SW Report: Accelerating Convolutional Neural Networks using Programmable Logic (IN2106, IN4345)

Hewei Gao, Lorenz Krakau, Tianle Ren

June 4, 2023

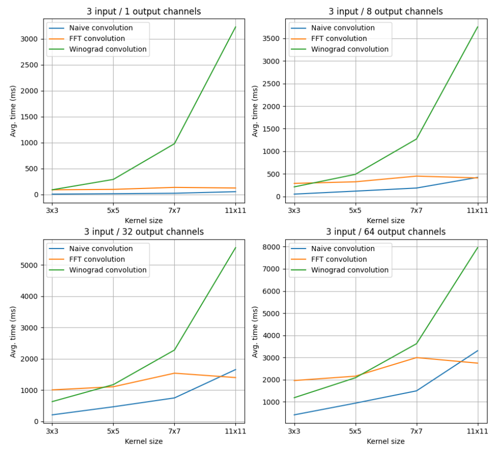


Figure 1: Benchmarks for 3 input channels

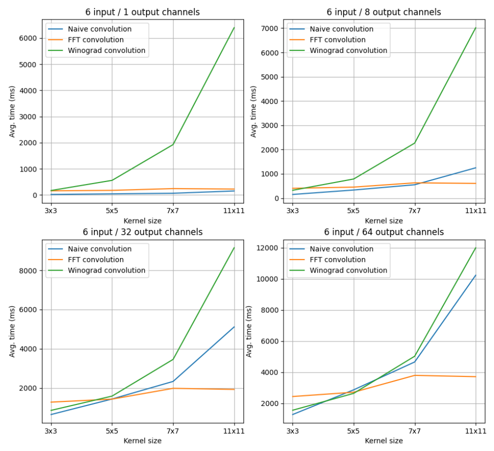


Figure 2: Benchmarks for 6 input channels

1 Supported Layers

1.1 Explanation of implementation

The CNN inference process is achieved by *cnn.cpp*. According to the layer structure, the input tensor is passed through various layers of the CNN. By applying switch-case, we read the layer type of each layer and call corresponding layer function, such as conv2d(), Linear(). These layer functions are defined in *kernals.cpp*, which is programmed with the help of mathematical expressions in $lab1.pdf$. It is worth mentioning that the convolutional kernel in this practical course are all square and adopt a stride of 1 for every convolutional layer. Therefore the single output of a convolutional layer is calculated as Eq.1

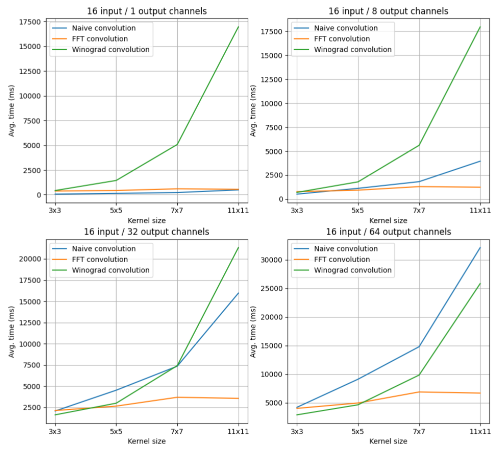
$$Zi,j,k = \sum_{c}^{X_C} \sum_{p}^{W_M} \sum_{q}^{W_N} x_{c,j+p,k+q} W_{i,c,p,q} \tag{1}$$

Additionally, since the padding function is implemented outside the convolution function, we leave the conv2d() to be always not padded.

2 Optimizing Convolution

From the benchmark results in 1,2,3,4 we can deduce a few observations:

- Naive convolution performs better in the cases with small number of input channel paired with a small kernel, for example, 3/1, 3/8, 6/1, 6/8, especially when small input channel with a large output channel, for example 3/32, 3/64.



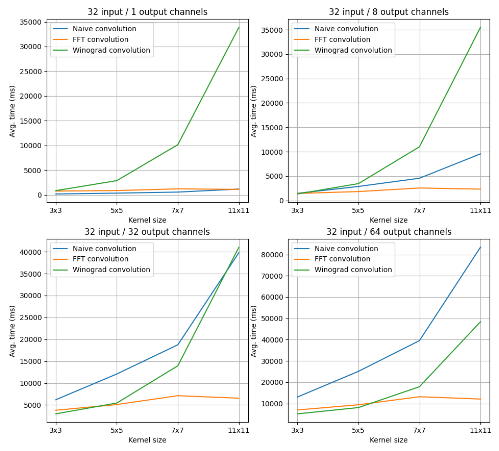


Figure 3: Benchmarks for 16 input channels Figure 4: Benchmarks for 32 input channels

- The two optimized algorithms always outperform the naive convolution when the layer has a large number of input and output channels, for example, 32/32 (#input channels/#output channels), 64/64, 16/64.
- Winograd becomes worse quickly with higher kernel sizes. FFT achieves better performance than Winograd when the convolutional layer has a large number of input/output channel and a large kernel size, such as 7*7 and 11*11.

Intuitively, the first two points are because FFT and Winograd both involve a certain level of preprocessing and postprocessing overhead (such as padding, data rearrangement, Fourier transform, inverse Fourier transform, etc.) that can become significant when the input and output channels are small. In contrast, the naive convolution method has no such overhead, and hence can be faster for small inputs. The third point might result from the different way of transforming the input tensor and weight tensors. Winograd's algorithm reduces multiplicative complexity by increasing additive complexity. It transforms tensors by applying matrix multiplication, which has an repidly increasing complexity with larger kernel size, while FFT transforms tensors by applying 2D-FFT with a complexity of $O(n) = N^2 * log(N)$, which doesn't change as much with the kernel size, making it better for larger kernels.

For now a very crude classification criterion was chosen for picking the methods:

```
if(input_channels < 6) {
return Optimization::None;
} else if(kernel_width > 7 && input_channels > 6) {
```

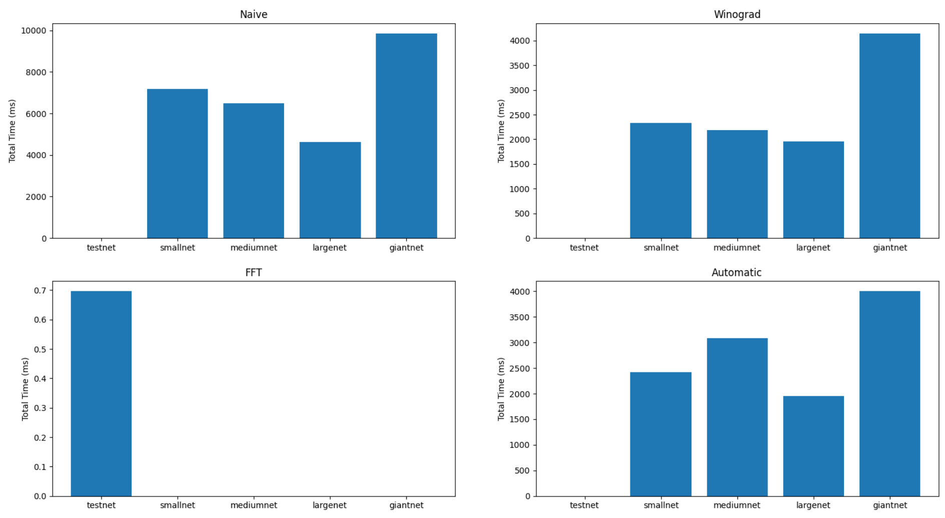


Figure 5: Final Benchmark Results (FFT Results omitted largely since they didn't finish in time

```
return Optimization::FFT;
} else if(input_channels > 6 && kernel_width < 7) {
return Optimization::Wino;
} else {
return Optimization::None;
}
```

Training a SVM on the benchmark data might yet better classification criteria, but couldn't be finished in time for this report.

3 Final Implementation

3.1 Visualizations of benchmarks results

The final benchmark results are shown in Figure 5.

4 Future work

As mentioned above, the algorithm to choose the best optimization schemes can still be improved. We will test more benchmarks and leverage them to train a SVM and get better classification criteria. Additionally, the FFT implementation can also be improved. Since all input in our case are real number, we can combine data of two tiles into one complex number and regard them as the real and imaginary part respectively.