A Library for Convolutional Neural Network Design

Petr Rek

Supervisor: prof. Ing. Lukáš Sekanina, Ph.D.

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Problem statement

The rapid progress in AI technologies based on deep and convolutional neural networks (CNN) has led to an enormous interest in efficient implementations of neural networks in low power embedded devices. The goal of this diploma thesis was to develop a new library for Convolutional Neural Networks (CNN) that allows the users to evaluate effects of using different data types and/or operations on CNN. The library, called TypeCNN, was developed and tested on commonly used data sets. Furthermore, this work also includes experiments with CNN based on fixed point representation.

TypeCNN library features

Convolutional neural network

- Layers – Convolutional, Pooling (average and max), Fully connected, Drop-out, Activation,
- Activation functions - ReLU, Leaky ReLU, Sigmoid, Tanh, Softmax,
- Loss functions – Mean squared error, Cross-entropy,
- Optimizers – SGD, Momentum, Nestorov momentum, Adagrad, Adam.

Additional features

- XML schema for CNN specification,
- Persistency module (loading and saving trained CNN),
- Both CLI and API,
- Parsers for different formats (IDX, PNG, Binary),
- Support for up to three data types in each layer, independent between layers,
- FixedPoint data type – fixed point representation with configurable bit length of integer and fractional parts.

Data type independence

The library contains three independent data type aliases:

- **WeightType** – data type used for weights during inference and when saving to a disk,
- **ForwardType** – data type used for all the computations done during inference,
- **BackwardType** – data type used while updating learnable parameters (gradients, precise weights, …).

Any data type that supports a set of predefined operations (arithmetical, logical, …) may be used, including user defined data types. That means that the user can also redefine these operations and support , for example, approximate operators.

As a demonstration, the TypeCNN library contains fixed point data type **FixedPoint<F, P>**, where F is the amount of bits for the integer part and P is the amount of bits for the fractional part. This representation is beneficial wherever speed and low power consumption is important. However it introduces problems such as overflow. It is suggested to first train the network using floating point data type, then convert all relevant values to the fixed point data type and fine-tune for a few epochs.

Replacing 32-bit floating point by 8-bit fixed point leads to a speedup of over 3 times in performing the MAC (multiply-accumulate) operation and simultaneously decreases its power consumption by over 30 times *[1]*.

[1] Gysel, P.; Pimentel, J.; aj.: Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks. IEEE Transactions on Neural Networks and Learning Systems, 2018

Results

Common data sets

The library was tested on the MNIST data set (accuracy of 99.37%) and the CIFAR-10 data set (accuracy of 73.59%). These results are acceptable with respect to the CNN architectures used.

Comparison with other libraries

The TypeCNN library was compared to other publicly available libraries. It performs well in training performance, but is slower, than those widely used. See table below.

Approximation using FixedPoint data type

It was shown that both NN and CNN can be used with representation included in this library. 16 or more bits are sufficient for CNN to perform with close to zero loss in precision. Lower bit widths can be used, however it depends on the task given and the architecture chosen. See experiments below.

Data type	Before retraining [%]	After retraining [%]
Double	98.60	99.17
Float	98.60	99.17
FixedPoint<16,16>	86.37	99.15
FixedPoint<8,8>	86.91	99.13
FixedPoint<4,4>	10.58	79.59

Table 1: Effects on CNN when trained using floating point data type for 10 epochs and then fine-tuned using given data type for 5 epochs

Data type	Before retraining [%]
Double	98.57
Float	98.62
FixedPoint<16,16>	98.79
FixedPoint<8,8>	98.67
FixedPoint<4,4>	64.38

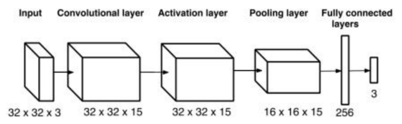
Table 2: Effects on CNN when trained using given data type from the beginning for 10 epochs

Data type	Before retraining [%]	After retraining [%]
FixedPoint<8,8>	86.91	99.13
FixedPoint<4,4>	81.97	98.97
FixedPoint<1,3>	31.85	98.61
FixedPoint<2,2>	10.05	97.67
FixedPoint<1,1>	9.80	93.77

Table 3: Effects on CNN using FixedPoint<8,8> as ForwardType and given data type as WeightType, trained on floating point for 10 epochs and fine-tuned for 5 epochs with these settings

Library	Training time [s]	Accuracy [%]
SimpleCNN	746	97.20
TinyDNN	151	98.23
Keras (TensorFlow)	744	98.34
TypeCNN	538	98.31

Table 4: Comparison of libraries using the same training settings


Input
Convolutional layer
Activation layer
Pooling layer
Fully connected layers
32 x 32 x 3
32 x 32 x 15
32 x 32 x 15
16 x 16 x 15
256
3