ACCELERATE DEEP LEARNING INFERENCE USING INTEL TECHNOLOGIES

OBJECT DETECTION USING INTEL® DISTRIBUTION OF OPENVINO™ TOOLKIT

August 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.

Intel, the Intel logo, Pentium, Celeron, Atom, Core, Xeon, Movidius, Saffron, and others are trademarks of Intel Corporation in the United States and other countries.

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

Smart Video Workshop Overview

Introduction

1. Introduction to Intel technologies for deep learning inference
2. Hardware acceleration techniques

Intel® Distribution of OpenVINO™ 101

Hardware Acceleration on laptop and devcloud

2. Basic End-to-End Object Detection Example

3./4./5. Hardware Acceleration with CPU, Integrated GPU, Intel® Movidius™ NCS, FPGA

Intel® Deep Learning Deployment Toolkit

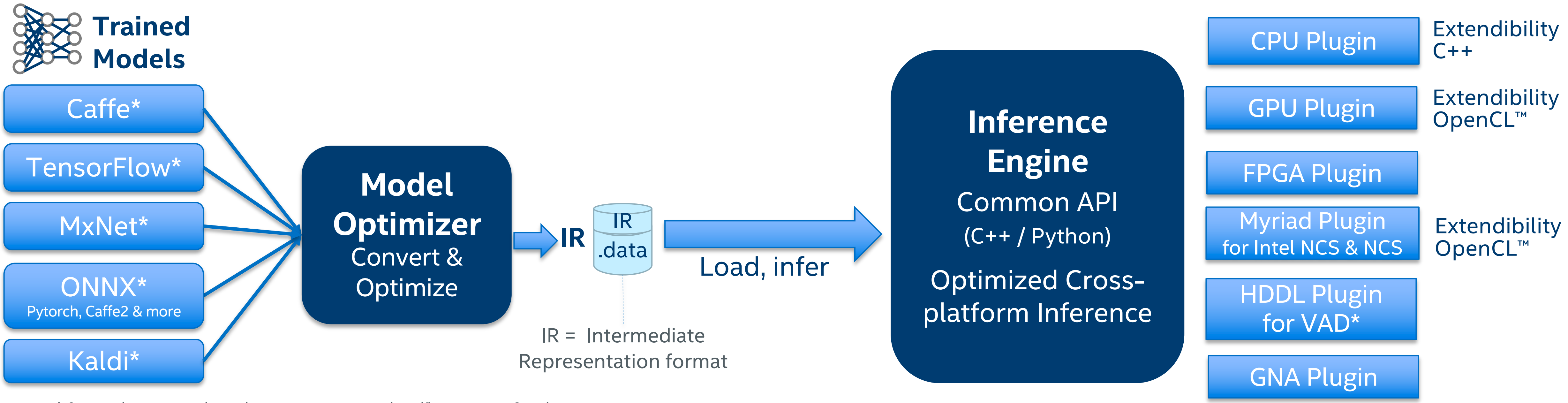
For Deep Learning Inference

Model Optimizer

- **What it is**: A Python*-based tool to import trained models and convert them to Intermediate representation.
- **Why important**: Optimizes for performance/space with conservative topology transformations; biggest boost is from conversion to data types matching hardware.

Inference Engine

- **What it is**: High-level inference API
- **Why important**: Interface is implemented as dynamically loaded plugins for each hardware type. Delivers best performance for each type without requiring users to implement and maintain multiple code pathways.

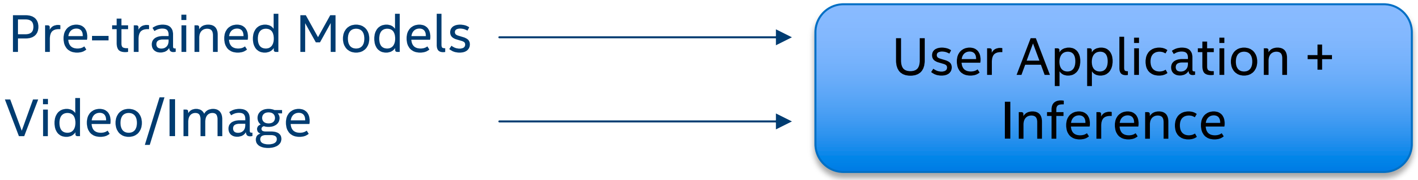


GPU = Intel CPU with integrated graphics processing unit/Intel® Processor Graphics

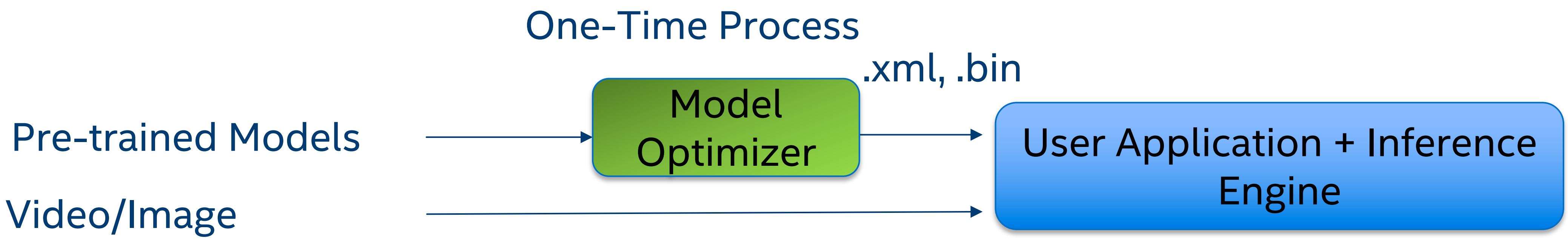
*VAD = Vision Accelerator Design Products (HDDL-R)

Deep Learning Application Deployment

Traditional

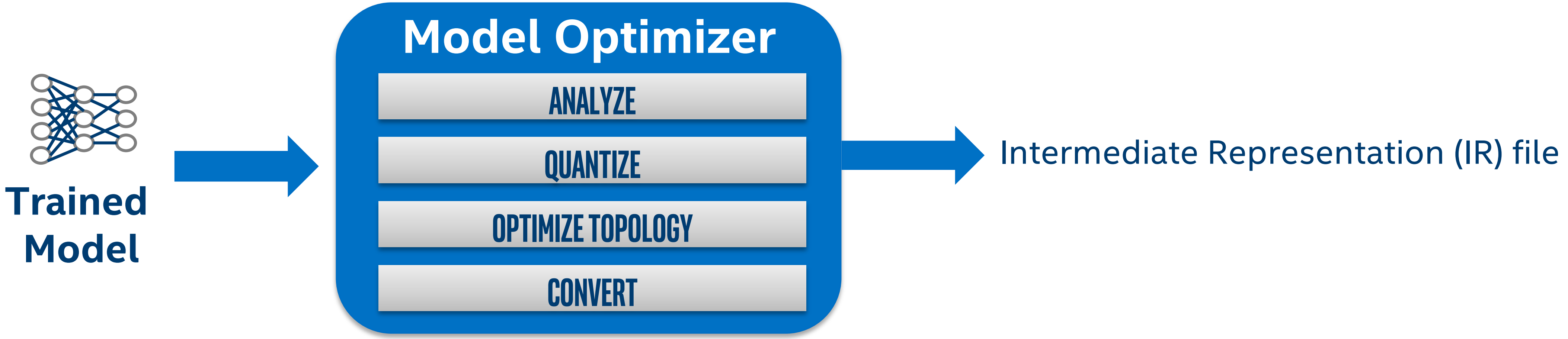


With Intel® Distribution of OpenVINO™ Toolkit



Model Optimizer

Improve Performance with Model Optimizer



- Easy to use, Python*-based workflow does not require rebuilding frameworks.
- Import Models from many supported frameworks: Caffe*, TensorFlow*, MXNet*, Kaldi*, exchange formats like ONNX* (Pytorch*, Caffe2* and others through ONNX).
- 100+ models for Caffe, MXNet, TensorFlow validated. Supports all ONNX* model zoo public models.
- Extends inferencing for non-vision networks with support of LSTM, Bert, GNMT, TDNN-LSTM, ESPNet and more.
- IR files for models using standard layers or user-provided custom layers do not require Caffe.
- Fallback to original framework is possible in cases of unsupported layers, but requires original framework.

Model Optimizer

Model optimizer performs generic optimization:

- Node merging
- Horizontal fusion
- Batch normalization to scale shift
- Fold scale shift with convolution
- Drop unused layers (dropout)
- FP16/Int8 quantization
- Model optimizer can add normalization and mean operations, so some preprocessing is 'added' to the IR

  --mean_values (104.006, 116.66, 122.67)

  --scale_values (0.07, 0.075, 0.084)

	FP32	FP16	FP11	INT8
CPU	yes	no	no	yes
GPU	yes	recommended	no	no
MYRIAD	no	yes	no	no
FPGA/DLA	no	yes	yes	no

Model Optimization Techniques

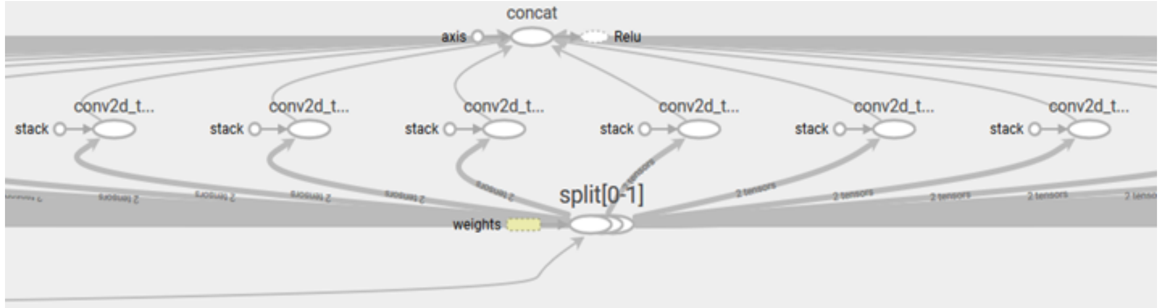
Linear Operation Fusing & Grouped Convolutions Fusing

Linear Operation Fusing: 3 stages

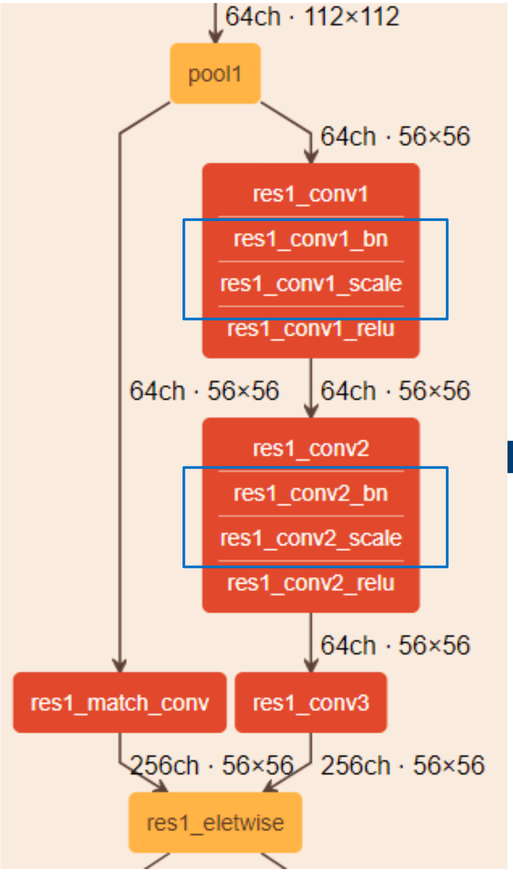
1. **BatchNorm and ScaleShift decomposition:** *BN* layers decomposes to *Mul->Add->Mul->Add* sequence; ScaleShift layers decomposes to *Mul->Add* sequence.
2. **Linear operations merge:** Merges sequences of Mul and Add operations to the **single** Mul->Add instance.
3. **Linear operations fusion:** Fuses Mul and Add operations to Convolution or FullybConnected layers.

Grouped Convolutions Fusing

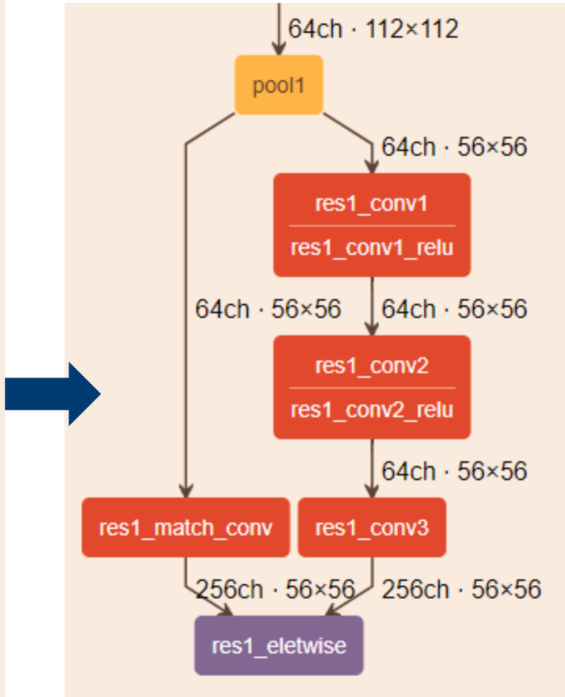
Specific optimization that applies for TensorFlow* topologies. (Xception*)



Split->Convolutions->Concat block from TensorBoard*



Caffe Resnet269 block (from Netscope)

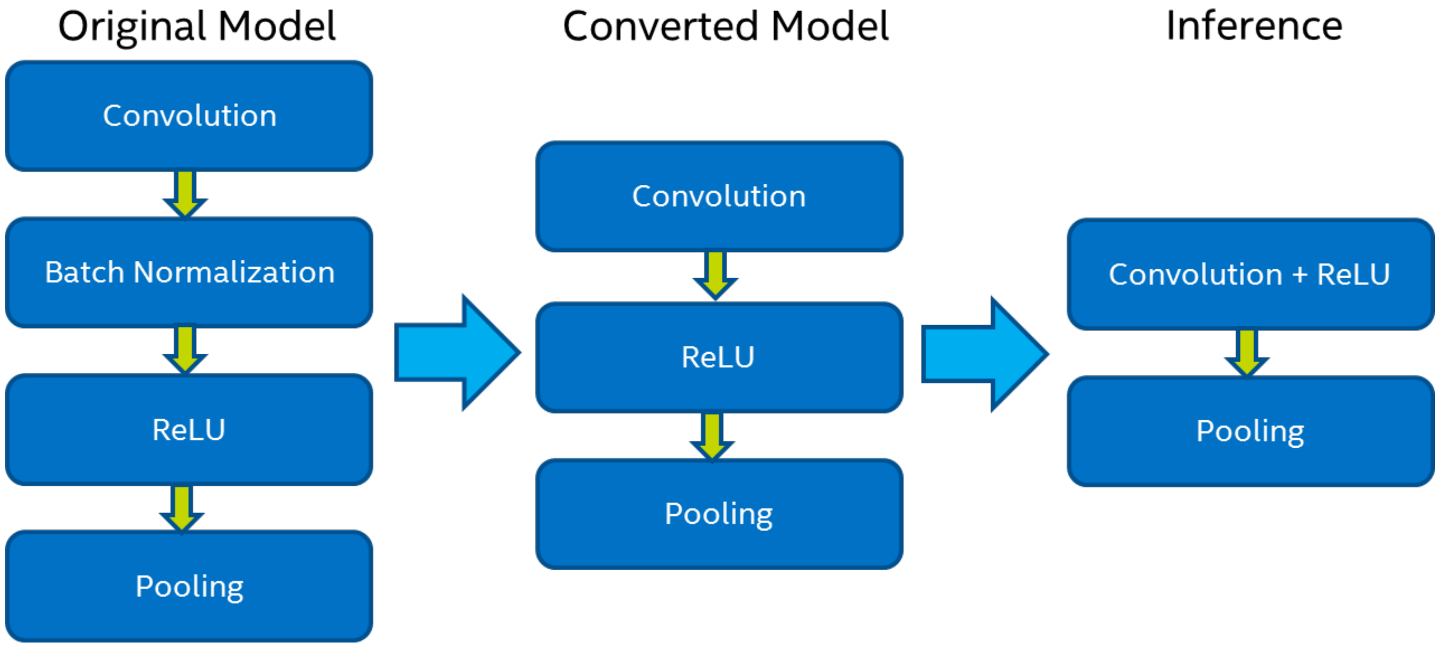


Merged Caffe* Resnet269 block (from Netscope*)

Model Optimizer: Linear Operation Fusing

Example

1. Remove Batch normalization stage.
2. Recalculate the weights to ‘include’ the operation.
3. Merge Convolution and ReLU into one optimized kernel.



Model Optimizer: Cutting Off Parts of a Model

Model optimizer can cut out a portion of the network:

- Model has pre/post-processing parts that cannot be mapped to existing layers.
- Model has a training part that is not used during inference.
- Model is too complex and cannot be converted in one shot.

Command line options

- Model Optimizer provides command line options --input and --output to specify new entry and exit nodes ignoring the rest of the model:
- --input option accepts a comma-separated list of layer names of the input model that should be treated as new entry points to the model;
- --output option accepts a comma-separated list of layer names of the input model that should be treated as new exit points from the model.

Intermediate Representation (IR)

Squeezenet1.1.caffemodel → Squeezenet1.1.bin

Squeezenet1.1.prototxt → Squeezenet1.1.xml

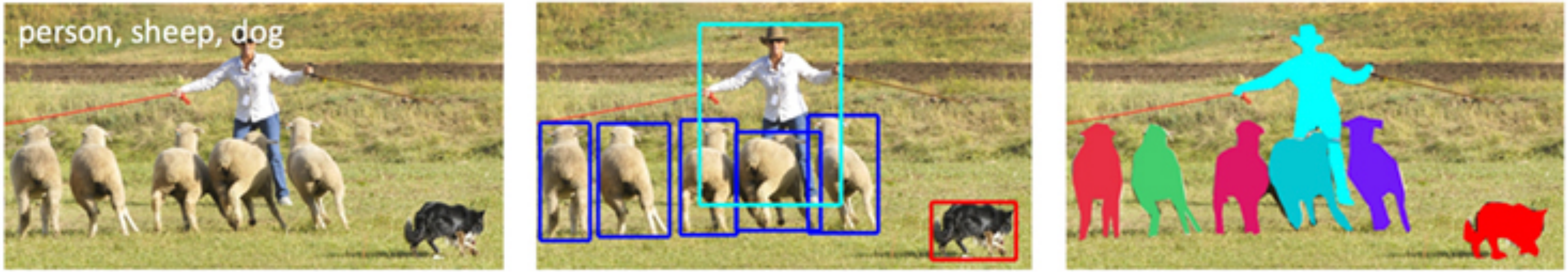
```
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param { shape: { dim: 1 dim: 3 dim: 227 dim: 227 } }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 64
    kernel_size: 3
    stride: 2
  }
}
```

```
<net batch="1" name="model" version="2">
  <layers>
    <layer id="100" name="data" precision="FP32" type="Input">
      <output>
        <port id="0">
          <dim>1</dim>
          <dim>3</dim>
          <dim>227</dim>
          <dim>227</dim>
        </port>
      </output>
    </layer>
    <layer id="129" name="conv1" precision="FP32" type="Convolution">
      <data dilation-x="1" dilation-y="1" group="1" kernel-x="3" kernel-y="3" output="64" pa
      <input>
        <port id="0">
          <dim>1</dim>
          <dim>3</dim>
          <dim>227</dim>
          <dim>227</dim>
        </port>
      </input>
      <output>
        <port id="3">
          <dim>1</dim>
          <dim>64</dim>
          <dim>113</dim>
          <dim>113</dim>
        </port>
      </output>
      <blobs>
        <weights offset="2275104" size="6912"/>
        <biases offset="4805920" size="256"/>
      </blobs>
```

Inference Engine

Inference on an Intel® Edge Systems

Many deep learning networks are available—choose the one you need.

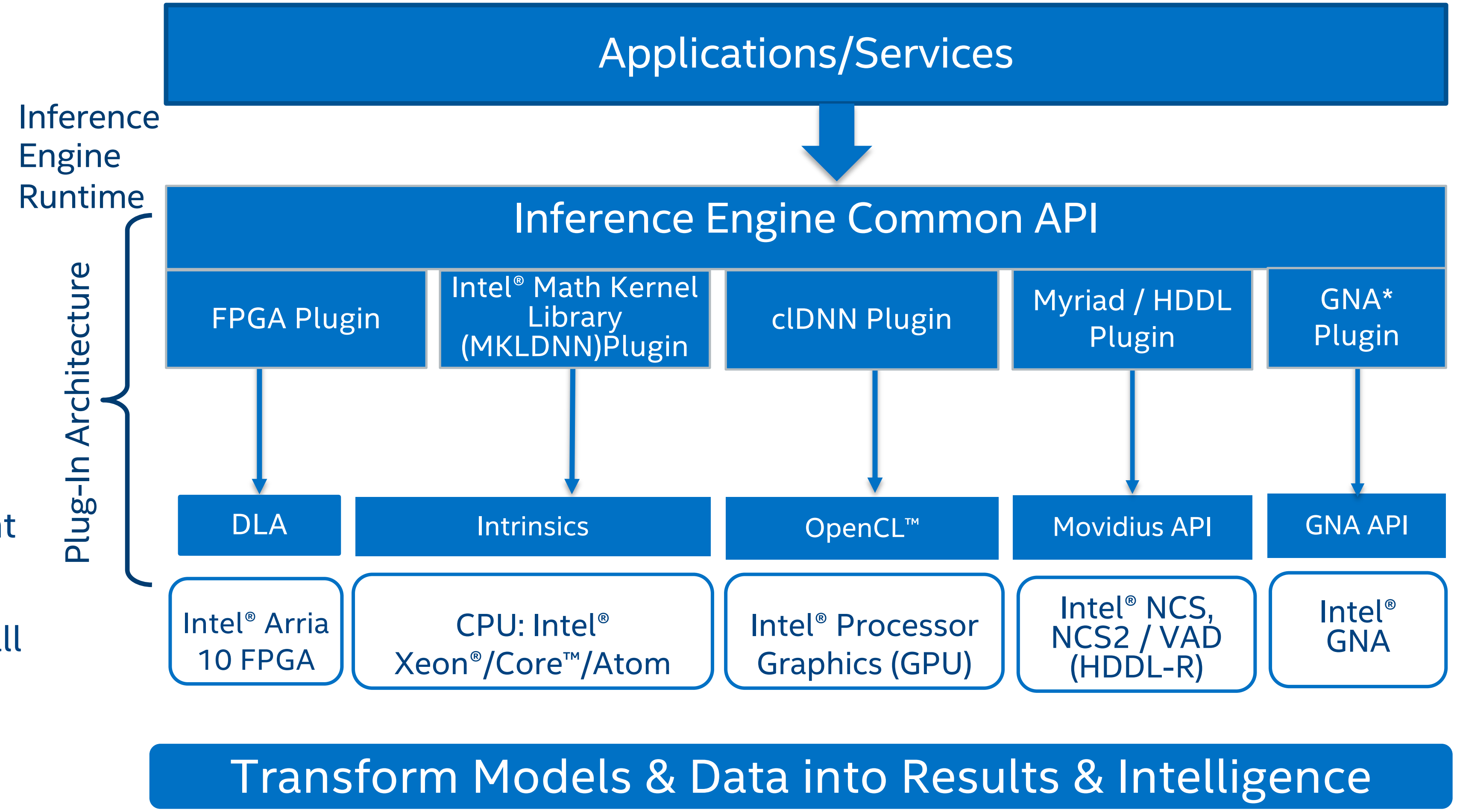


(a) classification (b) detection (c) segmentation

The complexity of the problem (data set) dictates the network structure. The more complex the problem, the more ‘features’ required, the deeper the network.

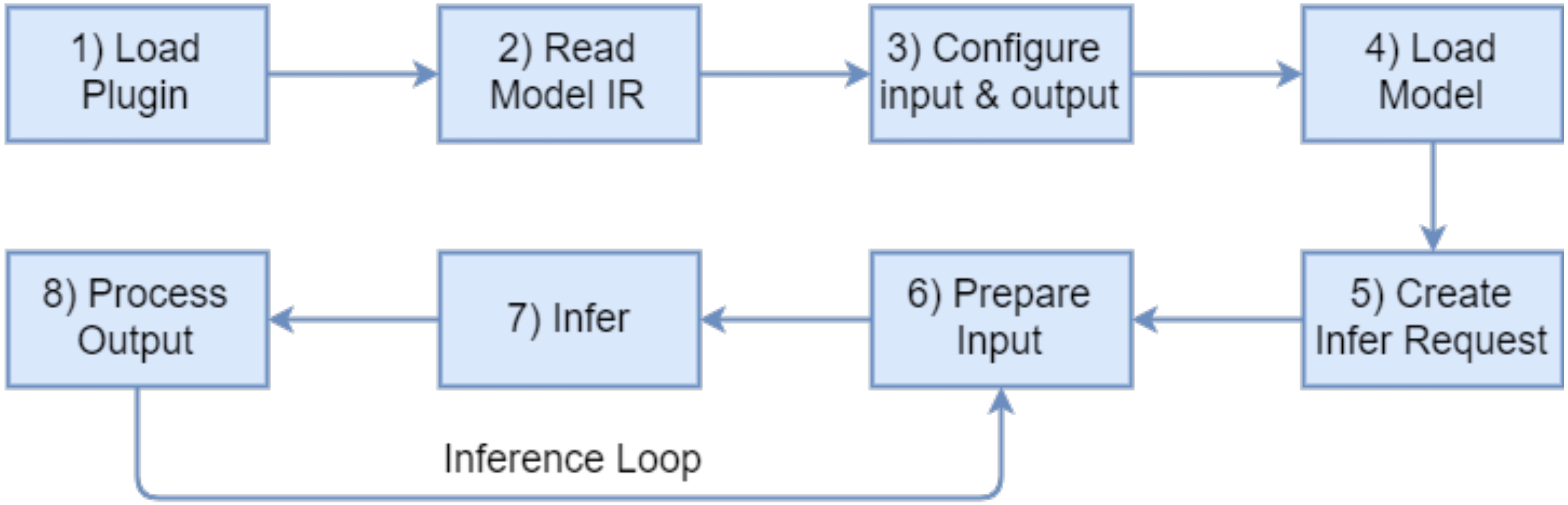
Optimal Model Performance Using the Inference Engine

- Simple & unified API for inference across all Intel® architecture
- Optimized inference on large IA hardware targets (CPU/GEN/FPGA)
- Heterogeneity support allows execution of layers across hardware types
- Asynchronous execution improves performance
- Futureproof/scale your development for future Intel® processors
- Supports serialized FP16 IR across all plugins / platforms (CPU inference remains at FP32)



GPU = Intel CPU with integrated graphics/Intel® Processor Graphics/GEN
GNA = Gaussian mixture model and Neural Network Accelerator

Application Workflow

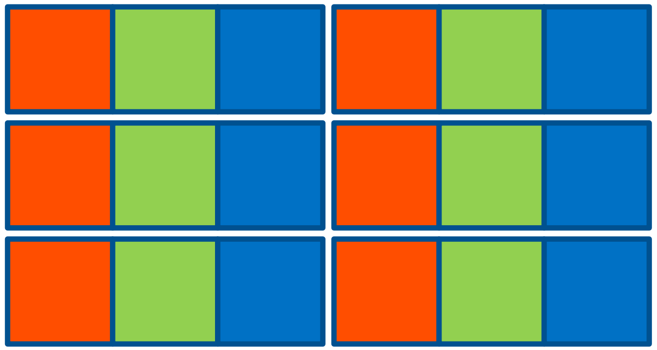


Pre-processing

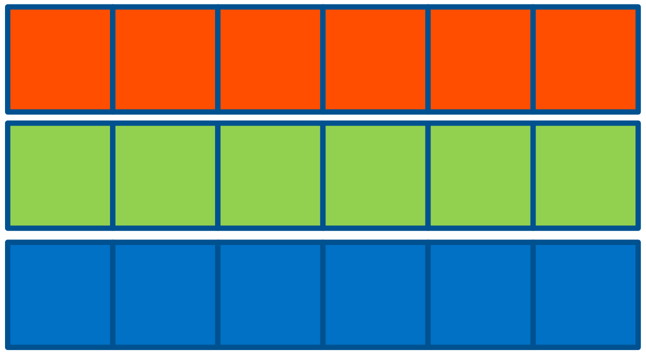
Most of the image formats are interleaved formats (RGB, BGR, BGRA, and so forth). Inference engine expects input to be in RGB planar format, such as:

R-plain, G-plain, B-plain

Interleaved



Planar



Post-processing

Developers are responsible for parsing inference output.

Many output formats are available. Some examples include:

- **Simple Classification (alexnet*):** an array of float confidence scores, # of elements=# of classes in the model
- **SSD:** Many “boxes” with a confidence score, label #, xmin,ymin, xmax,ymax

Unless a model is well documented, the output pattern may not be immediately obvious.

Lab1 - Basic End to End Object Detection Example

URL: https://github.com/intel-iot-devkit/smart-video-workshop/blob/master/object-detection/README.md

Objective: This Lab uses a Single Shot MultiBox Detector (SSD) on a trained mobilenet-ssd* Caffe model to walk you through the basic steps of using two key components of the Intel® Distribution of OpenVINO™ toolkit: Model Optimizer and Inference Engine.

Estimated Complete Time: 30min

Lab2 - Tensor Flow example

URL: https://github.com/intel-iot-devkit/smart-video-workshop/blob/master/advanced-video-analytics/tensor_flow.md

Objective: This Lab showcases how to convert and freeze the TensorFlow Inception_v1 model, and use the converted IR model with the Intel® Distribution of OpenVINO™ toolkit inference engine on a classification sample.

Estimated Complete Time: 30min

Smart Video Workshop Overview

Introduction

1. Introduction to Intel technologies for deep learning inference
2. Hardware acceleration techniques

Each module contains a hands-on lab exercise that introduces various Intel technologies to accelerate computer vision application with hardware heterogeneity.

Intel® Distribution of OpenVINO™ 101	2. Basic End-to-End Object Detection Example
Hardware Acceleration on laptop and devcloud	3./4./5. Hardware Acceleration with CPU, Integrated GPU, Intel® Movidius™ NCS, FPGA
Optimization	6. Optimization Tools and Techniques
Application	7. Advanced Video Analytics
Custom layers	8. Custom layers

CPU/GPU/FPGA/Intel® Movidius™ Neural Compute Stick

Hardware Heterogenity

Heterogeneous Dichotomies

Data vs. Task Parallelism

- Data parallelism between devices is usually explicit and extremely error-prone (like in OpenCL™).
- Inference engine heterogeneity support is inherently **task-oriented** (node-level).

Dynamic vs. Static

- **User-defined work split** vs. adaptive schemes (like load-balancing).
- For both, there is a question on the granularity (communications costs, next).

***Heterogeneity* is basically *Fallback*.**

Introducing Heterogeneous Plugin

The heterogeneous plugin enables computing for inference on one network on several devices. Purposes to execute networks in heterogeneous mode

- To utilize accelerators power and calculate heaviest parts of network on accelerator and execute not supported layers on fallback devices like CPU
- To utilize all available hardware more efficiently during one inference

The execution through heterogeneous plugin can be divided to two independent steps:

- Setting of affinity to layers (binding them to devices in InferenceEngine::ICNNNetwork)
- Loading a network to the Heterogeneous plugin, splitting the network to parts, and executing them through the plugin

These steps are decoupled. The setting of affinity can be done automatically using fallback policy or in manual mode.

Apply Device Affinities to Layers Automatically Using Fallback Policy (1 of 2)

```
$ ./object_detection_sample_ssd -m  <path_to_model>/Model.xml -i
<path_to_pictures>/picture.jpg -d HETERO:FPGA,CPU
```

The “priorities” defines a greedy behavior:

- Keeps all layers that can be executed on the device (FPGA)
- Carefully respects topological and other limitations
- Follows priorities when searching (for example, CPU)

Apply Device Affinities to Layers Automatically Using Fallback Policy (2 of 2)

```
1. InferenceEngine::PluginDispatcher dispatcher({ FLAGS_pp,
   archPath , "" });
2. InferenceEngine::InferenceEnginePluginPtr enginePtr;
3. enginePtr =
   dispatcher.getPluginByDevice("HETERO:FPGA,CPU");
4. InferencePlugin plugin(enginePtr);
5. CNNNetReader reader;
6. reader.ReadNetwork("Model.xml");
7. reader.ReadWeights("Model.bin");
8. auto executable_network = plugin.LoadNetwork(network,
   {});
```

Apply Device Affinities to Layers Manually

```
1.InferenceEngine::PluginDispatcher dispatcher({ FLAGS_pp,
  archPath , "" });
2.InferenceEngine::InferenceEnginePluginPtr enginePtr;
3.enginePtr =
  dispatcher.getPluginByDevice("HETERO:FPGA,CPU");
4.HeteroPluginPtr hetero(enginePtr);
5.hetero->SetAffinity(network, { }, &resp);
6.network.getLayerByName("qqq")->affinity = "CPU";
7.InferencePlugin plugin(enginePtr);
8. CNNNetReader reader;
9. reader.ReadNetwork("Model.xml");
10.reader.ReadWeights("Model.bin");
11.auto executable_network = plugin.LoadNetwork(network,
  {});
```

Lab3 - Hardware Heterogeneity

URL: https://github.com/intel-iot-devkit/smart-video-workshop/blob/master/hardware-heterogeneity/README.md

Objective: This example shows how to use hetero plugin to define preferences to run different network layers on different hardware types, then use option -pc to get performance data on each subgraph.

Estimated Complete Time: 20min

INFERENCE ENGINE MULTI DEVICE

Multi-Device support

OOB automatic load-balancing between devices (inference requests level)

- Fully general machinery: any devices combinations
  - **VCAA**, which is just **MULTI: HDDL,GPU**,*<optional>CPU*
  - **CPU+GPU** (very ubiquitous!)
  - Multiple MYX sticks, etc

As easy as “-d **MULTI**:HDDL,GPU” for cmd-line option of your favorite sample

Also C++ example (Python is similar)

```
Core ie; //NEW IE-CENTRIC API
ExecutableNetwork exec = ie.LoadNetwork(network,{{“DEVICE_PRIORITIES”, “HDDL,GPU”}}, “MULTI”);

auto plugin =  PluginDispatcher().getPluginByDevice("MULTI:CPU,GPU"); //Old (plugin-centric) API
ExecutableNetwork executable_network = plugin.LoadNetwork(network, config);
```

Multi-Device performance facts

Performance HDDL and GPU devices is combined ideally

- Primary target is VCAA
- Another target is with i5 (or i7) and HDDL-R

Multiple MYXs sticks also combined on 100%

CPU and GPU combine up to 97% (on computationally heavy topologies)

- Exact ratio depends on the mem BW, which is shared between the devices
- Also TDP, GPU driver polling and other variables that influence the equation
- **See benchmark_app for more details on the optimal setup**

Multi-Device implications: CPU is critical resource

Rule of thumb: every accelerator need a CPU core
for scheduling, data transfers, etc

We made initial revision of the threading in plugins and system services

- E.g. GPU driver polling and scheduling issues
  - use THROTTLING_HINT (**see benchmark_app**)

Would probably exclude CPU from inference entirely for VCAA (KBL-Y has just 2 cores)

- Just like GPU can be busy with decoding/resize/crop and contribute less in real scenarios

Intel® Movidius™ Neural Compute Stick 2

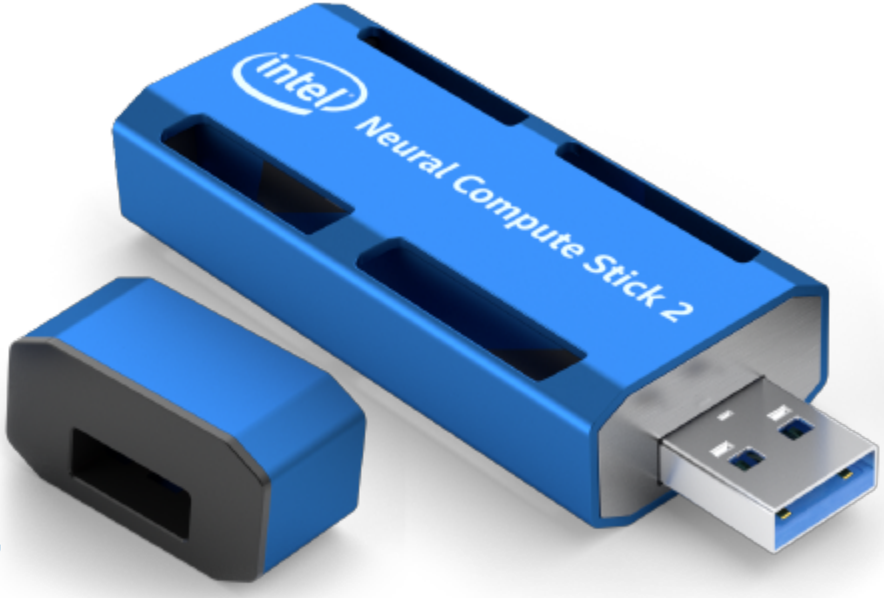
Intel® Movidius™ Neural Compute Stick 2

Intel® Movidius™ Neural Compute Stick 2

Redefining the AI Developer Kit

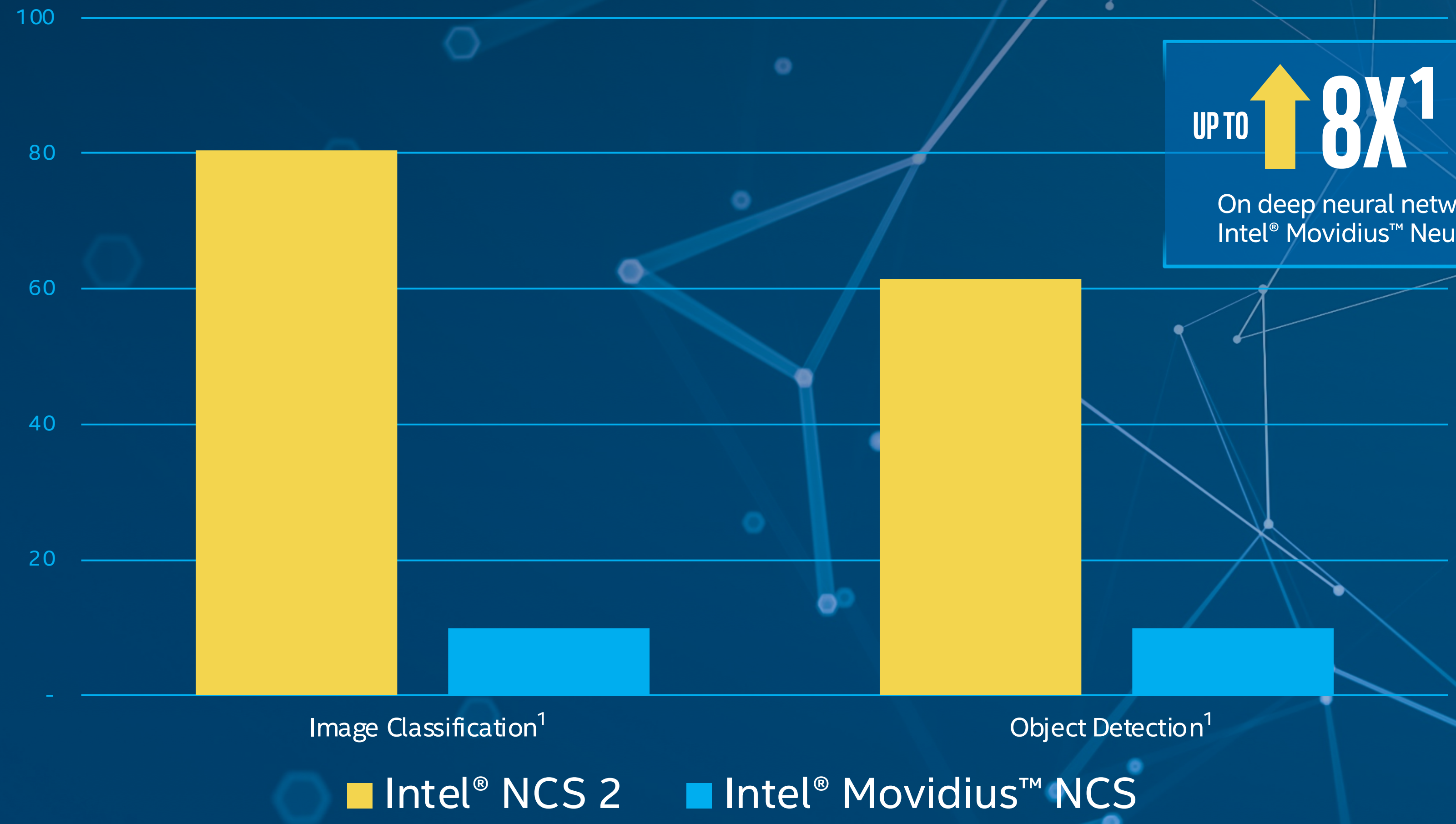
Technical Specifications×

- Processor: Intel® Movidius™ Myriad™ X Vision Processing Unit (VPU)
- Supported frameworks: TensorFlow* and Caffe*
- Connectivity: USB 3.0 Type-A
- Dimensions: 2.85 in. x 1.06 in. x 0.55 in. (72.5 mm x 27 mm x 14 mm)
- Operating temperature: 0° C to 40° C
- Compatible operating systems: Ubuntu* 16.04.3 LTS (64 bit), CentOS* 7.4 (64 bit), and Windows® 10 (64 bit)



INTEL® NEURAL COMPUTE STICK 2

MORE CORES. MORE AI INFERENCE



UP TO 8X[1] HIGHER PERFORMANCE

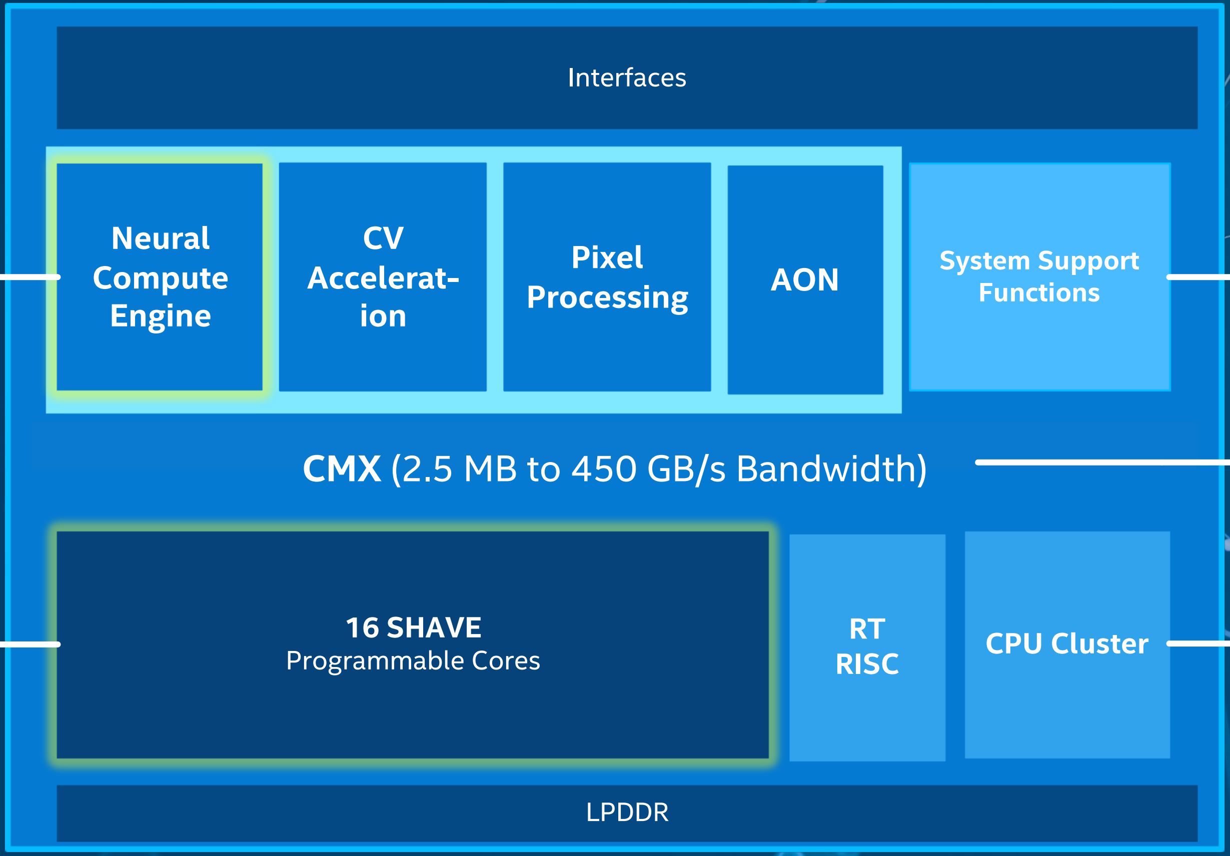
On deep neural networks compared to Intel® Movidius™ Neural Compute Stick

INTEL® NEURAL COMPUTE STICK 2: FEATURING THE INTEL® MOVIDIUS™ MYRIAD™ X VPU

A self-sufficient, all-in-one processor that features the powerful **Neural Compute Engine** and **16 programmable SHAVE cores** that deliver class-leading performance for deep neural network inference applications.

An entirely new deep neural network (DNN) inferencing engine that offers flexible interconnect and ease of configuration for on-device DNNs and computer vision applications

VLIW (DSP) programmable processors are optimized for complex vision & imaging workloads



System support functions operate frames, tiles, CODEC, compression and security

Homogeneous memory design for low-power, UL latency, sustained High Performance, and locally stored data

RISC Processors, RTOS Schedulers, Pipeline Managers, Sensor Control Frameworks

INTEL® DISTRIBUTION OF OPENVINO™ TOOLKIT
SUPPORTED LAYERS

View Documentation ▶ https://docs.openvinotoolkit.org/latest/_docs_IE_DG_supported_plugins_Supported_Devices.html

- Activation-Clamp
- Activation-ELU
- Activation-Leaky ReLU
- Active-PReLU
- Activation-ReLU
- Activation-ReLU6
- Activation-Sigmoid/Logistic
- Activation-TanH
- ArgMax
- BatchNormalization
- Concat
- Const
- Convolution-Dilated
- Convolution-Grouped
- Convolution-Ordinary
- Crop
- CTCGreedyDecoder*
- Deconvolution
- DetectionOutput*
- Eltwise-Max
- Eltwise-Mul
- Eltwise-Sum
- Flatten
- FullyConnected (Inner Product)
- GRN
- Interp
- LRN (Norm)
- MVN*
- Normalize*
- Pad*
- Permute
- Pooling(AVG,MAX)*
- Power
- PriorBox
- PriorBoxClustered
- Proposal
- PSROIPooling
- RegionYolo
- ReorgYolo
- Resample
- Reshape
- RNN
- ROIPooling
- ScaleShift*
- Slice
- SoftMax
- Split
- Tile

* Support is limited to the specific parameters. Refer to “Known Layers Limitation” section for the device from the list of supported.

Changed since last update

Lab4 - HW Acceleration with Intel® Movidius™ Neural Compute Stick

URL: https://github.com/intel-iot-devkit/smart-video-workshop/blob/master/HW-Acceleration-with-Movidious-NCS/README.md

Objective: This lab shows how the Intel® Distribution of OpenVINO™ toolkit provides hardware abstraction to run the sample object detection application which was built in previous modules on Intel® Movidius™ Neural Compute Stick.

Estimated Complete Time: 20min