Scaling the Scattering Transform: Deep Hybrid Networks

Edouard Oyallon, Eugene Belilovsky, Sergey Zagoruyko

Appendix: Fast GPU Implementation of the Scattering Transform

Previous implementations of the scattering transform are too slow to scale to large image datasets. Moreover, learning networks which operate on top of the scattering transform is impractical with current, CPU base, implementations. For this reason we develop a GPU based implementation of the scattering transform that is many orders of magnitude faster than existing implementations. Our implementation is compatible and integrated with two modern deep learning libraries pytorch (python) and torch (lua).

The bottleneck of the scattering implementations on CPU was generally both speed and memory consumption. Our implementation on GPU drastically improves both. It is thus necessary to quickly explain our algorithm: we show that by reorganizing the order of the computation of the algorithm, one can gain a large speed improvement.

Computing a scattering transform at order 2 requires computing each path $||x \star \psi_{p_1}| \star \psi_{p_2}|$ where p_1, p_2 are the parameters with increasing scales of the filters. Computing each path can be viewed as a computational tree, where the coefficient of the scattering transform before an averaging are the leaf of the tree, and the internal nodes are the modulus of the intermediary wavelet transform. The way the tree is traversed affects the computation time. In the existing CPU implementation, ScatNet [**?**], the traversal is done by first computing each internal node, storing the results of each internal node. In a second steps, the leafs are computed and stored. In terms of memory, this is not optimal since it requires storing the intermediate computations. Instead, we use an affix traversal of the tree. It reduces at its minimal the memory used, and allow the straightforward application of GPU primitives.

Our implementation, dubbed PyScatWave (python) and ScatWave2 (lua), is a GPU version of the scattering networks in PyTorch and Torch7, that is based on the observation above. ScatNetLight is a MATLAB version on CPUs, which uses as much as possible multithreading. PyScatWave on the other hand uses the library cuFFT as well as custom CUDA kernels to implement the core operations.The Table 1 reports the difference in computation time, for identical parameters and output representations (e.g. same sampling, same hyper parameters). The input corresponds to batches of 128 tensor, the two first dimensions being the size of the image, and the third the number of elements in the tensor (e.g. the color in the case of images) . For our comparisons we used 24 cores of an Intel Xeon 2.6GhZ for ScatNetLight and a GTX 1080 GPU for PyScatWave. The speed-up is at least an order of magnitude in all cases, and up to $225x$ speed-up in the case of larger images, similar to the size of those in imagenet.

In practice, on the CIFAR dataset, we find that even when performing the scattering transform on-the fly (versus caching its output) we can get a speedup of 30% in training time with scattering + WRN 12-8 versus WRN-16-8 in the experiments in Section 4.3.1. This speedup comes from learned convolutions operating on a spatial resolution of only 8×8. For imagenet our current implementation with on-the fly generation allows for training of Scattering and ResNet-10 at speeds analogous to Resnet-152.

We note that further optimizations are possible and that with appropriate infrastructure, effective caching mechanisms can allow training speeds to improve substantially. Using our software which we will release at time of publication there are several possibilites to further improve the low level CUDA routines to better exploit the parallelization of the scattering transform, as has been done for many key low-level CNN components. Finally, we note that the scattering transform is amenable to hardware implementation and deployment.

Table 1. Computation time (in seconds) for different input size, one being with MATLAB on 24 CPUs and the other with PyTorch on a single GPU. For sizes similar to imagenet, ScatNetLight also utilizes a larger amount of memory (5x or more).

Input Size (H $\times$ W $\times$ 3 $\times$ BatchSize)	**J**	**ScatNetLight (in s)**	**PyScatWave (in s)**	**Speed Up**
$32 \times 32 \times 3 \times 128$	2	2.5	0.03	8x
$32 \times 32 \times 3 \times 128$	4	13	0.20	65x
$128 \times 128 \times 3 \times 128$	2	16	0.26	62x
$128 \times 128 \times 3 \times 128$	4	52	0.54	96x
$256 \times 256 \times 3 \times 128$	2	160	0.71	225x