END2END VIDEO ANALYTICS & ADVANCED FEATURES

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 (1 OF 2)

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 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 (2 OF 2)

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.

AGENDA

END TO END VIDEO APPLICATION PIPELINE

OPENCV* FOR END TO END VIDEO APPLICATION

INFERENCE ENGINE INTEGRATION

INFERENCE ENGINE OBJECTS

ASYNC API

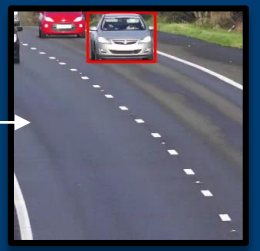
HETEROGENEOUS API

PERFORMANCE COUNTERS

EXTENSIONS

END 2 END VIDEO ANALYTICS

Decode → Pre-processing → **Inference** → Post-Processing → Encode



Decode	Pre-processing	Inference	Post-Processing	Encode
Encoded Streams Converted to Image Arrays (RGB etc.)	Resize/Normalize Frame/Change ColorSpace etc.	Run Forward Propagation/Prediction on DL Model	Add Annotations/Resize etc.	Encode the Decoded Frames to Save/Stream on Storage/Network

OPENCV* FOR EVERYTHING

Load Video	`cap = cv.VideoCapture(0)` `has_frame, frame = cv.read()`

Pre-Process Frame	`resized_frame = cv.resize(frame, (Config.MODEL_IMAGE_HEIGHT, Config.MODEL_IMAGE_WIDTH))`
	
	`blob = cv.dnn.blobFromImage(resized_frame, Config.MODEL_SCALE, (Config.MODEL_IMAGE_HEIGHT, Config.MODEL_IMAGE_WIDTH), (Config.MODEL_MEANS[0], Config.MODEL_MEANS[1], Config.MODEL_MEANS[2]))`

Inference	`network = cv.dnn.readNetFromTensorflow(Config.MODEL_WEIGHT_FILE, Config.MODEL_FILE)`
	`# Send blob data to Network`
	`network.setInput(blob)`
	
	`# Make network do a forward propagation to get recognition matrix`
	`out = network.forward()`

Post-Process Frame	`cv.putText(frame, label_text, (int(xmin), int(ymin)), cv.FONT_HERSHEY_SIMPLEX, 0.5, Config.LABEL_COLORS[label_index], 2)` `cv.rectangle(frame, int(xmin), int(ymin)),(int(xmax), int(ymax)),Config.LABEL_COLORS[label_index], thickness=3)`
Encode	`cv.VideoWriter(FOURCC('H','2','6','4')`

OPENCV – INFERENCE ENGINE BACKEND

- Open source version of **Intel® Distribution of OpenVINO™** is maintained under OpenCV* DLDT project.
  - https://github.com/opencv/dldt
- Open Source version landing page:
  - https://01.org/openvinotoolkit/overview

```
network.setPreferableBackend(cv.dnn.DNN_BACKEND_OPENCV)
network.setPreferableBackend(cv.dnn.DNN_BACKEND_INFERENCE_ENGINE)

network.setPreferableTarget(cv.dnn.DNN_TARGET_CPU)
network.setPreferableTarget(cv.dnn.DNN_TARGET_OPENCL)
```

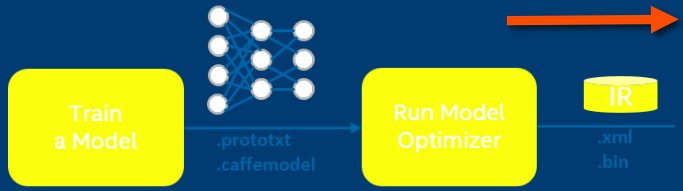
INFERENCE ENGINE INTEGRATION

- If there is an application already running with a custom inference framework, Inference Engine can be integrated easily.
- Compared to **OpenCV*** there are just couple more steps to add.
- Python sample for inference.

```
# Load Plugin
plugin = IEPlugin(device="CPU", plugin_dirs='/lib/dir')
# Create Network
net = IENetwork(model=model_xml, weights=model_bin)
# Load Executable Network
exec_net = plugin.load(network=net, num_requests=num_requests)
# Getting the input and outputs of the network
input_blob = next(iter(net.inputs))
out_blob = next(iter(net.outputs))
# Get infer results
infer = exec_net.infer(inputs={input_blob: images})
```

INFERENCE ENGINE – WORKFLOW

OFFLINE, ONE TIME RUN OF THE MODEL-OPTIMIZER



DECODE
RESIZE
COLOR CONVERSION..ETC

Create IE Instance

Load IE Plug-in

Read an IR Model

Set Target Device

Load model to plug-in

Allocate Input and Output Blobs

Read data into Input Blob

Inference

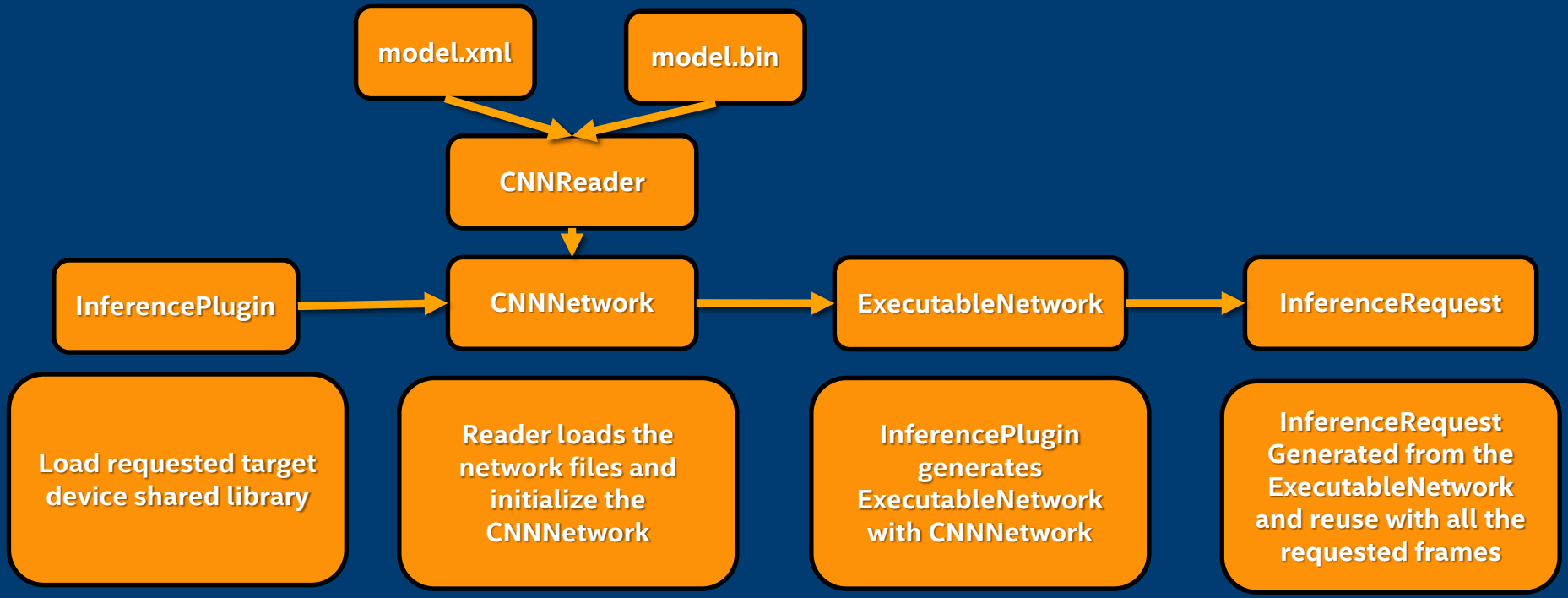
INTEGRATE IE (INFERENCE ENGINE) API TO YOUR APPLICATION

MKL-DNN PLUG-IN FOR CPU, CL-DNN FOR GPU

TARGET DEVICE COULD BE CPU/GPU/FPGA/MYRIAD

REPEAT INFERENCE FOR EACH FRAME

INFERENCE ENGINE API COMPONENTS - C++



INFERENCE ENGINE API – PUBLIC METHODS QUICK LOOKUP

InferencePlugin

LoadNetwork (ICNNNetwork &network, const std::map< std::string, std::string > &config)

Wraps original method IInferencePlugin::LoadNetwork(IExecutableNetwork::Ptr&, ICNNNetwork&, const std::map<std::string, std::string> &, ResponseDesc*).

LoadNetwork (CNNNetwork network, const std::map< std::string, std::string > &config)

Wraps original method IInferencePlugin::LoadNetwork(IExecutableNetwork::Ptr&, ICNNNetwork&, const std::map<std::string, std::string> &, ResponseDesc*).

Infer (const BlobMap &input, BlobMap &result)

Wraps original method IInferencePlugin::Infer(const BlobMap&, BlobMap&, ResponseDesc *resp)

GetPerformanceCounts () const

Wraps original method IInferencePlugin::GetPerformanceCounts.

AddExtension (InferenceEngine::IExtensionPtr extension)

Wraps original method IInferencePlugin::AddExtension.

SetConfig (const std::map< std::string, std::string > &config)

Wraps original method IInferencePlugin::SetConfig.

CNNNetwork

getPrecision () const

Wraps original method ICNNNetwork::getPrecision.

getOutputsInfo () const

Wraps original method ICNNNetwork::getOutputsInfo.

getInputsInfo () const

Wraps original method ICNNNetwork::getInputsInfo.

layerCount () const

Wraps original method ICNNNetwork::layerCount.

setBatchSize (const size_t size)

Wraps original method ICNNNetwork::setBatchSize.

getBatchSize () const

Wraps original method ICNNNetwork::getBatchSize.

operator ICNNNetwork & () const

An overloaded operator & to get current network. More...

setTargetDevice (TargetDevice device)

Sets tha target device. More...

ExecutableNetwork

ExecutableNetwork (IExecutableNetwork::Ptr actual)

GetOutputsInfo () const

Wraps original method IExecutableNetwork::getOutputsInfo.

GetInputsInfo () const

Wraps original method IExecutableNetwork::getInputsInfo.

reset (IExecutableNetwork::Ptr newActual)

reset owned object to new pointer, essential for cases when simultaneou networks not expected More...

CreateInferRequest ()

Wraps original method IExecutableNetwork::CreateInferRequest.

CreateInferRequestPtr ()

Wraps original method IExecutableNetwork::CreateInferRequestPtr. Mor

Export (const std::string &modelFileName)

Exports the current executable network so it can be used later in the Imp More...

GetMappedTopology (std::map< std::string, std::vector< PrimitiveInfo::Pt &deployedTopology)

Gets the mapping of IR layer names to implemented kernels. More...

operator IExecutableNetwork::Ptr & ()

QueryState ()

see original function InferenceEngine::IExecutableNetwork::QueryState

InferenceRequest

SetBlob (const std::string &name, const Blob::Ptr &data)

Sets input/output data to infer. More...

GetBlob (const std::string &name)

Wraps original method IInferRequest::GetBlob.

Infer ()

Wraps original method IInferRequest::Infer.

GetPerformanceCounts () const

Wraps original method IInferRequest::GetPerformanceCounts

SetInput (const BlobMap &inputs)

Sets input data to infer. More...

SetOutput (const BlobMap &results)

Sets data that will contain result of the inference. More...

SetBatch (const int batch)

Sets new batch size when dynamic batching is enabled in executable network that created this request. More...

InferRequest (IInferRequest::Ptr request)

StartAsync ()

Start inference of specified input(s) in asynchronous mode. More...

INFERENCE ENGINE - ASYNC API

- Inference requests to Inference Engine can be made either synchronously or asynchronously.
- When input object is defined, **`ExecutableNetwork`** can either use `Infer` or **`StartAsync`** method to get inference results from the network.
  - For each request a **unique id** should be assigned.
  - HW Level parallelization controlled with **`InferenceEngine`** itself.

```
# Start async inference for request_id
exec_net.start_async(request_id=my_request_id, inputs={input_blob: in_frame})

# Blocking wait for a particular request_id
if exec_net.requests[my_request_id].wait(-1) == 0
        # getting the result of the network
        res = exec_net.requests[my_request_id].outputs[out_blob]
```

INFERENCE ENGINE – HETEROGENEOUS API

- The API was originally invented to support “fallback” of non existing layers
- **Example**: FPGA does not have a layer and we’d like to run this layer on another device (CPU)
- Support allocating different layers to different devices.
- Performance consideration: memory traffic to/from the accelerator can kill the performance

(though there might be cases where it’s ok, first/last layers for example)

- So basically a plug-in that can choose a plug-in per layer..

```
plugin = IEPlugin(device=hetero_device, plugin_dirs='')

plugin.set_initial_affinity(net)
plugin.set_config({"TARGET_FALLBACK": hetero_device})

# Dump Graph for Affinity of Layers
if dot_graph:
    plugin.set_config({"HETERO_DUMP_GRAPH_DOT": "YES"})
```

INFERENCE ENGINE – LAYER AFFINITY

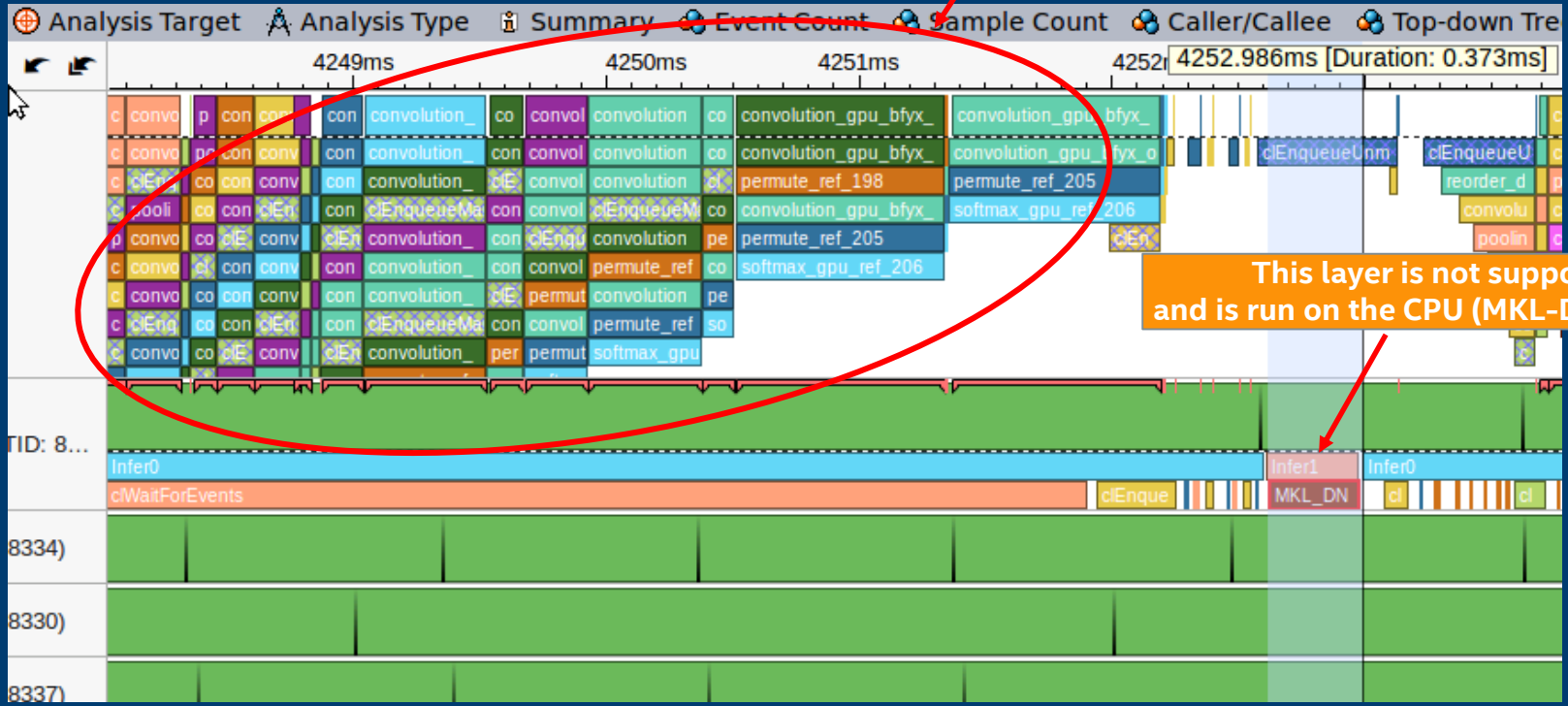
- Inference Engine allows you to manually assign target device for layers.
  - Manual decision for what target to run layer on.
- Below code shows how to check and assign value for layer.

```
for l in net.layers.values():
    print('Type: ', l.type, 'Device: ', l.affinity)

# Iterate over layers and check type or affinity
for l in net.layers.values():
    if l.type == ‘Convolution’:
       l.affinitiy = ‘GPU’
```

HETEROGENEOUS RUN ANALYSIS

Example (SSD) - vTune analysis..



INFERENCE ENGINE – PERFORMANCE COUNTER

- Inference Engine allows you to get a detailed report of layer execution times.
  - Performance metrics given as a dictionary format
- Below shows how to use Performance Counter and get report out.

```
# Print the performance counters
perf_counts = exec_net.requests[0].get_perf_counts()

print("Performance counters:")

for layer, stats in perf_counts.items():
    print(layer, ': ', stats)

# Example Output
#  {'layer_type': 'Concat', 'cpu_time': 0, 'status': 'NOT_RUN', 'real_time': 0,
'exec_type': 'unknown_FP32'}
```

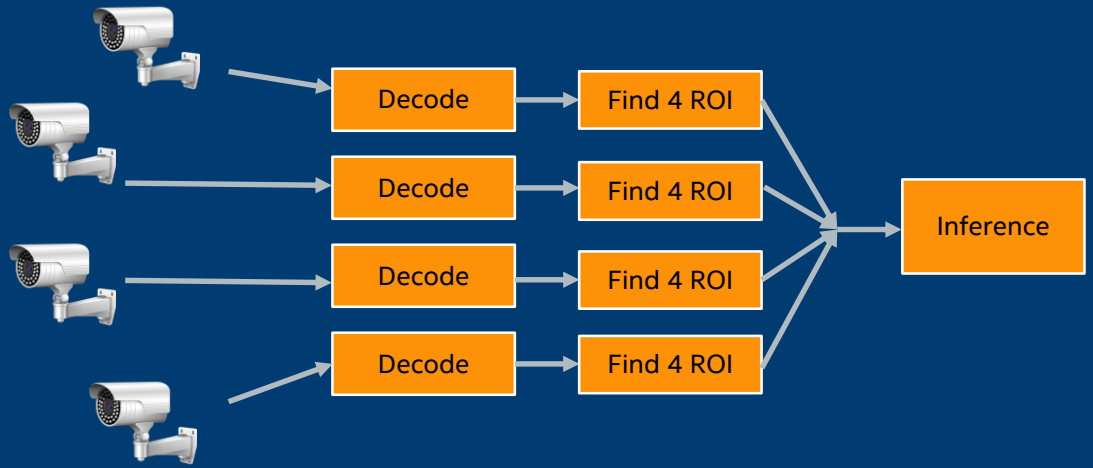
INFERENCE ENGINE – DYNAMIC BATCHING

- Processing a data batch is usually faster then sequentially feeding image by image to the network.
- Here's where dynamic batching comes into play. Instead of using a fixed batch size (big enough to process all faces at once), it is possible to limit it on every frame depending the amount of detected faces.

```
# Set Batch Size at the beginning
net = createNetwork(model_xml, model_bin, plugin)
net.batch_size = batch_size

# During the inference, you can dynamically set the batch size.
exec_net.requests[0].set_batch(inputs_count)
```

INFERENCE ENGINE – DYNAMIC BATCHING



Output Blob

[0, 0, 0.2, .2123, .123]
[0, 1, 0.2, .2123, .123]

All zeros

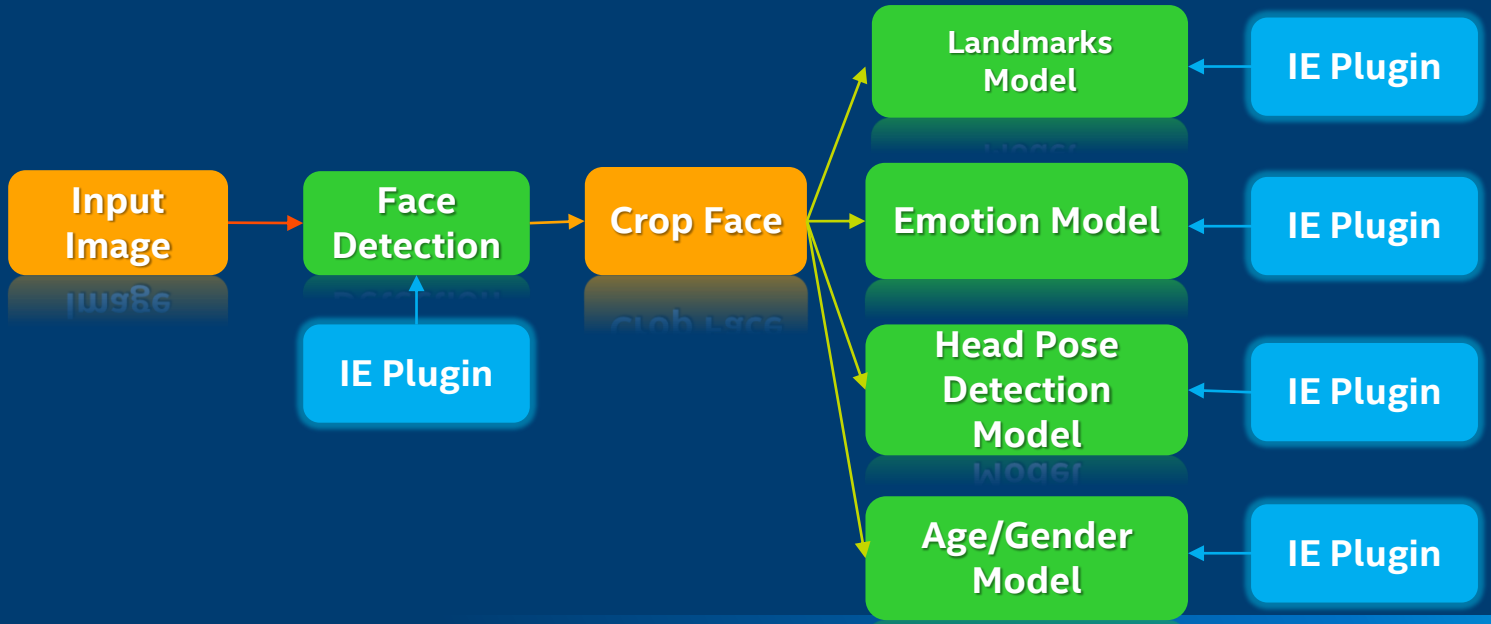
Higher Batch Size

[0, 0, 0.2, .2123, .123]
[1, 1, 0.2, .2123, .123]

Batch size 0 to N

MULTIPLE MODELS FOR INFERENCE

- Multiple models can run on different target devices.
  - Interactive Face Recognition Demo
  - License Plate Recognition (Security Barrier Demo)



CUSTOM LAYER EXTENSIONS

- A **layer** is defined as a convolutional neural network (CNN) building block implemented in the training framework (for example, Convolution in Caffe*).
- A **kernel** is defined as the corresponding implementation in the Inference Engine.

- The Inference Engine supports only CPU and GPU custom kernels
  - C++ for CPU target
  - OpenCL for GPU target
- **See**: `/opt/intel/computer_vision_sdk_2018.3.343/deployment_tools/inference_engine/samples/extension , add ext_primitive.cpp`
- Recompile the cpu_extension project
- Then, it is included in cpu_extension.so

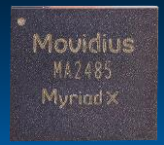
INTEL® NEURAL COMPUTE STICK 2

MORE CORES. MORE AI INFERENCE

INTRODUCING
INTEL® NEURAL COMPUTE STICK 2

A Plug-and-Play Deep Learning Development Kit

POWERED BY



Intel® Movidius™ Myriad™ X VPU delivers industry leading performance

+

OpenVINO™

Intel® Distribution of OpenVINO™ toolkit accelerates solution development and streamlines deployment

DELIVERING UP TO **8X**[1] **HIGHER PERFORMANCE**

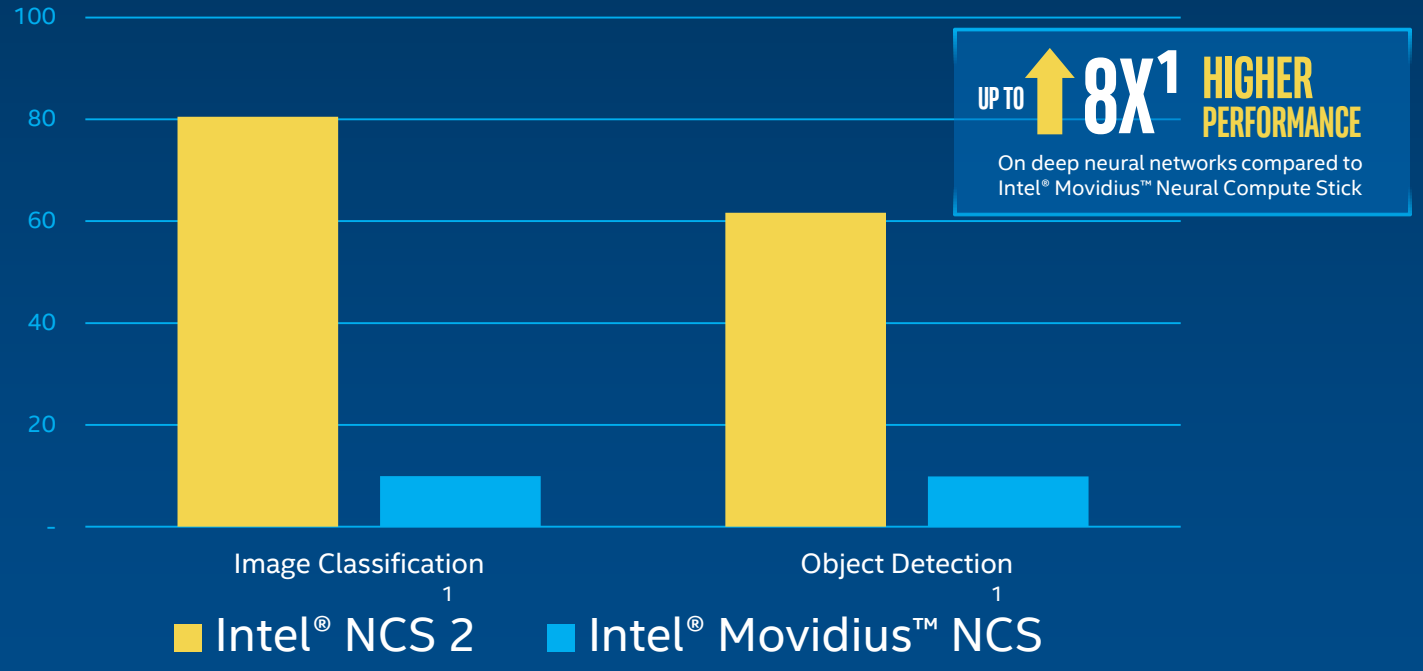
On deep neural networks compared to Intel® Movidius™ Neural Compute Stick

Order now for $99 MSRP*: Where to buy

Internet of Things Group

*MSRP is not a guarantee of final retail price. MSRP may be changed in the future based upon economic conditions.

INTEL® NEURAL COMPUTE STICK 2
MORE CORES. MORE AI INFERENCE



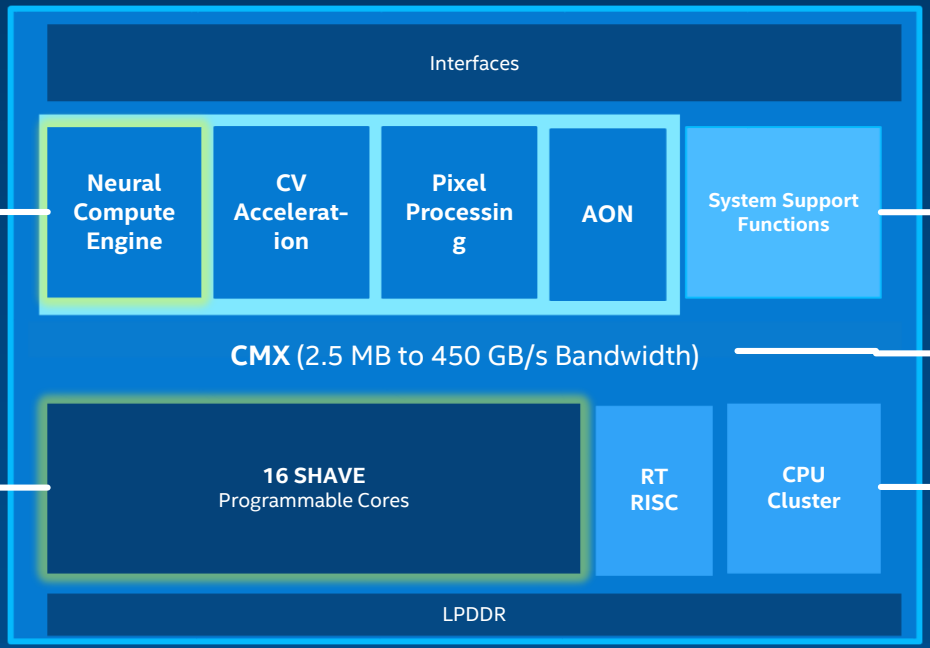
SPECIFICATION COMPARISON

Specifications	Intel® Movidius™ Neural Compute Stick	Intel® Neural Compute Stick 2
Vision Processing Unit (VPU)	The Intel® Movidius™ Myriad™ 2 VPU	The Intel® Movidius™ Myriad™ X VPU
Software development kit	The Intel® Movidius™ Neural Compute SDK	The Intel® Distribution of OpenVINO™ toolkit
OS support	Ubuntu* 16.04, Raspberry Pi* 3 Model B running Stretch desktop or Ubuntu 16.04 Virtual Box instance	Ubuntu* 16.04.3 LTS (64 bit), Windows 10 (64 bit), or CentOS 7.4 (64 bit)
Supported framework	TensorFlow* and Caffe*	TensorFlow* and Caffe*
Connectivity	USB 3.0 Type-A	USB 3.0 Type-A
USB stick dimensions (mm)	72.5mm X 27mm X 14mm	72.5mm X 27mm X 14mm
Operating temperature	0° - 40° C	0° - 40° C
Material Master Number	962297	964486
MSRP	$79 USD as of November 1, 2018	$99 USD November 14, 2018

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

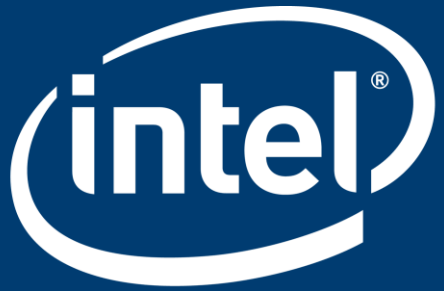


System support functions operate frames, tiles, CODEC, compression and security

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

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

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



Software

HANDS-ON LAB 2

HANDS-ON LAB

OpenVINO™

1. Log-in to your lab PC (intel/P@ssw0rd)
2. Open Firefox and **goto: localhost:8888**, run Jupyter Lab interface
3. Navigate to: `/home/intel/Workshop`
4. Click on **"E2E and Advanced Features – Lab2.ipynb"**

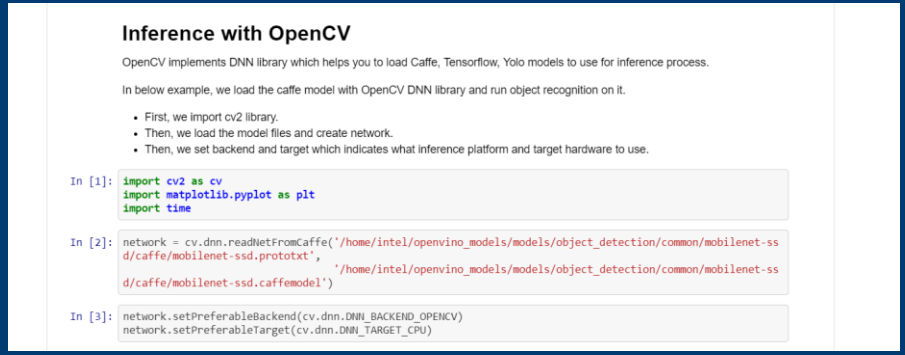
Jupyter Notebook:

- Jupyter notebook is an interactive scripting environment with Markdown support.
- Code part is active and runs on its own environment settings.

WHAT WE WILL COVER?

OpenVINO™

- Inference with OpenCV
- Object Detection Application with MobileNet SSD
  - OpenCV
  - OpenVINO
- Heterogeneous API
- Using Multiple Models on Multiple Platforms with Interactive Face Detection Application
- Dynamic Batching, Performance Counters



Inference with OpenCV

OpenCV implements DNN library which helps you to load Caffe, Tensorflow, Yolo models to use for inference process.

In below example, we load the caffe model with OpenCV DNN library and run object recognition on it.

- First, we import cv2 library.
- Then, we load the model files and create network.
- Then, we set backend and target which indicates what inference platform and target hardware to use.

In [1]:
```
import cv2 as cv
import matplotlib.pyplot as plt
import time
```

In [2]:
```
network = cv.dnn.readNetFromCaffe('/home/intel/openvino_models/models/object_detection/common/mobilenet-ssd/caffe/mobilenet-ssd.prototxt',
                                  '/home/intel/openvino_models/models/object_detection/common/mobilenet-ssd/caffe/mobilenet-ssd.caffemodel')
```

In [3]:
```
network.setPreferableBackend(cv.dnn.DNN_BACKEND_OPENCV)
network.setPreferableTarget(cv.dnn.DNN_TARGET_CPU)
```