Fast Training of Convolution Networks through FFTs

YICHUAN SU, ZHIJING YAO, and YING XU

Convolution network has been widely used in the machine learning. However, As the data set grows, the time required for convolution is very long. In order to have the ability to train the large data set, the Fast Fourier Transform algorithm provides an efficient solution. Here, we implement the algorithm on the GPU architecture. We did the analysis tests and compared them with the direct convolution method and torch.con2d() function to demonstrate the speedup brought by the FFT implementation.

Additional Key Words and Phrases: datasets, neural networks, FFT Algorithm

ACM Reference Format:
Yichuan Su, Zhijing Yao, and Ying Xu. 2022. Fast Training of Convolution Networks through FFTs. 1, 1 (December 2022), 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As a rapidly increasing number of tasks with more challenging workloads are out to be solved, models with greater complexity are required. As a result, more tremendous amounts of data are required to utilize and exploit the power of more advanced models fully. However, significantly more time will be needed for training and inference with a larger dataset. Compared to the earlier datasets with thousands or tens of thousands of samples, today's datasets are of the order of millions. This development brings enormous challenges in performing training and inference in a feasible time without hurting the performance.
In this paper, we demonstrate the implementation of FFT algorithm into conventional convolution computation. This idea is inspired by the idea that convolution can be performed as products in the Fourier domain. Therefore, training and inference can be effective by performing the calculation in the Fourier domain and reusing the transformed feature maps.

2 METHODOLOGY

This section discusses the implementation of convolution through the FFT algorithm and the implementation of the direct convolution algorithm on the Jupyter Notebook. Although the input and kernel take Fourier Transform first, as the input sizes, kernel sizes, and batch sizes increase, the FFT convolution will take less time than the direct convolution.

2.1 Convolution based on Fast Fourier Transform Implement

The algorithm of Fast Fourier Transform is based on the equation[2]:

$$f * g = \mathcal{F}^{-1}(\mathcal{F}(f) \cdot \mathcal{F}(g))$$

Authors' address: Yichuan Su; Zhijing Yao; Ying Xu.

In the code, the Fourier transform is done by the **torch.fft.rfft**. This computes the discrete Fourier transform for the real input. Then, the element point-wise product utilizes the **Einstein summation convention** in the pytorch[3]. Finally, the result will utilize **torch.fft.irfftn** to transform back to the original domain. The Figure 1 illustrates the aforementioned overall algorithm. For convenience to implement into the CNN, the FFT Convolution class has the following parameters:

- kernel size.
- output channel
- input channel

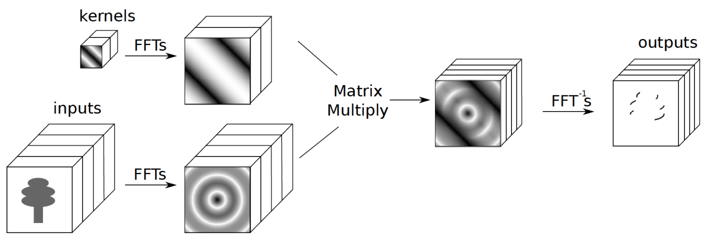

Fig. 1. Illustration of FFT Algorithm

2.2 Direct Convolution

The direct convolution is very straightforward. Unlike the method in the presentation, here, the direct method merely iterates the batch size, output channel, output height, and output weight[5]. The **unfold** function is not used here since, in the experiment, the **unfold** function speeds up the direct convolution. Similar to the FFT Convolution, The Direct Convolution class also has the following parameters:

- kernel size.
- output channel
- input channel

for the purpose of implementation in the CNN.

3 EVALUATION

Here, two experiments are performed on the Google Colab with cuda. The configuration of cuda for the first experiment is ’compute capability’: (7, 5), ’device name’: ’Tesla T4’. The configuration of cuda for the second experiment is ’compute capability’: (8, 0), ’device name’: ’A100-SXM4-40GB’. One experiment is doing the two methods of convolution computation. The other is to implement the two convolution methods within the CNN. Here the CNN has only one convolution lay. We define our own convolution network instead of using the VGG 16 discussed in the presentation. Since in VGG 16, the direct convolution will take a very long time to run. To demonstrate and reproduce the result in the paper[2] we are interested in, we replaced the VGG 16 with our network. The data sets are randomly generated by **torchvision** since the batch sizes and input sizes could be easily modified.

3.1 Convolution Computation

In this experiment, we did the convolution computation in the FFT convolution and Direct convolution. Three tests are done here. Firstly, the input will vary its batch sizes. Secondly, the input will vary its input sizes. Thirdly, the kernel sizes in the convolution will vary. The experiment results are shown below.

- **input size**.

For the simplicity, we choose batch size = 1, kernel size = 7, input channel = 1, output channel = 3. The range of the kernel is from 7 to 50.

As shown in Figure 2, the runtime of the FFT convolution is higher than the naive convolution at a small input size. However, with increasing input size, the performance gets better and better than the naive convolution.

- **kernel size**

For the simplicity, we choose batch size = 1, input channel = 1, output channel = 3, input size of the image is 32 x 32. The range of the kernel is from 2 to 14.

As shown in Figure 3, the performance of FFT convolution is generally way better than the naive convolution. With increasing kernel size, the performance of FFT convolution continues getting slightly improved.

- **batch size**

For the simplicity, we choose input size = 32, kernel size = 7, input channel = 1, output channel = 3. The range of the batch size is from 1 to 10.

As shown in Figure 4, the performance of FFT convolution is generally way better than the naive convolution. With increasing batch size, the performance of FFT convolution gets slightly improved.

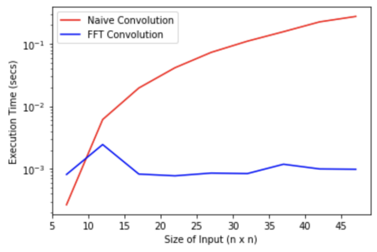

Fig. 2. Performance of FFT convolution and naive convolution with different input size

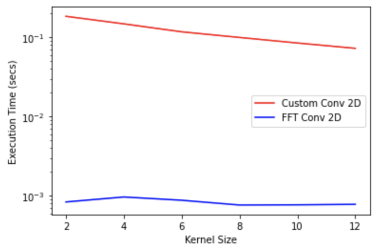

Fig. 3. Performance of FFT convolution and naive convolution with different kernel size

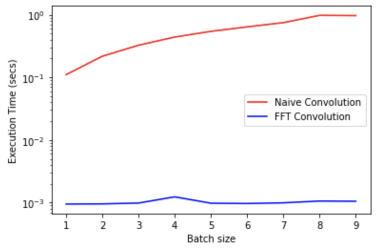

Fig. 4. Performance of FFT convolution and naive convolution with different batch size

3.2 Convolution with CNN

In this experiment, we did the convolution without CNN by varying batch sizes, input sizes, and kernel sizes [1, 4]. The experiment results are shown below.

- I. Input Sizes vs time: The batch size = 1, input channel = 3, output channel = 16 kernel size = 3.

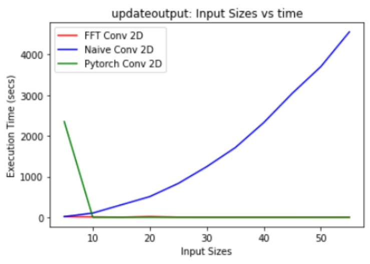

Fig. 5. Performance of FFT convolution, naive convolution, and Torch.Conv2d in the CNN with different input size. Compare their output speed

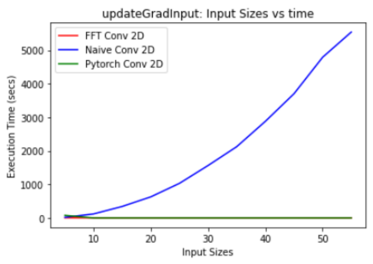

Fig. 6. Performance of FFT convolution, naive convolution, and and Torch.Conv2d in the CNN with different input size. Compare their computation speed of gradient

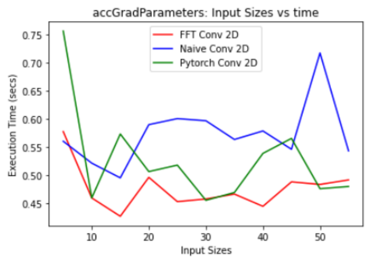

Fig. 7. Performance of FFT convolution, naive convolution, and and Torch.Conv2d in the CNN with different input size. Compare their update speed of weight

As shown in the graphs, for the updated output case and updated gradient case, the performance of FFT convolution is almost the same as the performance of PyTorch convolution, but we figured that it was reasonable. The explanation and further discussion are in the conclusion section. In the updated weight case, the FFT convolution performance is better overall than both the naive convolution and PyTorch convolution.

- II. Kernel Size vs Time: The batch size = 1, input channel = 3, output channel = 16, input size = 100

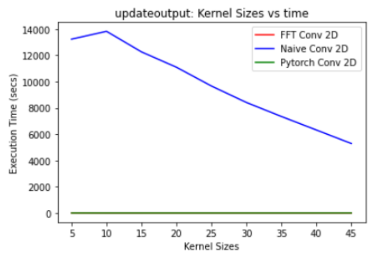

Fig. 8. Performance of FFT convolution, naive convolution, and Torch.Conv2d in the CNN with different input size. Compare their output speed

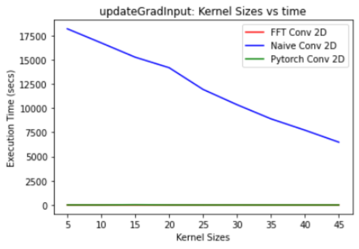

Fig. 9. Performance of FFT convolution, naive convolution, and and Torch.Conv2d in the CNN with different input size. Compare their computation speed of gradient

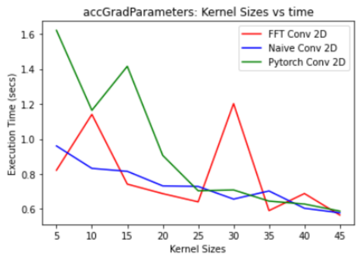

Fig. 10. Performance of FFT convolution, naive convolution, and and Torch.Conv2d in the CNN with different input size. Compare their update speed of weight

As shown in the graphs, for the updated output case and updated gradient case, the performance of FFT convolution is almost the same as the performance of PyTorch convolution, but we figured that it was reasonable. The explanation and further discussion are in the conclusion section. In the updated weight case, the performance of FFT convolution changes over kernel size: sometimes it is better than the other two methods, sometimes worse. This is because our batch size is set too small. With increasing kernel size, the number of iterations reduces, resulting in less time taken by direct convolution.

- III. Batch Size vs time The input channel = 3, output channel = 16, input size = 100, kernel size = 3

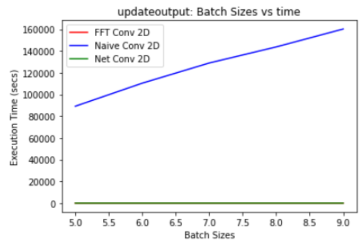

Fig. 11. Performance of FFT convolution, naive convolution, and Torch.Conv2d in the CNN with different input size. Compare their output speed

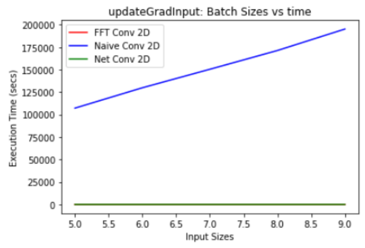

Fig. 12. Performance of FFT convolution, naive convolution, and and Torch.Conv2d in the CNN with different input size. Compare their computation speed of gradient

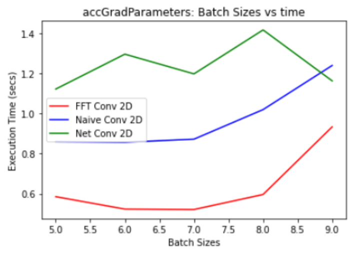

Fig. 13. Performance of FFT convolution, naive convolution, and and Torch.Conv2d in the CNN with different input size. Compare their update speed of weight

As shown in the graphs, for the updated output case and updated gradient case, the performance of FFT convolution is almost the same as the performance of PyTorch convolution, but we figured that it was reasonable. The explanation and further discussion are in the conclusion section. In the updated weight case, the FFT convolution performance is better overall than both the naive convolution and PyTorch convolution.

4 CONCLUSION AND DISCUSSION:

As shown in our evaluation section, FFT convolution generally shows a way better performance than naive convolution. However, for the updated gradient input vs. time and updated output vs. time with different input size cases, FFT convolution shows a similar performance to PyTorch convolution. This result meets our expectations and the results of the original paper. The evolution of PyTorch can probably cause the discrepancy.

We encountered several difficulties during the reproduction of the result of the paper and the implementation of FFT convolution. Some of them are solved, and some of them lead to further investigation and exploration. Firstly, when testing the performance of direct convolution, we accidentally used unfold functions which brought speedup to the direct computation.

Secondly, we set the VGG size too large at one point, resulting in too much time spent by direct convolution. To deal with this issue, we only built and tested one convolution layer, as the layer number was not specified in the paper, while a single convolution layer was enough to obtain a valid result of the comparison.

Thirdly, we were confused about how to perform a dot product without a for-loop. Eventually, we figured out that we could use the Einstein summation convention in order to utilize the built-in function of PyTorch.

After completing the full implementation and successfully verifying the results, we did gain many things. The most important takeaway was that we learned how to perform direct convolution without PyTorch. However, our results are still a little bit different from the results of the original paper. We suspect that it is due to the limitation of Colab itself. If the parameters are set as same as the original paper setting, it will take a long time to complete as it would be hard for direct convolution.

Our further work will explore the effect of the slight change between the direct convolution, FFT convolution, and PyTorch convolution with the same input, weight, and bias.

REFERENCES

[1] Ilya Sutskever Alex Krizhevsky and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. *NIPS* (March 2012), pages 1106–1114.

[2] Michael Mathieu and Mikael Henaff. 2014. Fast Training of Convolutional Networks through FFTs. (March 2014).

[3] Frank Odom. 2020. *Fourier Convolutions in PyTorch.* Retrieved Dec 13, 2022 from https://towardsdatascience.com/fourier-convolutions-in-pytorch-4cbd23c70005

[4] Koray Kavukcuoglu Ronan Collobert and Clement Farabet. 2011. Torch7: A matlab-like environment for machine learning. *NIPS* (March 2011).

[5] Anirudh Shenoy. 2019. *How Are Convolutions Actually Performed Under the Hood?* Retrieved Dec 13, 2022 from https://towardsdatascience.com/how-are-convolutions-actually-performed-under-the-hood-226523ce7fbf