CNN Quantization

Performance evaluation

Emanuele Ghelfi Emiliano Gagliardi

June 18, 2017

Politecnico di Milano

Contents

1 Purposes

2 Quantization

3 Our work

Problem

Accuracy/inference speed trade-off

- For real world application, convolutional neural network(CNN) model can take more than 100MB of space and can be computationally too expensive
- How to enable embedded devices like smart phones with the power of Neural Networks?
- There is the need of floating point precision?
  No, deep neural network tends to cope well with noise in their input
- Training still needs floating point precision to work, it is an iteration of little incremental adjustments of the weights

Solution

The solution is quantization.
Deep networks can be trained with floating point precision, then a quantization algorithm can be applied to obtain smaller models and speed up the inference phase reducing memory requirements:

- Fixed-point compute units are typically faster and consume far less hardware resources and power than floating-point engines
- Low-precision data representation reduces the memory footprint, enabling larger models to fit within the given memory capacity

Project purpose

Using a machine learning framework with support for convolutional neural networks:

- Define different kind of network
- Train
- Quantize
- Evaluate the original and the quantized models
- Compare them in terms of model size, cache misses, and inference time

What is quantization?

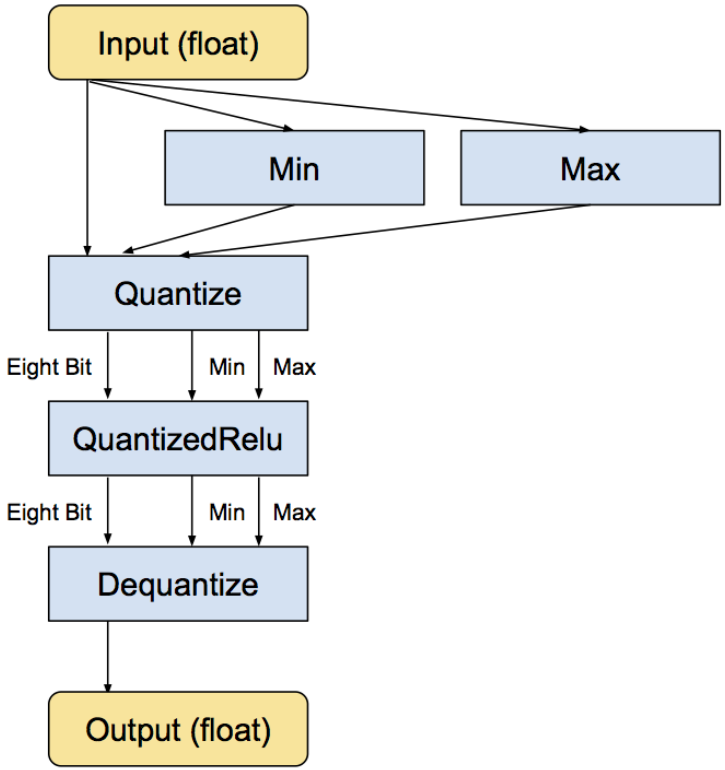
Developing this project we saw two different approaches:

- 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



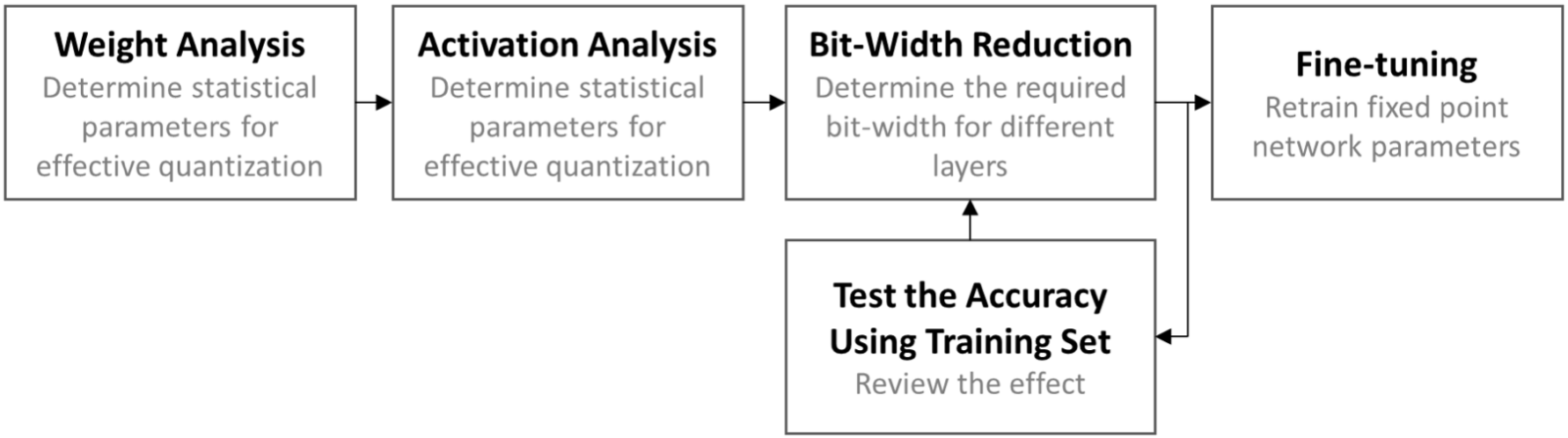
The resulting data structure is composed by an array containing the quantized value, and the two float min and max

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



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

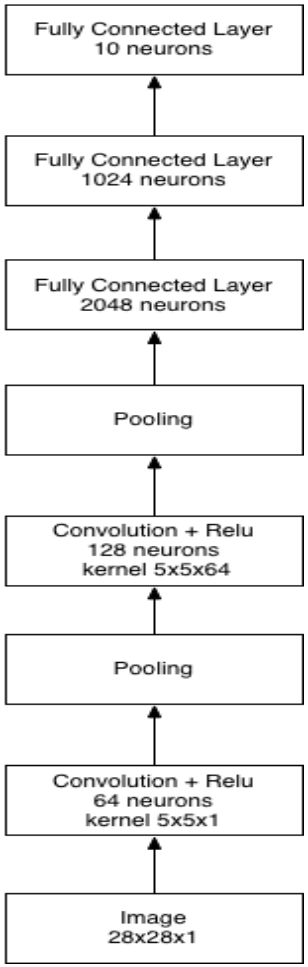
We tried different topologies of networks, to see how quantization affects different architectures

How we used the tensorflow quantization tool

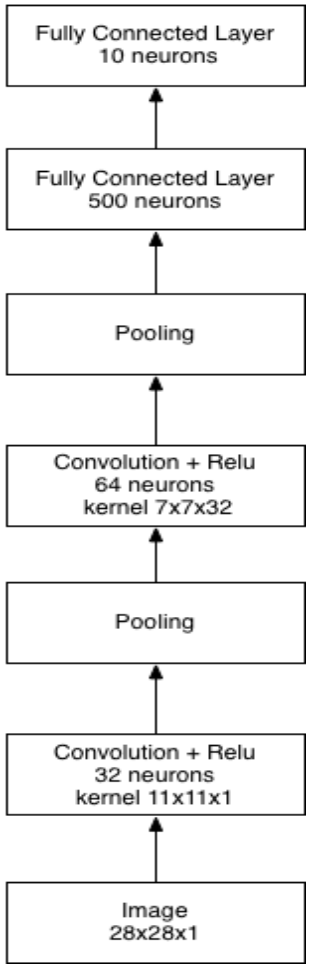
- We used python (with a bit OO, since we needed a way to use it with different networks)
- An abstract class defines the pattern of the network that the main script can handle
- The core methods of the pattern are
  - prepare: load the data and build the computational graph and the training step of the network
  - 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 on MNIST

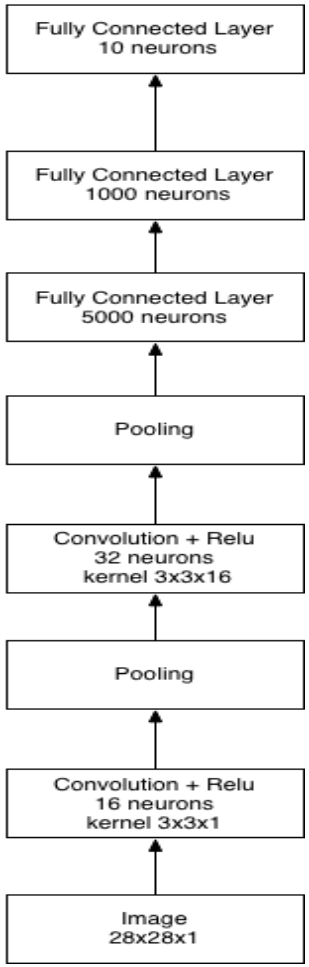
(a) Big



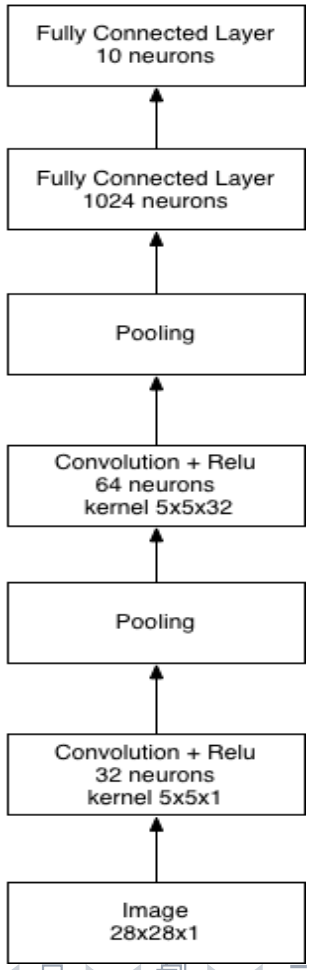
(b) More convolutional



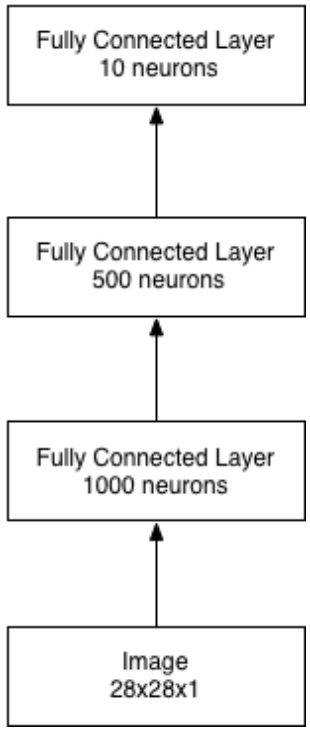
(c) More FC



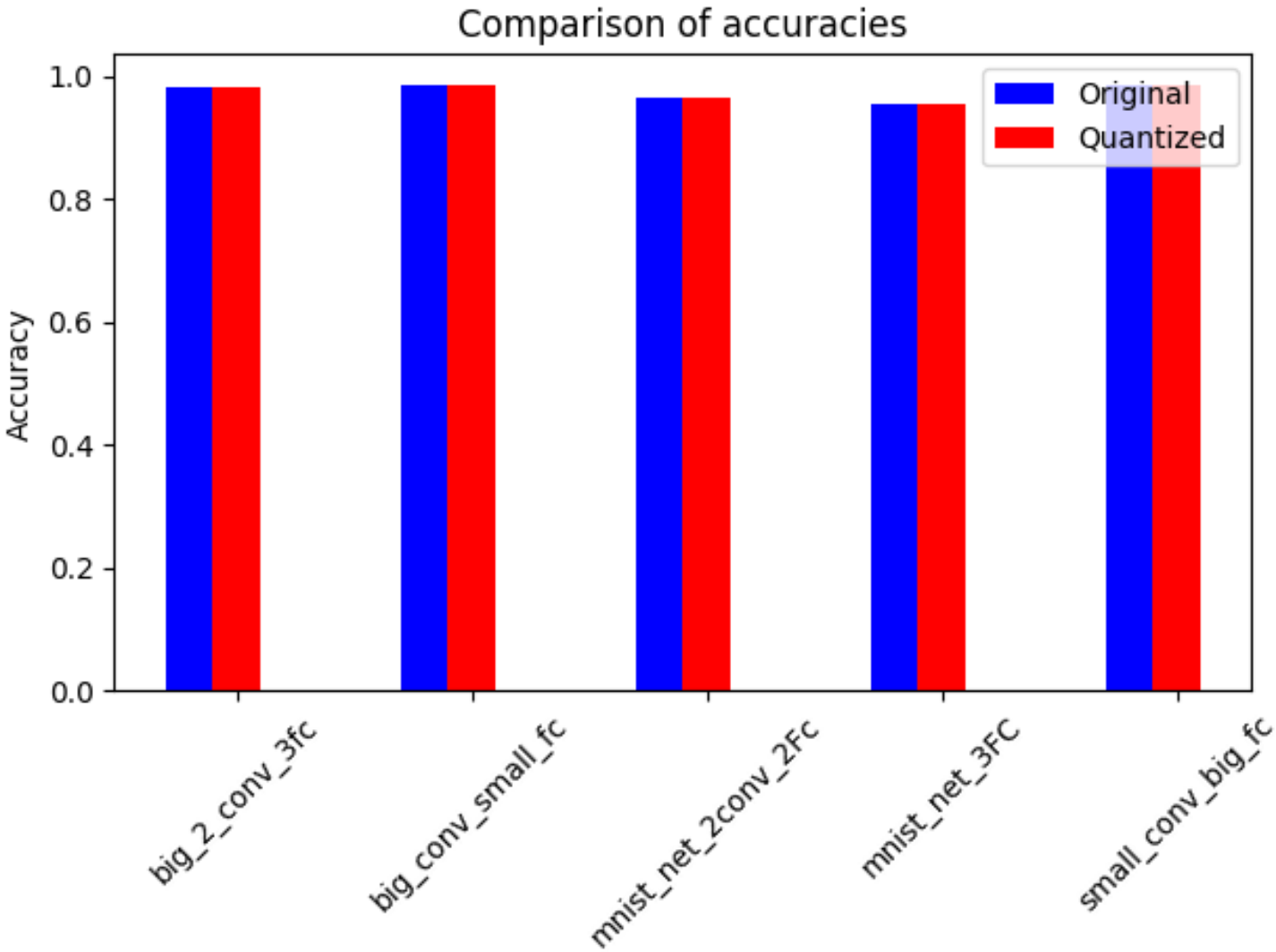
(d) Tf Example



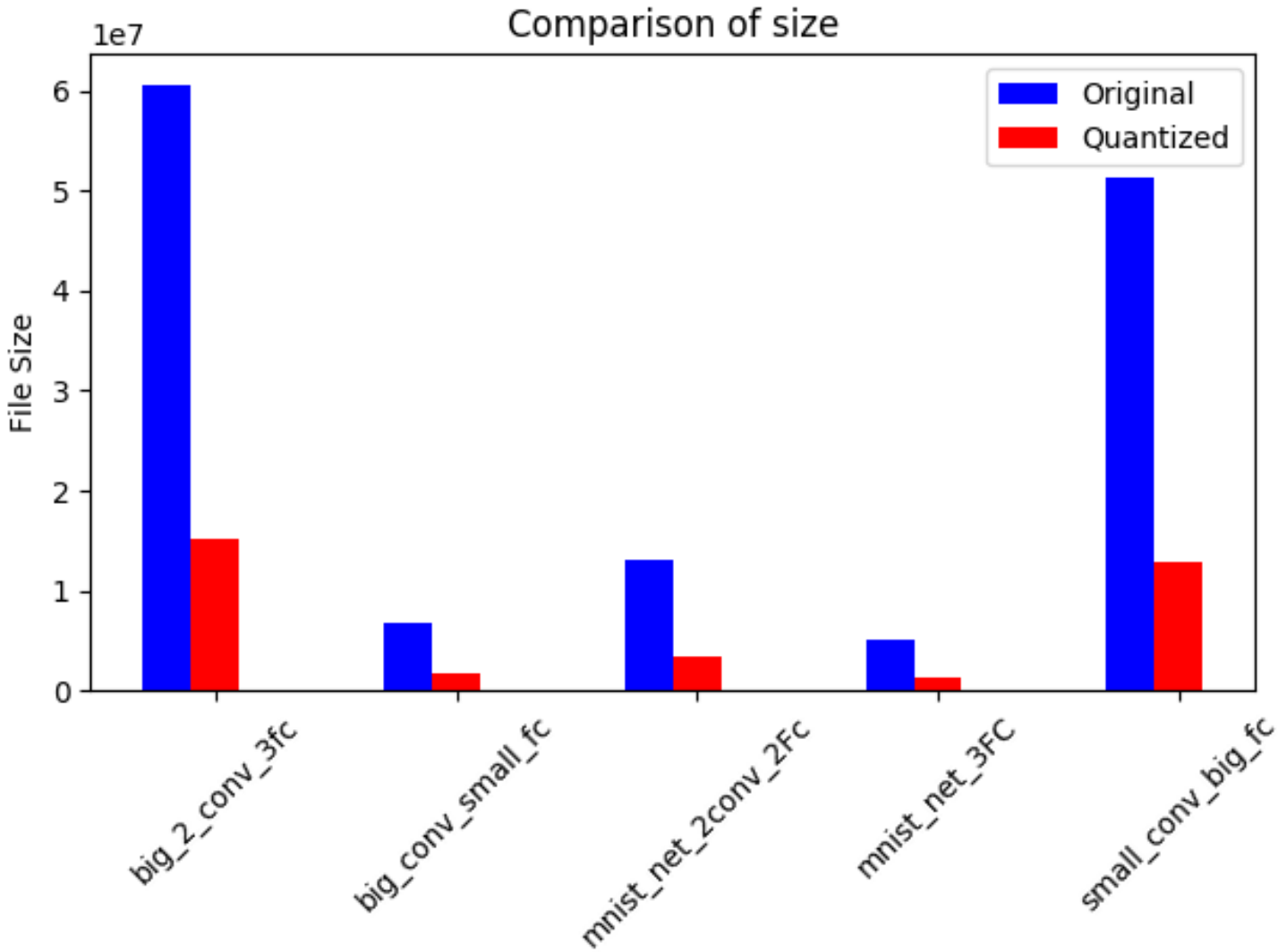
(e) Only FC



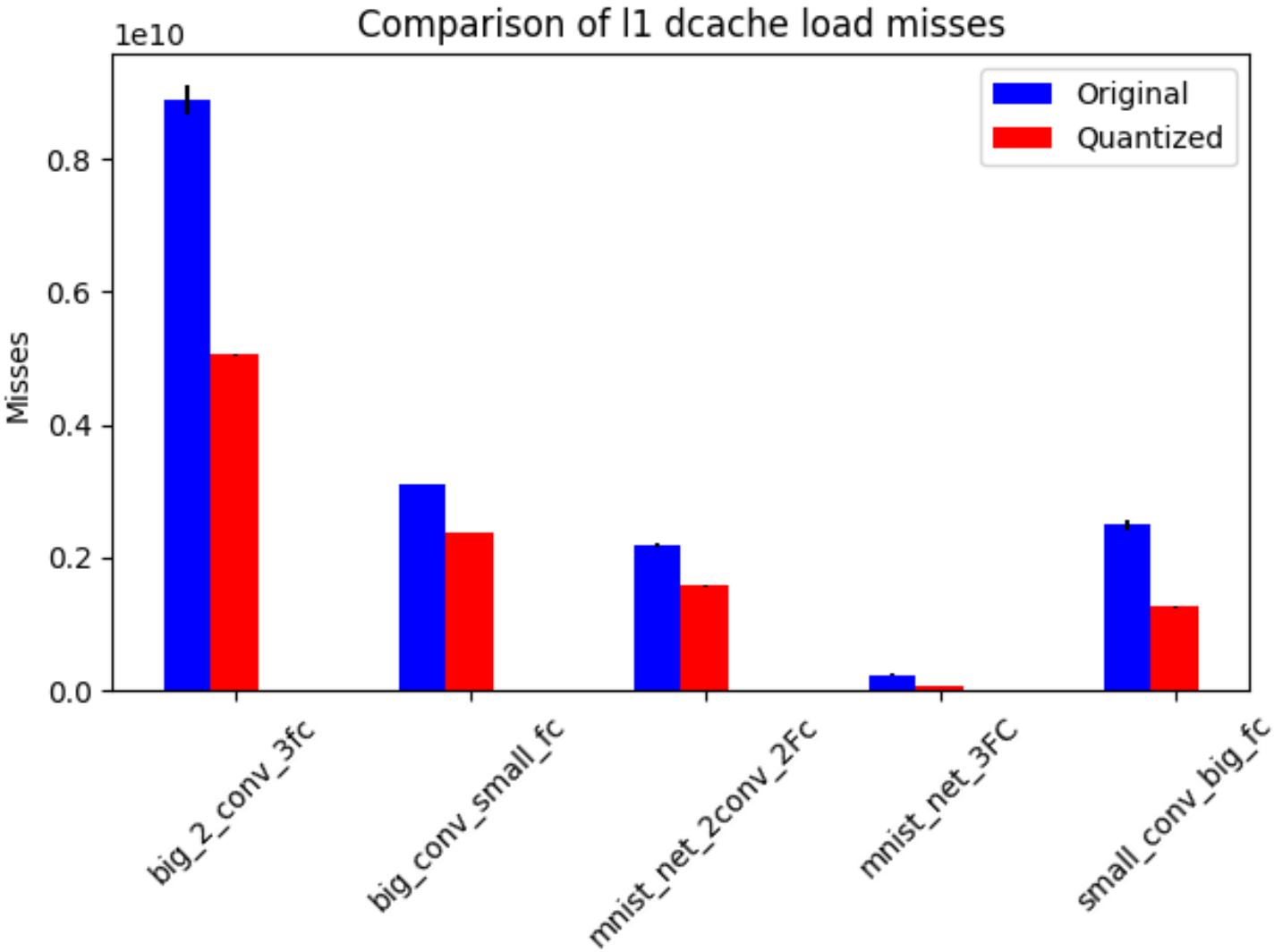
Some data on MNIST - accuracy



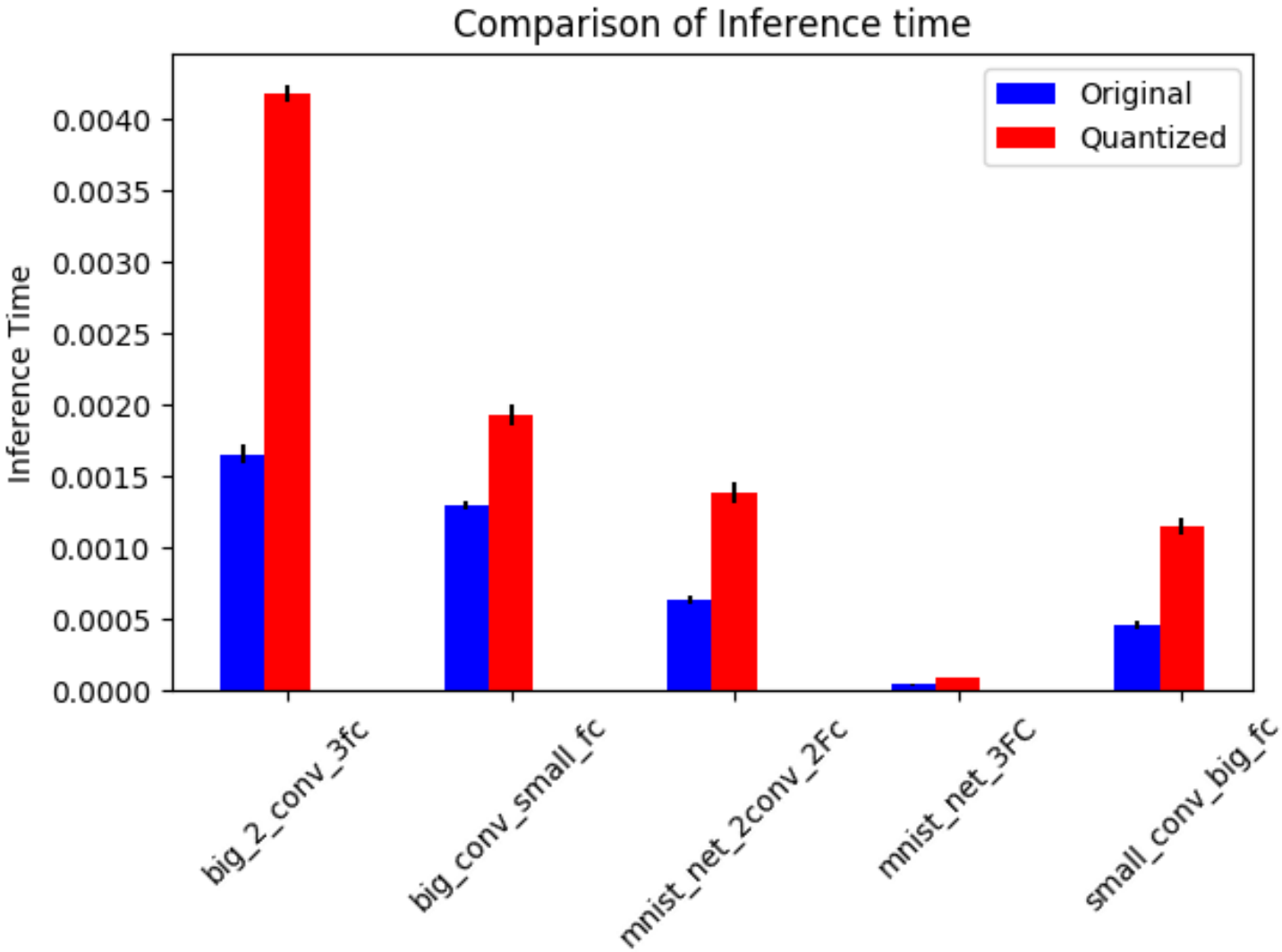
Some data on MNIST - models size



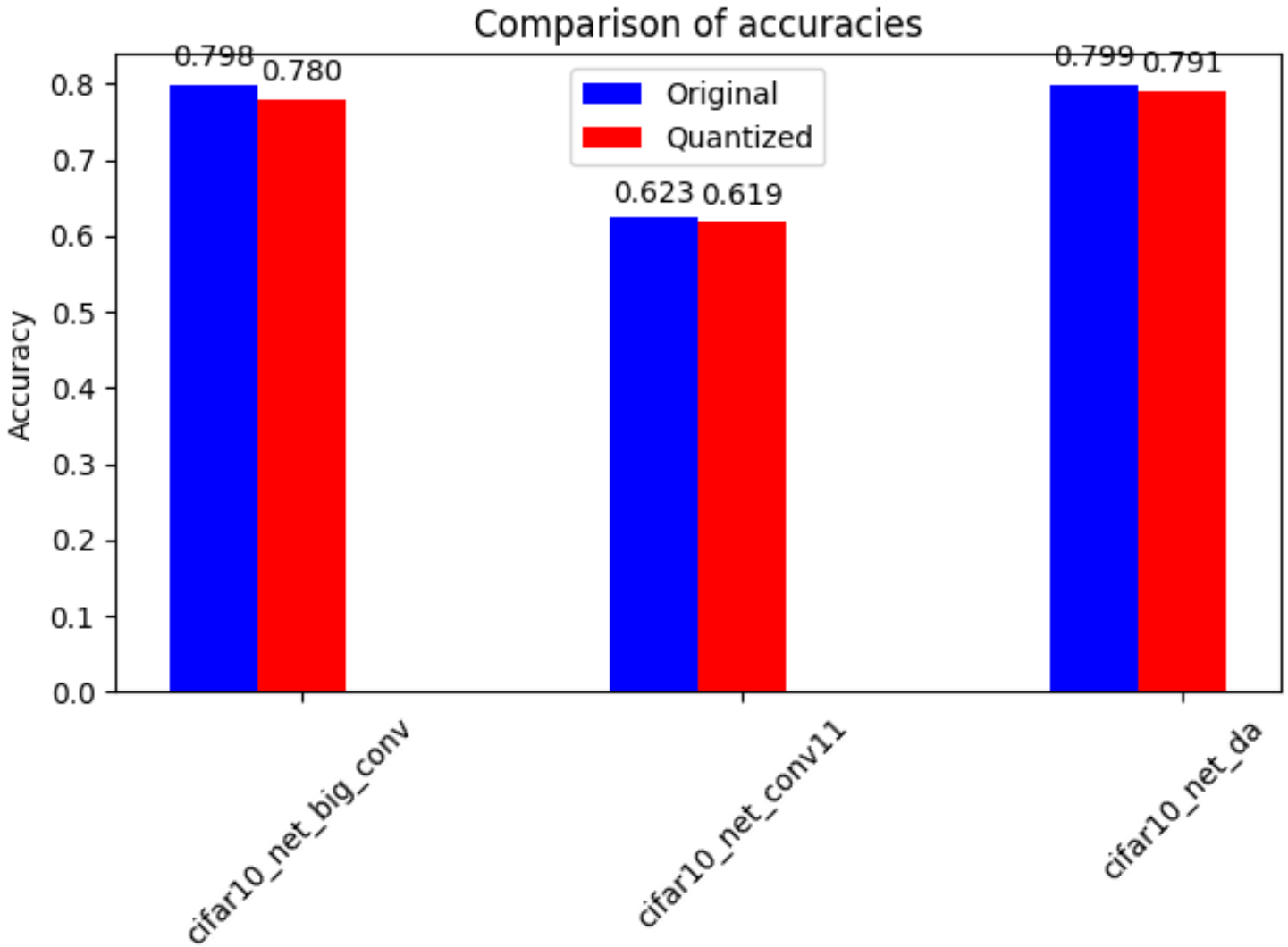
Some data on MNIST - data cache misses



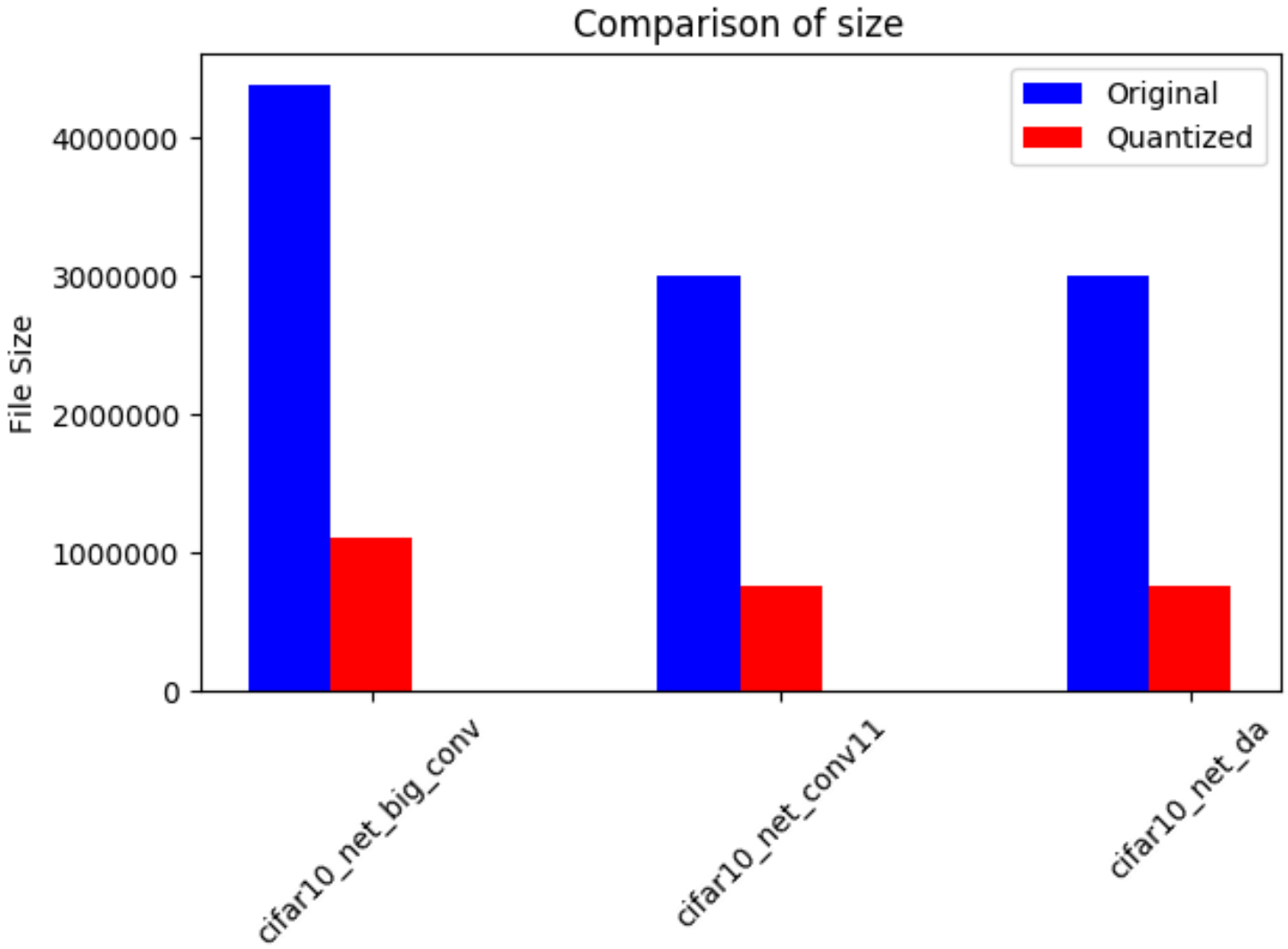
Some data on MNIST - inference time



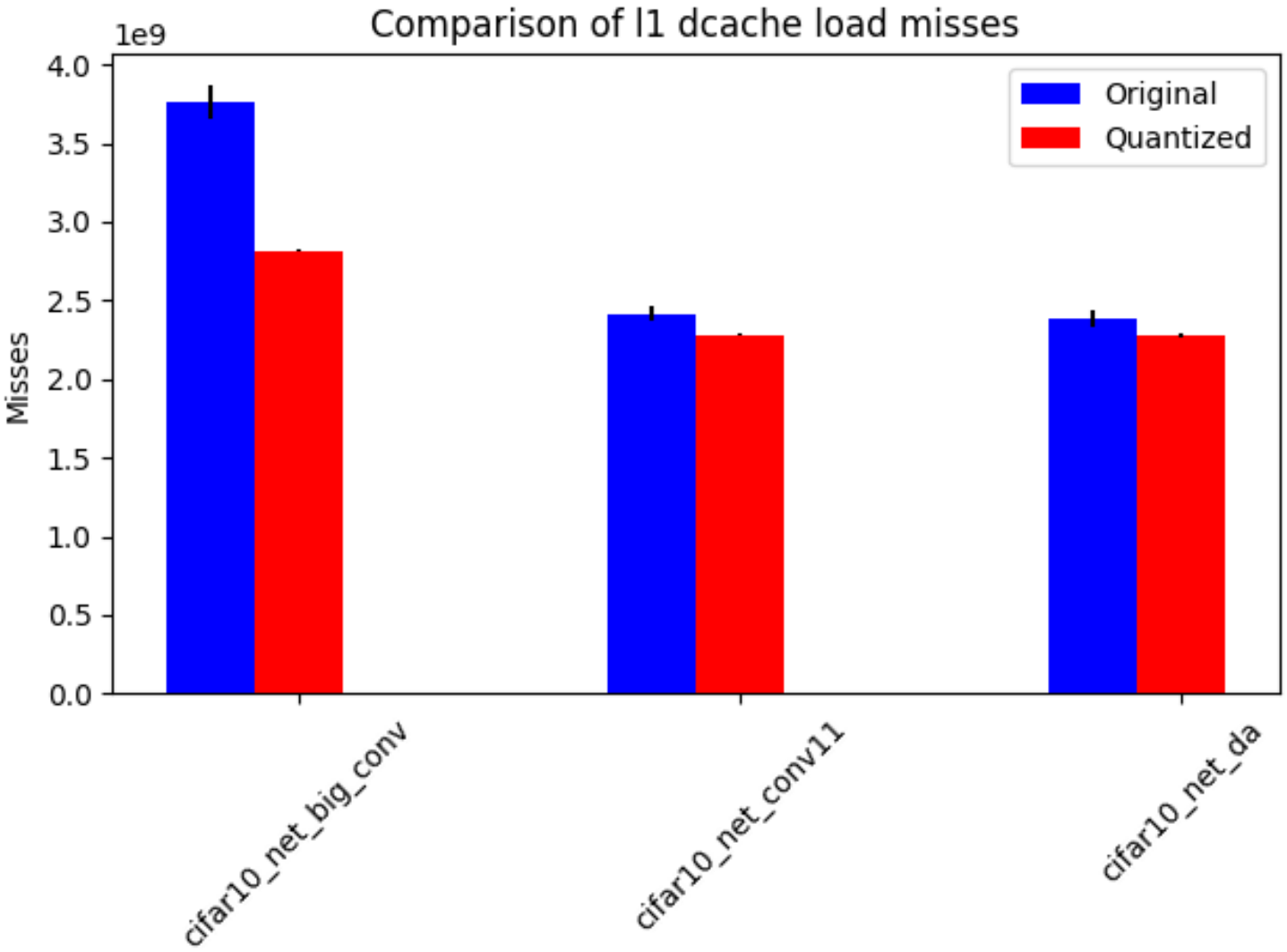
Some data on CIFAR10 - accuracy



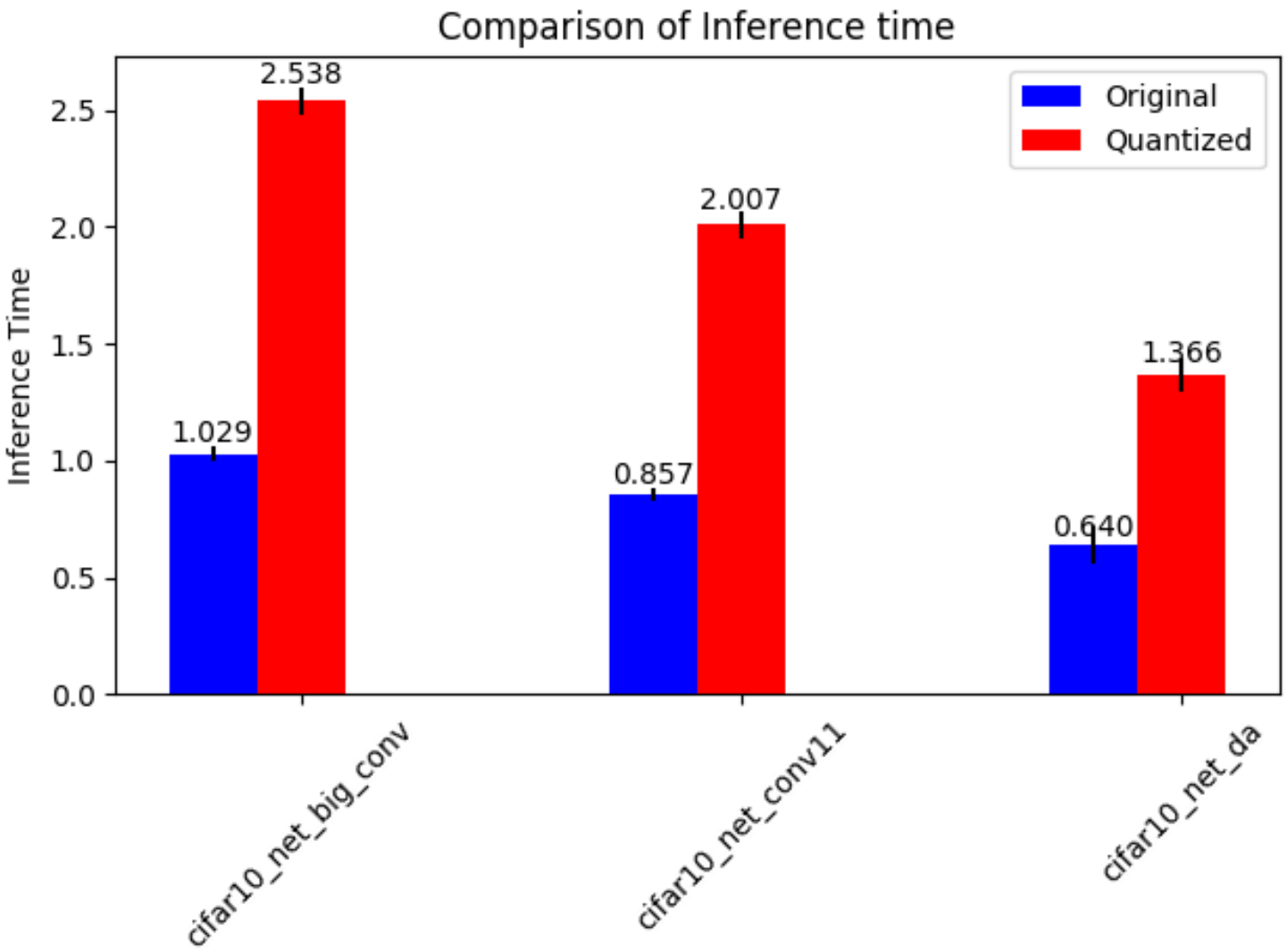
Some data on CIFAR10 - models size



Some data on CIFAR10 - data cache misses



Some data on CIFAR10 - 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

Original network on MNIST dataset:

[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

Quantized network on MNIST dataset:

[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 quantization: https://www.tensorflow.org/performance/quantization https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tools/graph_transforms
- Ristretto: http://lepsucd.com/?page_id=621
- Github repository of the project: https://github.com/EmilianoGagliardiEmanueleGhelfi/CNN-compression-performance
- Deep Learning with Limited Numerical Precision (Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan)