An Efficient Halide Framework for Fixed-Point CNNs Targeting ARM Cores

Background
Convolutional Neural Networks (CNNs) are being widely used and deployed in practical applications in domains such as computer vision, natural language processing, and recommendation systems. In particular, CNNs achieve outstanding image object recognition accuracy, setting new records for object detection and classification competitions every year. However, the high accuracy of CNNs comes with a high computational cost, and a high-power consumption as well. A hand optimization to a specific CNN is possible. However, it is usually tedious, time consuming, and case by case. Because of this there is a great need for an efficient and easy-to-use framework to “automatically” optimize CNNs to a given target platform.

Halide [1] is a programming language designed to make it easier to write high-performance image processing code on modern machines. It achieves this by splitting the application code into a functional part and a scheduling part. The functional part describes the application's operations, such as convolution or reduction etc. The scheduling part allows the programmer to independently specify the order of the operations. In this way different loop transformation techniques, such as reordering, tiling, and fusion can be applied easily without (accidentally) changing the program's behavior. Halide's current front-end is embedded in C++. Compiler targets include x86/SSE, ARM v7/NEON, CUDA, and OpenCL.

Assignment
In this assignment, given a CNN Caffe model, two main tasks need to be carried out.

1. (a) Generate ARM target code of the given CNN (floating point data type) with the Halide framework, and profile it in both single core and multicore ARM CPUs. (b) Improve the performance by applying loop/data optimization approaches using Halide.
2. (a) Upgrade the framework to support 4-bit/8-bit/16-bit fixed point CNNs. As more and more practical applications are using narrow-bit fixed point implementations, instead of more time/memory demanding floating point implementations. (b) Profile and improve the performance.

Contacts
Luc Waeijen  lj.w.waeijen@tue.nl  ES Group, Flux 4.076
Yifan He  y.he@tue.nl  ES Group

Note: if you find this assignment interesting, we may also extend it to a master-project assignment by adding extra tasks.