CNN Quantization

Performance evaluation

Emanuele Ghelfi   Emiliano Gagliardi

June 17, 2017

Politecnico di Milano

Contents

Project purpose

Using a machine learning framework with support for convolutional neural networks

- Define different kind of networks
- Train
- Quantize
- Evaluate the original and the quantized models
- Make a comparison in term of size of the model, cache misses, and inference time

What is quantization?

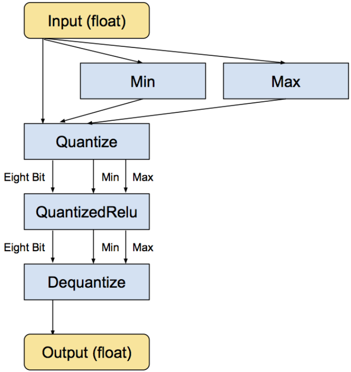
Developing this project we saw two different approaches:

- Tensorflow
- Caffe Ristretto

Tensorflow quantization

Unsupervised approach

- Get a trained network
- Obtain for each layer the min and the max of the weights value
- Represent the weights distributed linearly between the minimum and maximum with 8 bits precision
- The operations have to be reimplemented for the 8-bit format



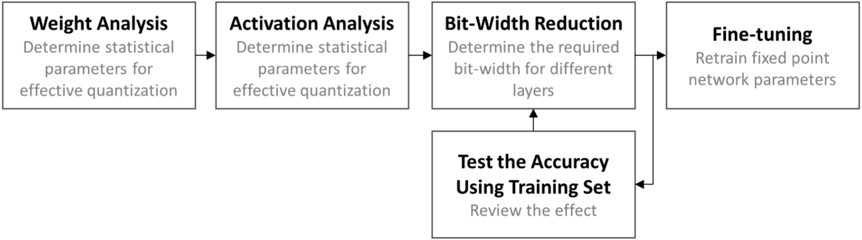
The resulting data structure is composed by an array containing the quantized value, and the two float min and max

Caffe Ristretto quantization

Supervised approach

- Get a trained network
- Three different methods:
  - Dynamic fixed point: a modified fixed-point format
  - Minifloat: bit-width reduced floating point numbers
  - Power of two: layers with power-of-two parameters don't need any multipliers, when implemented in hardware
- Evaluate the performance of the network during quantization in order to keep the accuracy higher than a given threshold
- Support for training of quantized networks (fine-tuning)

Caffè Ristretto quantization



Why quantize?

Accuracy/inference speed trade-off

- Deep networks tend to cope very well with high levels of noise in their inputs.
- It seems that there is no need of floating point precision
- Training still needs floating point precision to work, it is an iteration of little incremental adjustments of the weights

So deep networks are trained with floating point precision, then a quantization algorithm can be applied to obtain smaller models and speed up the inference phase

Caffe ristretto

The results obtained with Ristretto on a simple network for the Mnist dataset are not so satisfying...

network	accuracy	model size (MB)	Time (ms)	LL_d misses (10^6)	L1_d misses (10^6)
Orginal	0.9903	1.7	29.2	32.098	277.189
Dynamic f. p.	0.9829	1.7	126.41	42.077	303.209
Minifloat	0.9916	1.7	29.5	37.149	282.396
Power of two	0.9899	1.7	61.1	35.774	280.819

Linux running on macbook pro, cachegrind tool for cache statistics. Intel i5 2.9 GHz, L3 cache 3MB, 16 GB ram.

- The quantized values are stored in float size after the quantization
- The quantized layers implementation works with float variables:
  - perform the computation with low precision values stored in float variables
  - quantize the results, still stored in float variables

Tensorflow

The quantization is better supported

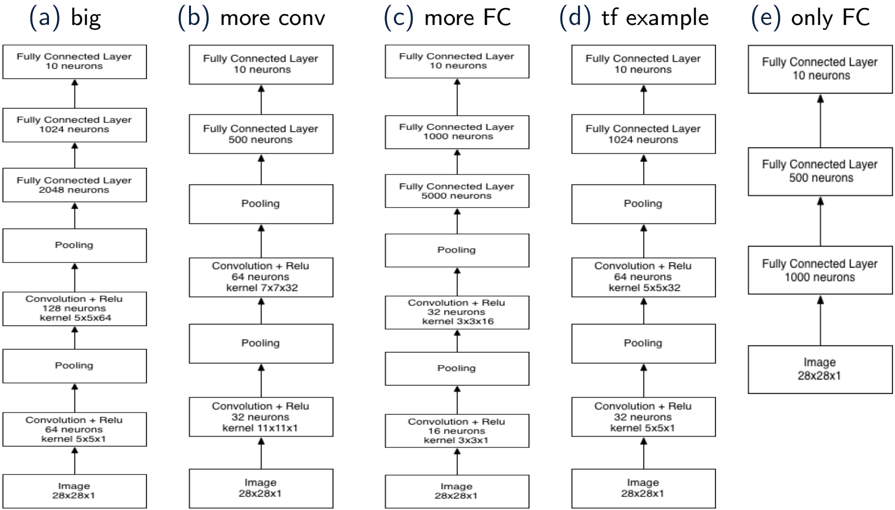
- The quantized model is stored with low precision weights
- Some low precision operations are already implemented

We tried different topologies of networks, to see how quantization affect different architectures

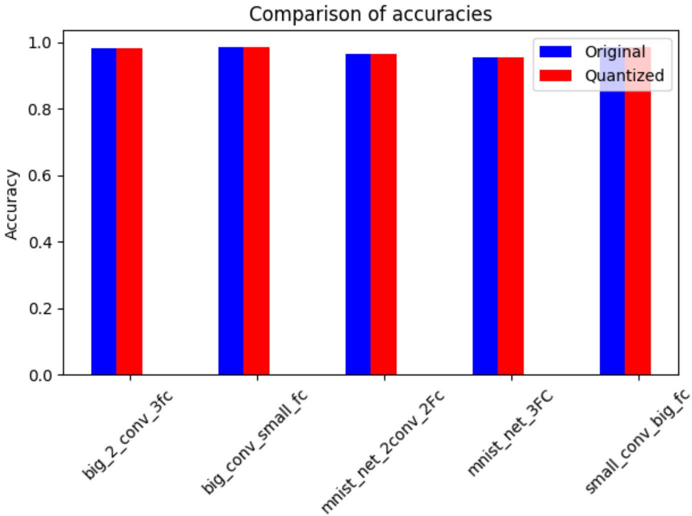
How we used the tensorflow quantization tool

- We used python (with a bit OO, since we needed a way to use it with different networks)
- An abstract class defines the pattern of the network that the main script can handle
- The core methods of the pattern are
  - prepare: build the computational graph of the network and the training step, load the data
  - train: iterate the train step
- The main script takes in input an instance and:
  - calls prepare and train
  - quantizes the obtained network
  - evaluates the accuracy
  - evaluates cache performance using linux-perf
  - plots the data

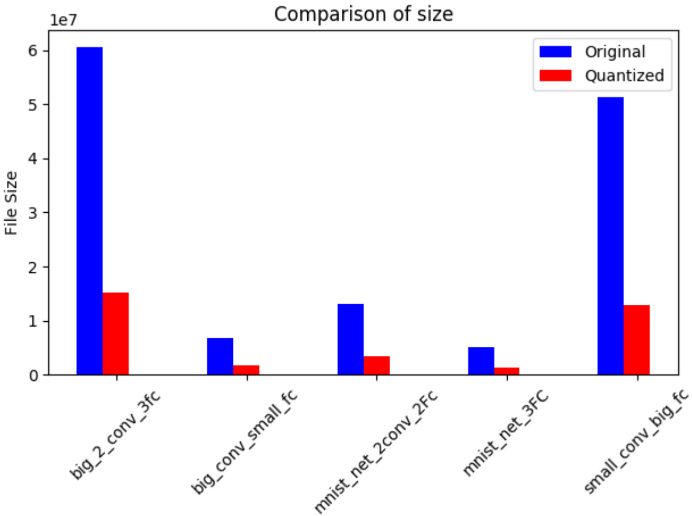
Topologies


(a) big
Fully Connected Layer 10 neurons
Fully Connected Layer 1024 neurons
Fully Connected Layer 2048 neurons
Pooling
Convolution + Relu 128 neurons kernel 5x5x64
Pooling
Convolution + Relu 64 neurons kernel 5x5x1
Image 28x28x1
(b) more conv
Fully Connected Layer 10 neurons
Fully Connected Layer 500 neurons
Pooling
Convolution + Relu 64 neurons kernel 7x7x32
Pooling
Convolution + Relu 32 neurons kernel 11x11x1
Image 28x28x1
(c) more FC
Fully Connected Layer 10 neurons
Fully Connected Layer 1000 neurons
Fully Connected Layer 5000 neurons
Pooling
Convolution + Relu 32 neurons kernel 3x3x16
Pooling
Convolution + Relu 16 neurons kernel 3x3x1
Image 28x28x1
(d) tf example
Fully Connected Layer 10 neurons
Fully Connected Layer 1024 neurons
Pooling
Convolution + Relu 64 neurons kernel 5x5x32
Pooling
Convolution + Relu 32 neurons kernel 5x5x1
Image 28x28x1
(e) only FC
Fully Connected Layer 10 neurons
Fully Connected Layer 500 neurons
Fully Connected Layer 1000 neurons
Image 28x28x1

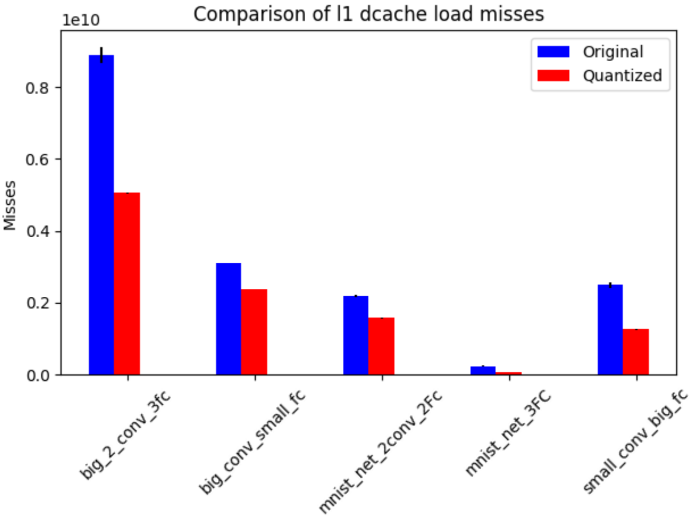
Some data - accuracy



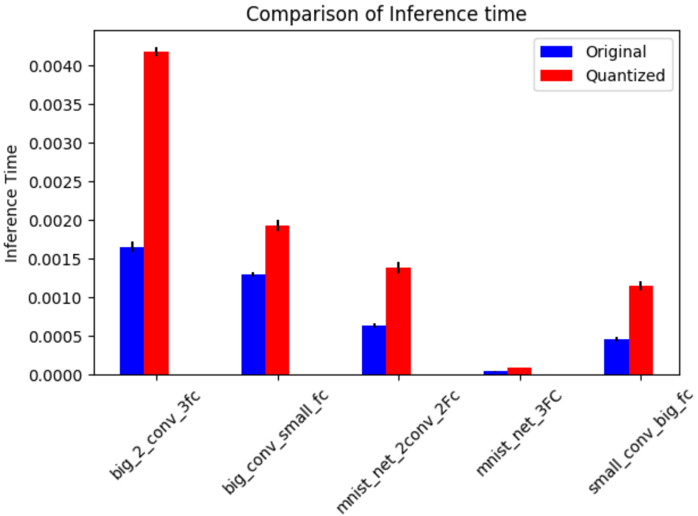
Some data - models size



Some data - data cache misses



Some data - inference time



Why is the inference time worst?

- We see an improvement in performance only for the size of the model, and so for the data cache misses
- Inference time and last level cache misses are worst in quantized networks

From the tensorflow github page:
Only a subset of ops are supported, and on many platforms the quantized code may actually be slower than the float equivalents, but this is a way of increasing performance substantially when all the circumstances are right.

Original net - tensorflow benchmark tool

[Node type]	[count]	[avg ms]	[avg %]
Conv2D	2	62.098	73.109%
MatMul	2	16.089	18.942%
Add	4	4.248	5.001%
MaxPool	2	1.453	1.711%
Relu	3	1.006	1.184%
Const	10	0.022	0.026%
Reshape	2	0.008	0.009%
_Retval	1	0.004	0.005%
_Arg	1	0.004	0.005%
NoOp	1	0.004	0.005%
Identity	1	0.003	0.004%

Quantized net - tensorflow benchmark tool

[Node type]	[count]	[avg ms]	[avg %]
QuantizedConv2D	2	637.514	74.337%
QuantizedMatMul	2	188.589	21.990%
QuantizeV2	4	7.491	0.873%
RequantizationRange	4	5.128	0.598%
Dequantize	4	4.476	0.522%
Add	4	4.296	0.501%
Requantize	4	3.801	0.443%
QuantizedMaxPool	2	2.203	0.257%
QuantizedRelu	3	1.398	0.163%
Min	4	1.241	0.145%
Max	4	1.224	0.143%
NoOp	1	0.147	0.017%
Const	20	0.042	0.005%
Reshape	4	0.020	0.002%
QuantizedReshape	2	0.014	0.002%
_Arg	1	0.007	0.001%
_Retval	1	0.005	0.001%

References

- Tensorflow: https://www.tensorflow.org
- Ristretto: http://lepsucd.com/?page_id=621
- Github repository of the project: https://github.com/EmilianoGagliardiEmanueleGhelfi/CNN-compression-performance