CNN optimization

Rassadin A.

01.2017 - 02.2017

What to optimize?

- Training stage time consumption (CPU / GPU)
- Inference stage time consumption (CPU / GPU)
- Training stage memory consumption
- Inference stage memory consumption
- Training stage power consumption
- Inference stage power consumption

Methods' classification

By the accuracy loss:

- *lossless*;
- optimization with accuracy loss;
- optimization-accuracy trade-off.

By the approach type:

- architectural;
- operational;
- computational;
- hardware.

By the restrictions:

- architecture-dependent;
- architecture-independent.

By the optimization type:

- speed;
- memory consumption;
- energy consumption.

By the implementation:

- runtime implementation;
- two-step (training -> optimization);
- sequential (training -> optimization -> re-training).

Methods overview: the general-kind optimization

- continuous architecture improvement (evolution)

  convolution spread up, replacement FC with convolutions, 1x1 convolutions, residual connections etc.
- hardware / driver optimization

  special-purpose processing and memory units (Google TPU, Nervana Engine, Movidius VPU, SnapDragon 820 etc.)
- optimization-precision trade-off

  vDNN, FP16, INT8
- special-purpose frameworks

  NNPack, tiny-dnn, Darknet
- general framework optimizations

Library	Class	Time (ms)	forward (ms)	backward (ms)
Nervana-neon-fp16	ConvLayer	230	72	157
Nervana-neon-fp32	ConvLayer	270	84	186
TensorFlow	conv2d	445	135	310
CuDNN[R4]-fp16 (Torch)	cudnn.SpatialConvolution	462	112	349
CuDNN[R4]-fp32 (Torch)	cudnn.SpatialConvolution	470	130	340
Chainer	Convolution2D	687	189	497
Caffe	ConvolutionLayer	1935	786	1148
CL-nn (Torch)	SpatialConvolutionMM	7016	3027	3988
Caffe-CLGreenTea	ConvolutionLayer	9462	746	8716

Methods overview: additional optimization

- Pruning

  Han et al. 2016, Molchanov et al. 2016

- Distillation The Knowledge

  Hinton et al. 2014, Romero et al. 2014

- Weights Hashing / Quantization

  Chen et al. 2015, Han et al. 2016

- Tensor Decompositions: TT, CP, Tucker, …

  Lebedev et al. 2015, Kim et al. 2015, Novikov et al. 2015, Garipov et al. 2016

- Binarization

  Courbariaux / Hubara et al. 2016, Rastegari et al. 2016, Merolla et al. 2016, Hou et al. 2017

- Architectural tricks (*simple* but yet *powerful* architecture)

  Hong et al. 2016, Iandola et al. 2016 etc.

- The *silver bullet* architecture --it's a kind of maaagic..

  Hasanpour et al. 2016

Distillation the knowledge

The most significant papers:

- *Distilling the Knowledge in a Neural Network, Hinton et al. 2014;*
- *FitNets: Hints for Thin Deep Nets, Romero et al. 2014.*

The idea:

- Transfer (**distilling**) the predictive power of well-trained network or ensemble of networks to lightweight one.

The receipt:

- Train a reference, probably cumbersome, model (network or an ensemble of networks) with big generalization ability.
- Train a single, probably thinner, network to imitate the predictions of the cumbersome one

Disadvantages:

- Still demand in sufficient resources for training
- Sequential optimization

Advantages:

- Optimization-accuracy trade-off

Weights Hashing / Quantization

The most significant papers:

- *Compressing Neural Networks with the Hashing Trick, Chen et al. 2015;*
- *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, Han et al. 2016.*

The idea:

- Equal weights (in terms of some magnitude) receiving the same hash.

Advantages:

- Optimization-accuracy trade-off

Tensor Decompositions

The idea:

- Decomposition of original tensors to lower-rank ones which speedups computations.

Disadvantages:

- Strong mathematics inside

Advantages:

- Strong mathematics inside
- Optimization-accuracy trade-off

Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition, Lebedev et al. 2015:

- Non-linear least squares for low-rank CP-decomposition -> fine-tuning

Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications, Kim et al. 2015:

- Rank selection with variational Bayesian matrix factorization -> Tucker decomposition on kernel tensor -> fine-tuning

Tensorizing Neural Networks, Novikov et al. 2015, Ultimate tensorization: compressing convolutional and FC layers alike, Garipov et al. 2016

- Decomposition of convolutional and FC leyers’ weights with TT technique

Binarization

The most significant papers:

- *Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1, Courbariaux / Hubara et al. 2016;*
- *Deep neural networks are robust to weight binarization and other non-linear distortions, Merolla et al. 2016;*
- *XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks, Rastegari et al. 2016;*
- *Loss-Aware Binarization Of Deep Networks, Hou et al. 2017.*

The idea:

- Weights' (activations, inputs) values binarizing with the ***Sign(x)*** (possible variations) function which gives its compact representation and allows bitwise operations.

Disadvantages:

- Specific GPU implementation in order to reduce computations via bitwise operations

Advantages:

- Architecture-independent.

Pruning

The most significant papers:

- *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, Han et al. 2016;*
- *Pruning Convolutional Neural Networks for Resource Efficient Transfer Learning, Molchanov et al. 2016.*

The idea:

- Removing weights with the minimal impact to the prediction.

Advantages:

- Very basic approach
- Optimization-accuracy trade-off

Architectural tricks

The idea:

- Using modern techniques or architectural tricks makes architecture computationally-efficient but yet *powerful*.

Disadvantages:

- Limitation in architectural variations
- Possibly framework upgrading (not necessarily)
- Task-specific architecture (not necessarily)

Advantages:

- No additional tricks: it *should* works every time the same

SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and %3c0.5 MB model size, Iandola et al. 2016:

- Introducing *Fire* module

PVANet: Lightweight Deep Neural Networks for Real-time Object Detection, Hong et al. 2016:

- Using a bunch of modern techniques making architecture be computationally-efficient but yet powerful
- C.ReLU, Inception, Deconv, ~20 layers

Architectural tricks

Tiny Darknet, Joseph Redmon & Darknet:

- “It's only 28 MB but more importantly, it's only 800 million floating point operations. The original Alexnet is 2.3 billion. Darknet is 2.9 times faster and it's small and it's 4% more accurate.”

2016 - BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks, Teerapittayanon et al. 2016:

- Adding additional side branch classifiers allows prediction results to exit the network early via these branches with high confidence

Comparisons. Implementations

	completeness	framework	customizable	framework customization	references
Distillation the knowledge	*implementable*	-	-	-	-
HashedNets	full	Torch	yes	no	[1]
Deep Compression	decode, AlexNet	Caffe	no	no	[1]
Ristretto	dead?	Caffe	yes	yes	[1]
CP-Decomposition	arch-partial	Caffe / Matlab	yes	no	[1]
TensorNet	full	Theano (Lasagne), Matlab, TensorFlow	yes	no	[1], [2]
BinaryNet	full	Theano, Torch	yes	no	[1], [2]
Binary-Weight-Network	full	Torch	yes	no	[1]
XNOR-Net	full	Torch	yes	no	[1]

Comparisons. Implementations

	completeness	framework	customizable	framework customization	references
SqueezeNet	more than needed	Caffe + MXNet, Keras etc.	yes	no	[1] + [2], [3], [4]
PVANet	partial, R-CNN	Caffe	yes	yes	[1]
Tiny Darknet	full	Darknet	yes	no	[1]
BranchyNet	*implementable*	-	-	-	-

Comparisons. Optimization type

	Train - Memory	Train - Speed	Inference - Memory	Inference - Speed
Distillation the knowledge	-	-	+	+
HashedNets	?	?	+	?
Deep Compression	-	-	+	+
CP-Decomposition	-	-	+	+
TensorNet	?	-	+	?
BinaryNet	-	-	+	+
Binary-Weight-Network	-	-	+	+
XNOR-Net	-	+	+	+
SqueezeNet	N/A	N/A	+	?
PVANet	N/A	N/A	+	+
Tiny Darknet	N/A	N/A	+	+
BranchyNet	-	-	-	+

Comparisons. Scores

	Memory reduction while training	Memory reduction while inference	Inference speedup	Accuracy gain	Baseline model	Dataset
FitNets	-	36	13.36	-1.17	Maxout	CIFAR-10
HashedNets	-	64	?	0,24	*same-size*	MNIST
Deep Compression	-	49 (~4)	?	0.33	VGG-16	ImageNet
CP-Decomposition	-	12	4.5	-1	AlexNet	ImageNet?
TensorNet	?	80	?	-1.1	*simple*	CIFAR-10
BinaryNet	~32 (theoretical)		3.4~23	1.53	Maxout	CIFAR-10
Binary-Weight-Network	-	67	58 (CPU)	-8.5	ResNet-18	ImageNet
XNOR-Net	-			-18.1		
SqueezeNet	?	50	1.	0.3	AlexNet	ImageNet
Tiny Darknet	?	60	2.9	1.5		
BranchyNet	-	-	1.9	-1,53	ResNet-110	CIFAR-10

Several Conclusions

- **Pruning** is a general optimization approach, applicable to every architecture and, probably, most efficient by the complexity reduction. Unfortunately, it's still not common..
- Every standalone architecture (already optimal or not) can become a baseline to every other optimization approach.
- Using simplified architectures justified only if it gives sufficient result on your task.
- **BranchyNet** reveals a kind of general way for optimization, so it can be applied with every other method.
- From the *Binarization* methods, **XNOR-Net** is the best decision when accuracy is less important, otherwise - **BWN**.
- **SqueezeNet** more preferable than Tiny Darknet because of Darknet implementation.
- **DeepCompression** is hard-estimated because of critical impact of the pruning.
- **DeepCompression** is *two-stage* optimization while the **HashedNets** - runtime.
- **CP-decomposition** is more general approach while the **TensorNet** can give a superior performance.
- *Tensor Decomposition* techniques and especially *Binarization* ones are most promising for the nearest progress.

A *Super-Optimization-Scheme*

1. Training a super-ensemble with **Snapshot Ensemble** and **vDNN** with most-powerful framework
2. **Distillation The Knowledge** to the lightweight (**fully-convolutional**, with (**wide-**)**residual** or **dense** connections etc. etc.) **Branchy**N**et**-like architecture
3. **Pruning**
4. **Binarization**
5. **Tensor Decomposition**
6. Extra-optimized inference (**low-precision** calculations, **optimized platform** etc.)

Experiments. Formulation

Baseline - visual emotion recognition, Levi et al. 2015. Unfortunately, original EmotiW 2015 dataset not available and Radboud Faces Database was used instead for training and evaluation.

- **CP-decomposition**: decomposition of every convolutional layer of the author's pretrained RGB model, evaluation on whole RaFD dataset.
- **HashedNets**, **BWN**, **XNOR-Net**: learning from scratch on RGB images from RaFD dataset (cropped by face) using originally proposed VGG-S architecture and Torch; no data augmentation, independent and balanced train / test sets.
- **TensorNet**: TensorFlow..
- **SqueezeNet**: learning from scratch on RGB images from RaFD dataset (cropped by face) using originally proposed VGG-S architecture and Keras (Theano); data augmentation (Z-score, rotation, zoom, horizontal flipping), independent and balanced train / test sets.

Experiments. CP-decomposition

Characteristic:

- very *home-made* code;
- **Matlab** dependency redundant;
- manual fine-tuning?

The setting:

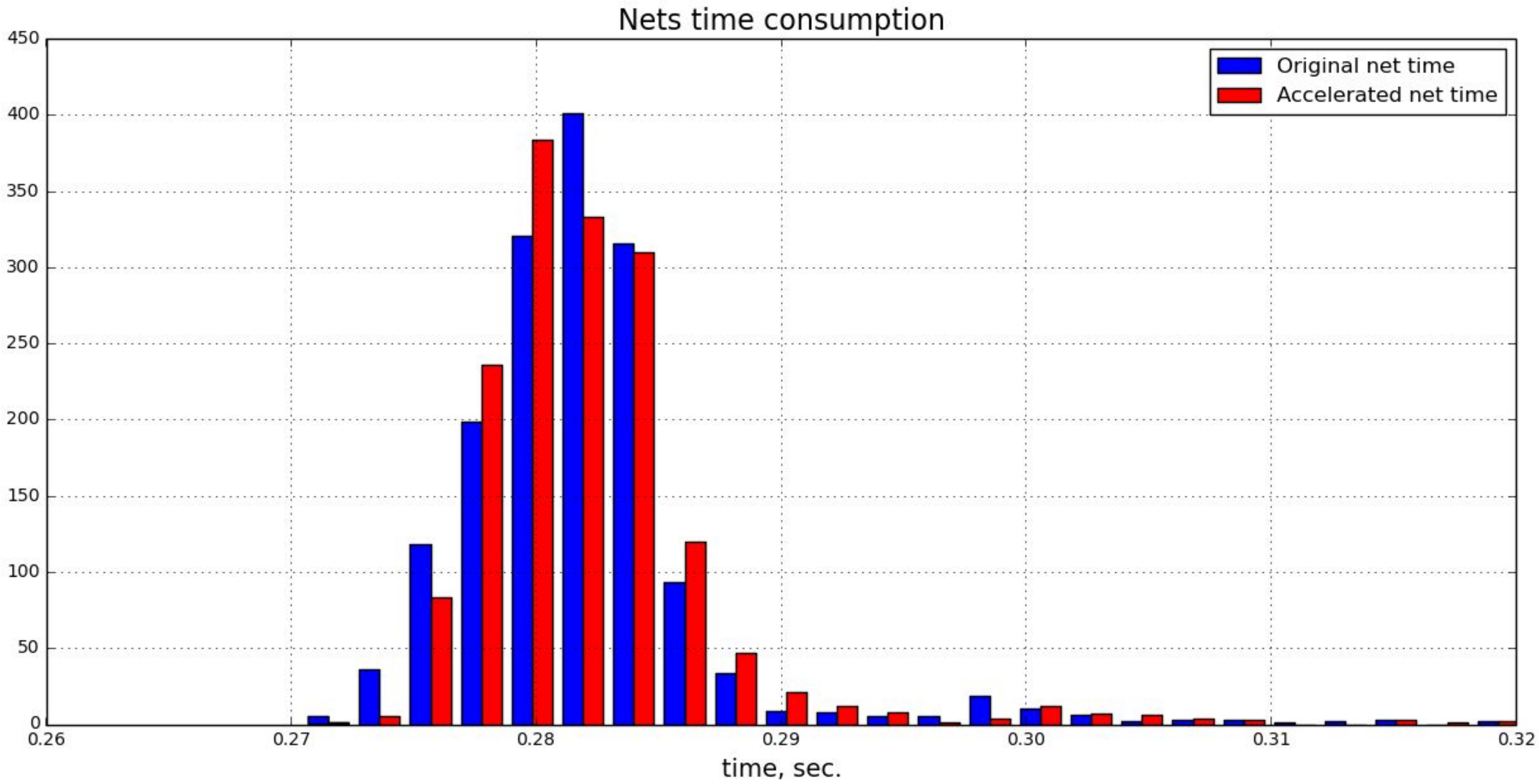
- decomposition of every convolutional layer;
- the last **Caffe** state (*master* branch);
- accuracy metric - prediction proximity between original and accelerated models, call *similarity*;
- speedup metrics: prediction time both on CPU and GPU in comparison with the baseline, GPU memory consumption (directly from `nvidia-smi`).

Conclusions:

- insufficient similarity loss only for the 1st convolutional layer decomposition;
- iterative process possibly can give more more optimization but accuracy loss still expecting a high.

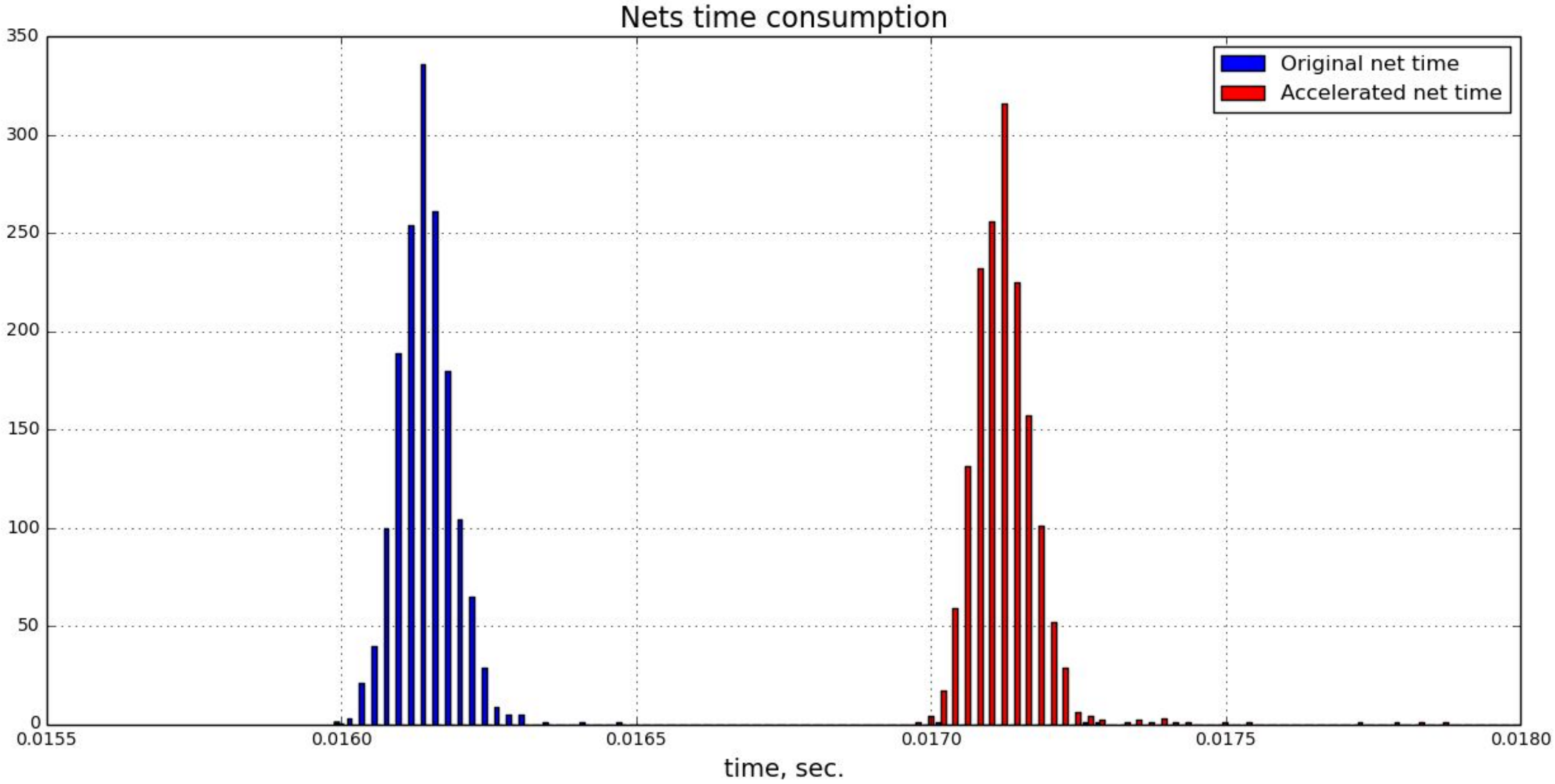
Experiments. CP-decomposition

RANK=16, decomposition only for the 1st convolutional layer, similarity = 95.4%



Experiments. CP-decomposition

RANK=16, decomposition only for the 1st convolutional layer, similarity = 95.4%



Experiments. HashedNets

Characteristic:

- modern **CUDA** / **gcc** incompatibility: worked on 1 machine from 4 with manual fixes;
- well-done code in the rest;
- **an issue:** unable to save the model file.

The setting:

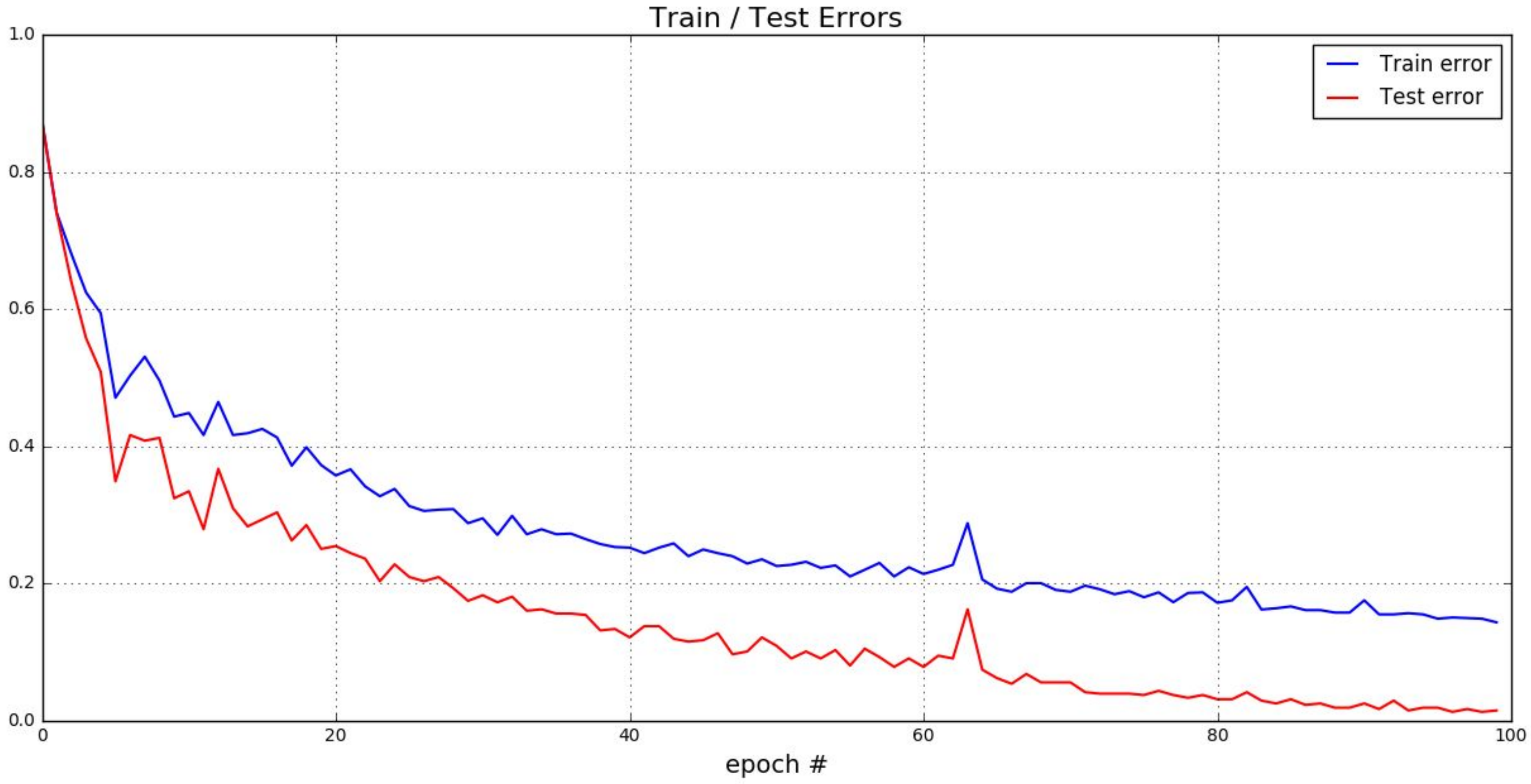
- SGD with momentum, fixed? lr-pocily without regularization;
- 100 epochs for the training;
- accuracy metrics: train / test losses and accuracies;
- speedup metrics: prediction time (GPU-only) in comparison with the baseline, averaged over 10 runs with 2 (minimal and maximal) mini-batch sizes, GPU memory consumption (directly from `nvidia-smi`).

Conclusions:

- explicit training slowdown.

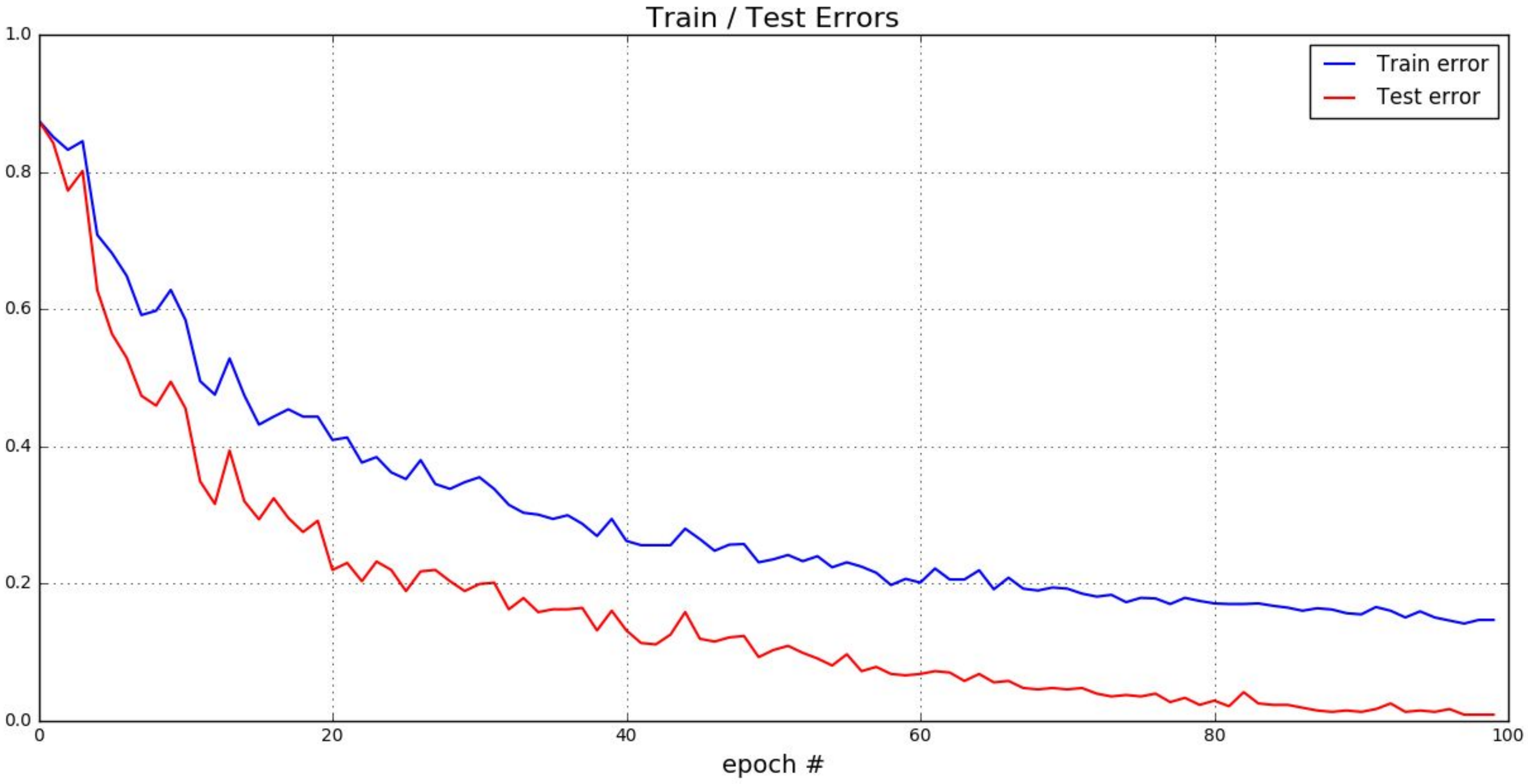
Experiments. HashedNets

Baseline test accuracy: 98.77%, **HashedNet** test accuracy: 99.18%



Experiments. HashedNets

Baseline test accuracy: 98.77%, **HashedNet** test accuracy: 99.18%



Experiments. XNOR-Nets

Characteristic:

- the algorithm itself very simple, but implementation overloaded.

The setting:

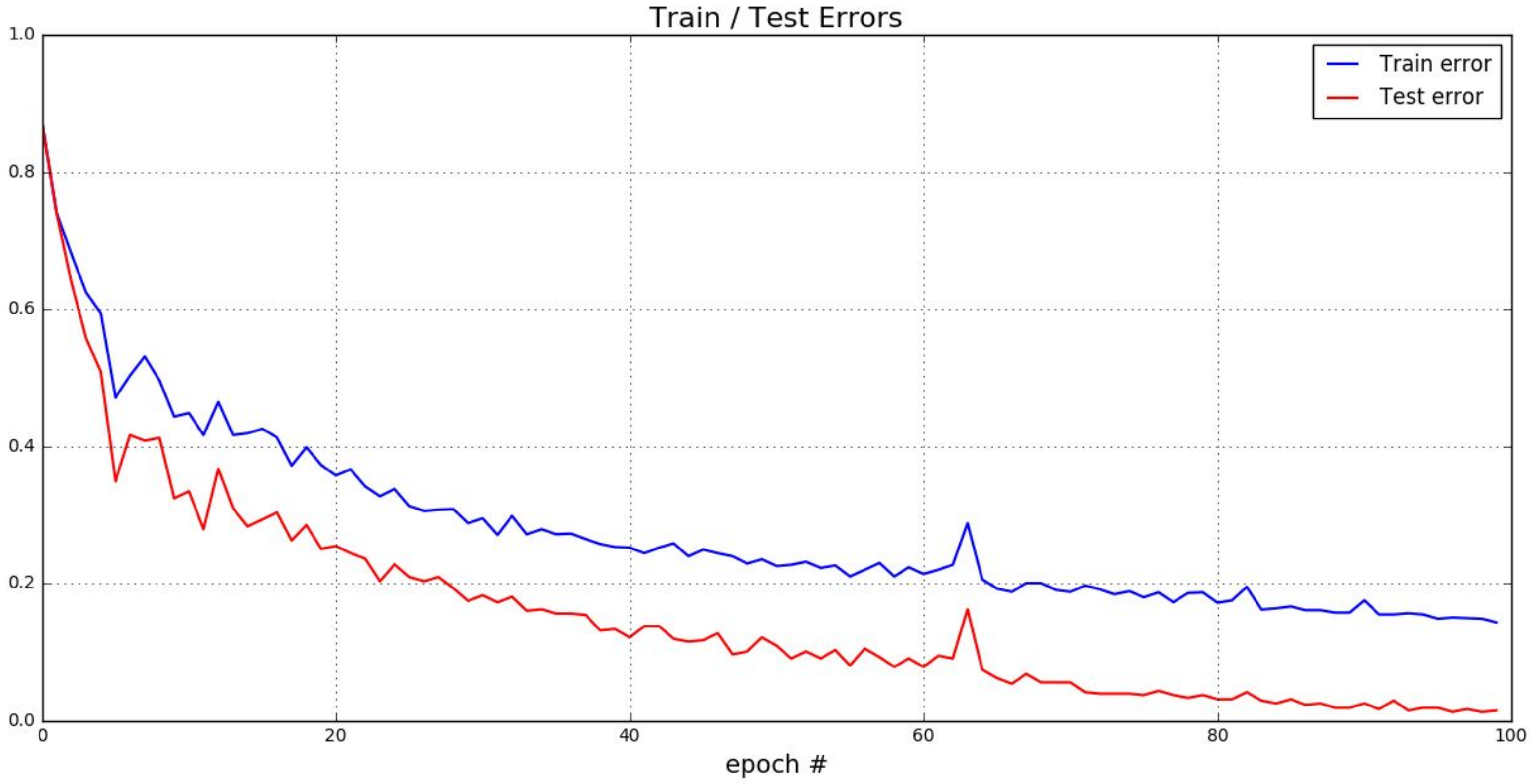
- SGD with momentum, fixed? lr-pocily without regularization;
- 100 epochs for the baseline; 25 epochs for the **BWN**, 100 epochs for the **XNOR-Net**;
- XNOR-Net layers (ordering) configuration according the paper and build-in example (AlexNet): 1st *conv-bn-poll* block followed by reordering;
- accuracy metrics: train / test losses and accuracies;
- speedup metrics: prediction time (GPU-only) in comparison with the baseline, averaged over 10 runs with 2 (minimal and maximal) mini-batch sizes, GPU memory consumption (directly from `nvidia-smi`).

Conclusions:

- BWN training speedup (~4 times);
- XNOR-Net is more compact.

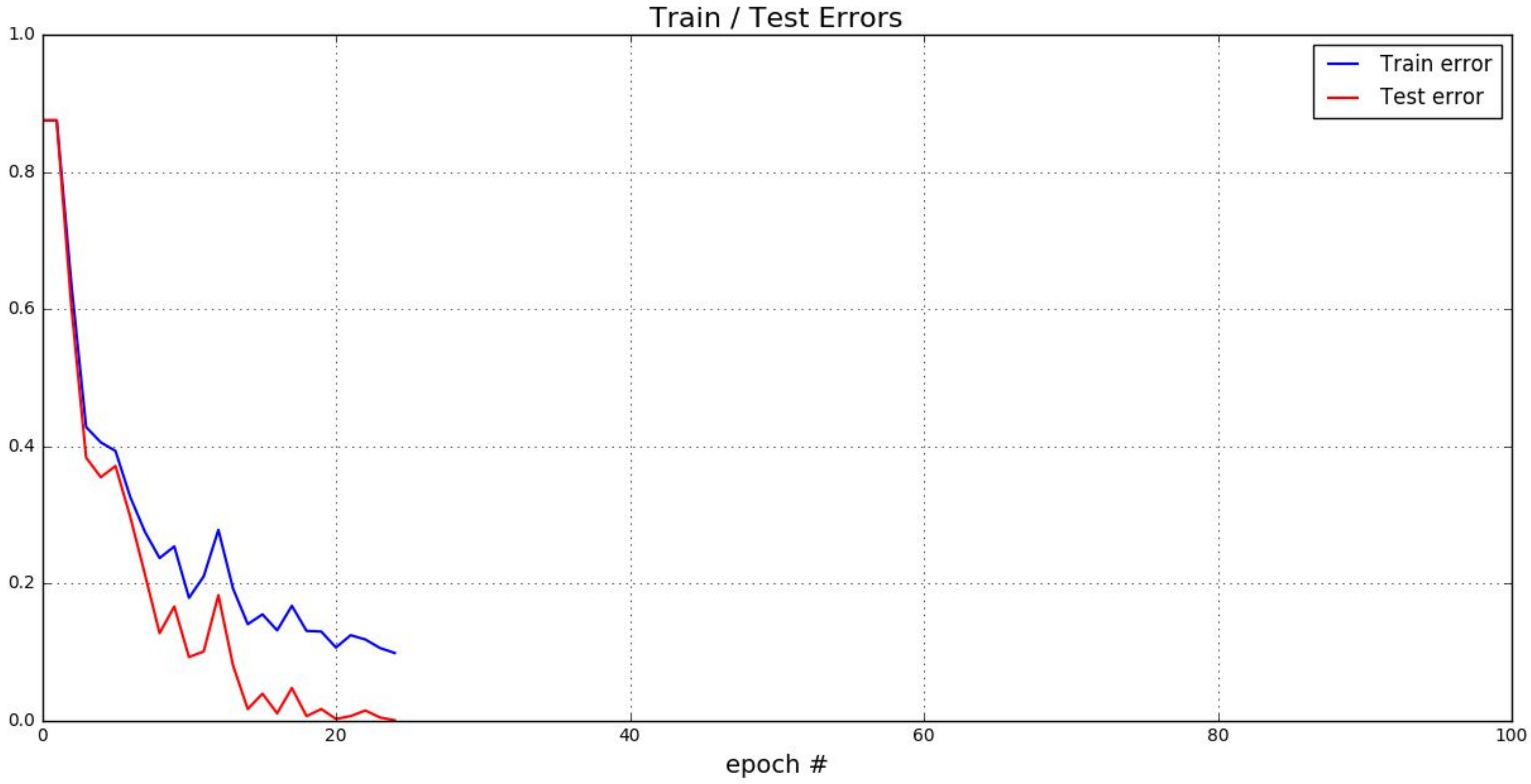
Experiments. XNOR-Nets

Baseline test accuracy: 98.77%, **BWN** test accuracy: 100%, **XNOR-Net** test accuracy: 97.34%



Experiments. XNOR-Nets

Baseline test accuracy: 98.77%, **BWN** test accuracy: 100%, **XNOR-Net** test accuracy: 97.34%



Experiments. XNOR-Nets

Baseline test accuracy: 98.77%, **BWN** test accuracy: 100%, **XNOR-Net** test accuracy: 97.34%

Experiments. Scores

	Training speedup	Memory reduction while inference	Inference speedup	Accuracy gain	Parameters reduction
HashedNets	-	?	?	0.41%	?
CP-Decomposition	N/A	0% / −2.28%	-	-4.6% (*similarity*)	0.0099%
TensorNet	?	?	?	?	?
Binary-Weight-Network	4x	0%	0% / -3.2%	1.23%	0
XNOR-Net	+	2.4%	-1.8% / -0.1%	-1.43%	0.0008%
SqueezeNet	?	?	?	?	?