XNOR nets report

Authors: Michal Hradiš, Honza Kohut
Faculty of Information Technology, Brno University of Technology

Abstract

This document summarizes implementations of XNOR nets and related experimental results. Training implementation is provided in CAFFE. For inference of the trained networks, a custom standalone C++ code is available. The inference code has minimal dependencies, is well optimized, and supports wide range of network architectures.

Table of contents

Code repositories

- Caffe implementation of XNORNet training https://github.com/kohuthonza/caffe, with short documentation at https://github.com/kohuthonza/caffe/wiki
- C++ inference using logical operations https://github.com/DCGM/XNOR-inference, with short documentation at https://github.com/DCGM/XNOR-inference/wiki. The inference code does not have any dependencies except Eigen library which is used for efficient floating point matrix multiplication.
- Network definitions are at https://github.com/kohuthonza/caffe-net-generator/tree/master/definitions
- Results and trained networks: TODO

Caffe implementation

The XNORNet Caffe extension consists of two layers **BinaryConvolution** and **Signum** which have both GPU and CPU implementations. All computation is done in float numbers. The layers are implemented in files:

- **Signum** - signum_layer.cpp (CPU), signum_layer.cu (GPU)
- **BinaryConvolution** - binary_conv_layer.cpp (CPU), cudnn_binary_conv_layer.cpp (GPU), cudnn_binary_conv_layer.cu (GPU). GPU implementation works only with cuDNN, if cuDNN isn't avalible and GPU mode is set, the CPU implementation is used.
- Binary fully connected layer is not provided, use **BinaryConvolution** with appropriate kernel size (1x1) instead.

The **XNOR nets** (Rastegari et al., 2016) approximate floating point convolution by binary convolution where both inputs and weights are limited to values {-1, +1}. The results are scaled to reflect the energy of the individual convolution filters.

XNOR convolution "layer sequence" has to be composed of three layers **BNorm**->**Signum**->**BinaryConvolution** (see Figure 1). Batch normalization centers the previous activations around zero to minimize information loss. **Signum** layer binarizes the activations to {-1, +1} - this operation is non-linear and no additional activation function (e.g. ReLU) is needed. Pooling can be added after **BinaryConvolution** - to pool float (integer values). Pooling of binary values would result in extreme information loss.

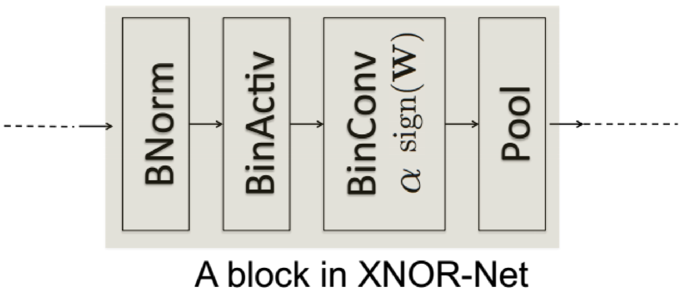


Figure 1. The basic building block of XNORNet. The Pooling layer is optional. The block produces real-valued outputs. From (Rastegari et al., 2016).

BinaryConvolution layer keeps floating point weight tensor which is updated during learing. In forward pass, the weight tensor is binarized. The approximation of the floating point convolution $I * W$ is with binary filter weights is:

$$I * W \approx (I * B)\,\alpha;\ B = sign(W);\ \alpha = \frac{1}{n}||W||_{l1}$$

where B is a binarized weight tensor, and α is a floating point constant for each convolution filter which maintains the energy of the filters. This approximation is defined by equations (4) and (6) in (Rastegari et al., 2016).

BatchNorm-Signum. The full XNOR approximation in the original paper (Rastegari et al., 2016) in equation (11) is:

$$I * W \approx (sign(I) * sign(W))\ \odot K\alpha$$

where K captures energy of each image position $K_{ij} = \frac{1}{n}\left\|X_{ij}\right\|_{l1}$ where X_{ij} is the filter-sized activation map patch (all channels) centered around pixel ij. The symbol $\odot$ indicates elementwise multiplication after K is repeated for each output channel of the convolution. This operation best preserves local energy; however, the term K is omitted by the authors of XNORNets in the reference implementation[1]. It adds a relatively large number of floating-point operations and it would prevent purely integer/binary network inference (see Section Binary Inference). The K term does not seem to be needed and we omit it.

Gradients of binarization (Signum). Signum function does not have a gradient - it is 0 everywhere except at the discontinuity at 0. However, "pseudo" gradients can be used and the networks still train. The basic option is to consider the $sign$ function as identity during backpropagation. In (Rastegari et al., 2016, page 6 bottom), the authors advocate setting the gradient of $sign(r)$ to 0 if $|r| > 1$. The idea is to ignore gradient of large inputs as it is "imposible" to change tem anyway, and to try to improve values which are close to 0. Our

[1] https://github.com/allenai/XNOR-Net/blob/master/newLayers/BinActiveZ.lua

implementation uses this “trick” in the **Signum** layer[2] and in **BinaryConvolution** layer for the weights. However, for weigths, small gradient is retained even when the weight is greater than 1.[3] Anyway it does not have much effect for the filter weights as they are generally smaller than 1.

Caffe Layers Cheat Sheet

Signum

- Computes signum(x) - output is +1, -1
- Cannot be used inplace (input is needed for backward pass)
- If value of input is in range (-1, 1) the gradient is passed otherwise the gradient is multiplied by zero
- If value of input is in range (-1, 1) the gradient is passed otherwise the gradient is multiplied by zero
- Signum_layer.cpp, signum_layer.cu

```
layer {
  name: "name"
  type: "Signum"
  bottom: "bottom"
  top: "top"
}
```

BinaryConvolution

- Binarizes weights and computes convolution.
- Needs cuDNN.
- Definition is the same as in normal convolution with additional binary_convolution_param.
- param **update_weight_diff -** Set to TRUE to match the reference implementation. Gradients of weights with values outside <-1;+1> are significantly reduced. This option does not have much effect as weights are normally within the range <-1;+1>. This option should be irrelevant for per-weight adaptive learning algorithms such as Adam.
- param **scale_weight_diff -** Set to TRUE to match the reference implementation. It is the scaling from line 85 in https://github.com/allenai/XNOR-Net/blob/master/util.lua. This

[2] As in https://github.com/allenai/XNOR-Net/blob/master/newLayers/BinActiveZ.lua lines 14, 15.
[3] https://github.com/allenai/XNOR-Net/blob/master/util.lua line 78 - `updateBinaryGradWeight()`

constant scaling is not part of the original paper and is almost 1 for higher channel numbers. This option did not have any measurable effect in our experiments.

- Binary_conv_layer.cpp, cudnn_binary_conv_layer.cpp, cudnn_binary_conv_layer.cu

```
layer {
  name: "name"
  type: "BinaryConvolution"
  bottom: "bottom"
  top: "top"
  param { lr_mult: val decay_mult: val}
  param { lr_mult: val decay_mult: val}
  convolution_param {
      bias_term: bool
      kernel_size: val
      num_output: val
      stride: val
      pad: val
      group: val
      weight_filler {
          type: "type"
          std: val
      }
      bias_filler {
          type: "type"
          value: val
      }
  }
  binary_convolution_param {
     update_weight_diff: bool (default: true)
     scale_weight_diff: bool (default: true)
  }
}
```

Network architectures

First and last layer of all networks should not be binarized. This is consistent with the original paper (Rastegari et al., 2016) as stated in Section 4.1 and with the reference implementation.

The **Caffe implementation** allows any network architecture. However, only architectures which follow the XNOR-Net basic building block in Figure [XNOR block] make sense. The sequence of layers should mostly be …->BNorm->Signum->BinConv(->Pooling)->BNorm->Signum.

- **BinConv** should be preceeded by **Signum** layer - otherwise it is not a XNOR network.

- **Signum** layer should be preceeded by **BatchNorm** layer to retain the most activation information in the **Signum** layer (and to conform to the original XNOR paper).
- Any pooling should be done right after **BinConv** layer (not on binary values).

Transition from floating point to binary. A straightforward transformation of the first layers of standard networks is: CONV -> BNorm -> ReLU -> BNorm -> BinConv …
Some notes:

- This layer progression contains two **BatchNorm** layers and two non-linearities and could be repaced just by CONV -> BNorm -> Signum -> BinConv. However, our preliminary experiments showed slight reduction in accuracy for this simplified version.
- XNOR networks generally need many channels which may be too many for the first floating point layer filters (resulting in suboptimal speed). A good alternative would be to use just few filters in the first layer and expand the channels with a second floating point layer with 1x1 filters.

Transition from binary to floating point. The output of **BinConv** can be used the same way as output of standard convolution layer.

Examined architectures

We examined two architectures. Simple linear network similar to VGG-16 (Simonyan and Zisserman, 2014) and a variant of ResNet (He et al., 2016).

VGG. The VGG networks are simple linear networks with 3x3 convolutional filters. The starting layers are CONV -> BNorm -> ReLU -> BNorm -> Signum -> BinConv.

ResNet. The original ResNets (He et al., 2015) are not much suitable for XNORNets. As shown in Figure [ResNet blocks] (a), the nonlinearity is after the elementwise addition, which would severely restrict the information flow when exchanged to **Signum**. An updated version of ResNets which moves all operations to the “computation” branch and removes any non-linearity from the “identity” branch [ResNet blocks] (b) (He et al., 2016) is more suitable. Here, the order of layers matches the order of XNORNet layers and the information flow between ResNet blocks is not limited by binarization.
One issue of the chosen ResNet architecture is how to change the number of channels (e.g. when reducing spatial resolution by pooling). We used two options. On CIFAR, the number of channels is changed by a binary convolution on the straight path. On PFC faces, we decided the to use floating-point convolutions with 1x1 filters between ResNet blocks: ResBlock -> CONV -> Pooling -> ResBlock -> ResBlock. These 1x1 floating point convolutions are slower, but retain

more information. These layers are not that expensive - these layers amount to 12.3% of computational time for one of the best XNOR networks[4] on PFC.

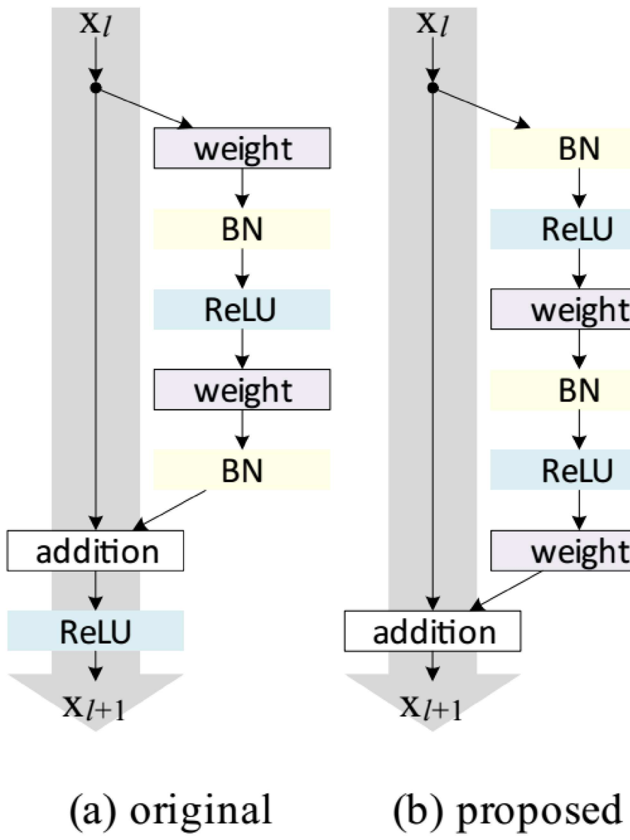


Figure [ResNet blocks]. The identity mapping ResNet block proposed by He et al.(2016). The block (b) is suitable for XNORNets - ReLU layer are replaced by Signum layers.

Binary Inference

A Caffe model is transformed such that it would be most efficient to compute inference with binary operations. Normal floating point layers are mostly preserved and their behavior does not change. Layers after **BinaryConvolution** get transformed depending on the network structure. If possible, layers get merged to minimize computation. **BinaryConvolution** by itself has integer outputs in the inference code.

Network transformation related to XNOR nets are:

[4] BIN_C128S2_2RES128_P2_2RES192_P2_2RES256_P4_2RES4096_FC1024_FC6300

- Any sequence of BinConv (*-> Pooling | BNorm | ReLU) -> Signum gets transofrmed into BinConv (*-> Pooling) -> Signum by merging all scaling, biases (BNorm) and ReLU into the Signum threshold.
- BinConv which is not followed by Signum is transformed into BinConv -> Bias and produces floating point tensors.

Other transformations include (if not removed):
- BNorm becomes Scale -> Bias

Branching restrictions
- Concatenation is supported both on floating point and binary tensors. However, it is limited to concatenating channels (not in spatial domain). Channels have to be divisible by 8 for the binary concatenation.
- Elementwise layer is limited to addition and to floating point values.

Datasets

Two datasets were used in experiments - CIFAR-10 and PFC face dataset.

PFC dataset was created by combining three face datasets: PubFig, FaceScrub a CASIA WebFace (Stratil, 2017). Common identities were merged and identities from LFW dataset were removed. The PFC dataset includes 317,328 images of 6,154 people. The images were geometrically normalized (rotation, translation, scale) to 64x64 resolution, such that positions of the eyes would be (22,22) and (22,42).

For this study, the number of images of a single person was limited to 200 and 10% of the images were reserved for testing. The minimal number of images per identity is 15. The final training set contains 254,963 images and the testing set 29,046 images. Plot showing the image counts for individual identities is shown in Figure [Dataset distribution]. Note that a maximum classification accuracy of a blind classifier on the test set is only 0.069% despite the imbalance in the dataset.

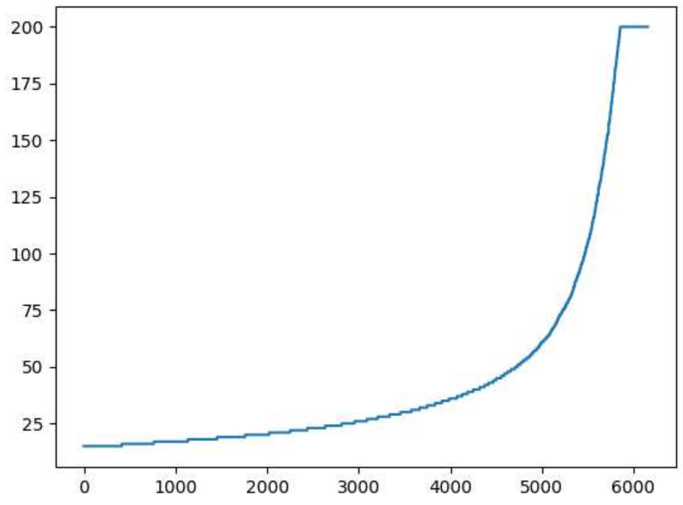


Figure [Dataset distribution]. The number of images per identity in the face dataset. TODO

Experiments

Network architecture strings

Network architecture is described by strings such as BIN_C032S2_2RES32_P2_2RES64_P2_2RES128_P4_2RES4096_FC1024_FC6300. The meaning is:

- C - convolution, number of filters (S2 - spatial stride 2)
- 2C32 - two convolutional layers with 32 filters
- If not stated otherwise, all convolution kernels are 3x3.
- RES - resnet block, number of filters
- 2RES32 - 2x ResNet block with 32 filters
- P2 - pooling layer with stride 2
- FC1024 - fully connected layer with 1024 channels
- BIN_ - XNOR network - All convolutions in the network are binary except the first layer and the last layers. PFC networks have two floating point layers at the end to reduce computational complexity due to 6300 output classes.
- All inference time measurements of PFC networks were without the last two floating point layers for both XNOR network and floating point networks. The reason is that these two layers dominate the computational time and would not be present in other networks (e.g. gender and age classification) or would be much smaller (e.g. facial fingerprint extraction).

Some network as named e.g. VGG.32. These networks follow the structure of VGG networks with stages of 2 convolutional layers followed by pooling. The number of channels doubles between the stages.

General notes

- Speed measured on Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz.
- Inference of floating point networks were done using Intel fork of Caffe https://github.com/intel/caffe. This branch is optimized for CPU computation and uses MKL. It is up to 10x faster on CPUs compared to the master branch.
- All 4 cores are utilized in the experiments. Batch size for Intel Caffe is 4 as higher batch sizes improved speed only marginally. Binary inference uses 4 parallel threads.

Convolution inference speed

This experiment measures speed of pure matrix multiplication in single precision float numbers vs. in binary XNOR operations on standard CPU. The tool used for the measurements is testGEMM.cpp. Library Eigen was used for the floating point matrix multiplication. The core code for the binary multiplication is almost the same as in the binary inference code and it computes value of each output pixel in the inner loop and uses 64-bit popcnt instruction:

```
for(int k = 0; k < kerSize; k++){
        cumul += _mm_popcnt_u64(input[k] ^ filter[k]);
}
results[p * fCount + f] = cumul;
```

The full results are in spredsheet and they were computed on 4 core Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz. The experiment used 4 independent computation threads running in parallel and the results are reported per thread (multiply by 4 to get the total speed).

Setup. The computation speeds were measured for a range of image sizes, number of input and output channlels. These three number define the size of matrices in the multiplication - first matrix is [pixels, #input_channels] and the second matrix is [#input_channels, #output_channels]. #input_channels is in fact the size of the convolutional filters times the number of channels of the input activation map (it would be 576 for 3x3 filter and 64 input channels).

Results. The measured speedup ranges from 2.46x up to 117x, while 80% of the speedups are between 3.7x and 25x (see Figure [Speedups]). The speed of binary convolution is relatively low for low number of input channels (64 - that is 1x1 convolution with only 64 input filters). The floating point convolution is slower when some of the dimension of matrices in the multiplication is low (see Figure [Speed]). Full scatter plot of the speeds is in Figure [Speeds_compare].

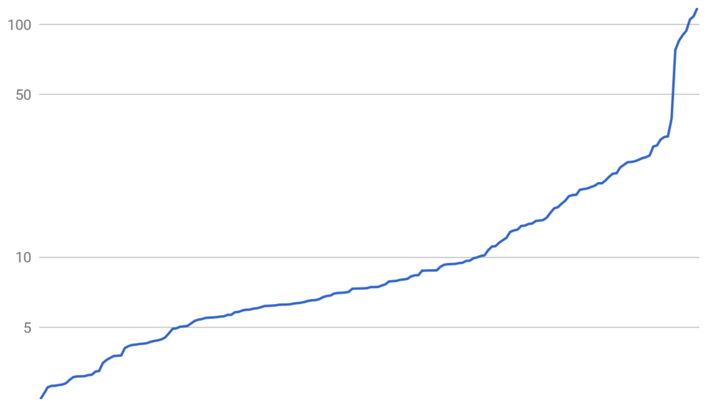
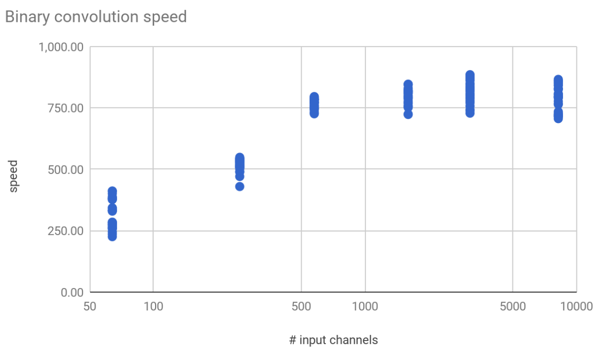


Figure [Speedups]. Speedups of binary convolution vs. floating point convolution.



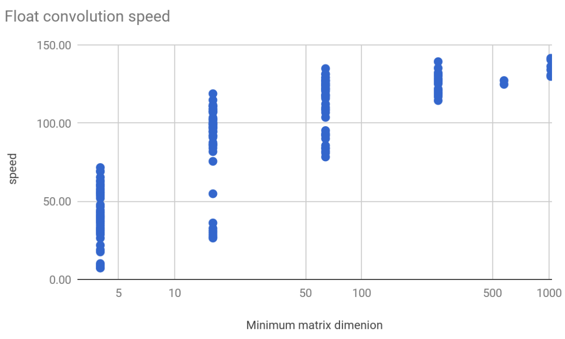


Figure [Speed]. Speed of binary convolution with respect to number of input channels and float convolution speed with respect to minimum dimension of the matrices in matrix multiplication.

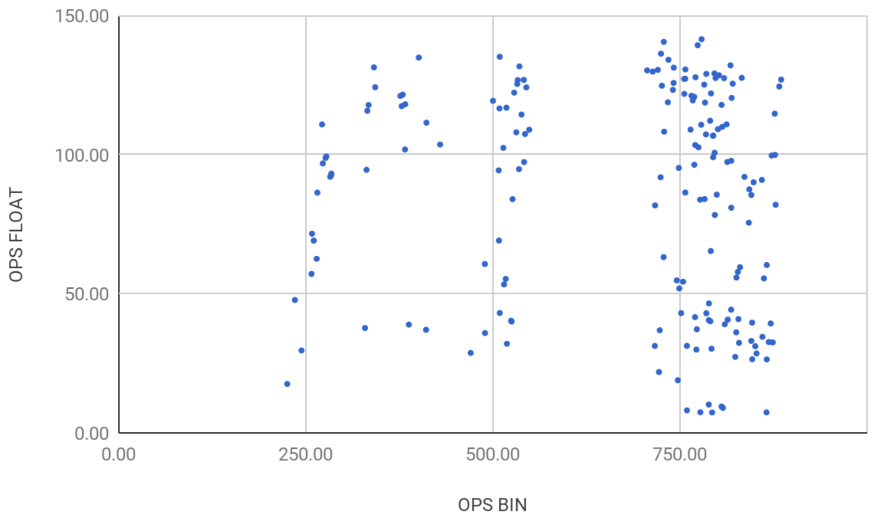


Figure [Speeds_compare]. Speeds of floating point convolution and equivalent binary convolutions on 3.30GHz CPU. The speeds are in millions of equivalent floating point operations per second per thread (multiply by number of available threads and adjust to the target CPU frequency to get a rough expected speed).

Full network binary inference speedup

This section shows speedup of the binary inference code compared to floating point inference in Intel Caffe. The speedups are in the range 2-10x. Lower speedups are achieved in networks which have more floating point operations. These are network with more filters in the first layer or ResNets on PFC which use floating point convolution to increase number of channels.

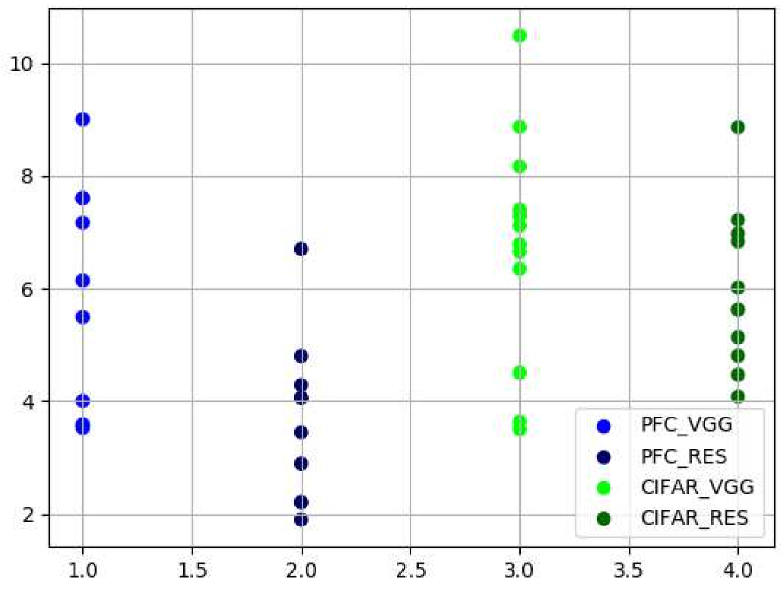


Figure [inference_speedup]. Y-axis show inference speedup. X-axis distinguishes different types of networks.

The difference can be seen in the binary inference times of the next two networks. The first network processes one image in 11.2ms and its first layer takes 5.788 ms which is 51% of the time (in the experiment, 4 images were processed in parallel). The second network takes 7.2 ms as a whole and the first layer 1.9 ms (25%). The respective speedups are 3.59x and 9. In the following timings, the blue part represents binary part of the networks.

BIN_1xC128_P2_1xC128_P2_1xC256_P2_FC1024_FC2048_FC512_FC6300
conv1.1-nobias | runtime: 5.788
pool1-float | runtime: 0.38534

signum2.1 | runtime: 0.258631
conv2.1-int | runtime: 2.39226
pool2-int | runtime: 0.177783
signum3.1 | runtime: 0.0590335
conv3.1-int | runtime: 1.43666
pool3-int | runtime: 0.0805233
reshape-int | runtime: 0.053666
signum4.1 | runtime: 0.0414077
conv4.1-int | runtime: 0.440113
signum4.2 | runtime: 0.00305643
conv4.2-int | runtime: 0.0604529
conv4.2-alfa | runtime: 0.0065613
conv4.2 | runtime: 0.00458729
conv4.2-inplace0-bias | runtime: 0.00338247
conv4.2-inplace0 | runtime: 0.00445118
conv4.2-inplace1 | runtime: 0.00382282

BIN_2xC32_P2_2xC64_P2_2xC128_P2_FC2048_FC4096_FC512_FC6300
conv1.1-nobias | runtime: 1.90916
signum-1.2 | runtime: 0.267778
conv1.2-int | runtime: 1.53519
pool1-int | runtime: 0.159606
signum2.1 | runtime: 0.0703179
conv2.1-int | runtime: 0.64008
signum2.2 | runtime: 0.10252
conv2.2-int | runtime: 0.705865
pool2-int | runtime: 0.0718005
signum3.1 | runtime: 0.0318742
conv3.1-int | runtime: 0.370664
signum3.2 | runtime: 0.0603226
conv3.2-int | runtime: 0.603715
pool3-int | runtime: 0.041454
reshape-int | runtime: 0.0277299
signum4.1 | runtime: 0.012786
conv4.1-int | runtime: 0.38416
signum4.2 | runtime: 0.00391297
conv4.2-int | runtime: 0.179678
conv4.2-alfa | runtime: 0.00732426
conv4.2 | runtime: 0.00515427
conv4.2-inplace0-bias | runtime: 0.00429722
conv4.2-inplace0 | runtime: 0.00565005
conv4.2-inplace1 | runtime: 0.00615179

Batch size

This experiment examines training with different mini-batch sizes on PFC dataset with relatively small residual network. Both XNOR and float networks train better with larger batch sizes. In general, XNOR networks benefit more from larger batch size. However the improvement is stronger on training sed and the accuracy saturates on test set. Experiments on PFC were done with batch size of 128 in order to be able to train even larger networks.

Most XNOR networks on CIFAR were trained with batch size 128 except the largest ones which were trained with batch size 64, and the largest XNOR ResNet was trained with 16 due to GPU memory constraints. The batch size 64 did not affect results significantly on CIFAR. The batch size 16 probably degraded accuracy of the network significantly. VGG float networks were trained with batch size 64 and float ResNets with batch size 128.

Learning rate was gradually reduced during training of the networks. We did not observe further improvement when training beyond the point reached in the experiments.

Dataset: PFC faces
Network: both BINARY and FLOAT
C032S2_2RES32_P2_2RES64_P2_2RES128_P4_2RES4096_FC1024_FC6300
Training: Adam, lr 0.008, lr_step 50000 by 0.666, iterations 125000

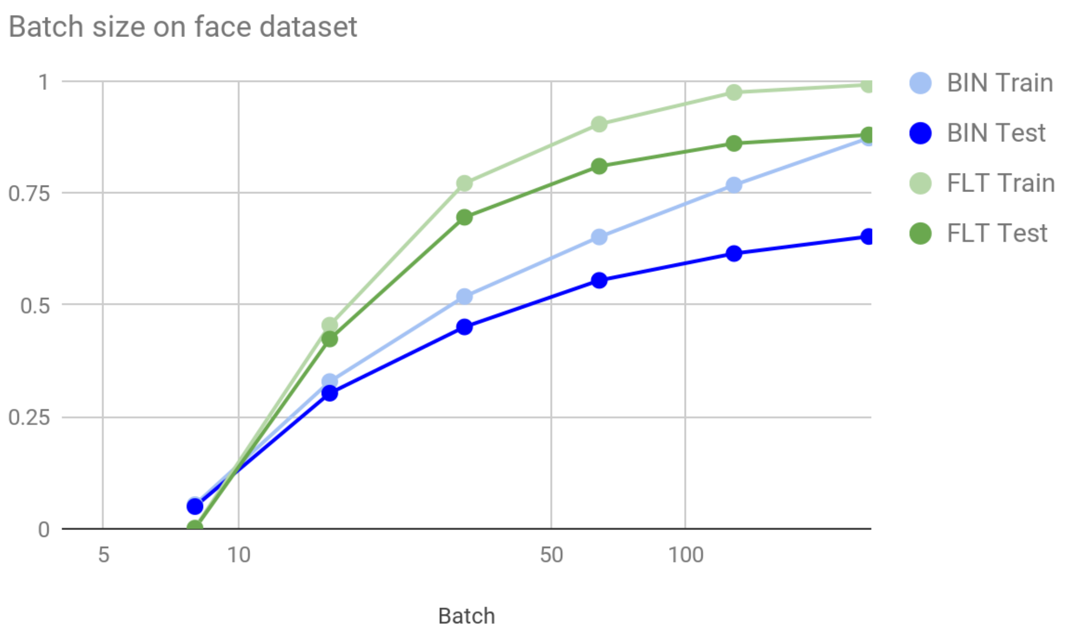


Figure [batch_size]. Training with different batch sizes. X-axis show batch size. X-axis show classification accuracy.

Training algorithm

We have experimented with training algorithms - mainly SGD and Adam. On CIFAR, the results were similar and the final experiments used SGD in most cases. On PFC, the XNOR networks did not train well with SGD and all the reported experiments use Adam.

CIFAR-10

Speed and accuracy of networks on CIFAR-10 dataset is shown in Figure [CIFAR_RESULTS]. On this task, the float networks provide better absolute accuracy 91.8%[5], while the best XNOR network has 88.7%[6]. A float network running at the speed of the top XNOR network has accuracy 0.90% (the inference speeds are 543 and 551). The float and XNOR networks become comparable at higher speeds (after ~2000 fps). More detailed results follow.

The largest XNOR ResNet[7] was trained with batch size of only 16 due to GPU memory constraints. This probably reduced its accuracy. However, networks of such sizes are becoming impractical to train.

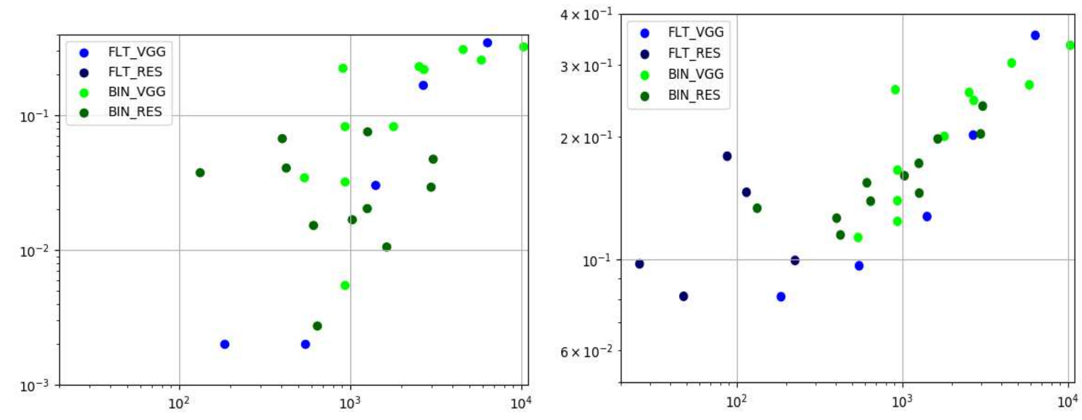


Figure [CIFAR_RESULTS]. XNOR nets vs float nets on CIFAR-10. Showing speed as images per second (x-axis) and classification error (y-axis). Training data (left), testing data (right). BIN - XNOR; FLT - float.

FLOAT VGG network size

[5] 2xC64_P2_2xC128_P2_2xC256_P2_2xC512_FC256_FC256_FC10 and C128_3x2RES128_P2_FC1024_FC10_DROP30
[6] BIN_2xC128_P2_2xC256_P2_2xC256_P2_FC1024_FC1024_FC10
[7] BIN_C256_3x2RES256_P2_FC1024_FC10

This table compares float networks with VGG architecture of different size. Larger networks have higher accuracy, but the performance saturates at the 64 channels in the first layer - training accuracy is already 1.0.
The networks follow the basic VGG structure with with 3x3 convolutions and dropouts before each pooling layer. The input to fully connected layer is 4x4:
2xC4_P2_2xC8_P2_2xC16_P2_2xC32_FC256_FC256_FC10,
2xC8_P2_2xC16_P2_2xC32_P2_2xC64_FC256_FC256_FC10, ...

Filters in first layer	Test acc.	Train acc.	Fps
4	0.646	0.655	6383
8	0.798	0.833	2690
16	0.873	0.970	1417
32	0.904	0.999	551
64	**0.919**	**1**	186

XNOR VGG network size
XNOR VGG-like networks require relatively large number of filters at the beginning of the network and do not necessarily require doubling of channels in deeper layer. However, this effect is probably limited to the CIFAR dataset due to strong tendency to overfit the small dataset.

The binary fully connected layers have to significantly larger compared to float network. We 1024 for XNOR nets and only 256 for float nets.

The number of filters has to be much higher compared to float networks. Our XNOR “VGG.64” has similar accuracy as float VGG.8 and the best XNOR network “VGG.128.256” has accuracy between VGG.32 and VGG.16.

The XNOR networks still benefit from dropout which manages to limit overfitting. However, this may not be needed on large datasets.

Network	Test acc.	Train acc.	Fps
2xC64_P2_2xC64_P2_2xC64_P2_FC1024_FC1024_FC10	0.8	0.918	1802
2xC128_P2_2xC128_P2_2xC128_P2_FC1024_FC1024_FC10	0.861	0.995	939
2xC128_P2_2xC128_P2_2xC128_P2_FC1024_FC1024_FC10_DROP30	0.876	0.918	939
2xC128_P2_2xC256_P2_2xC256_P2_FC1024_FC1024_FC10_DROP30	**0.887**	**0.966**	543

Float ResNets

Float ResNets had to be larger than VGG nets. The best results were achieved by ResNet with 128 channels in all ResNet blocks and with 2 blocks in each stage (network with 256 channels did overfit).

Network	Test acc.	Fps
C64_3x3RES64_P2_FC1024_FC10	0.8210999942	88
C128_3xRES128_P2_FC1024_FC10_DROP30	0.9005999959	226
C128_3x2RES128_P2_FC1024_FC10_DROP30	**0.9187999976**	48
C256_3x2RES256_P2_FC1024_FC10_DROP30	0.9024999952	26
C64_2RES64_P2_2RES128_P2_2RES256_P2_FC1024_FC10	0.8539999944	115

XNOR ResNets - Filter count
XNOR ResNets peaked at 128 channels. However, Network with 256 channels was trained on small mini-batches of 16 samples which definitely reduced its accuracy.

Network	Train acc.	Test acc.	Fps
BIN_C64_3x2RES64_P2_FC1024_FC10	0.839766	0.983203	1034.19
BIN_C128_3x2RES128_P2_FC1024_FC10_DROP30	0.885234	0.959375	425.153
BIN_C256_3x2RES256_P2_FC1024_FC10_DROP30	0.866562	0.9625	133.47

XNOR ResNest - depth
XNOR ResNets with more ResNet blocks per pooling stage generally perform better. The effect is not significant after 2 blocks per stage. The difference between one and two blocks is very significant.

Network	Train acc.	Test acc.	Fps
BIN_C64_3xRES64_P2_FC1024_FC10	0.797	0.971	2987
BIN_C64_3x2RES64_P2_FC1024_FC10	0.840	0.983	1034
BIN_C64_3x3RES64_P2_FC1024_FC10	0.846	0.985	614
BIN_C128_3xRES128_P2_FC1024_FC10_DROP30	0.855	0.925	1271
BIN_C128_3x2RES128_P2_FC1024_FC10_DROP30	0.885	0.959	425
BIN_C32_RES32_P2_RES64_P2_RES128_P2_FC1024_FC10	0.763	0.953	3072
BIN_C32_2RES32_P2_2RES64_P2_2RES128_P2_FC1024_FC10	0.828	0.980	1265
BIN_C64_RES64_P2_RES128_P2_RES256_P2_FC1024_FC10	0.803	0.989	1644
BIN_C64_2RES64_P2_2RES128_P2_2RES256_P2_FC1024_FC10	0.861	0.997	647

PFC faces

Speed and accuracy of face recognition networks is shown in Figure [PFC_RESULTS]. On this task, the float networks provide better speed-accuracy tradeoff with accuracy lower by ~2.5% across the whole range of speeds. The best achieved accuracy with float network is 89%[8] while best XNOR networks have 86.5%[9] and 85.5%[10] accuracy. The best float network with accuracy around 86% are able to process 240[11] or 280[12] images per second while the best XNOR network only process 96 and 134 images per second. More detailed results follow.

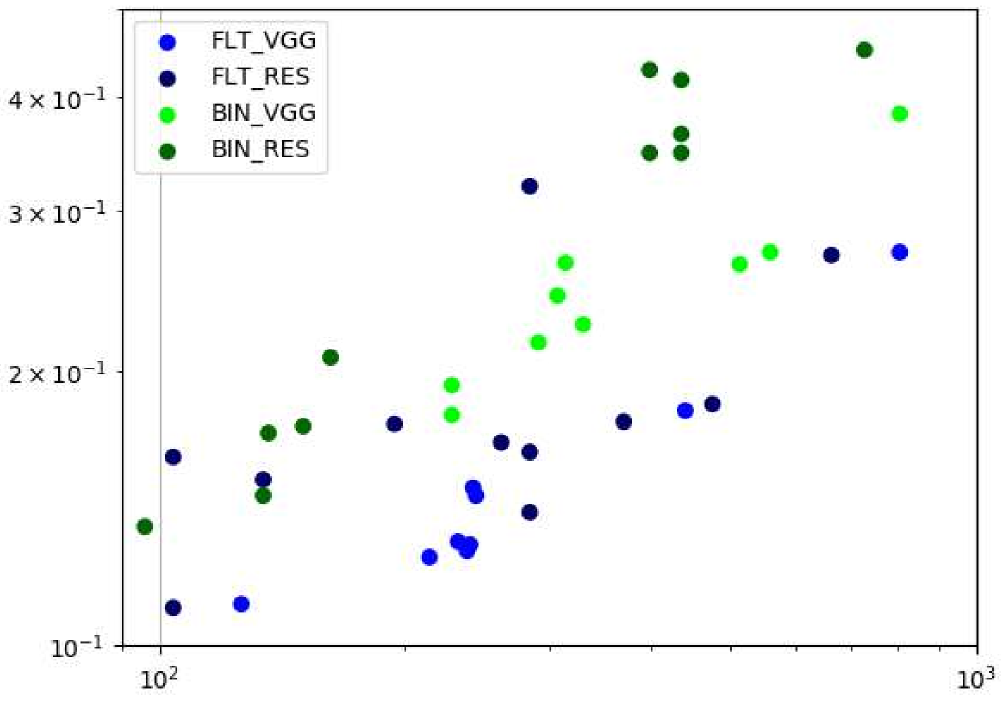


Figure [PFC_RESULTS]. XNOR nets vs float nets on PFC. Showing speed (x-axis) and classification error on testing set (y-axis). BIN - XNOR; FLT - float.

[8] VGG.48 and FLT_C032S2_6RES32_P2_6RES64_P2_6RES128_P4_6RES4096_FC1024_FC6300
[9] BIN_C128S2_C1_2RES128_P2_2RES192_P2_2RES256_P4_2RES4096_FC1024_FC6300
[10] BIN_C128S2_2RES128_P2_2RES192_P2_2RES256_P4_2RES4096_FC1024_FC6300
[11] VGG.32
[12] FLT_C032S2_2RES32_P2_2RES64_P2_2RES128_P4_2RES4096_FC1024_FC6300

FLOAT VGG network size

This table compares float networks with VGG architecture of different size. Larger networks have higher accuracy, but the performance saturates at the 48 channels in the first layer - training accuracy is already 1.0.

The architecture is 2xC64_P2_2xC128_P2_2xC256_P2_FC2048_FC4096_FC512_FC6300.

Filters in first layer	Test accuracy	Image per second
8	0.73	805
16	0.819	440
32	0.875	214
48	0.889	126

XNOR VGG network size

Larger networks have higher accuracy. The training accuracy for the network with 64 filters is already 99%. Larger network was not tested as it did not fit into memory.

Filters in first layer	Test accuracy	Image per second
16	0.617	805
32	0.738	513
64	0.812	228

FLOAT ResNet network depth

This tab compares float ResNets with different number of ResNet blocks in each stage. The general architecture is C032S2_**X**RES32_P2_**X**RES64_P2_**X**RES128_P4_**X**RES2048_FC1024_FC6300, where X is the number of blocks per stage. The results show no significant differences beyond 2 blocks per stage.

ResNet blocks per stage	Test accuracy	Image per second
1	0.816	475
2	0.837	284
3	0.825	194
6	0.839	104

XNOR ResNet network depth

Similarly to floating point networks, no improvement is observed beyond 2 blocks per stage

ResNet blocks per stage	Test accuracy	Image per second

1	0.55	729
2	0.583	435
3	0.572	398

FLOAT ResNet block bottleneck

ResNet blocks are composed of two convolutional layers. The first layer does not necessarily have to have the same number of channels as the direct route. Speed can be improved by lowering the number of filters and creating a bottleneck. The compared networks are FLT_C032S2_2RES32_P2_2RES64_P2_2RES128_P4_2RES4096_FC1024_FC6300/.

The 0.5 bottleneck improves speed at a slight accuracy cost. The 0.25 bottleneck does not improve speed much. All ResNets on PCF further use 0.5 bottleneck.

ResNet channel bottleneck ratio	Test accuracy	Image per second
0.25	0.833	298
0.5	0.837	284
1.0	0.848	134

XNOR ResNet network channel width

More channels in XNOR ResNets improve results. However, even larger network become impractical to train. We already had to restrict the number of channels in the deeper layers of the

NET	Test accuracy	Image per second
C032S2_2RES32_P2_2RES64_P2_2RES128_P4_2RES4096_FC1024_FC6300/	0.583	435
C064S2_3RES64_P2_3RES128_P2_3RES256_P4_3RES4096_FC1024_FC6300/	0.793	162
C128S2_2RES128_P2_2RES192_P2_2RES256_P4_2RES4096_FC1024_FC6300/	0.854	134

XNOR ResNet - start with 2 float convolutions

This experiment explores a network which starts with two floating point layers where the second one has 1x1 kernels. This extension results in slight improvement at significant speed reduction (30% for this network). An option would be to decrease the number of filters in the first layer.

NET	Test accuracy	Image per second

BIN_C128S2_C1_2RES128_P2_2RES192_P2_2RES256_P4_2RES4096_FC1024_FC6300/	0.865	96
BIN_C128S2_2RES128_P2_2RES192_P2_2RES256_P4_2RES4096_FC1024_FC6300/	0.854	134

Conclusions

- XNOR networks are probably not able to reach the same accuracy as floating point networks even with increased size on harder recognition tasks which have enough data to prevent severe overfitting. On some problems, they could provide competitive speed-accuracy ratio on PC-CPU, but would probably significantly overcome floating point networks only if SSE/AVX registers could be used for inference of the XNOR networks. XNOR network would be suitable for platforms which allow fast bit population count and offer only slow floating point matrix multiplication (no SSE/AVX).
- For the face PFC recognition problem, accuracy could have been probably improved with larger networks, however such networks would already be too large to train on a single GPU.
- On CPUs, the observed speedups due to binarization in the current 64-bit implementation for suitable networks are around 5x. This relatively low speedup is due to very efficient floating point convolution (GEMM) from MKL which uses AVX registers and instructions and which is able to process 8 float numbers at once on the target CPU. This instruction-level parallelism would suggest speedup of XNOR nets of 8x (XNOR nets are computed using 64-bit instructions). This expected speedup is reduced by floating point layers of the XNOR networks. The range of speedup of the convolution itself may be significantly higher and lower due to different efficiencies of floating point and binary convolutions for different input and filter sizes/counts.

References

(Stratil, 2017) STRATIL, Jan. Hluboké neuronové sítě pro rozpoznání tváří ve videu. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Hradiš, Ph.D.

(Rastegari et al., 2016) Mohammad Rastegari and Vicente Ordonez and Joseph Redmon and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. ECCV, 2016.

(Simonyan and Zisserman, 2014) K. Simonyan, A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv technical report, 2014

(He et al., 2016) Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Identity Mappings in Deep Residual Networks. arXiv:1603.05027, 2016.

(He et al., 2015) Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2015.