Fast Algorithms for Convolutional Neural Networks

Problem: What is the paper trying to solve?

- Deep convolutional neural networks take GPU-days of computation to train on large data sets. Pedestrian detec□tion for self driving cars requires very low latency. Image recognition for mobile phones is constrained by limited pro□cessing resources. The success of convolutional neural net□works in these situations is limited by how fast we can com□pute them. Conventional FFT based convolution is fast for large filters, but state of the art convolutional neural net□works use small, 3 × 3 filters.

Key idea: What is the main idea in the solution?

- We introduce a new class of fast algorithms for convolutional neural networks using Winograd's minimal filtering algorithms. The algorithms compute minimal complexity convolution over small tiles, which makes them fast with small filters and small batch sizes. We benchmark a GPU implementation of our al□gorithm with the VGG network and show state of the art throughput at batch sizes from 1 to 64.

Novelty: What is different from previous work, and why?

- This paper introduces a new class of fast algorithms for convolutional neural networks based on the minimal filter□ing algorithms discovered by Toom [14] and Cook [4] and generalized by Winograd [16]. The algorithms can reduce the arithmetic complexity of a convnet layer by up to a fac□tor of 4 compared to direct convolution. Almost all of the arithmetic is performed by dense matrix multiplies of suf- ficient dimensions to be computed efficiently, even when the batch size is very small. The memory requirements are also light compared to the conventional FFT convolution algorithm. These factors make practical implementations possible. Our implementation for NVIDIA Maxwell GPUs achieves state of the art throughput for all batch sizes mea□sured, from 1 to 64, while using at most 16MB of workspace memory.
- Matrix multiply has efficient implementations on CPU, GPU, and FPGA platforms, owing to its high computational intensity. Thus we have arrived at the practical implemen□tation for the fast algorithm listed in Algorithm 1.

Critique: Is there anything you would change in the solution or the way that the solution is presented/evaluated?

- Never thought that the repetitive numbers in the convolution kernel would be useful to reduce computation complexity.