DSXplore: Optimizing Convolutional Neural Networks via Sliding-Channel Convolutions

Yuke Wang, Boyuan Feng, Yufei Ding
UC Santa Barbara, CA, USA

yuke_wang@cs.ucsb.edu
Diverse Convolutions in CNNs
Challenges

First, there are limited DSC designs to balance accuracy performance and the size of computation/parameters.

Second, there is a lack of efficient implementation support for new factorized kernels.

- Derived from the **DW+PW** design for standard convolution replacement. The more effective DSC schemes that can potentially deliver better accuracy and model size trade-offs still remain uncovered.
- For example, we can further reduce the computation cost and parameter size by combining the group convolution (GC) (dividing the input and output channel into the same number of groups and only applying standard convolution within each group) with PW.

- Rely on the deep-learning infrastructure with standard/group convolutions for their factorized kernel implementation.
- For example, the DW convolution can be expressed as the extreme case of the GC with the number of groups equal the input channels, while the PW convolution can be expressed as another special case of standard convolution with the $1 \times 1$ kernel spatial dimension.
- Therefore, the better factorized kernel that may bring better accuracy and lower computation and memory costs but not in the above categories cannot leverage the existing convolutional primitives for an effective implementation.
Contributions

- A novel Sliding-channel Convolution (SCC) design.
  - Balance the accuracy performance and the reduction of computation and memory cost.
  - Enormous design exploration space with parameterized design strategy.

- An optimized GPU-implementation tailored for SCC design.
  - Output-centric forward and input-centric backward optimization
  - Optimization based on the convolutional specialty (cyclic channel) of its filters.

- Seamless integration with the original Pytorch framework.
  - Drop-in replacement of the existing DSCs to facilitate the training and inference of an end-to-end fashion.

- Extensive Experiments.
  - Better accuracy and lower computation/memory cost compared with the existing DSC.
Comparison with existing kernels

![Diagram showing comparison among PW, GPW, and SCC with 3 channels each.]

**TABLE I: Comparison among SCC, PW, and GPW.**

<table>
<thead>
<tr>
<th>Convolution</th>
<th>FLOPs</th>
<th>Params.</th>
<th>Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PW†</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>GPW*</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>SCC</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

†: PW can be seen as SCC with 1 group with 100% channel overlapping of adjacent filters.

*: GPW can be seen as SCC with $m$ groups with 0% channel overlapping of adjacent filters. $m$ is a parameter that can be determined by users.
Pytorch-based SCC Design

(a) Channel-Stack

(b) Convolution Stack
Optimized CUDA SCC Kernel
Channel-cyclic Optimizations

Algorithm 1: Compute channel indexes of one cycle.

1. channel_map = {};
2. group_width = input_channel/num_groups;
3. start, end = 0, group_width;
4. start_v, end_v = start, end;
5. cyclic_dist = 0;
6. for oid = 0; oid < output_channel; oid++ do
   7. item = (start, end);
   8. if item not in channel_map then
      9. channel_map.add(item);
      10. cyclic_dist ++;
   11. end
   12. else
      13. break;
   14. end
   15. start_v = end_v - int(overlap * group_width);
   16. end_v = start_v + group_width;
   17. start = start_v % input_channels;
   18. end = end_v % input_channels;
19. end

Algorithm 2: DSxplode Channel-cyclic Optimization

1. thread_id = blockIdx.x * blockDim.x + threadIdx.x;
2. opt_channel_id = get_output_channel(thread_id);
3. idx = output_channels % cyclic_dist;
4. start, end = channel_map[idx];

Fig. 6: Case study of cyclic-channel optimization on Pytorch-based implementations for $C_{in} = 4$, $cg = 2$, $co = 50\%$. 
**Experiment: Accuracy**

**TABLE II: Accuracy comparison of CNNs on CIFAR-10.**

<table>
<thead>
<tr>
<th>Model</th>
<th>Implementation</th>
<th>MFLOPS</th>
<th>Param. (M)</th>
<th>Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG16</td>
<td>Origin</td>
<td>314.67</td>
<td>14.73M</td>
<td>92.64</td>
</tr>
<tr>
<td></td>
<td>DSXplore</td>
<td>94.39</td>
<td>0.85M</td>
<td>92.60</td>
</tr>
<tr>
<td>VGG19</td>
<td>Origin</td>
<td>399.75</td>
<td>20.02M</td>
<td>93.88</td>
</tr>
<tr>
<td></td>
<td>DSXplore</td>
<td>114.22</td>
<td>1.16M</td>
<td>92.71</td>
</tr>
<tr>
<td>MobileNet</td>
<td>Origin</td>
<td>67.31</td>
<td>3.19M</td>
<td>92.05</td>
</tr>
<tr>
<td></td>
<td>DSXplore</td>
<td>45.29</td>
<td>1.63M</td>
<td>92.02</td>
</tr>
<tr>
<td>ResNet18</td>
<td>Origin</td>
<td>581.63</td>
<td>11.17M</td>
<td>95.75</td>
</tr>
<tr>
<td></td>
<td>DSXplore</td>
<td>298.63</td>
<td>0.54M</td>
<td>94.81</td>
</tr>
<tr>
<td>ResNet50</td>
<td>Origin</td>
<td>2036.01</td>
<td>23.52M</td>
<td>95.82</td>
</tr>
<tr>
<td></td>
<td>DSXplore</td>
<td>1469.64</td>
<td>12.81M</td>
<td>95.67</td>
</tr>
</tbody>
</table>

**TABLE III: Accuracy comparison (ImageNet) for ResNet50.**

<table>
<thead>
<tr>
<th>Network</th>
<th>MFLOPs</th>
<th>Param.</th>
<th>Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>4130</td>
<td>23.67M</td>
<td>76.56</td>
</tr>
<tr>
<td>DSXplore</td>
<td>2550</td>
<td>14.34M</td>
<td>75.91</td>
</tr>
</tbody>
</table>
Experiment: **Accuracy (Cont’d)**

**TABLE IV: Comparison of different settings on MobileNet.**

<table>
<thead>
<tr>
<th>Network</th>
<th>MFLOPs</th>
<th>Param.</th>
<th>Acc.(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (DW+PW)</td>
<td>67.31</td>
<td>3.19M</td>
<td>92.05</td>
</tr>
<tr>
<td>DW+GPW-cg2</td>
<td>45.29</td>
<td>1.63M</td>
<td>90.11</td>
</tr>
<tr>
<td>DW+GPW-cg4</td>
<td>34.28</td>
<td>0.84M</td>
<td>88.88</td>
</tr>
<tr>
<td>DW+GPW-cg8</td>
<td>28.78</td>
<td>0.45M</td>
<td>82.69</td>
</tr>
<tr>
<td>DW+SCC-cg2-co25%</td>
<td>45.29</td>
<td>1.63M</td>
<td>92.02</td>
</tr>
<tr>
<td>DW+SCC-cg2-co50%</td>
<td>45.29</td>
<td>1.63M</td>
<td>91.36</td>
</tr>
<tr>
<td>DW+SCC-cg4-co25%</td>
<td>34.28</td>
<td>0.84M</td>
<td>90.63</td>
</tr>
<tr>
<td>DW+SCC-cg4-co50%</td>
<td>34.28</td>
<td>0.84M</td>
<td>90.60</td>
</tr>
<tr>
<td>DW+SCC-cg8-co25%</td>
<td>28.78</td>
<td>0.45M</td>
<td>88.92</td>
</tr>
<tr>
<td>DW+SCC-cg8-co50%</td>
<td>28.78</td>
<td>0.45M</td>
<td>89.23</td>
</tr>
</tbody>
</table>
Experiment: Performance

Fig. 7: Runtime performance comparison on CIFAR10. Note that speedup is normalized \textit{w.r.t.} Pytorch-Base Implementation.

Fig. 8: Runtime performance comparison on ImageNet. Note that speedup is normalized \textit{w.r.t.} Pytorch-Opt Implementation.
Experiment: Additional Studies (cont’d)

Fig. 9: Back-propagation optimization.

Fig. 10: Channel-cyclic optimization.
Experiment: Additional Studies (con’d)

Fig. 11: The performance impact of the number of groups (cg). Note that we set co = 50% and the runtime is normalized w.r.t the performance at cg = 1.

Fig. 12: The performance impact of the input-channel overlapping ratio (co). Note that we set cg = 2 and the runtime is normalized w.r.t the performance at co = 10%.

Fig. 13: Impact of batch size on training performance.
Thank You

Q & A