MXNet Graph Optimization and Quantization based on subgraph and MKL-DNN

- **Credit to Zhennan for this proposal** 😎

Purpose

Two advanced features, fused computation and reduced-precision kernels, are introduced by MKL-DNN in the recent version [1,2]. These features can significantly speed up the inference performance on CPU for a broad range of deep learning topologies.

However, MXNet still cannot benefit from them because of the limitation of graph representation and the lack of graph optimization previously. Fortunately, the new subgraph feature [3] of MXNet makes these improvements possible now.

This document serves the purpose to illustrate the subgraph based solution to leverage MKL-DNN's new features into MXNet in Q3'2018. In general, the new solution will partition MKL-DNN operators into a subgraph, and then replace those MKL-DNN operators with fused kernels if possible, within the subgraph. Furthermore, users can choose the quantization flow to accelerate the inference procedure of their models by using the reduced-precision kernels of MKL-DNN, such as INT8 kernels.

Milestone

The whole project of this proposal is targeting to be released in **1.4.0** of MXNet. Three steps in the above can be considered as milestones of this project and used to track the process.

- DONE, PR#12157 the subgraph branch into the master branch based on [3]
- DONE, MKL-DNN fusion implementation into subgraph branch
- DONE, PR#12530 (PR#12663) submit the quantization flow implementation into the master branch

Enable this new feature with master branch CI: 213ab09e7a2924da436c0d0526d62fefeeea6aa7

```
$ make -j USE_OPENCV=1 USE_MKLDNN=1 USE_BLAS=mkl
```

```
$ export MXNET_SUBGRAPH_BACKEND=MKLDNN
```

Refer: https://github.com/apache/incubator-mxnet/blob/master/MKLDNN_README.md

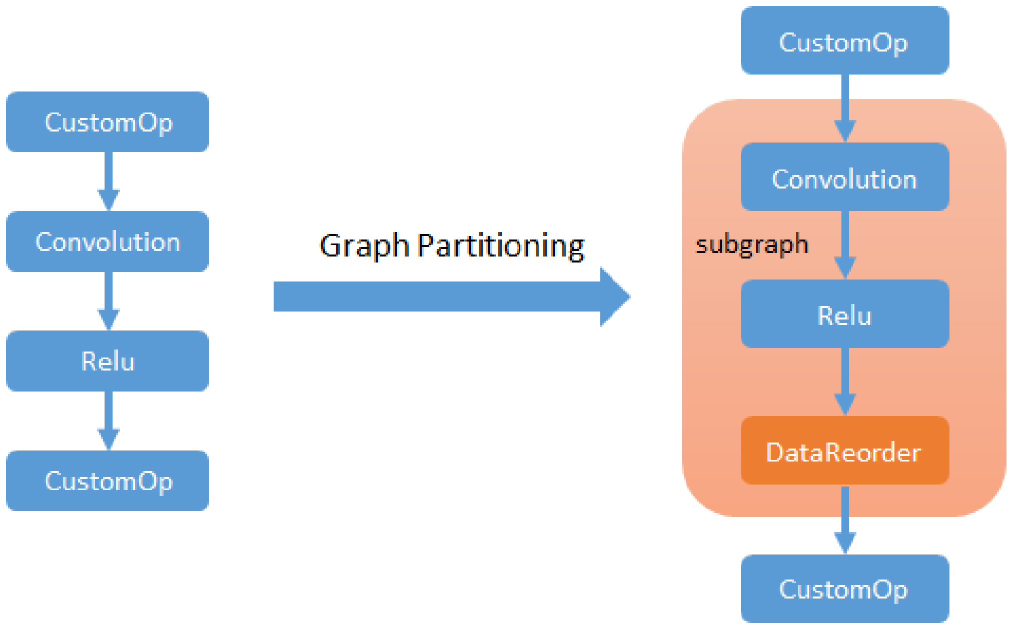
Workflow

Step 1. Partition MKL-DNN operators into the subgraph

This is one of the main purposes of the subgraph. More details are described at "Unified integration with external backend libraries" [3] . To achieve this, we can list all operators MKL-DNN supported and pass it to DefaultSubgraphProperty. After applying graph partitioning, all MKL-DNN operators will group into subgraph node, and data format will be converted between MKL-DNN internal format and NDArray default format on the subgraph boundary automatically. Subgraph border is a bit different according to executing mode.

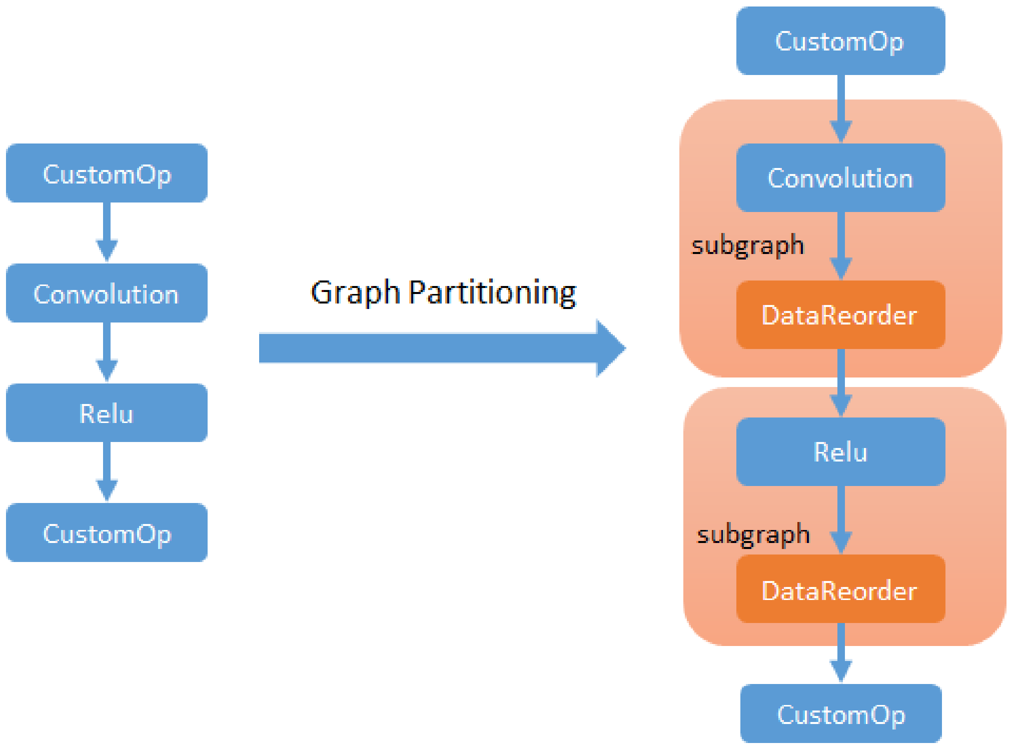
1. Symbolic mode

Subgraph will try to cover adjacent MKL-DNN operators as much as possible.



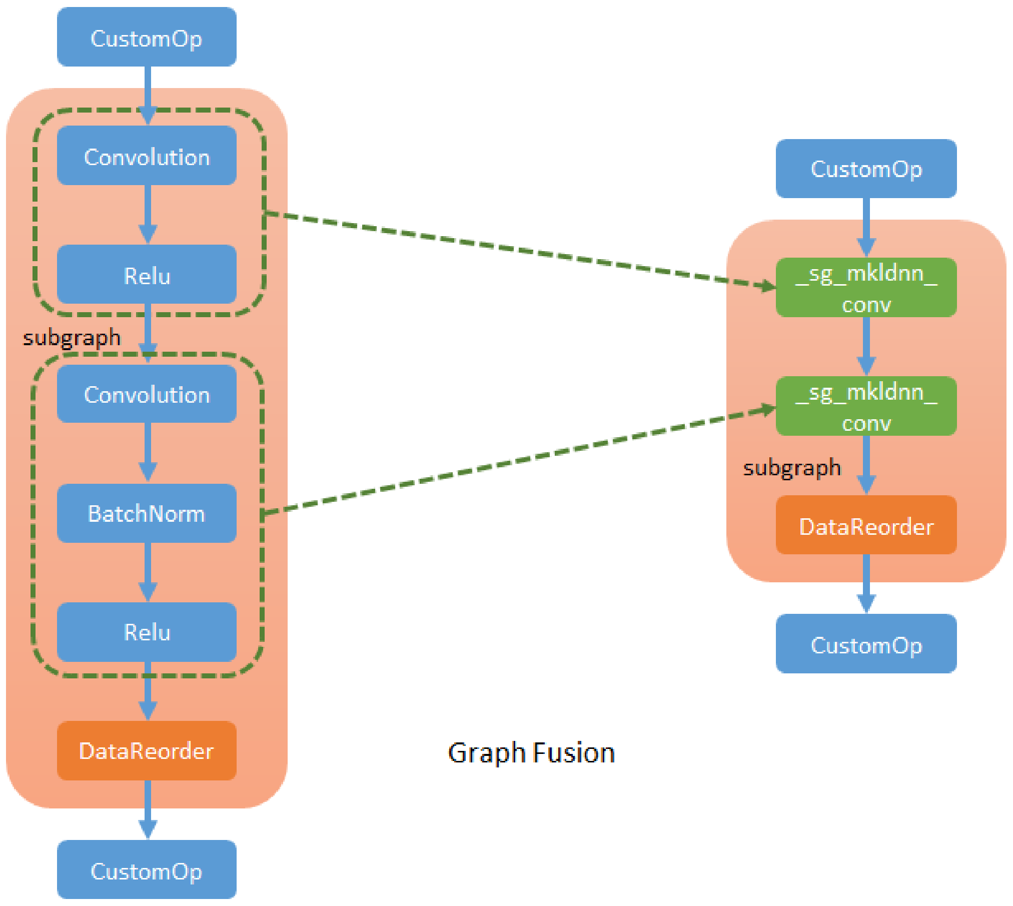
2. Imperative mode

Each MKL-DNN operator will execute in an independent subgraph to ensure MKL-DNN internal format won't be exposed outside subgraph.



Step 2. MKL-DNN operator fusion

MKL-DNN library supports running several certain patterned operators in a single execution. Such as convolution + relu. We can define new SubgraphSelector to capture such operator pattern and generate new MKL-DNN specific operator to replace the original ones.



New fused operators are standalone operators which represent the operations MKL-DNN library defined. For example, in MKL-DNN library, convolution can support post operations that allow executing relu following convolution. So, for convolution + relu fusion, we will create a MKL-DNN convolution (named _sg_MKL-DNN_conv) and describe relu as a post operation of it.

```
NNVM_REGISTER_OP(_sg_MKL-DNN_conv)
.set_attr<FStatefulComputeEx>("FStatefulComputeEx<cpu>",   SgMKL-DNNConvOpForward)
```

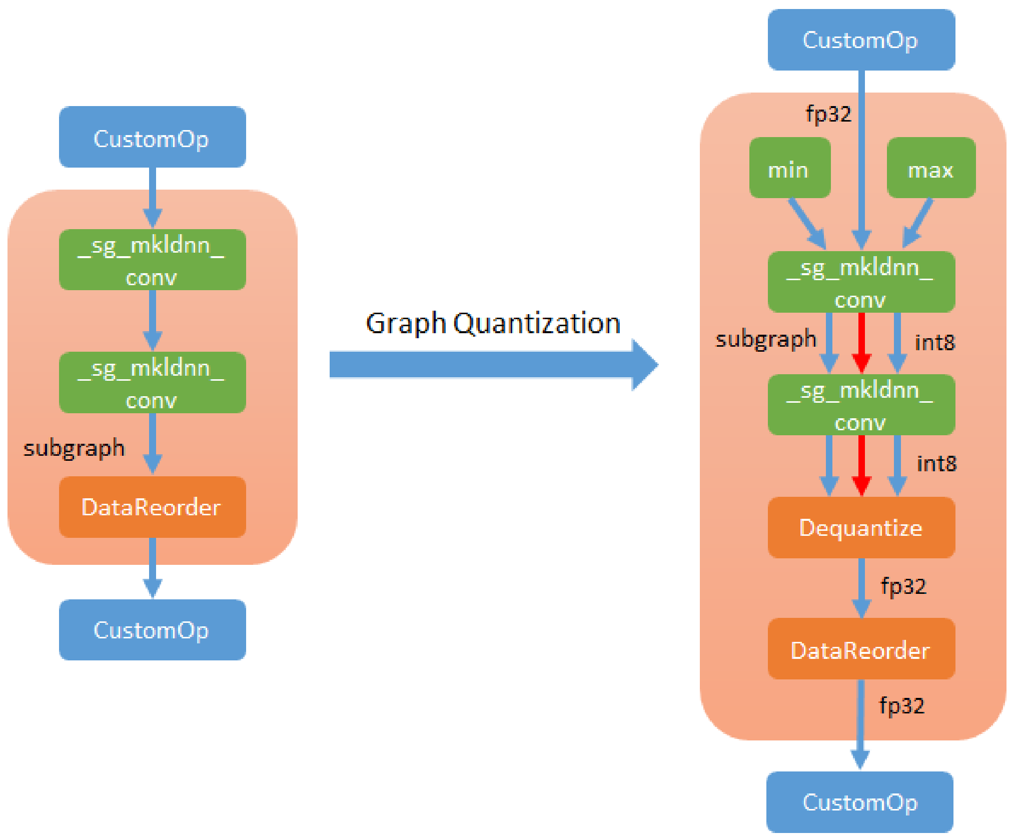
In general, new fused operators follow the abstraction of MKL-DNN library, and subgraph fusion pass is the lowering process to convert NNVM common graph to MKL-DNN graph. Fusion pass only happens inside the subgraph created by step1, so won't affect MXNet default operators.

Step 3. Quantization

MKL-DNN supports most reduced precision primitives in convolutional neural networks, especially for fused primitives. MXNet INT8 inference consists of two steps:

1. Prepare INT8 model, quantize parameters and collect calibration data

We perform this step only once. To create INT8 model based on FP32 model, we will run QuantizeGraph pass inside subgraph to replace FP32 operators with INT8 operators if MKL-DNN supports and insert dequantize operator on proper position.



2. Run INT8 inference

When INT8 symbols and parameters are ready for inference, the user can run inference on new input data which is the same as before.

Accuracy Validation

Regardless of quantization, subgraph solution won't introduce accuracy lost itself, on the contrary, it will enhance framework stability when using MKL-DNN.

NETWORK	**FP32**		**FP32 with Fusion**	
	Top1	Top5	Top1	Top5
Resnet-152 by MKL-DNN	77.16	92.98	77.16	92.98
Inception by MKL-DNN	72.36	90.58	72.36	90.58

For quantization enabled case, MKL-DNN can get huge performance improvement within less than 0.5% accuracy loss comparing with FP32 with entropy mode calibration.

Accuracy data from the internal branch are shown in below table [4].

	FP32		**INT8**				
			No Calibration		**Calibration using 5 batches**		**Calibration Method**
NETWORK	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**	
Resnet-152 by GPU	77.19	93.01	75.56	92.32	75.58	92.24	**Threshold by min/max**
					75.65	92.35	**Threshold by entropy loss**
Resnet-152 by MKL-DNN	77.16	92.98	76.59	92.71	76.44	92.69	**Threshold by min/max**
					76.76	92.79	**Threshold by entropy loss**
Inception-bn by GPU	72.38	90.61	71.98	90.26	71.78	90.26	**Threshold by min/max**
					71.98	90.36	**Threshold by entropy loss**
Inception-bn by MKL-DNN	72.36	90.58	72.21	90.50	72.16	90.43	**Threshold by min/max**
					72.19	90.45	**Threshold by entropy loss**

Performance

On performance part, both fusion and quantization can provide the huge improvement on various kinds of topologies.

Below is the inference performance data(images/sec) from the internal branch based on SKX-8180 1 socket with batch size 64.

Topology	Base	With Fusion	With Fusion + Quantization
resnet50-v1	208.17	348.80	568.25
resnet50-v2	198.87	256.10	
vgg-16	92.11	101.00	150.92
inception-bn	475.74	658.75	836.94
inception-v3	175.97	227.29	327.39
inception-v4	88.99	109.15	
inception-resnet-v2	104.13	124.75	
mobilenet 1.0	668.71	1380.64	1788.95
squezenet	708.63	849.27	975.90

Testcase

The quantization solution is still under development. We will share more data when it's ready.

Tests need to cover 2 parts. First one is the graph conversion test. We need to ensure that:

Step	Criterion
1	All MKL-DNN operators are partitioned into one or more subgraphs according to executing mode.
2	Desired patterns can be captured and desired fused operators will be created.
3	Quantization pass can convert desired operators to the quantized version with the correct data connection.

Another one is the unit test for MKL-DNN specific fused operators. The test should cover all the fusion scenarios to ensure the fused operators can provide the accurate result.

Q & A

- How will a user invoke the feature - what is the workflow, commands and API's to use?
  By the environment variable in the first version, MXNET_SUBGRAPH_BACKEND. This feature will be enabled after its mature.
- What API enhancements (or changes) are you planning? Any concerns about backward compatibility?
  This change applied the API provided by Proposal [3] and NO backward compatibility issues.
- What are the planned unit and integration tests? E2E testing with complete models is good, but I think we need also small tests we can execute with PR checks and regular regression.
  There will be the fully tests including unit test and model-level test. The unit test will be added to check the correctness for both generated graph and the output of fused OP.
  The model-level test will check the training and inference accuracy for the new change.
- Please add as well non-CV test cases.
  Sure, we have tested the RNN, GAN, RL network as well. For example, validate the performance and functionality by sockeye models (GNMT, transformers).
- Am I right in assuming that users will choose operators completely independent of the underlying backends?
  Not exactly. The backend flow doesn't changes with subgraph at this stage.
  The user specific the backend by building variable (USE_MKLDNN, USE_CUDNN) and ctx (gpu,cpu) at runtime.
- Does that mean that if I want a convolution, for example, mxnet will automatically choose between cpu, GPU and mkldnn? If so, how is this done exactly?
  NO, following the current MXNet backend usage, user can't switch single OP between different backend.
- I think it is important that we have a clear separation in between operators and their different implementations. In the end, users should be able to define a network graph and it's our responsibility to find the most optimal way to execute it.
  Agree. Finally, we should reach this status but it's not the goal in this step.

Reference

1. Intel MKL-DNN, Introduction to Low-Precision 8-bit Integer Computations
   http://intel.github.io/mkl-dnn/ex_int8_simplenet.html
2. Intel MKL-DNN Post OPs Struct Reference
   http://intel.github.io/mkl-dnn/structmkldnn_1_1post__ops.html
3. Da Zheng, Unified integration with external backend libraries
   https://cwiki.apache.org/confluence/display/MXNET/Unified+integration+with+external+backend+libraries
4. Wu Jun, Model Quantization with Calibration
   https://github.com/apache/incubator-mxnet/files/1662073/quantization_github.pptx
   https://github.com/apache/incubator-mxnet/pull/9552