Deep Neural Network Compiler and Inference Framework for microcomputers and microcontrollers

Gunjan Nandy
Machine Learning Engineer Intern
AI Tech System
gunjan.nandy1@gmail.com

Nikhil Tiwari
Machine Learning Engineer Intern
AI Tech System
nikhilcbse97@gmail.com

Hrishikesh Murali
Machine Learning Engineer Intern
AI Tech System
hrishi98.m@gmail.com

Subham Sarkar
Machine Learning Engineer Intern
AI Tech System
kingsubham27@gmail.com

Nalin Shani
Machine Learning Engineer Intern
AI Tech System
nalinshani14@gmail.com

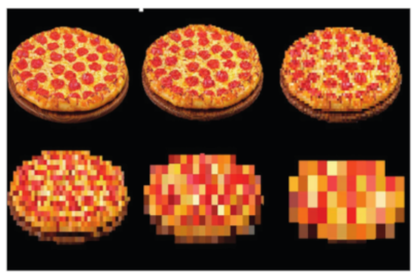
Vishal Yadav
Machine Learning Engineer Intern
AI Tech System
vvyy346@gmail.com

Summary

By bringing deep learning models to tiny microcontrollers, we can boost the intelligence of billions of devices that we use in our lives, without relying on expensive hardware or reliable internet connections. Imagine smart appliances that can adapt to your daily routine, intelligent industrial sensors that understand the difference between problems and normal operation, and magical toys that can help kids learn in fun and delightful ways.

Important Aspects of the Deep Neural Network Compiler

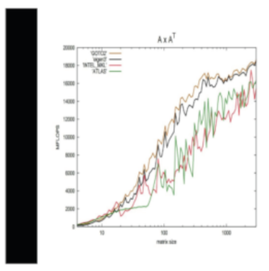
Operator Fussion



Quantization

Adding new operator is very easy which will help in improving the compiler with time

Scalability

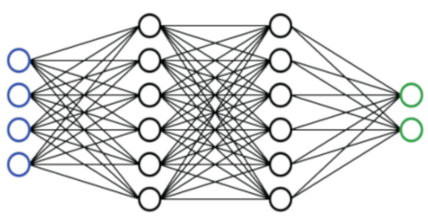


Memory Management

Design

Step 1.

We take Neural Network as an input



Where you can use multiple platforms like Tensorflow or pytorch etc. to develop your neural network.

PYTORCH TensorFlow Caffe2 mxnet

Step 2.

Then the Neural Network is sent to ONNX 3.0

Our compiler uses ONNX as a protobuf format, which is directly translated into a high-level compute graph with operators as nodes and data flowing through these nodes as tensors.

Step 3.

Followed by our operators and Graphs Optimizations

Operators

Graph Optimizations

These operators are being written in C++ for performance reasons. They are also ported to python interface for quick testing and possibility of tuning compiler in to a full framework with python interface.

Step 4.

Main difference between neural networks computation graph and traditional compiler IR graph is the number of high-level operators. Traditional compiler has a small and fixed set of operators, whereas neural networks computation graph operators are over 200 and new are being added every day. This dictates our design choice of adding new operators decentralized and independent of overall compiler.

LLVM Bleeding Edge Compiler Technology

NVIDIA CUDA

OpenCL

Step 5.

and more...

The compiler turns neural networks into an executable bundled with model parameters ready to run on embedded devices such as the raspberry pi, odroid, arduino, risc-V and other controllers with small form factor.

Technologies Used

The Eigen project started when some hackers from the large KDE meta-project realized the need for a single unified matrix library.

ONNX enables developers to share models independet of the framework they use.

ONNX

Future Work

We are dedicated in adding more and more operators to the compiler so that it can achieve state of the art performance in terms of speed and efficiency.

References

Rohit Sharma et. Al, DNNC: Deep Neural Network Compiler, 2019

Guennebaud, G., Jacob, B., et al.: Eigen v3 (2010).

Bai, Junjie and Lu, Fang and Zhang, Ke and others, ONNX 3.0: 2019

Do have a look at our work on GITHUB bit.ly/dnnCompiler

Feel free to have a look at the notebook

Want to see our work live ?