# IMPLEMENT CUSTOM LAYERS WITH INTEL® DISTRIBUTION OF OPENVINO™ TOOLKIT

September 2019

# Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness or any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804.

# Legal Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software, or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at www.intel.com.

Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

This document contains information on products, services, and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications, and roadmaps.

Any forecasts of goods and services needed for Intel's operations are provided for discussion purposes only. Intel will have no liability to make any purchase in connection with forecasts published in this document.

Arduino* 101 and the Arduino infinity logo are trademarks or registered trademarks of Arduino, LLC.

Altera, Arria, the Arria logo, Intel, the Intel logo, Intel Atom, Intel Core, Intel Nervana, Intel Xeon Phi, Movidius, Saffron, and Xeon are trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

# Legal Notices and Disclaimers

This document contains information on products, services, and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications, and roadmaps. Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software, or service activation. Learn more at intel.com or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Statements in this document that refer to Intel's plans and expectations for the quarter, the year, and the future are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel's results and plans is included in Intel's SEC filings, including the annual report on Form 10-K.

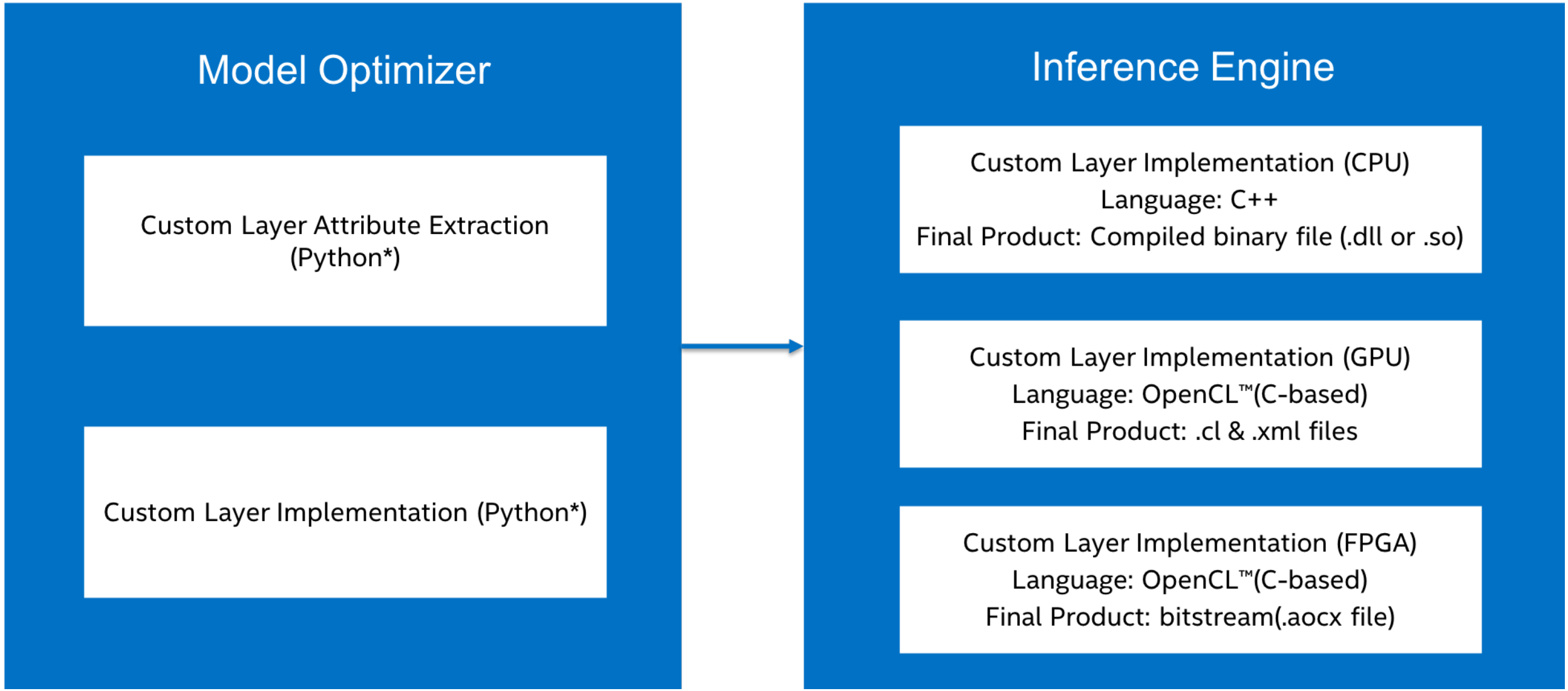
The products described may contain design defects or errors, known as *errata,* which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

# Custom Layer Workflow



# Extension Generator Tool

Extension == Custom Layer == Kernel

- Python* based tool (extgen.py)
- Generates template/stub files with core functions for creating custom layers.

# Extension Generator Tool

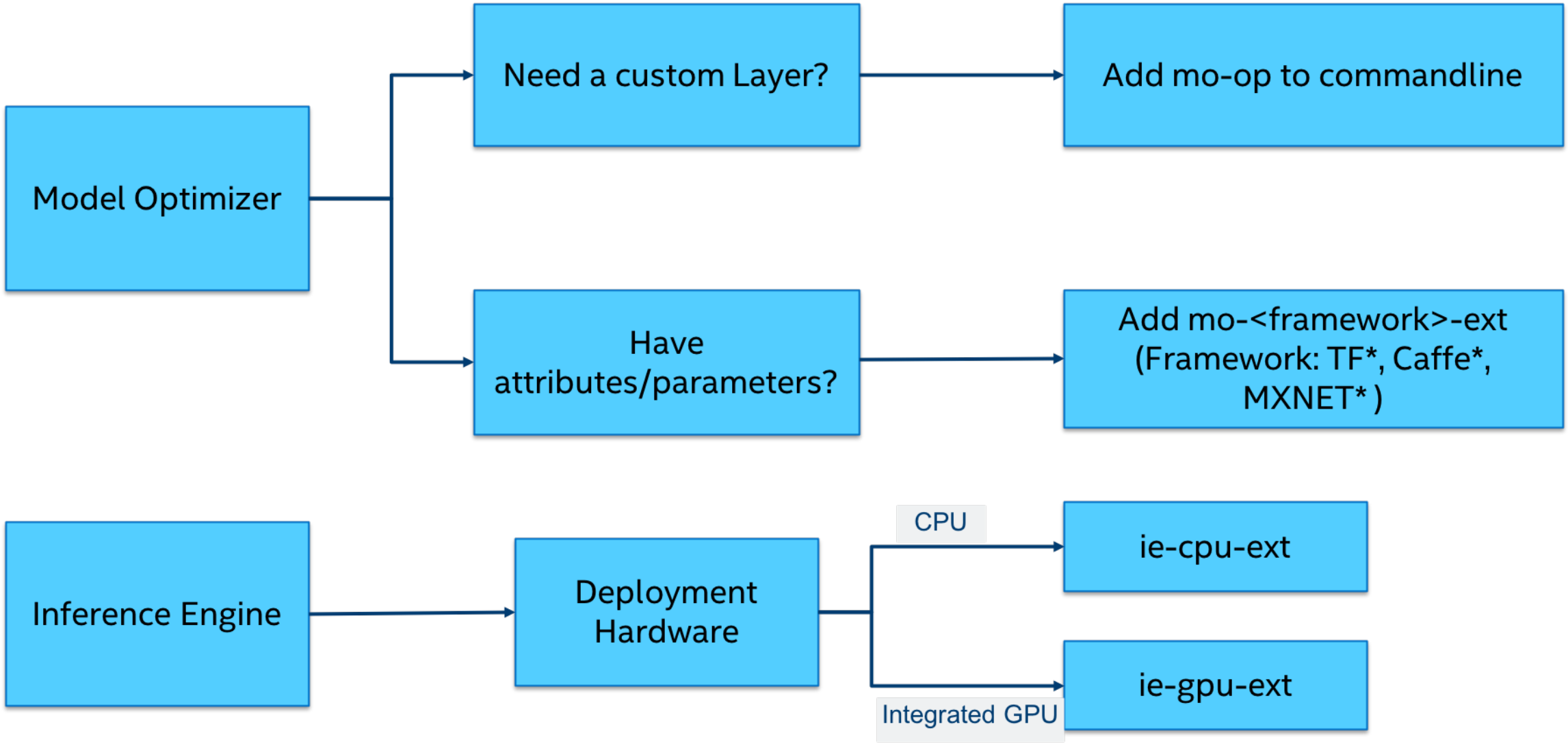
To run the tool in the interactive mode, specify the following parameters:

- mo-op - To generate a Model Optimizer operation
- mo-caffe-ext - To generate a Model Optimizer Caffe* extractor
- mo-mxnet-ext - To generates a Model Optimizer MXNet* extractor
- mo-tf-ext - To generate a Model Optimizer TensorFlow* extractor
- ie-cpu-ext - To generate an Inference Engine CPU extension
- ie-gpu-ext - To generate an Inference Engine GPU extension
- output_dir - To set an output directory. If not specified, the current directory is used by default.

Example

```
python3 extgen.py new mo-op mo-tf-ext ie-cpu-ext
```

# Extension Generator Flow Chart



# Extending Custom Layer to Model Optimizer

# Directory Structure for Model Optimizer and Inference Engine extension files

- /user_mo_extensions
  - __init__.py
  - /front
    - /caffe
      - __init__.py
    - /mxnet
      - __init__.py
    - /tf
      - __init__.py
  - /ops
    - __init__.py
- /user_ie_extensions
  - /cpu
  - /gpu

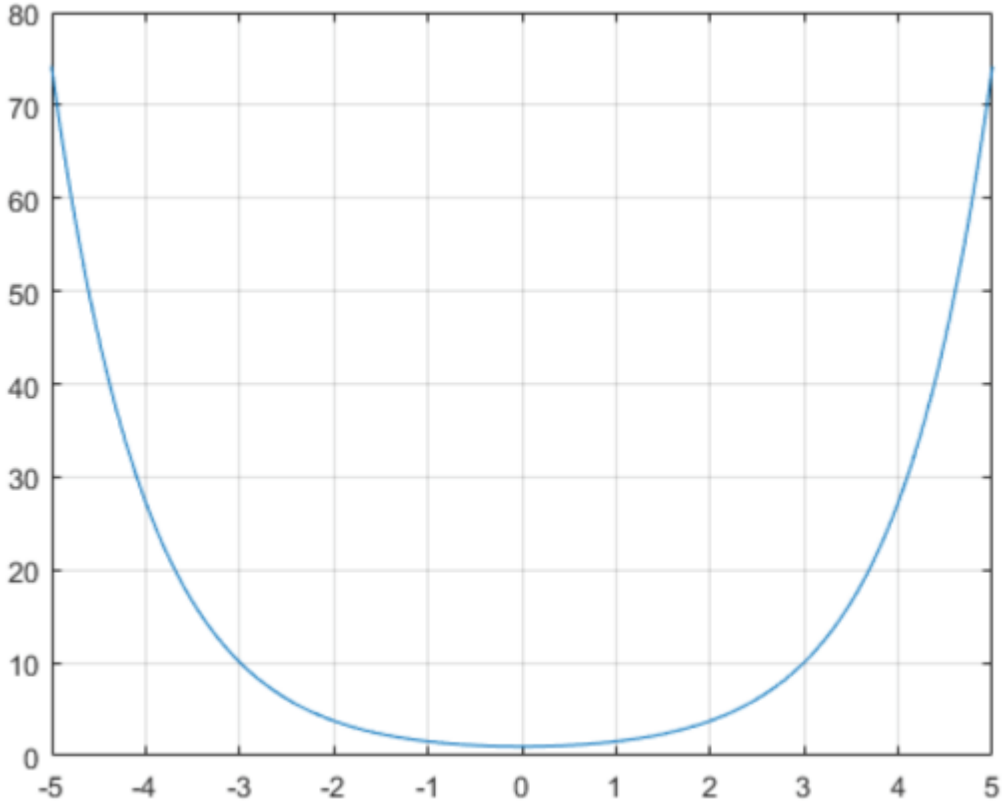
Extraction of Model's attributes for the targeted framework

Custom Layer implementation to calculate output shapes and attributes in IR

Inference Implementation of Custom Layer

# Custom Layer Example: Cosh – Hyperbolic cosine

$$\cosh x = \frac{e^{x} + e^{-x}}{2}$$



# Ops Folder-cos.py

```python
from mo.front.common.partial_infer.elemental import copy_shape_infer
from mo.ops.op import Op
from mo.graph.graph import Node

class Cosh(Op):
    op = 'Cosh'
    enabled = True

    def __init__(self, graph, attrs: dict):
        super().__init__(graph, {
            'type': __class__.op,
            'op': __class__.op,
            'infer': Cosh.infer,
        }, attrs)

    @staticmethod
    def infer(node: Node):
        # we just set the same shape to the output
        copy_shape_infer(node)
```

Represents if the operation should be used by Model Optimizer or excluded

Constructor is automatically generated

Sets the Output shape to the Input shape

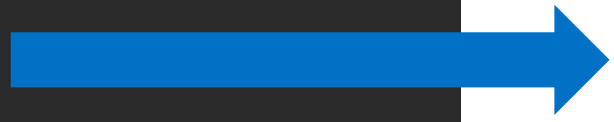
# Front/<Framework>: cosh_ext.py

**Framework**: TensorFlow*, Caffe*, MXNet*, ONNX*

```
from mo.front.extractor import FrontExtractorOp
from mo.ops.cosh import Cosh


class CoshFrontExtractor(FrontExtractorOp):
    op = 'Cosh'
    enabled = True

    @staticmethod
    def extract(node):
        Cosh.update_node_stat(node)
        return __class__.enabled
```



Constructor is automatically generated

# Generate IR with Custom Layer files using Model Optimizer

- The --extension arguments are used by Model Optimizer to parse the files in the directory provided on the command line as shown below.

```
python3 mo.py --input_model <model> --extension <path to directory with files>
```

```
python3 mo.py --input_model <model> --extension ../extension_generator/
```

# Extending Custom Layer to Inference Engine

# Inference Engine CPU extension (.cpp)

```
namespace InferenceEngine {
namespace Extensions {
namespace Cpu {

class CoshImpl: public ExtLayerBase {
public:
    explicit CoshImpl(const CNNLayer* layer) {
        try {
            if (layer->insData.size() != 1 || layer->outData.empty())
                THROW_IE_EXCEPTION << "Incorrect number of input/output edges!";

            addConfig(layer, {{ConfLayout::PLN, false, 0}}, {{ConfLayout::PLN, false, 0}});
        } catch (InferenceEngine::details::InferenceEngineException &ex) {
            errorMsg = ex.what();
        }
    }
```

addConfig is important!!

# AddConfig

- addConfig(layer, {{ConfLayout::PLN, false, 0}}, {{ConfLayout::PLN, false, 0}});

addConfig(layerpointer, DataConfig Vector(input), DataConfig Vector(Output)

- Layer pointer
- ConfLayout is the memory layout for the input and output data which can be PLN(planar), ANY(finds the best suitable option), etc. for more information look at IE_Common.h
- False indicates that the flag for determination of the constant memory is disabled
- 0 indicates the index of in-place memory. If -1 memory cannot be in-place.

# Inference Engine CPU extension Cont. (.cpp)

```
    StatusCode execute(std::vector<Blob::Ptr>& inputs, std::vector<Blob::Ptr>& outputs,
                       ResponseDesc *resp) noexcept override {
        // Add implementation for layer inference here
        // Examples of implementations are in OpenVINO samples/extensions folder
        float* src_data = inputs[0]->buffer();
        float* dst_data = outputs[0]->buffer();

        SizeVector dims = inputs[0]->getTensorDesc().getDims();

        int N = static_cast<int>((dims.size() > 0) ? dims[0] : 1);
        int C = static_cast<int>((dims.size() > 1) ? dims[1] : 1);
        int H = static_cast<int>((dims.size() > 2) ? dims[2] : 1);
        int W = static_cast<int>((dims.size() > 3) ? dims[3] : 1);

        //hyperbolic cosine is given by : (e^x + e^-x)/2
        parallel_for(N*C*H*W, [&](int ii) {
                dst_data[ii] = (exp(src_data[ii]) + exp(-src_data[ii]))/2;
        });
        return OK;
    }

private:
};

REG_FACTORY_FOR(ImplFactory<CoshImpl>, Cosh);

}  // namespace Cpu
}  // namespace Extensions
}  // namespace InferenceEngine
```

Create input and output buffers for data

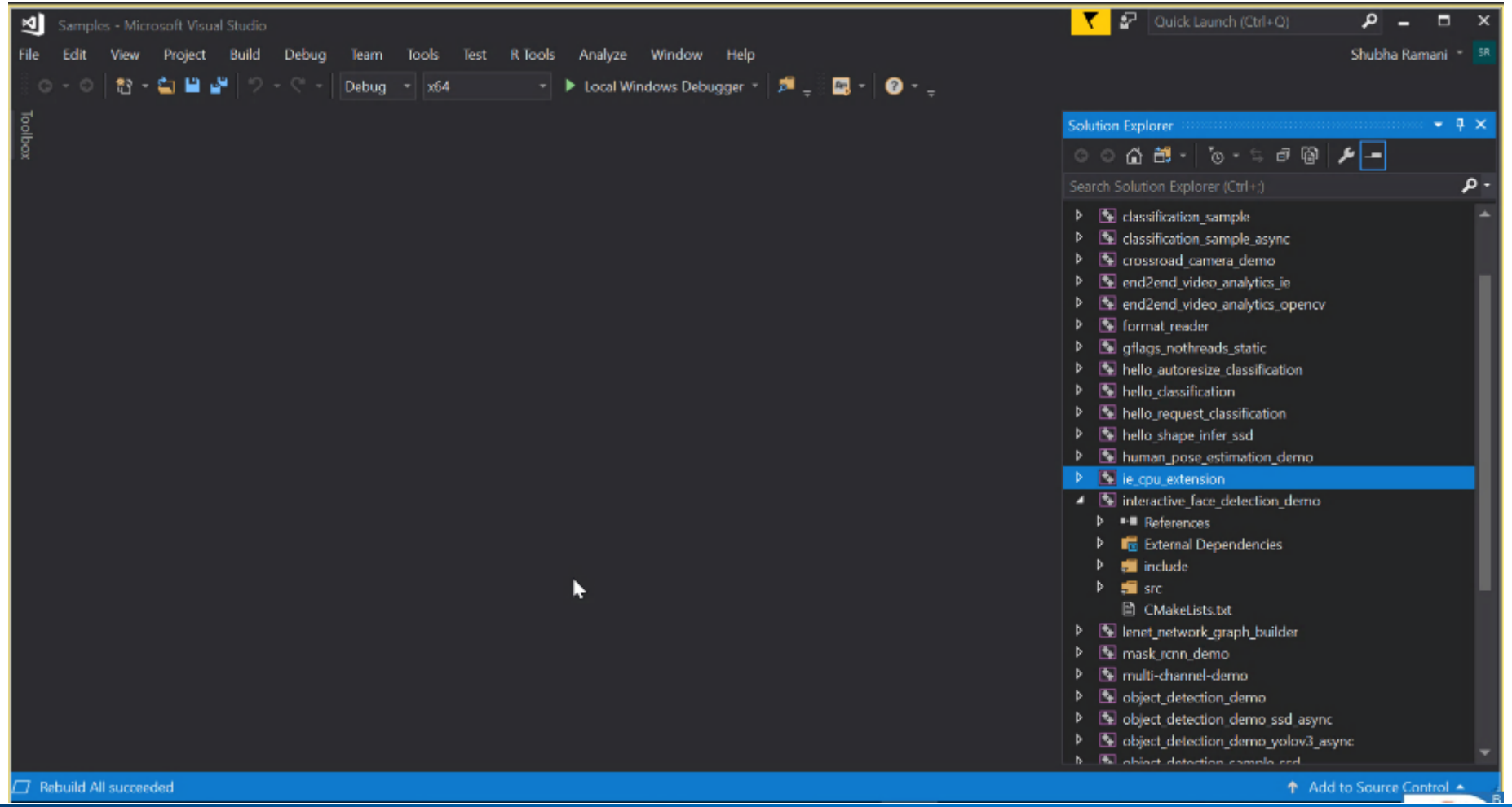
Get input dimensions:
N: Number of images (batch)
C: # of channels (feature/depth)
H: Height of image
W: Width of Image

Implementation of operation for inference across tensor

Register custom layer to IE

# Compiling CPU custom Layer

- Recompile the cpu_extension project
- The cpu_extension.so or .dll gets updated



# Inference Engine GPU custom layer (.cl)

Custom Layer == Kernel

```
#pragma OPENCL EXTENSION cl_khr_fp16 : enable

__kernel void Cosh(const __global INPUT0_TYPE* input,
                   __global OUTPUT0_TYPE* output)
{
        // global index definition set in the XML configuration file
        const uint idx = get_global_id(0);
        const uint idy = get_global_id(1);
        const uint idbf = get_global_id(2);
        const uint feature = idbf%OUTPUT0_DIMS[1];
        const uint batch = idbf/OUTPUT0_DIMS[1];

        const uint in_id = batch*INPUT0_PITCHES[0] + feature*INPUT0_PITCHES[1] +
                           idy*INPUT0_PITCHES[2] + idx*INPUT0_PITCHES[3] + INPUT0_OFFSET;
        const uint out_id = batch*OUTPUT0_PITCHES[0] + feature*OUTPUT0_PITCHES[1] +
                           idy*OUTPUT0_PITCHES[2] + idx*OUTPUT0_PITCHES[3] + OUTPUT0_OFFSET;

        INPUT0_TYPE value = input[in_id];
        output[out_id] = (exp(value) + exp(-value))/2;
}
~
```

Directive enables half floating point support

Get index in tensor from current global ID
Definition set in configuration XML file

Get input and output index based on BFYX format

Implementation of operation for inference

# Inference Engine GPU custom layer Cont. (.xml)

- Registration of Custom Layer to Inference Engine

```
<CustomLayer name="Cosh" type="SimpleGPU" version="1">
  <Kernel entry="Cosh">
    <Source filename="cosh_kernel.cl"/>
  </Kernel>
  <Buffers>
    <Tensor arg-index="0" type="input" port-index="0" format="BFYX"/>
    <Tensor arg-index="1" type="output" port-index="0" format="BFYX"/>
  </Buffers>
  <CompilerOptions options="-cl-mad-enable"/>
  <WorkSizes global="X,Y,B*F"/>
</CustomLayer>
```

Configuration of Custom Layer:

- name - name of the layer type should be identical to the name in IR.
- type and version must always be these values shown.

# Inference Engine GPU custom layer Cont. (.xml)

- Registration of Custom Layer to Inference Engine

```
<CustomLayer name="Cosh" type="SimpleGPU" version="1">
  <Kernel entry="Cosh">
    <Source filename="cosh_kernel.cl"/>
  </Kernel>
  <Buffers>
    <Tensor arg-index="0" type="input" port-index="0" format="BFYX"/>
    <Tensor arg-index="1" type="output" port-index="0" format="BFYX"/>
  </Buffers>
  <CompilerOptions options="-cl-mad-enable"/>
  <WorkSizes global="X,Y,B*F"/>
</CustomLayer>
```

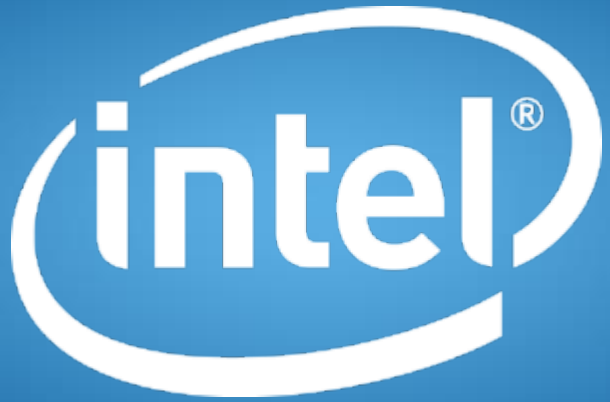
filename – Name of the file containing OpenCL™ source code. Path is relative to your executable.

OpenCL Launch Global NDRange

# Running Samples with Custom Layers Libraries

<sample.exe> -i <path to image> -m <path to model> **-l <path to custom layer library>** -d CPU

<sample.exe> -i <path to image> -m <path to model> **-c <path_to_xml_config_file>** -d GPU


intel