Fast Training of Convolutional Networks through FFTs

Problem: What is the paper trying to solve?

1. Convolutional networks are one of the most widely employed architectures in computer vision and machine learning. In order to leverage their ability to learn complex functions, large amounts of data are required for training. Training a large convolutional network to produce state-of-the-art results can take weeks, even when using modern GPUs. Producing labels using a trained network can also be costly when dealing with web-scale datasets.

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

1. In this work, we present a simple algorithm which accelerates training and inference by a significant factor, and can yield improvements of over an order of magnitude compared to existing state-of-the-art implementations. This is done by computing convolutions as pointwise products in the Fourier domain while reusing the same transformed feature map many times. The algorithm is implemented on a GPU architecture and addresses a number of related challenges.

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

1. Although it has long been known that convolutions can be computed as products in the Fourier do□main, until recently the number of feature maps used in convolutional networks has been too small to make a method like ours effective. Previous work in the 90â€™s [1] explored the possibility of using FFTs to accelerate inference at the first layer of a trained network, where the Fourier transforms of the filters could be precomputed offline. However, this was not used during training, possibly because the number of feature maps used at the time was too small to make the overhead of computing FFTs at every iteration worthwhile. When the number of feature maps is large, as is the case for modern convolutional networks, using FFTs accelerates training and inference by a significant factor and can lead to a speedup of over an order of magnitude.
2. Although conceptually straight forward, a number of challenges relating to GPU implementation needed to be addressed. First, current GPU implementations of the FFT such as cuFFT are designed to parallelize over individual transforms. This can be useful for computing a limited number of transforms on large inputs, but is not suitable for our task since we are performing many FFTs over relatively small inputs. Therefore, we developed a custom CUDA implementation of the CooleyTukey FFT algorithm which enabled us to parallelize over feature maps, minibatches and within each 2-D transform. Note that 2-D FFTs lend themselves naturally to parallelization since they can be decomposed into two sets of 1-D FFTs (one over rows and the other over columns), and each set can be done in parallel.

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

1. Maybe more experiments will be convincible.