Optimization for Convolutional Neural Network(CNN) Implementing Neural Network and Other Language in Halide

1. Introduction: What is Convolutional Neural Network and Halide

What is Convolutional Neural Network(CNN)

CNN is a neural network consisting of mainly convolution layer and pooling layer. A convolutional layer is a layer uses convolution as an operator for a matrix. giving the example below, suppose that you have 5x5 input image and consider convolutional operation by applying the following 3x3 filter to such a matrix. The original input image is multiplied element by element from the upper left to the lower right. The result of this calculation is called a feature map.

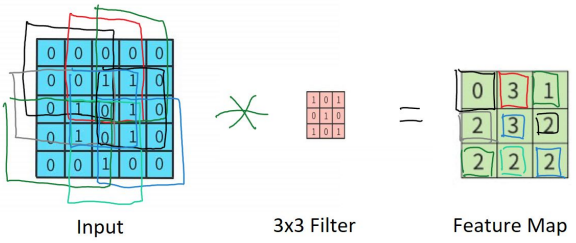

Fig 1.1: Convolution example

Pooling layer is usually applied after convolutional layer. It takes the biggest number from the 2x2 matrix to compresses the information and down-sample it. It has the benefits of being robust against a minute position, suppressing excessive learning, and reducing calculation cost.

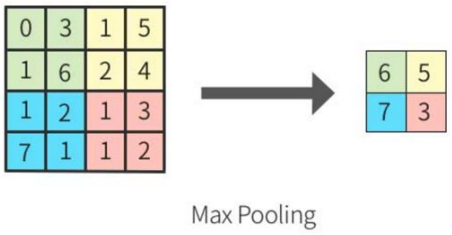

Fig 1.2: Max pooling example

Since CNN was originally studied for handwritten character recognition of postal code, it is often used for image recognition. In natural language processing, it has come to be applied to emotion analysis, text classification, and translation. Giving some actual examples, it has been used by Facebook for facial detection for tagging system, Google for image search and voice recognition, Line/Spotify for recommendation system.

What is Halide

Halide is a programming language dedicated to image processing, that is domain specific language(DSL). Unlike general programming languages such as C++, Python, and Java, DSL is specialized for specific tasks. This language is embedded in C++ so that it runs as if linking libraries and using functions/classes in C++. What is great about Halide is that you can

generate a faster code than the code written by an outstanding person in speeding up image processing, in a simple description.

Here is the blur example. This is the implementation of blur operation in C++, which takes 9.94ms per megapixel.

```cpp
// Normal C++ Blur Operation - 9.94ms per megapixel

void blur(const Image &in, Image &blurred) {
    Image tmp(in.width(), in.height());
    for (int y = 0; y < in.height(); y++)
        for (int x = 0; x < in.width(); x++)
            tmp(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;
    for (int y = 0; y < in.height(); y++)
        for (int x = 0; x < in.width(); x++)
            blurred(x, y) = (tmp(x, y-1) + tmp(x, y) + tmp(x, y+1))/3;
}
```

Fig 1.3: blur operation in C++

If you would like to optimize this code, you will need the following scheduling to speed up 10 times faster. The problem of this scheduling is that if you change the algorithm or hardware, you have to reschedule everything. Also, this optimization can be done by only a professionally skilled coder.

```cpp
// Scheduled C++ Blur Operation - 09ms per megapixel

void fast_blur(const Image &in, Image &blurred) {
 m128i one_third = _mm_set1_epi16(21846);
    #pragma omp parallel for
    for (int yTile = 0; yTile < in.height(); yTile += 32) {
        m128i a, b, c, sum, avg;
        m128i tmp[(256/8)*(32+2)];
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
        m128i *tmpPtr = tmp;
    for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in(xTile, yTile+y));
    for (int x = 0; x < 256; x += 8) {
        a = _mm_loadu_si128(( m128i*)(inPtr-1));
        b = _mm_loadu_si128(( m128i*)(inPtr+1));
        c = _mm_load_si128(( m128i*)(inPtr));
        sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
        avg = _mm_mulhi_epi16(sum, one_third);
        _mm_store_si128(tmpPtr++, avg);
        inPtr += 8;
        }
    }

    tmpPtr = tmp;

    for (int y = 0; y < 32; y++) {
        m128i *outPtr = ( m128i *)(&(blurred(xTile, yTile+y)));
        for (int x = 0; x < 256; x += 8) {
            a = _mm_load_si128(tmpPtr+(2*256)/8);
            b = _mm_load_si128(tmpPtr+256/8);
            c = _mm_load_si128(tmpPtr++);
            sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
            avg = _mm_mulhi_epi16(sum, one_third);
            _mm_store_si128(outPtr++, avg);
            }
        }
    }
    }
}
```

Fig 1.4: blur operation in scheduled C++

This is the Halide code. Not only it is as fast as scheduled C++ code, but it also divides the algorithm and schedule section to simplify the implementation. This separation makes possible easily to schedule based on different hardware, CPU, GPU, and FPGA. For the algorithm part, you only need to write the main calculation part in a simple way. For the scheduling part, it describes the order of calculation for vectorization, loop order, and optimization.

```cpp
//Halide Blur Operation - 0.9ms per megapixel

Func halide_blur(Func in) {
    Func tmp, blurred;
    Var x, y, xi, yi;

// The algorithm
    tmp(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;
    blurred(x, y) = (tmp(x, y-1) + tmp(x, y) + tmp(x, y+1))/3;

// The schedule
    blurred.tile(x, y, xi, yi, 256, 32)
    .vectorize(xi, 8).parallel(y);
    tmp.chunk(x).vectorize(x, 8);
    return blurred;
}
```

Fig 1.5: blue operation in Halide

2. Problem Statement

Although it is true that a lot of libraries are used for a convolutional neural network such as TensorFlow and PyTorch, CNN requires machines to calculate tens of convolutions and poolings. Sometimes, the processing speed of the program in python or other languages are not fast enough to operate those complicated calculations. Our service provides optimized code as a solution.

3. Product/Service: Giving optimized code written in Halide or Other languages

By utilizing Halide and optimizing technique, we provide professional services to realize massive calculation processing in a shorter time. Our company has several possibilities of product/service, a) optimization for CNN in Halide and development for `Halide Code` file, b) own project from open source codes, c) research for new speeding up technology, d)optimizing in other languages.

a) Optimization for CNN in Halide and development for `Halide Code` file

The following flow is the basics of our service. This is one line of a PyTorch code given by a client.

```
m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
```

After receiving codes, we search for the Pytorch website to see how `nn.Conv2d` is defined.

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out_j}, k) \star \text{input}(N_i, k),$$

Then, in the end, we can translate it into the C++ and Halide code below.

```
Func f;
RDom r(0, weight_shape[0], 0, weight_shape[1], 0, weight_shape[2]);
f(c, x, y, n) = sum(r, in(r.x, x*stride - pad + r.y, y*stride - pad + r.z, n) * weight(r.x, r.y, r.z, c)) + bias(c);
```

After implementing in Halide, we will start creating the `Halide Code` file to define all popular operations such as convolution, pooling, binary convolution, rectified linear units, fully connected, and Softmax. The `Halide Code` file functions will be developed as many as our company optimizes the code.

b) Own project from open source codes

Not only optimizing the client's code, but we can also start our own project by using the open source code. As the first step of our project, we will translate YOLO(You Only Look Once) algorithm into Halide. It is an algorithm for image recognition, announced in 2016, which can process in real time with high accuracy. The fast detection system can be used by a surveillance camera, drone camera, and facial detection system for tagging so we can productize the fast YOLO software for those purposes. The YOLO processing is completed in one CNN network so learning is relatively easy to translate for us.

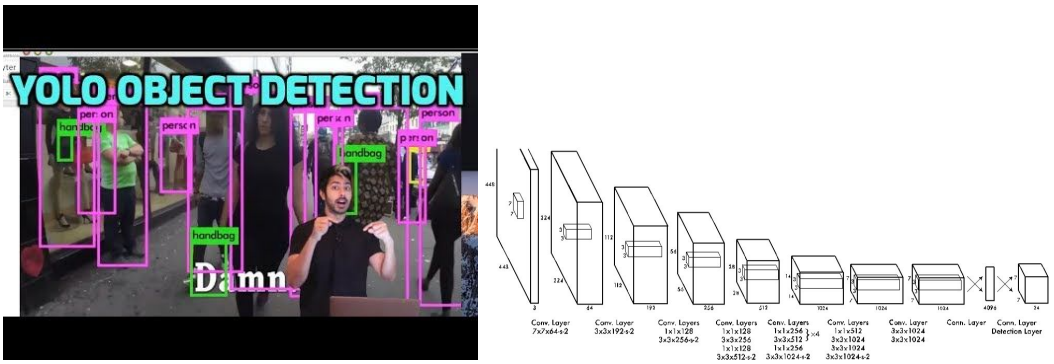

Fig 3.1: YOLO example

c) Research for new speeding up technology

As a part of the research for developing our technology, we try implementing a new CNN optimization from state-of-the-art articles. One example is XNOR-Network algorithm - an efficient approximation to a standard convolutional neural network. It runs a state-of-art network on a CPU in real time. The technique mainly approximates convolution by using binary arithmetic operation.

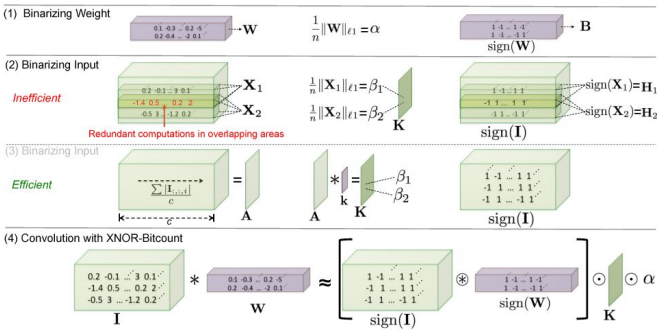

Fig 3.2: XNOR-Net example

d) Optimizing in other languages.

We will translate any other languages from our client into optimized required code. This business style is following: 1) our employees talk to customers on a meeting about their need for their code, such as speed, memory usage, and specific restriction. 2) We provide our solution to the requirement. 3) We implement the optimized codes and deliver the product. Although Halide and CNN optimization is one of our strengths, our goal is to speed up anything by using the best technology for the improvement so we will provide a solution except CNN and Halide.

4. Competition - Speed and rival companies

The first competition is the processing speed of the original source code. Our service is valuable only if the speed of the new code is extremely faster than the received code. The second one is other companies specialized for optimization. The solution to both competitions is to hire a lot of skillful software engineer specialized for optimization and to let them research for the new technology from current articles. The ‘extremely faster code’ can be achieved only by implementing an effective speeding up technique. We will be required to be a pioneer of Halide and other speeding up technology.

5. Development

Automatically Convertible Halide Generator
Not only optimizing codes but also we can create the product to help speed up - Automatically Convertible Halide Generator. This system automatically converts any other language source codes into Halide codes. It will reduce our time and effort to change the given code from the beginning. Since it cannot be as accurate as the human's work, our employees will fix the codes to make it even better.

Halide Compiler to Generate Hardware Description Language(HDL)
The second idea is Halide compiler to generate HDL automatically. For specific hardware such as FPGA, we will make a new compiler for Halide to generate HDL for more optimization.

6. Distribution.
There are three types of distribution, productization, open source code and the order to optimize codes. YOLO implementation is the example of productization - we create the software for a specific purpose and sell it to other companies. For the open source code, we will upload our part of Halide source code (`Halide Code` file) as open source code to spread the efficiency of Halide. Since Halide is still not popular language in the current IT field, we need to inform the effectivity of the language. Publishing as open source will help us get more client for optimization in the end. The order to optimize codes is to talk to the client, discuss the possible solution, implement it after receiving codes and deliver the faster codes.

7. Reference

"Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines", MIT CSAIL, http://people.csail.mit.edu/jrk/halide-pldi13.pdf

"fukushima1981." *Qiita*, https://qiita.com/fukushima1981

Aki. "'Introduction to Image Processing Programming by Halide' Material Disclosure." *Halide to FPGA*, 27 Nov. 2017, https://www.halide2fpga.com/halide_programming_tutorial/

"XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Network", University of Washington, https://arxiv.org/pdf/1603.05279.pdf

"You Only Look Once: Unified, Real-Time Object Detection" University of Washington, https://arxiv.org/pdf/1506.02640.pdf

DeepAge. "Understand the classic Convolutional Neural Network from scratch." *DeepAge*, DeepAge, 7 Nov. 2016, https://deepage.net/deep_learning/2016/11/07/convolutional_neural_network.html

Moriyama, Naoto. "Introduction to Neural Network." *LinkedIn SlideShare*, 31 May 2016, https://www.slideshare.net/naotomoriyama/ss-62582878?next_slideshow=1

"YOLO Algorithm - Object Detection." *Coursera*, Rice University, https://www.coursera.org/lecture/convolutional-neural-networks/yolo-algorithm-fF3O

0

"Going deeper with convolutions" Google Inc.
https://arxiv.org/pdf/1409.4842.pdf