

1 ARTIFACT DESCRIPTION APPENDIX: SOL: TRANSPARENT NEURAL NETWORK ACCELERATION PLATFORM

1.1 Abstract

This appendix contains the technical details for our experiments.

1.2 Description

1.2.1 Check-list (artifact meta information).

- **Program:** Python and C++ libraries
- **Compilation:** Python 3.5.3, GCC 6.3.0-18 and ISPC 1.9.2
- **Binary:** C++ library and JIT compiled OpenMP+ISPC libraries
- **Data set:** GoogleNet and Resnet-18
- **Run-time environment:** Debian 9.5 (Kernel 4.9.0-3)
- **Hardware:** 2x Intel Xeon E5-2637 v4
- **Output:** runtime comparison between PyTorch with and without Sol
- **Experiment workflow:** build program, run test scripts
- **Publicly available?:** no

1.2.2 *How software can be obtained (if available).* As SOL is still under development, code is unavailable at the moment.

1.2.3 *Hardware dependencies.* SOL requires at least an x86/64 capable Intel CPU.

1.2.4 *Software dependencies.* We evaluated SOL on Debian 9.5 (Kernel 4.9.0-3), GCC 6.3.0-18, ISPC 1.9.2¹, PyTorch 0.4.1², Python 3.5.3 and Intel MKL-DNN³ 0.15.

1.2.5 Datasets.

- GoogleNet⁴
- Resnet-18⁵

1.3 Experiment workflow

To run our experiments, we used the following python script. To enable the DNN optimizations, we used an environmental variable (SOL_NODNN) to activate or disable the DNN optimizations. In this case, all layers that are not optimized by the DFP optimizations use the default PyTorch implementations. Due to the current structure of SOL, the DFP optimizations cannot be disabled.

```
#!/usr/bin/python3.5
import torch
import torch.nn as nn
import sol.pytorch as sol
import torchvision.models as models
from timeit import default_timer as timer
from googlenet import GoogLeNet
```

```
torch.manual_seed(0)
```

```
if (False):
    size = [0, 3, 32, 32]
```

```
pm = GoogLeNet()
else:
    size = [0, 3, 224, 224]
pm = models.__dict__["resnet18"]()

sm = sol.optimize(pm, size)
pm.NAME = 'PyTorch'
sm.NAME = 'Sol'
pm.eval()
sm.eval()

with torch.no_grad():
    for model in [sm, pm]:
        for x in [16, 32, 64, 128, 256]:
            T = torch.rand(x, *size[1:])
            time = 9999999999
            for x in range(0, 20):
                start = timer()
                model(data)
                end = timer()
                time = min(time, end-start)
            print('%-8s_%3i_%.72f' % (model.NAME,
                                    x, time*1000))
```

¹<http://ispc.github.io>

²<https://pytorch.org>

³<https://github.com/intel/mkl-dnn>

⁴<https://github.com/kuangliu/pytorch-cifar/blob/master/models/googlenet.py>

⁵<https://github.com/pytorch/vision/tree/master/torchvision>