Supplement: Band-limited Training and Inference for Convolutional Neural Networks

1. Implementation details

We present details on the map reuse, CUDA implementation and shifting of the DC coefficient.

1.1. Map Reuse

We divide the input map M (with half of the map already removed due to the conjugate symmetry) into two parts: upper D1 and lower D2. We crop out the top-left (S1) corner from D1 and bottom-left (S2) corner from D2. The two compressed representations S1 and S2 are maintained separately for the backward pass. In the backward pass, we pad the two corners S1 and S2 to their initial sizes D1 and D2, respectively. Finally, we concatenate D1 and D2 to get the FFT map M', where the high frequency coefficients are replaced with zeros.

If the memory usage should be decreased as much as possible and the filter is small, we can trade the lower memory usage for the longer computation time and save the filter in the spatial domain at the end of the forward pass, followed by the FFT re-computation of the filter in the backward pass. The full frequency representation of the input map (after padding) is bigger than its spatial representation, thus the profitability of re-computing the input to save the GPU memory depends on the applied compression rate.

We also contribute a fast shift of the DC coefficients either to the center or to the top-left corner. The code for the element-wise solution uses two for loops and copy each element separately. For the full FFT map, we divide it into quadrants (I - top-right, II - top-left, III - bottom-left, IV - bottom-right). Then, we permute the quadrants in the following way: I $\rightarrow$ III, II $\rightarrow$ IV, III $\rightarrow$ I, IV $\rightarrow$ II.

1.2. CUDA

We use $min(max\ threads\ in\ block, n^2)$ threads per block and the total number of GPU blocks is Sf', where S is the mini-batch size, f' is the number of output channels, and n is the height and width of the inputs. Each block of threads is used to compute a single output plane. Intuitively, each thread in a block of threads incrementally executes a complex multiplication and sums the result to an aggregate for all f input channels to obtain a single output cell (x, y).

Additional optimizations, such as maintaining the filters only in the frequency domain or tiling, will be implemented in our future work.

2. Experiments

2.1. Experimental setup

For the experiments with ResNet-18 on CIFAR-10 and DenseNet-121 on CIFAR-100, we use a single instance of P-100 GPU with 16GBs of memory.

We also use data from the UCR archive, with the main representative: 50 words time-series dataset with 270 values per data point, 50 classes, 450 train data points, 455 test data points, 2 MB in size. One of the best peforming CNN models for the data is a 3 layer Fully Convolutional Neural Network (FCN) with filter sizes: 8, 5, 3. The number of filter banks is: 128, 256, 128. [1].

Our methodology is to measure the memory usage on GPU by counting the size of the allocated tensors. The direct measurement of hardware counters is imprecise because PyTorch uses a caching memory allocator to speed up memory allocations and incurs much higher memory usage than is actually needed at a given point in time.

2.2. DenseNet-121 on CIFAR-100

We train DenseNet-121 (with growth rate 12) on the CIFAR-100 dataset.

In Figure 1 we show small differences in test accuracy during training between models with different levels of energy preserved for the FFT-based convolution.

In Figure 2 we show small differences in accuracy and loss between models with different convolution implementations. The results were normalized with respect to the values obtained for the standard convolution used in PyTorch.

2.3. Reduced Precision and Bandlimited Training

In Figure 4 we plot the maximum allocation of the GPU memory during 3 first iterations. Each iteration consists of training (forward and backward passes) followed by test-

[1]http://bit.ly/2FbdQNV

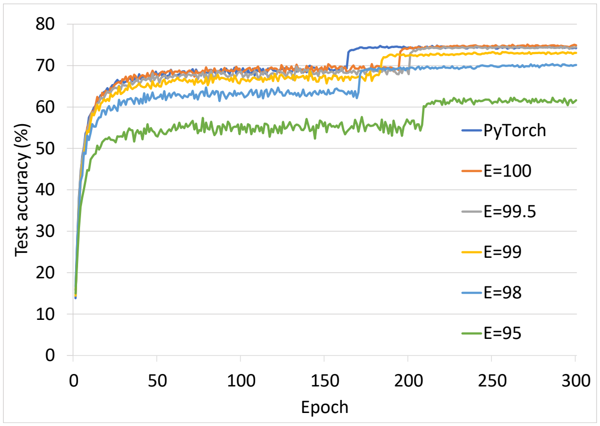

Figure 1. Comparing test accuracy during training for CIFAR-100 dataset trained on DenseNet-121 (growth rate 12) architecture using convolution from PyTorch and FFT-based convolutions with different energy rates preserved.

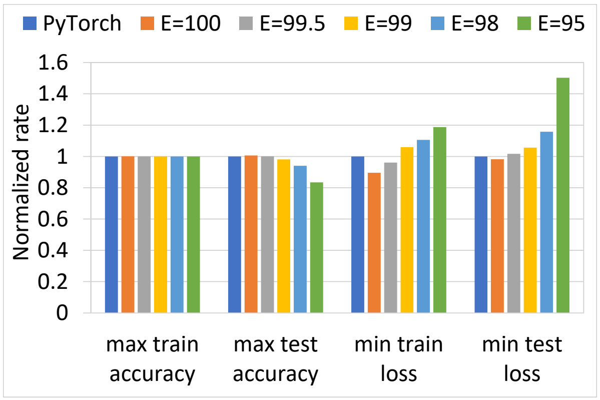

Figure 2. Comparing accuracy and loss for test and train sets from CIFAR-100 dataset trained on DenseNet-121 (growth rate 12) architecture using convolution from PyTorch and FFT-based convolutions with different energy rates preserved.

ing (a single forward pass). We use CIFAR-10 data on ResNet-18 architecture. We show the memory profiles of RPA (Reduced Precision Arithmetic), bandlimited training, and applying both. A detailed convergence graph is shown in Figure 3.

2.4. Resource usage vs accuracy

The full changes in normalized resource usage (GPU memory or time for a single epoch) vs accuracy are plotted in Figure 5.

FFT-ed tensors in PyTorch place the lowest frequency coefficients in the corners. For compression, we extract parts of a tensor from its top-left and bottom-left corners, then concatenate them. For the decompression part, we fill in the discarded parts with zeros. These operations have a non-negligible cost. The overhead might be minimized in 2D case if the DC component would be moved to the center and the compression would only extract the middle coefficients (by slicing and without a need to stitch together different parts of a tensor) so that the compressed tensor would remain contiguous.

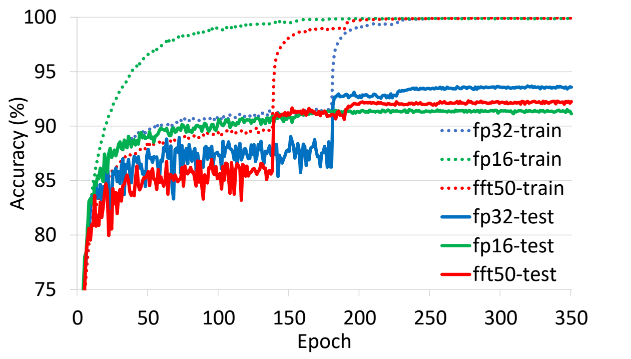

Figure 3. Train and test accuracy during training for CIFAR-10 dataset trained on ResNet-18 architecture using convolution from PyTorch (fp32), mixed-precision (fp16) and FFT-based convolutions with 50% of compression for intermediate results and filters (fft50). The highest test accuracy observed are: 93.69 (fp32), 91.53 (fp16), 92.32 (fft50).

DenseNet-121 shows a significant drop in GPU memory usage with relatively low drop in accuracy. On the other hand, ResNet-18 is a smaller network and after about 50% of the compression ratio, other than convolutional layers start dominate the memory usage. The convolution operation remains the major computation bottleneck and we decrease it with higher compression.

For ResNet-50 on ImageNet, we observe a substantial drop in memory usage after applying our compression and similar behavior in terms of the time per epoch to the ResNet-18 model on CIFAR-10.

2.5. Dynamic Changes of Compression

Deep neural networks can better learn the model if the compression is fixed and does not change with each iteration depending on the distribution of the energy within the frequency coefficients of a signal. However, the dynamic compression based on the energy preserved shows that at the beginning of the training the network is focused on the low frequency coefficients and as the training unfolds, more and more coefficients are taken into account. The compression based on preserved energy does not steadily converge to a certain compression ratio but can decrease significantly and abruptly even at the end of the training process (especially, for the initial layers). We observe that the compression can be applied more effectively to the first layers and the deeper the layers the less compression can be applied (for a given

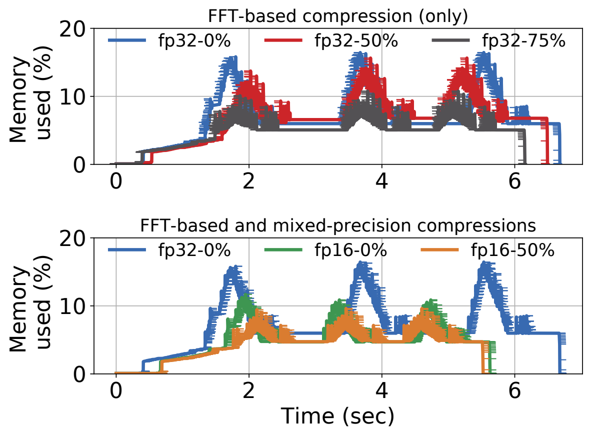

Figure 4. Memory used (%) for the first 3 iterations (train and test) with mixed-precision and FFT-based compression techniques. Mixed precision allows only a certain level of compression whereas with the FFT based compression we can adjust the required compression and accuracy. The two methods can be combined (fp16-50%).

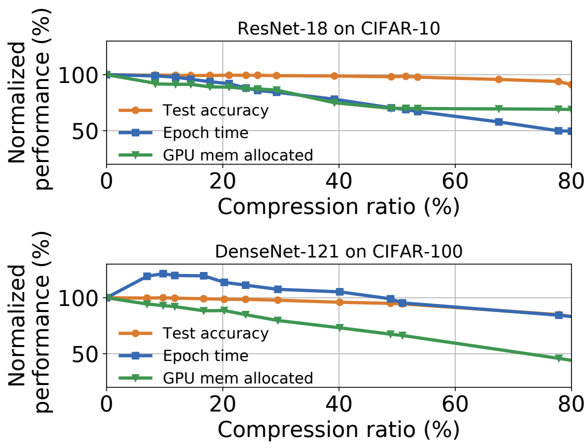

Figure 5. Normalized performance (%) between models trained with different FFT-compression ratios.

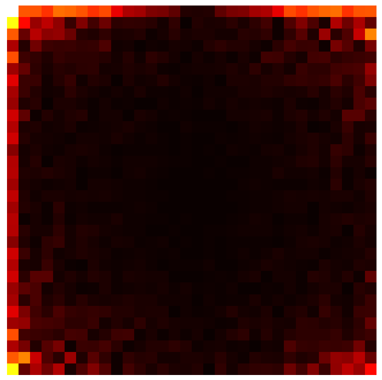

Figure 6. A heat map of absolute values (magnitudes) of FFT coefficients with linear interpolation and the max value colored with white and the min value colored with black. The FFT-ed input is a single (0-th) channel of a randomly selected image from the CIFAR-10 dataset.

energy level preserved).

The dynamic and static compression methods can be combined. We determine how much compression should be applied to each layer via the energy level required to be saved in each layer and use the result to set the static compression for the full training. The sparsification in the Winograd domain requires us to train a full (uncompressed) model, then inspect the Winograd coefficients of the filters and input maps and zero-out these of them which are the smallest with respect to their absolute values, and finally retrain the compressed model. In our approach, we can find the required number of coefficients to be discarded with a few forward passes (instead of training the full network), which can save time and also enables us to utilize less GPU memory from the very beginning with the dynamic compression.

2.6. Compression based on Preserved Energy

There are a few ways to compress signals in the frequency domain for 2D data. The version of the output in the frequency domain can be compressed by setting the DC component in the top left corner in the frequency representation of an image or a filter (with the absolute values of coefficients decreasing towards the center from all its corners) and then slicing off rows and columns. The heat maps of such a representation containing the absolute value of the coefficients is shown in Figure 6.

The number of preserved elements even for 99% of the preserved energy is usually small (from 2X to 4X smaller than the initial input). Thus, for the energy based compression, we usually proceed starting from the DC component and then adding rows and columns in the vertically mirrored L fashion. It can be done coarse-grained, where we just take into account the energy of the new part of row or column to be added, or fine-grained, where we add elements one by one and if not the whole row or column is needed, we zero-out the remaining elements of both an activation map and a filter.

2.7. Visualization of the compression

We present the visualization of our FFT-based compression method in 7. The magnitude is conveniently plotted in a logarithmic scale (dB).

2.8. Energy Based Compression for ResNet-18

Figure 8 shows the linear correlation between the accuracy of a model and the energy that was preserved in the model

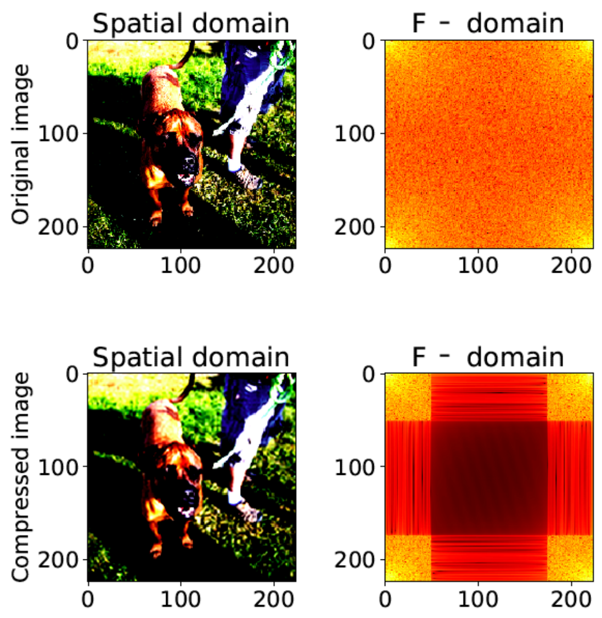

Figure 7. We present an original image from the ImageNet dataset with label: bull mastiff in the first row and the 50% compressed Fourier domain in the second row. The heat maps of magnitudes of Fourier coefficients are presented in a logarithmic scale (dB) with linear interpolation and the max value is colored with white while the min value is colored with black. The Fourier-ed representation is plotted for a single (0-th) channel.

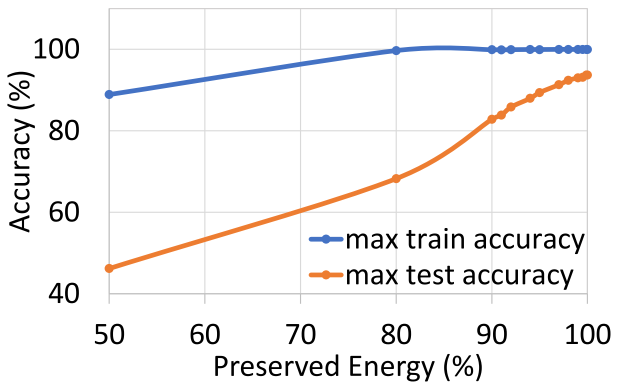

Figure 8. The linear correlation between the accuracy of a model and the energy that was preserved in the model during training and testing.

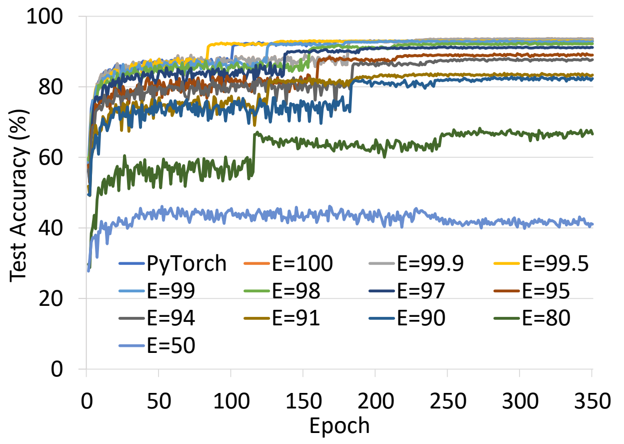

Figure 9. The test accuracy during the training process of the ResNet-18 model on the CIFAR-10 dataset.

during training and testing. Each point in the graph requires a fool training of a model for the indicated energy level preserved.

Figure 9 shows the test accuracy during the training process of the ResNet-18 model on the CIFAR-10 dataset.

Figure 10 shows the train accuracy during the training process of the ResNet-18 model on the CIFAR-10 dataset.

2.9. Training vs. Inference Bandlimiting

To further corroborate the previous point, consider a scheme where we train the network with one compression ratio and test with another (Figure 19).

We observe that the network is most accurate when the compression used for training is the same that is used during testing. We used the Friedman statistical test followed by the post-hoc Nemenyi test to assess the performance of multiple compression ratios during inference over multiple datasets. Figure 14 shows the average rank of the test accuracies of different compression ratios during inference across 25 randomly chosen time-series data from the UCR Archive. The training was done while preserving 90% of the energy. Inference with the same compression ratio (90%) is ranked first. meaning that it performed the best in the majority of the datasets. The Friedman test rejects the null hypothesis that all measures behave similarly, and, hence, we proceed with a post-hoc Nemenyi test, to evaluate the significance of the differences in the ranks. The wiggly line in the figure connects all approaches that do not perform statistically differently according to the Nemenyi test. We had similar findings when training was done using no compression but compression was later applied during inference (see Figure 15). In other words, the network *learns how to best leverage a band-limited operation to make its predictions*.

Even so its performance degrades gracefully for tests with the compression level further from the one used during training. In our opinions, the smooth degradation in performance is the most valuable property of band-limiting. An outer optimization loop can tune this parameter without worrying

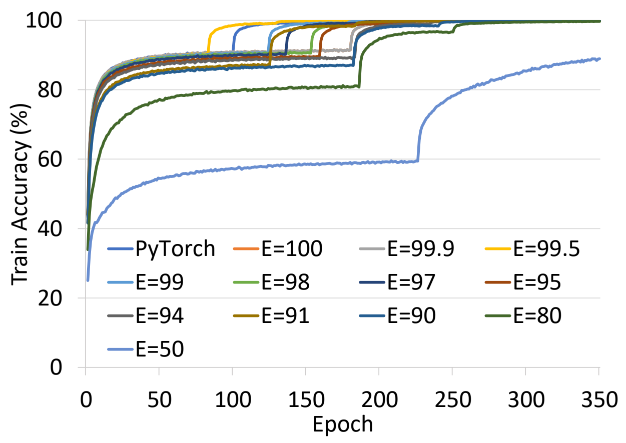

Figure 10. The train accuracy during the training process of the ResNet-18 model on the CIFAR-10 dataset.

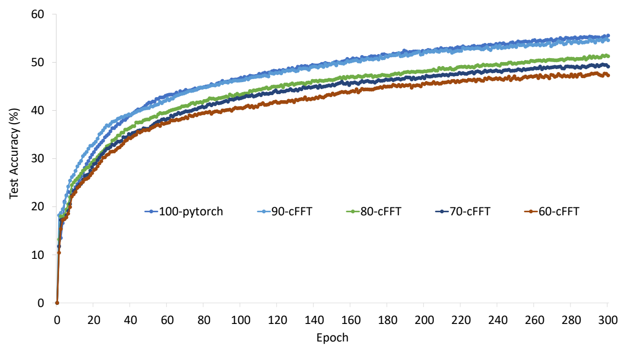

Figure 11. A comparison of 2D convolution operation implemented in PyTorch and FFT version for different percentage of preserved energy (on the level of a batch).

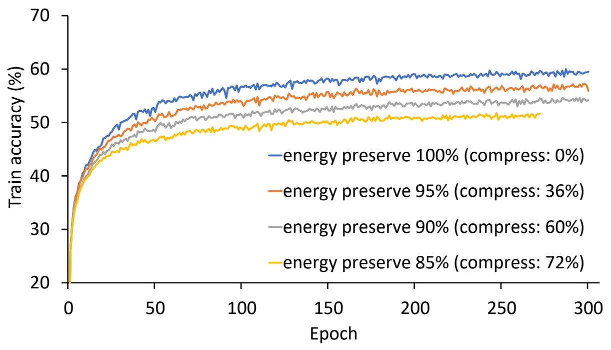

Figure 12. Train accuracy for CIFAR-10 dataset on LeNet (2 conv layers) architecture Momentum 0.9, batch size 64, learning rate 0.001 .

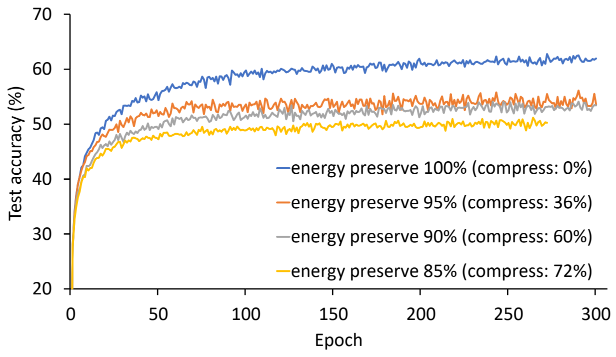

Figure 13. Test accuracy for CIFAR-10 dataset on LeNet architecture (2 conv layers, momentum 0.9, batch size 64, learning rate 0.001.

about training or testing instability.

2.10. Error incurred by 2D convolution with compression

We tried to measure how accurate the computation of the convolution result is when the compression is applied. An image from CIFAR-10 dataset (3x32x32) was selected and an initial version of a single filter (3x5x5, Glorot initialization). We did convolution using PyTorch, executed our convolution with compression for different compression ratios, and compared the results. The compression was measured relatively to the execution of our FFT-based convolution without any compression (100% of the energy of the input image is preserved). The results show that for 2D convolution the relative error is already high (about 22.07%) for a single index discarded (the smallest possible compression of about 6%). However, after the initial abrupt change we observe a linear dependence between compression ratio and relative error until more than about 95% of compression ratio, after which we observe a fast degradation of the result.

We plot in Figure16 fine-grained compression using the top method (the coefficients with lowest values are zeroed-out first). For a given image, we compute its FFT and its spectrum. For a specified number k of elements to be zeroed-out, we find the k smallest elements in the spectrum and zero-out the corresponding elements in the in the image. The same procedure is applied to the filter. Then we compute the 2D convolution between the compressed filter and the image. We do not remove the elements from the tensors (the sizes of the tensors remain the same, only the smallest coefficients are zeroed-out). The plots of the errors for a given compression (rate of zeroed-out coefficients) are relatively smooth. This shows that our method to discard coefficients and decrease the tensor size is rather coarse-grained and especially for the first step, we remove many elements.

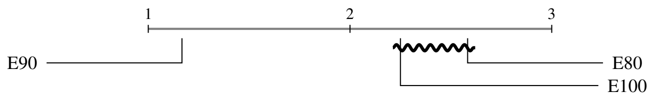

Figure 14. Ranking of different compression ratios (80%, 90%, and 100% energy preserved) during inference with model trained using no compression (90% of energy preserved)

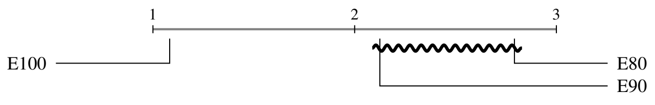

Figure 15. Ranking of different compression ratios (80%, 90%, and 100% energy preserved) during inference with model trained using no compression (100% of energy preserved)

We have an input image from CIFAR-10 with dimensions (3x32x32) which is FFT-ed (with required padding) to tensor of size (3x59x30). We plot the graph in 10 element zero-out step, i.e. first we zero-out 10 elements, then 20, and so on until we reach 5310 total elements in the FFT-ed tensors). The compression ratio is computed as the number of zeroed-out elements to the total number of elements in FFT-ed tensor. There are some dips in the graph, this might be because the zeroed-out value is closer to the expected value than the one computed with imprecise inputs. With this fine-grained approach, after we zero-out a single smallest coefficients (in both filter and image), the relative error from the convolution operation is only 0.001%. For the compression ratio of about 6.61%, we observe the relative error of about 8.41%. In the previous result, we used the lead method and after discarding about 6.6% of coefficients, the relative error was 22.07%. For the lead method, we were discarding the whole rows and columns across all channels. For the fine-grained method, we select the smallest elements within the whole tensor.

2.11. Time-series data

We show the accuracy loss of less than 1% for 4X less average GPU memory utilization (Figures: 17 and 18) when training FCN model on 50 words time-series dataset from the UCR archive.

In Figure 19 we show the training compression vs. inference compression for time-series data. This time we change the compression method from static to the energy based, however, the trend remains the same. The highest test accuracy is achieved by the model with the same energy preserved during training and testing.

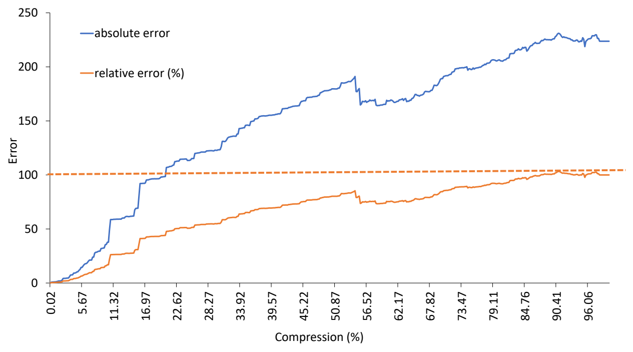

Figure 16. A comparison of the relative (in %) and absolute errors between 2D convolution from PyTorch (which is our gold standard with high numeric accuracy) and a fine-grained top compression method for a CIFAR-10 image and a 5x5 filter (with 3 channels).

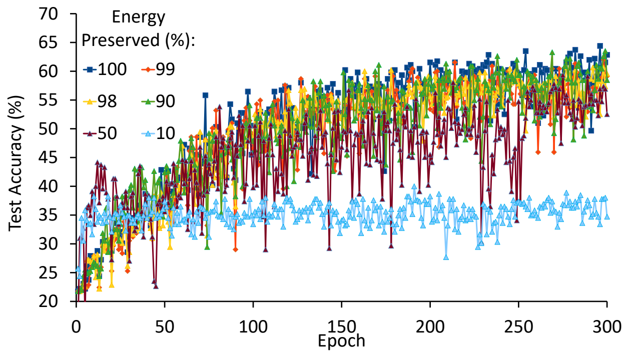

Figure 17. Test accuracy on a 3 layer FCN architecture for 50 words time-series dataset from the UCR archive.

2.12. Shapes of filters, activations and gradients in the frequency domain.

We visualized the input maps (activations) and filters in the frequency domain for convolution operations. We chose dataset 50 words from the UCR archive, the FCN network with 3 convolution layers (each followed by a ReLU and batch normalization) and preserve 90% of the energy in the activations.

Neural networks are biased towards low frequency functions, thus they tend to better approximate the smooth functions or natural images with small amount of noise. The question is if our lead compression method can be applied to the filters. We observe a transformation of the filter (in the last layer of FCN) from a random shape to a slope starting high for low frequencies and going down with increasing frequencies, which suggests that the network learned the low frequency values of the filters as constrained by our compression.

The representation of the input signal in the frequency do-

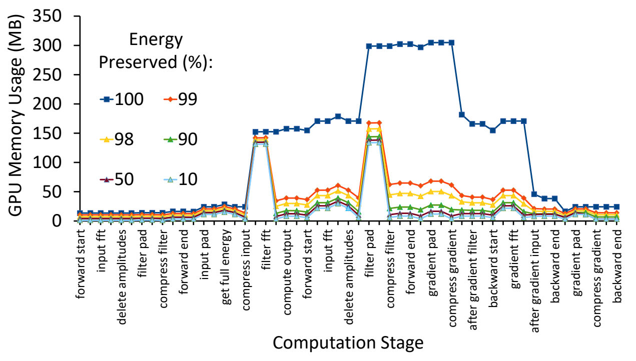

Figure 18. GPU memory usage (in MB) during training for a single forward and backward pass through the FCN network using 50 words dataset.

Table 1. Max. and Avg. memory usage with corresponding compression ratio for each layer. We compute the compression ratio as: $\frac{\text{input size}}{\text{FFT compressed size}}$. For example, for the first convolutional layer (conv1), which has 270 input values, there are 1024 values in the frequency domain, thus for 100% compression ratio we obtain: $\frac{270}{1024} = 0.26$, for 99% compression ratio: $\frac{270}{181} = 1.49$, where 181 is the length of the compressed signal.

Memory usage (MB)		Compression ratio		
Max.	Avg.	conv1	conv2	conv3
305	118	0.26	0.27	0.27
168	41	1.49	2.21	2.76
158	34	1.99	2.96	3.71
144	25	4.23	6.62	9.40
138	21	11.29	27.80	47.00
134	17	33.88	139.00	141.00

main shows the highest (signal) values for the low frequency values. The most important outcome is that the data inputs (activations) after the convolution operation retain their shape and the (signal) highest values are preserved for the low frequency values. This is because when we consider the convolution operation in the frequency domain, the element-wise multiplication between the input that has very small or zero values for all frequencies but the low ones, hence irrespective of the filter shape, the high signal values should remain only for the low frequency values after the convolution. We also plot the spectra of the flowing back gradients in the time and spectral domains. The gradients are generated for the output. The output from the convolution operation preserves the property that the highest values of the signal remain in the regime of low frequencies, thus also the gradients follow the same pattern. The only problematic part of the analysis is that the filters are relatively small and they do have high signal values even for the high frequencies (that usually represent noise for natural images), for

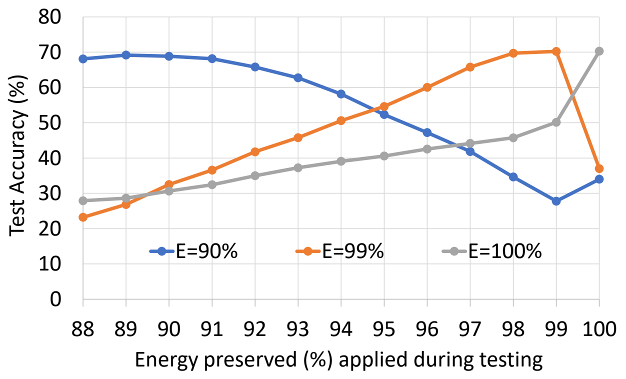

Figure 19. We train three models on the time-series dataset uWaveGestureLibrary_Z. The preserved energy during training for each of the models is 90%, 99% and 100% (denoted as E=X% in the legend of the graph). Next, we test each model with energy preserved levels ranging from 88% to 100%. We observe that the highest accuracy during testing is for the same energy preserved level as the one used for training and the accuracy degrades smoothly for higher or lower levels of energy preserved.

example, for the 1st layer of FCN. Our method to deal with the problematic behavior would be to try to initialize the filters in the frequency domain with higher values for lower frequency and manage the filters only in the frequency domain, since the transformations of the filter back and forth between time and spectral domain are unnecessary.

2.13. Robustness to Adversarial Examples

We present the most relevant adversarial methods that were executed using the foolbox library. Our method is robust to decision-based attacks (GaussianBlur, Additive Uniform or Gaussian Noise) but not to the gradient-based attacks (e.g., GradientSign).

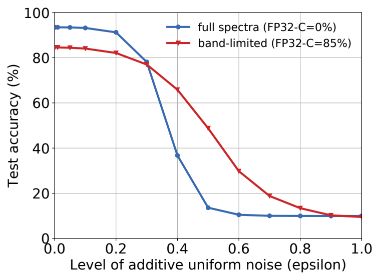

Figure 20. Input test images are perturbed with additive uniform noise, where the epsilon parameter is changed from 0 to 1. The more band-limited model, the more robust it is to the introduced noise. We use ResNet-18 models trained on CIFAR-10.