binarizes both input data and filters and replaces addition and subtraction operations with binary operations XNOR and bitcount. The authors report a potential of $\approx 58\times$ faster execution than standard AlexNet. The ImageNet top-1 validation accuracy is 12.5% lower than of AlexNet, 44.2% instead of 56.7% [4]. (Note that the top-1 validation accuracy reported by [11] for a single CNN is 59.3%.)

Rastegari et al. [4] have also tried binarizing more advanced ResNet and GoogLenet architectures. However, the best performing binary-weight networks reach lower top-1 accuracy than their full precision counterparts, lower by 8.5% for ResNet-18 and by 5.8% for GoogLenet.

Merolla et al. [12] showed that although the full precision weights are quite different from their binarized versions after training a binary weight network, the test errors on CIFAR-10 are surprisingly similar for full precision CNN and its binary weight variant. They discovered that CNNs performing well with binary weights are robust to other nonlinear distortions at test time as well. Stochastic nonlinear projections applied to the weights during training in the forward pass give even more robust network weights than the signum nonlinearity [12]. With a stochastic training weight projection with clipping ("Tr-StochM-C"), they surpass the full precision state-of-the-art on CIFAR-10. However, the same method applied to AlexNet and ImageNet has worse accuracy than the binary weight network of Rastegari et al. [4].

2.2. Ternary Weight Networks (TWNs)

Li et al. [13] with their ternary weight networks improved the accuracy over binary weight networks by using weights -1, 0 or $+1$ instead of just -1, $+1$. Similar to XNOR-Net [4], they scale each filter with a single full precision value.

Zhu et al. [2] proposed more accurate trained ternary weight networks using trained ternary quantization (TTQ) method. They scale each nonzero weight value (-1 or $+1$) by a separate full precision scaling factor, having two scaling factors per layer instead of one per filter. (Merolla et al. [12] have found that per filter and per layer scaling factors yield similar results.)

Although ternary weight networks require more memory for weight storage than binary weight networks (2 bits per weight vs. 1 bit), they give sparse representations which can help to reduce computation and energy requirements. Zhu et al. [2] report best validation accuracy at 30-50% weight sparsity (30-50% of the weights are zeros). The most accurate networks have different sparsity at different layers [2].

2.3. Compact or Compressed CNNs

Iandola et al. [5] proposed a compact SqueezeNet CNN architecture which is 50× smaller than AlexNet but obtains the same top-1 ImageNet accuracy (57.5%). In addition, they compress it with deep compression [14] another 10× so that it is 510× smaller than AlexNet (only 0.47 MB model size instead of 240 MB) while still maintaining AlexNet accuracy.

Even more compact architectures, such as Xception [6] or MobileNets [15], utilize depthwise separable convolutions. They are based on the hypothesis that spatial and channelwise convolutions can be performed separately.

Deep compression by [14] compresses neural networks by means of pruning, trained quantization and Huffman coding. Pruning reduces the number of parameters in AlexNet 9× and in VGG-16 13× by setting the small weights to zero during training. The remaining parameters are fine-tuned after that. They then cluster the weights of each layer into 64 or 256 clusters. These clustered weights are fine-tuned by training (each cluster by the sum of its weights gradients). The result of the quantization process is 4× or 5.3× smaller CNN model where each weight is represented by an 8 or 6 bit index into the respective layer lookup table. Huffman coding of the weight indexes follows as the last step of the compression process, yielding a 20-30% size reduction. When all three methods are combined together, they compress AlexNet 35× (from 240 MB to 6.9 MB) and VGG-16 49× (from 552 MB to 11.3 MB). Both compressed models have the same accuracy as their non-compressed counterparts on ImageNet.

The network after pruning [14] is not only smaller but also faster as dense vector-matrix multiplication required by fully connected layers is currently slower in GPU and CPU BLAS libraries than sparse vector-matrix multiplication. The evaluation of fully connected layers was 3× faster on average on a CPU with pruned weights. The pruned convolutional layers, however, will not be significantly faster given current standard CPU or GPU hardware and BLAS packages. This may be a disadvantage especially for compact networks without fully connected layers like SqueezeNet [5].

A research article by [16] suggests removing whole neurons instead of only single weights. They obtain 2-3× less parameters by doing that as well as an accuracy comparable to VGG-16. The advantage of their approach over the one of [14] is a direct positive impact on inference speed when evaluating the trimmed network using dense matrix multiplication on a GPU. What is more, neuron trimming may be combined with weight pruning to obtain even more compressed CNN models.

3. Methods

We have focused on combinations of several methods, namely depthwise separable convolutions, trained ternary quantization and binary activations. In what follows, these strategies will be described in more detail.

3.1. Depthwise Separable Convolutions

A depthwise separable convolution, popularized by [6], is a module consisting of two convolutional layers meant for building convolutional neural networks. It is based on the hypothesis that spatial and channelwise convolutions can be performed separately. For example, a convolutional layer with N $3{\times}3{\times}C$ filters can be replaced by a layer with C $3{\times}3{\times}1$ filters and a layer with N $1{\times}1{\times}C$ filters. The first layer is computing a separate spatial convolution (e.g. $3{\times}3{\times}1$) on each of the C input channels. It is followed by a convolutional layer with N filters of size $1{\times}1{\times}C$, where N is the desired output dimension.

When evaluating the effects of depthwise separable convolutions in networks like ResNet-20, we replace with them all convolutional layers except the first one. This can be justified by the fact that the first layer usually contains only quite small $3{\times}3{\times}3$ filters, thus the model size savings would not be very large.

3.2. Trained Ternary Quantization (TTQ)

To the best of our knowledge, out of the current 1 or 2 bit quantization methods, ternary weight networks by [2] preserve the classification accuracy the most when comparing them to full precision weight networks. We use ternarization the same way as [2]. A full precision real scalar weight $\tilde{w}_l(i)$ with a within-layer index i and from layer l is projected to a ternary weight $w_l^t(i)$ in the forward pass using

$$w_l^t(i) = \begin{cases} W_l^p, & \text{if } \tilde{w}_l(i) > \Delta_l \\ 0, & \text{if } |\tilde{w}_l(i)| \leq \Delta_l \\ -W_l^n, & \text{if } \tilde{w}_l(i) < -\Delta_l, \end{cases} \tag{1}$$

$$\Delta_l = t \times \max_i(|\tilde{w}_l(i)|). \tag{2}$$

Like in [2], we use $t = 0.05$ as a constant for the whole network. W_l^p and W_l^n are positive and negative scaling weights, shared by ternary weights within layer l. They are set to 1.0 before the training starts and updated during the training with gradients [1]

$$\frac{\partial L}{\partial W_l^p} = \sum_{i \in I_l^p} \frac{\partial L}{\partial w_l^t(i)}, \tag{3}$$

$$\frac{\partial L}{\partial W_l^n} = -\sum_{i \in I_l^n} \frac{\partial L}{\partial w_l^t(i)}, \tag{4}$$

where L is the loss function of the network, $I_l^p = \{i|\tilde{w}_l(i) > \Delta_l\}$, $I_l^n = \{i|\tilde{w}_l(i) < -\Delta_l\}$ are the sets of indices to positive and negative ternary weights. The weight gradients,

$$\frac{\partial L}{\partial \tilde{w}_l(i)} = \begin{cases} \frac{\partial L}{\partial w_l^t(i)} \times W_l^p, & \text{if } \tilde{w}_l(i) > \Delta_l \\ \frac{\partial L}{\partial w_l^t(i)} \times 1, & \text{if } |\tilde{w}_l(i)| \leq \Delta_l \\ \frac{\partial L}{\partial w_l^t(i)} \times W_l^n, & \text{if } \tilde{w}_l(i) < -\Delta_l, \end{cases} \tag{5}$$

are applied to the full precision weights $\tilde{w}_l(i)$ in each training step.

3.3. Binary Activations

The simplest binary activation is a deterministic signum function (a binary threshold) [3]

$$x_b = \text{Sign}(x) = \begin{cases} +1, & \text{if } x \geq 0, \\ -1, & \text{if } x < 0, \end{cases} \tag{6}$$

which maps a full precision activation x to a binary activation x_b.

[1]Note the minus sign in the equation for the gradient of W_l^n. It was left out in the original TTQ paper.

This binary activation is not fully differentiable. Approximating the gradients is one way of adapting it to the stochastic gradient optimization algorithms, which are widely used in the training of neural networks.

The gradient of the network loss function L w.r.t. the signum activation output x_b (equation 6) is

$$g = \frac{\partial L}{\partial x_b}. \tag{7}$$

We need to approximate the gradient of the loss w.r.t. the signum activation input x. One way of doing that is

$$g_b = \frac{\partial L}{\partial x} = \begin{cases} g, & \text{if } |x| \leq 1 \\ 0, & \text{if } |x| > 1, \end{cases} \tag{8}$$

as applied by [3]. In other words, we approximate the signum function by a linear function in $[-1, 1]$ and by a constant function elsewhere.

3.4. Dithering Binary Activation

We introduce *dithering activation* based on ordered dithering with the hope of reducing the loss of information caused by binary quantization.

Ordered dithering is a method of image quantization which randomizes quantization errors. It preserves lower frequencies at the expense of higher frequencies, yielding a binary image which is perceptibly as similar to the original image as possible. In ordered dithering, the quantization error randomization is achieved by adding a dithering mask to the real pixel values and quantizing the result, in our case using a binary threshold at zero. We choose ordered dithering over error diffusion because it is spatially stable (one changed pixel value does not affect neighbouring pixels). The computation of ordered dithering is also easily parallelizable.

Many different quantization masks exist (Bayer [17] or Void-and-Cluster [18] among others). We choose to use *a dither* quantization mask published in public domain in [19]. It does not exhibit visible grid-like artifacts like the Bayer mask. Furthermore, dithering masks of any size are easily computable. The mask (type 4) value $m \in [-1, +1]$ for a grayscale image pixel at non-negative i_x, i_y pixel coordinates is

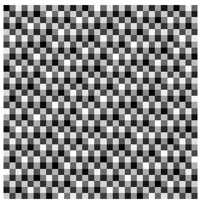
$$m_{i_x,i_y} = 2.0 m'_{i_x,i_y} - 1.0, \tag{9}$$

$$m'_{i_x,i_y} = \frac{((i_x + i_z * 67) + i_y * 236) * 119 \& 255}{255.0} \tag{10}$$

where $*$ is integer multiplication, & is binary bitwise AND operation. All operations with floating point operands (2.0, 255.0, 1.0) are floating point. i_z is the channel number. We quantize one single channel (grayscale) image $x_{i_x,i_y} \in [-1,+1]$ to obtain a binary image

$$x_b = \text{Sign}(x + m). \quad (11)$$

When there are more feature maps (channels) coming into one convolutional layer, we repeat equations 10, 11 for each channel with $i_z = 1$, applying the same mask m each time by default. We call this 2D dithering. In one experiment, we have also tested 3D dithering with different dithering masks for different channels, by setting i_z to the channel index. Thus the full precision input has always the same dimensions as the binary output.


(a) a dither mask

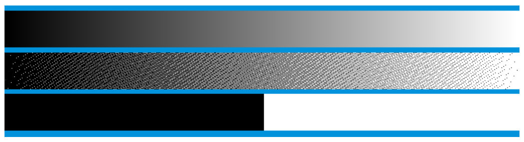
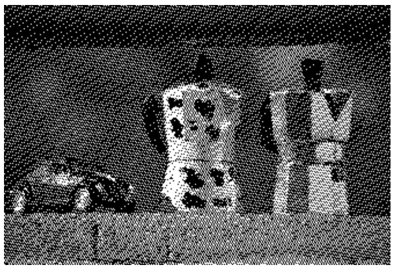

(b) colour ramps

Figure 1: Left: a 32×32 pixels large sample of the quantization mask m_{i_x,i_y}. Right: horizontal colour ramps. They are from top to bottom: grayscale, binary-quantized by a dither, binary-quantized by thresholding.

A sample of the dithering mask can be seen in figure 1. Figure 1 also shows how a horizontal grayscale colour ramp is quantized to binary values by dithering and by simple thresholding. Figure 2 shows an example of a grayscale image and its binary dithered and thresholded versions. Dithering intuitively helps preserve information when there are dark objects on dark backgrounds or light objects on light backgrounds. As dithering introduces a constant noise, high contrast images may be better served with simple thresholding.

We apply dithering activations in convolutional neural networks after the first layer, after spatial subsampling (e.g. max pooling) and before the last layer which has spatial input (spatial size larger than one). Dithering does not have to be before every layer because its effects can be seen in several subsequent layers. We argue that the first binary activation should include dithering, in order to preserve as much information as possible. Max pooling makes data in the spatial dimensions more uniform. We expect dithering to restore more information after max pooling than a simple thresholding activation.

(a) grayscale


(b) binary, a dither

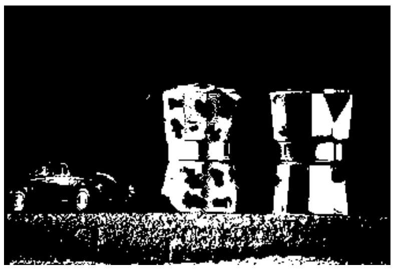

(c) binary, threshold

Figure 2: Illustration of ordered dithering and thresholding of a grayscale image from ImageNet. The image size is 334×224 pixels.

4. Results

4.1. Adapting SqueezeNet v1.1 to CIFAR-10

The CIFAR-10 dataset contains images of a uniform size of 32×32 pixels and three colour channels. In our experiments, we train and evaluate three different CNN architectures, namely BCNN [3] ResNet-20 (the version used in [2]) and SqueezeNet v1.1 [5].

ResNet-20 and BCNN are networks already developed for CIFAR-10 but SqueezeNet was designed for larger input images. Thus in case of SqueezeNet, we upscale the small CIFAR-10 input images to 128×128 pixels.

4.2. Ternary Weights on CIFAR-10

Our setup for training ternary weight networks is almost identical to the approach of [2]. All networks are at first trained full precision, then fine-tuned using TTQ. Learning rate is set to 0.1 for the first 81 epochs, then decreased by a factor of ten after epochs 81, 123 and 150. The momentum parameter was set to 0.9, we use L2-normalized weight decay of 0.0002, batch size 128. The only exception is that we do not apply data augmentation, in order to simplify any future attempts for reproduction of our results. When not stated otherwise, we apply TTQ to all convolutional and fully connected layers except for the first convolutional layer of a given network.

We validate our implementation by training ResNet-20 with full precision and ternary weights. Our obtained results are compared to the original results from [2] in table 1. The effect of data augmentation is significant, but we can see that the absolute accuracy difference between both ternary and full precision network pairs is at most 0.72%.

Under the same training conditions, ternary weight SqueezeNet network has almost the same accuracy as the full precision one. However, when we evaluate ResNet-20 with depthwise separable convolutions (DW-sep.), the ternary weight model loses 7.17% in accuracy on the full precision one. We can preserve more accuracy by ternarizing only the 1×1 convolutional layers and keeping the rest full precision. The accuracy loss is then 2.17% at model size 32kB.

The same results are displayed in figure 3, as points coloured according to the type of weights (ttq, full precision float) and the presence of depthwise separable convolutions (dw).

4.3. Ternary Weights and Binary Activations on CIFAR-10

In order to test ternary weights combined with binary activations as well as our binary dithering activation, we first reproduce the results of [3]. This means training a larger BCNN network on CIFAR-10 with both weights and activations binary. Our training hyperparameters are identical to the CIFAR-10 Theano experiment in [3], which means that the learning rate is exponentially decayed every epoch from 0.001 at epoch one down to 0.0000003 at the final epoch number 500. The square hinge loss computed from a batch of 50 images is optimized by the Adam optimizer.

The results of BCNN are shown in table 2. The accuracy of 88.60% achieved by [3] is within the confidence bounds of our result, $88.16\pm0.68\%$ [2]. Courbariaux et al. [3] state that their full precision BCNN

Setup	Size [kB]	CIFAR-10 [%]
Standard results, 32 bit weights		
*SqueezeNet, ours	2899	88.03
*ResNet-20	1073	86.00
ResNet-20 [2]	1073	91.77
*ResNet-20, DW-sep.	146	85.87
Ternary weights (TTQ)		
*SqueezeNet	188	88.10
*ResNet-20	69	86.72
ResNet-20 [2]	69	91.13
*ResNet-20, DW-sep.	11	78.70
TTQ only for 1x1 convolutions		
*ResNet-20, DW-sep.	32	83.70

Table 1: Network size and test accuracy on CIFAR-10 for full precision and ternary weights. Evaluating networks with standard and depth-wise separable convolutions. Results marked by * come from our own experiments.

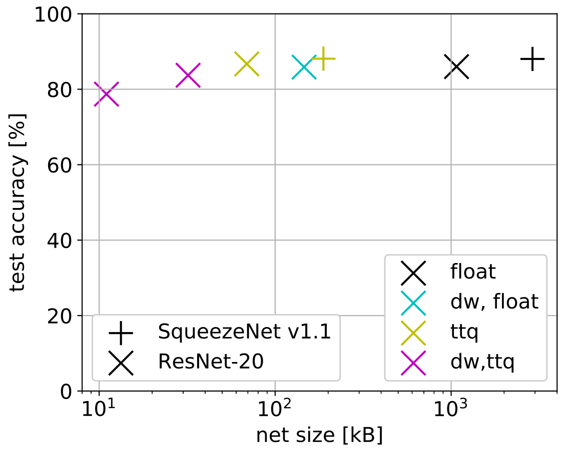


Figure 3: Model size and test accuracy plot for neural networks on the CIFAR-10 dataset.

[2]The confidence is twice the standard deviation based on three trials. Each trial starts from a different random weight initialization. Each time, the network from the training epoch with the best validation accuracy is chosen to be tested.

Setup	Size [kB]	CIFAR-10 [%]
Standard results, 32 bit weights		
*SqueezeNet	2899	88.03
BCNN [3]	56088	?
Binarized Neural Networks		
*SqueezeNet	97	73.87
*BCNN	1766	88.16 ± 0.68
BCNN [3]	1766	88.60
*BCNN 2D dithering	1766	87.29
Ternary Weights and Binary Activations		
*SqueezeNet	188	78.44
*SqueezeNet 2D dit.	188	79.42
*SqueezeNet 3D dit.	188	79.69

Table 2: Network size and test accuracy on CIFAR-10 for full precision, binary and ternary weights. The networks with reduced weight precision have binary activations. When provided, the confidence is twice the standard deviation based on three trials. Results marked by * come from our own experiments.

has almost the same test accuracy as the binarized variant.

In our SqueezeNet v1.1 experiments, we only increase the exponentially decayed learning rate schedule ten times to 0.01 at epoch one and to 0.000003 at epoch 500, due to the deeper network (eighteen weight layers in SqueezeNet instead of nine in BCNN). The rest of the training setup remains unchanged.

We discover that the accuracy of the more compact SqueezeNet v1.1 architecture drops significantly more than in case of BCNN when binarized, from 88.03% down to 73.87%. We can improve this result to 78.44% by using ternary weights (TTQ) together with binary activations, see table 2. Still, the accuracy drop remains significant.

The default 2D variant of the dithering activation is tested on the binarized BCNN network. It causes an absolute accuracy drop of 0.87% when compared to simple binary thresholding activation. We record a mild absolute accuracy improvement of 0.98% thanks to 2D and of 1.25% thanks to 3D dithering activations on ternary weight SqueezeNet v1.1 (table 2).

5. Conclusion

We have shown that by combining efficient depth-wise separated convolutions and ternary weights, small networks can be trained on CIFAR-10. One of the smallest ResNet-based networks has only 32kB model size while achieving 83.70% test accuracy and only 2.17% absolute accuracy loss to its full precision version.

Ternary weights and binary activations improve on binarized networks introduced by [3]. Nevertheless, the full precision to all-quantized accuracy gap remains quite large (8.3% absolute) in case of the compact SqueezeNet architecture. The binarized BCNN network works better (only 1% accuracy loss as reported by [3]) likely because it is 19× larger than SqueezeNet, providing higher parameter and activation redundancy. We have not compared our results to the better XNOR-net baseline [4], because most of their experiments were performed on the ImageNet benchmark using larger networks and we have not been able to reproduce their methods sufficiently well.

We have introduced binary *dithering activation*, as a method motivated by better information preservation in quantized feature maps. However, our experiments show that the default 2D dithering brings overall only a mild accuracy improvement. In addition, the attempt to preserve even more information in the feature maps by 3D dithering yields almost the same classification accuracy as 2D dithering on SqueezeNet.

The higher performance boost achieved by dithering on SqueezeNet (1.25% absolute vs. -0.87% on BCNN) could be explained by its upscaled input images (128×128 vs. 32×32 pixels in BCNN). An upscaled image has more low frequency components which can be better preserved by dithering binarization. Whether this larger classification accuracy improvement thanks to dithering applies to higher resolution images as well remains an open research question.

Acknowledgements

I would like to thank to Václav Hlaváč and Radoslav Škoviera for text corrections and suggestions, as well as to the three anonymous reviewers for inspiring insights into the content of this paper.

This work was supported by the European Regional Development Fund under the project Robotics for Industry 4.0 (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000470).

One part of the computations was perfomed on CTU CIIRC resources. The other part of the computations was performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) at PDC (Tegner computer) and HPC2N (Kebnekaise computer).

References

[1] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009. 1

[2] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *ICLR 2017 poster*, 2017. [Online]. Available: http://arxiv.org/abs/1612.01064 1, 2, 3, 4, 5, 6

[3] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016. 1, 2, 4, 5, 6, 7

[4] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542. 1, 2, 7

[5] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016. 1, 3, 5

[6] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *arXiv preprint arXiv:1610.02357*, 2016. 1, 3

[7] O. Holesovsky, "Compact convnets with ternary weights and binary activations," Master's thesis, KTH, 2017. 1

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/ 2

[9] PyTorch, "Pytorch," http://pytorch.org/, 2017. 2

[10] Wikipedia, "Multiply–accumulate operation," https://en.wikipedia.org/wiki/Multiply-accumulate_operation, 2017. 2

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105. 2

[12] P. Merolla, R. Appuswamy, J. Arthur, S. K. Esser, and D. Modha, "Deep neural networks are robust to weight binarization and other non-linear distortions," *arXiv preprint arXiv:1606.01981*, 2016. 2

[13] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016. 2

[14] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015. 3

[15] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017. 3

[16] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016. 3

[17] B. Bayer, "An optimum method for two-level rendition of continuous-tone pictures," IEEE International Conference on Communications, archived from the original (PDF) on 2013-05-12, https://web.archive.org/web/20130512190753/http://white.stanford.edu:80/~brian/psy221/reader/Bayer.1973.pdf, 1973. 4

[18] R. A. Ulichney, "Void-and-cluster method for dither array generation," in *IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology*. International Society for Optics and Photonics, 1993, pp. 332–343. 4

[19] Øyvind Kolås, "a dither," http://pippin.gimp.org/a_dither/, 2017. 4