VIDEO ANALYTICS IN RETAIL

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

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.

Agenda

- ✓ Video Analytics in Retail
- ✓ Enhanced Video Analytics on edge using deep learning and Intel OpenVINO™ Toolkit
- ✓ Object Detection Using OpenVINO™ Toolkit
- ✓ Questions and Answers

Video Analytics use cases in Retail?

- Counting faces
- Demographics
- Movement detection
- Expression detection
- Face Recognition
- Security
- Monitor wait time
- And So on

VIDEO ANALYTICS ON EDGE USING
DEEP LEARNING AND INTEL OPENVINO™ TOOLKIT

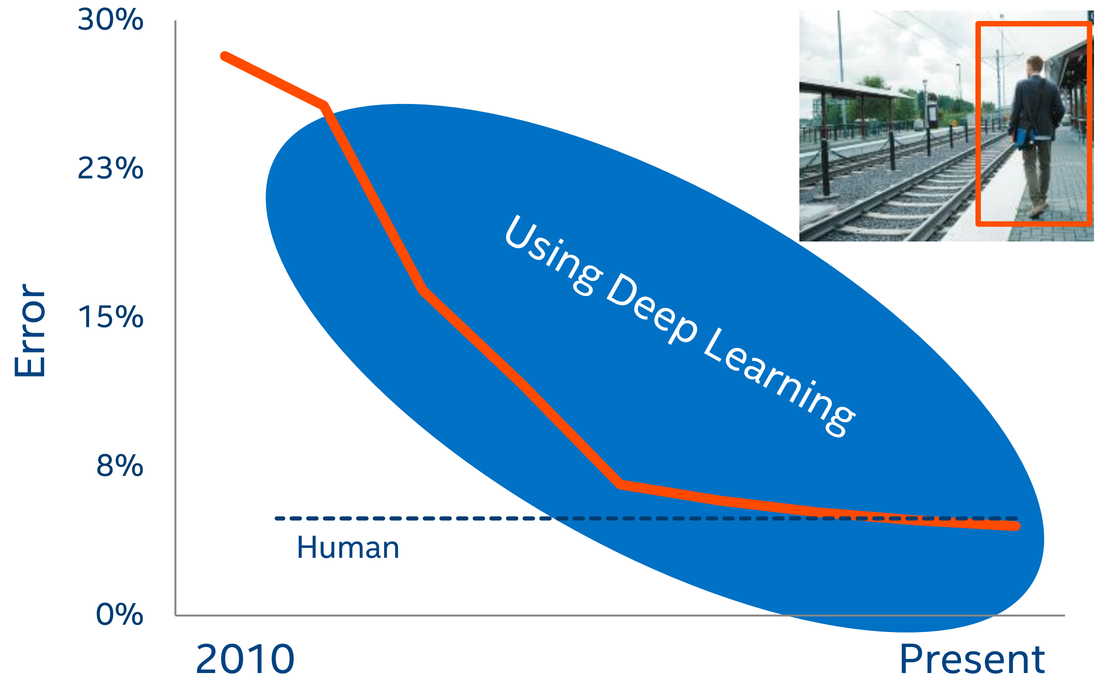
Video: The “Eye of IOT”

Use of video, computer vision, and deep learning is growing rapidly

Deep Learning Usage Is Increasing

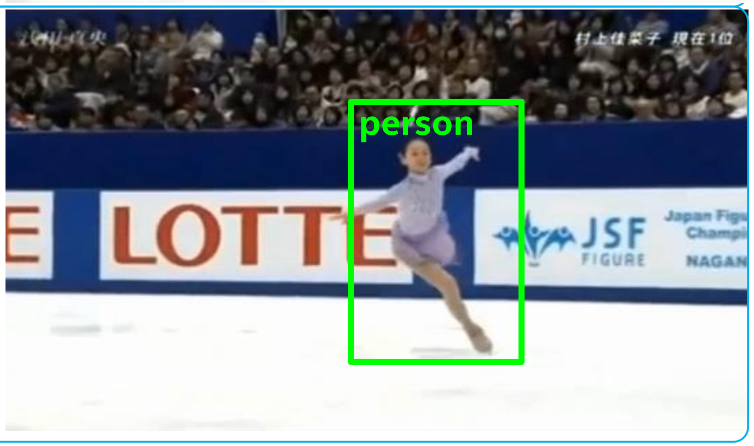
Deep learning revenue is estimated to grow from $655M in 2016 to $35B by 2025[1].

Image Recognition



Traditional Computer Vision
Object Detection

Deep Learning Computer Vision
Person Recognition

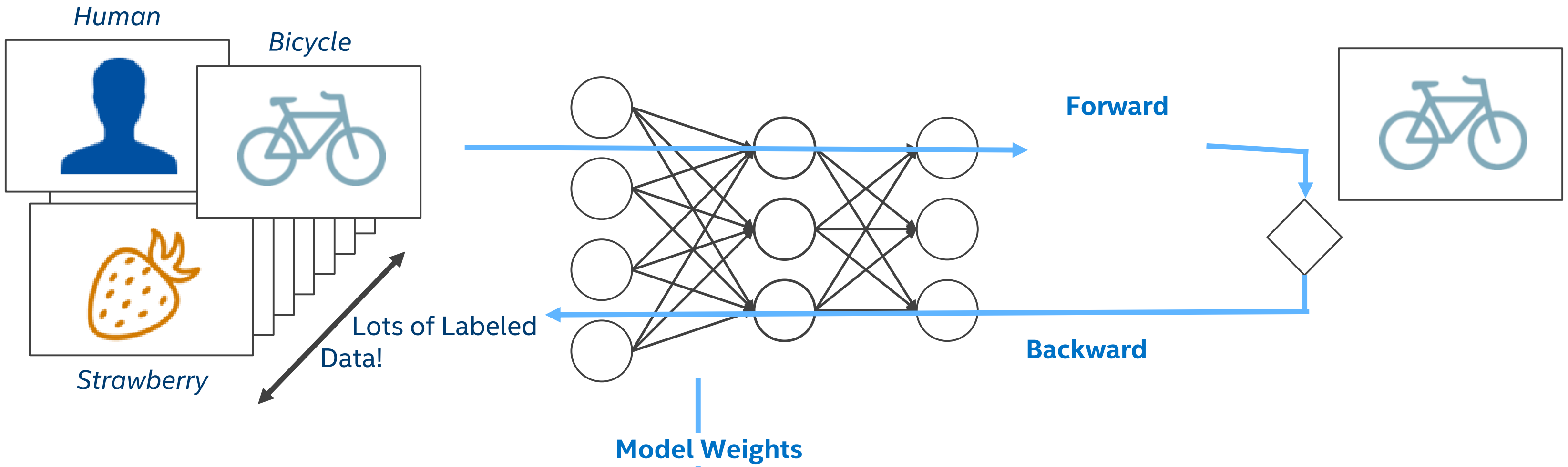


Market Opportunities + Advanced Technologies Have Accelerated Deep Learning Adoption

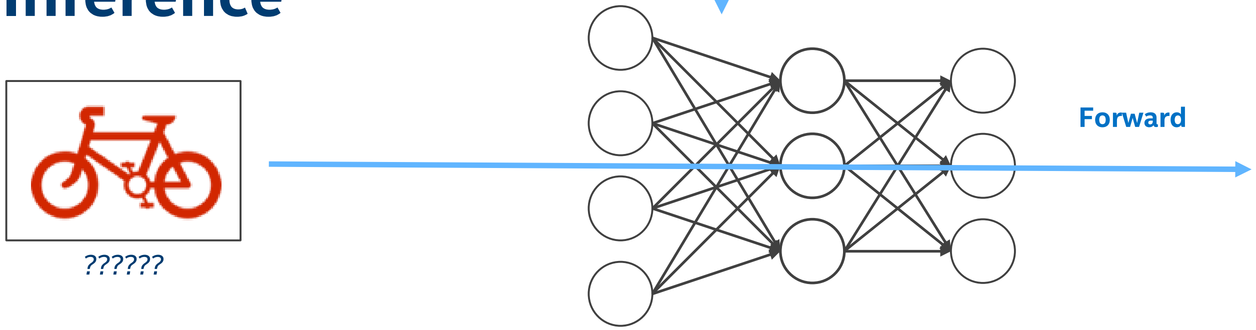
[1]Tractica* 2Q 2017

Deep Learning: Training vs. Inference

Training

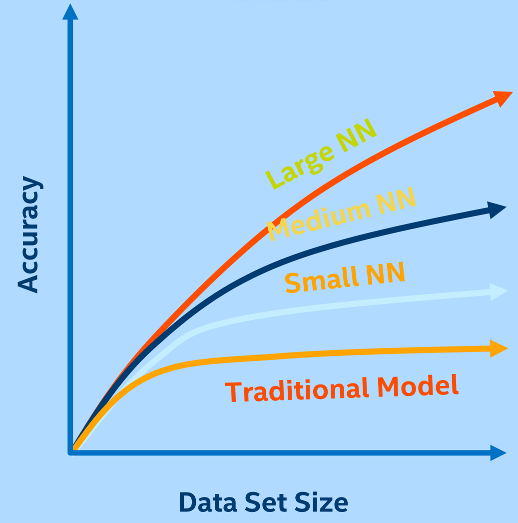


Inference



Did You Know?

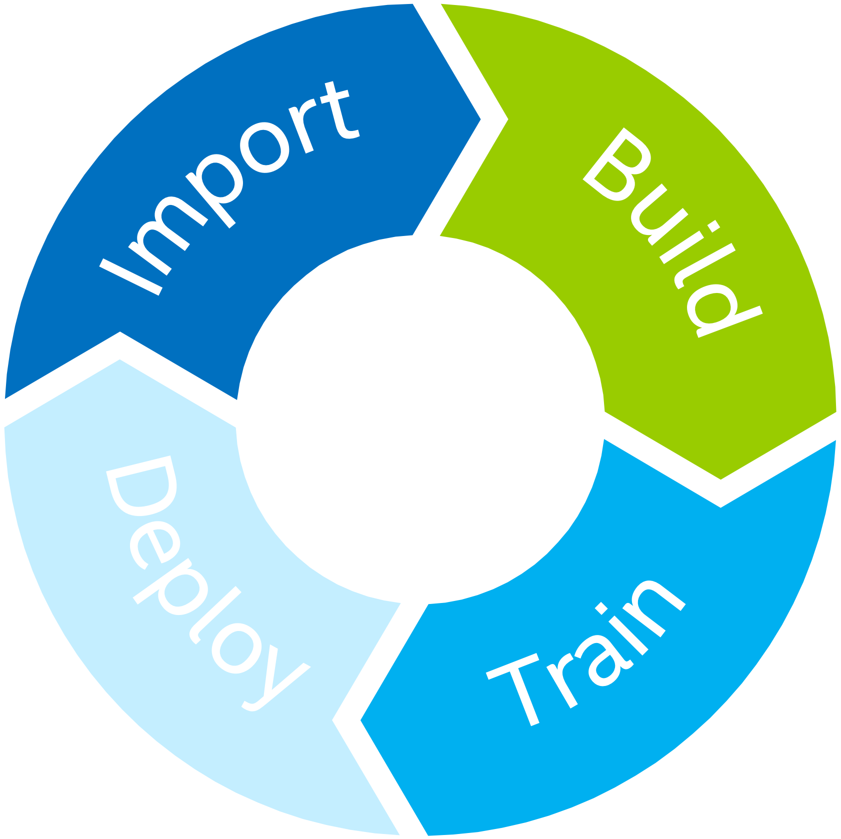
Training requires a very large data set and deep neural network (many layers) to achieve the highest accuracy in most cases



Artificial Intelligence Development Cycle

Data aquisition and organization

Integrate trained models with application code



Create models

Adjust models to meet performance and accuracy objectives

Intel® Deep Learning Deployment Toolkit Provides Deployment from Intel® Edge to Cloud

Intel Internet of Things (IoT) Video Portfolio

Intel Invests in AI, Computer Vision, and Deep Learning for IoT

Smart Cameras	Video Gateways/NVRs	Data Center/Cloud

FPGA Solutions from Intel

+

Intel® Movidius™

Intel® Media SDK, OpenVINO™ Toolkit

Industry's Broadest Media and Computer Vision and Deep Learning Portfolio

Open Visual Inference and Neural Network Optimization (OpenVINO™) Toolkit and Components

TensorFlow*
Caffe*
mxnet*

Trained Models

Inside the OpenVINO™ Toolkit

Intel® Deep Learning Deployment Toolkit

Model Optimizer
Convert and Optimize

IR

Inference Engine
Optimized Inference

Component tools

Traditional Computer Vision—All SDK Versions
Optimized Computer Vision Libraries

OpenCV* | **OpenVX***

Increase Processor Graphics Performance—Linux* Only

Intel® Media SDK
(Open Source Version)

OpenCL™
Intel Integrated Graphics Drivers and Runtimes

Linux for FPGA Only

FPGA Runtime Environment (RTE)
(from Intel® FPGA SDK for OpenCL™)

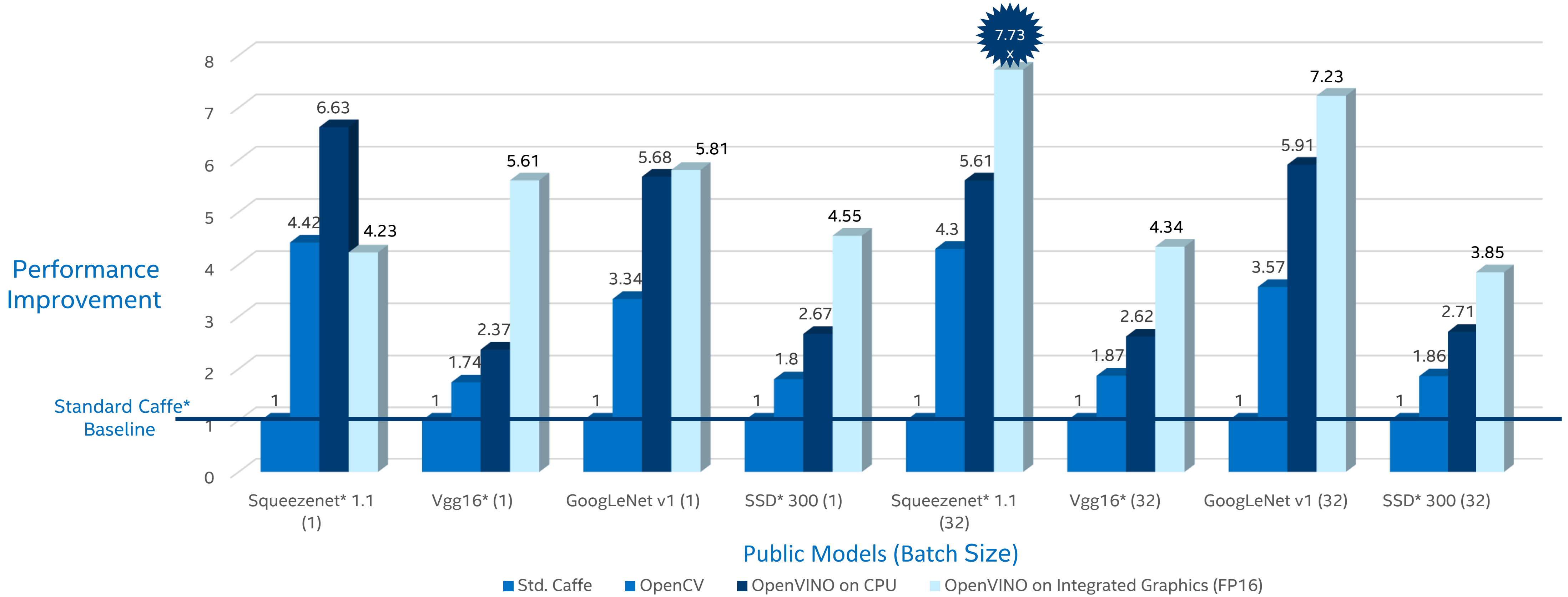
Bit Streams

CPU | **GPU** | **FPGA** | **VPU**

GPU = CPU with Intel Integrated Graphics Processing Unit
VPU = Intel® Movidius™ Vision Processing Unit

Performance Improvement Using the OpenVINO™ Toolkit

Comparison of Frames per Second (FPS)



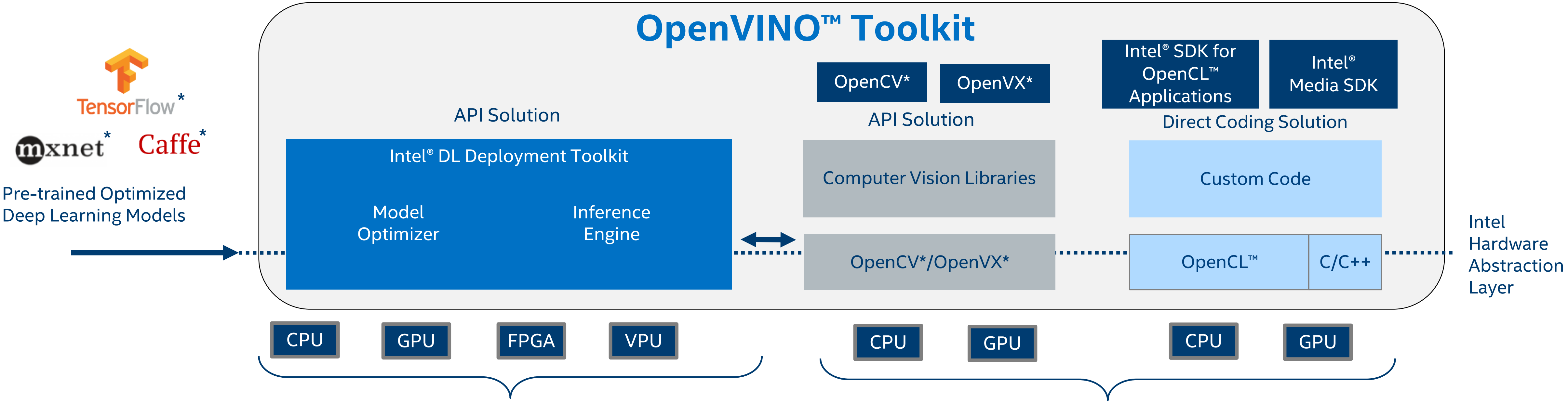
Faster Results on Intel Hardware

[1]Accuracy changes can occur w/FP16

The benchmark results reported in this deck may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks. **Configuration:** Intel® Core™ i7-6700K CPU @ 2.90 GHz fixed, GPU GT2 @ 1.00 GHz fixed Internal ONLY testing, performed 4/10/2018 Test v312.30 – Ubuntu* 16.04, OpenVINO™ 2018 RC4. Tests were based on various parameters, such as model used (these are public), batch size, and other factors. Different models can be accelerated with different Intel hardware solutions, yet use the same Intel software tools. Benchmark Source: Intel Corporation.

Deep Learning vs. Traditional Computer Vision

OpenVINO™ Toolkit End-to-End Vision Pipeline



DEEP LEARNING Computer Vision

- Based on the application of a large number of filters to an image to extract features.
- Features in the object(s) are analyzed with the goal of associating each input image with an output node for each type of object.
- Values are assigned to output node representing the probability that the image is the object associated with the output node.

TRADITIONAL Computer Vision

- Based on selection and connections of computational filters to abstract key features and correlating them to an object.
- Works well with well-defined objects and controlled scene.
- Difficult to predict critical features in larger number of objects or varying scenes.

End to End Process of computer vision with OpenVINO™

Train

Train a DL model.
Currently supports:
- Caffe*
- Mxnet*
- TensorFlow*

Prepare Optimize

Model optimizer:
- Converting
- Optimizing
- Preparing to inference

(device agnostic, generic optimization)

Inference

Inference engine lightweight API to use in applications for inference.

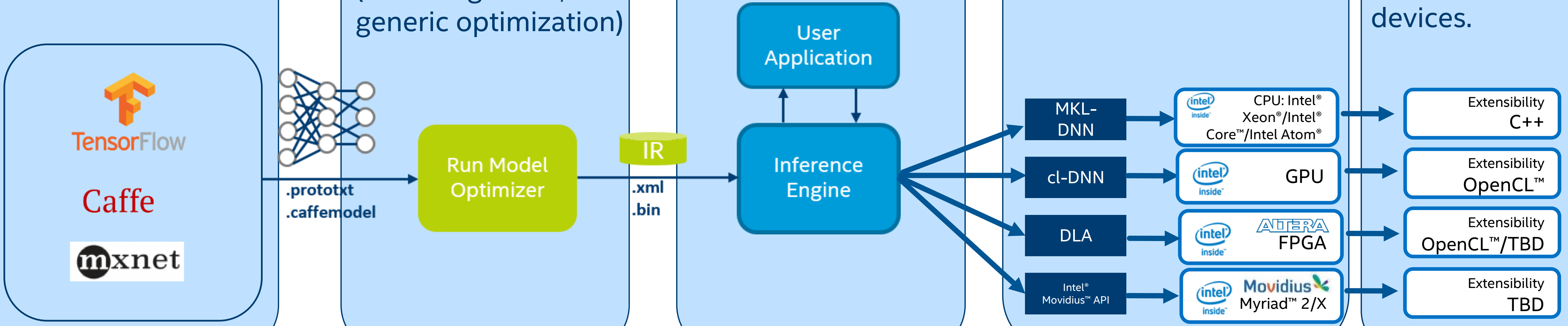
Optimize/ Heterogeneous

Inference engine supports multiple devices for heterogeneous flows.

(device-level optimization)

Extend

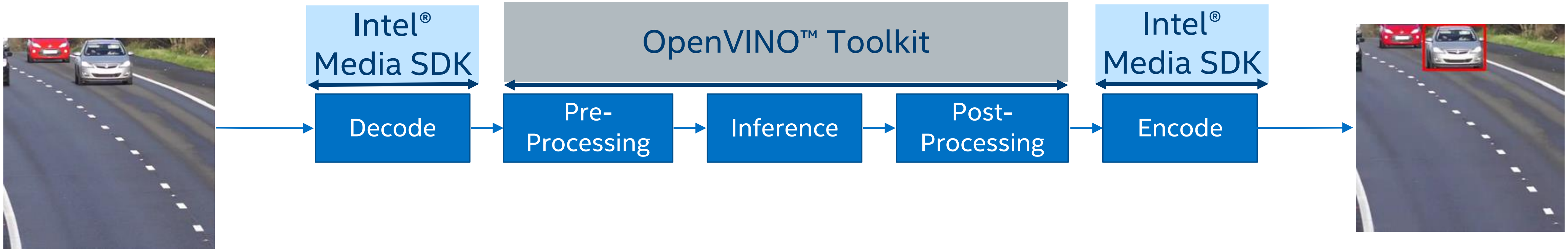
Inference engine supports extensibility and allows custom kernels for various devices.



Accelerate Streaming Performance, Integrate Video Analytics

Computer Vision Needs Intel® Media SDK

Using Intel® Media SDK and the OpenVINO™ toolkit together enables customers to build high performance, intelligent vision solutions.



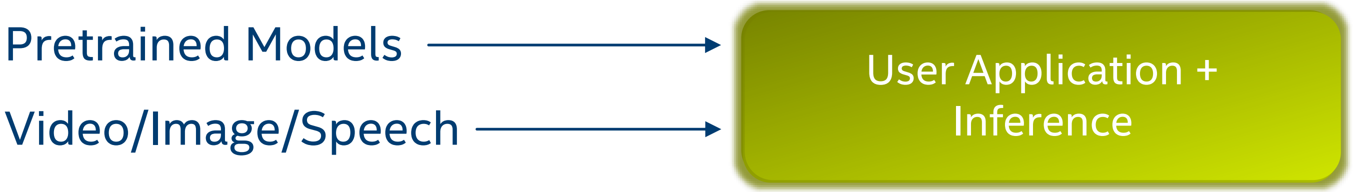
Object Detection Using OpenVINO™ Toolkit

May 2018

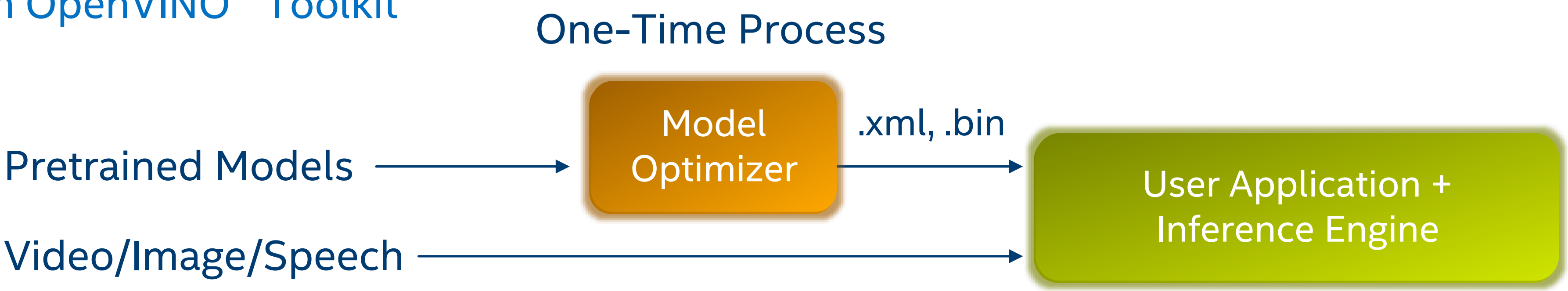
Core and Visual Computing Group

Deep Learning Application Deployment

Traditional



With OpenVINO™ Toolkit

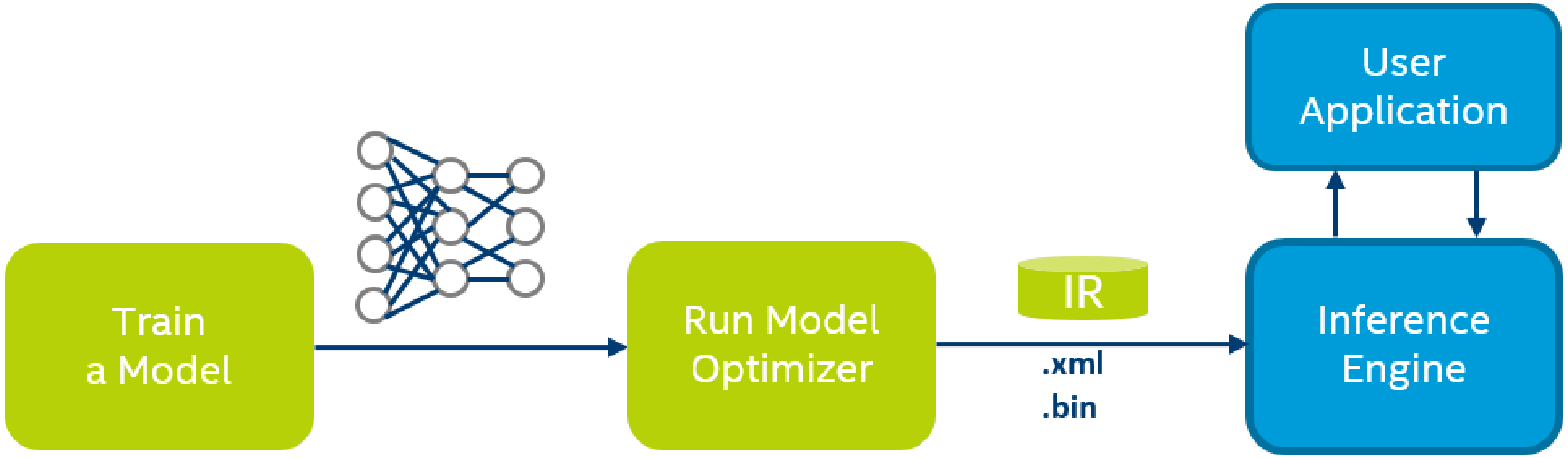


Model Optimizer

Core and Visual Computing Group

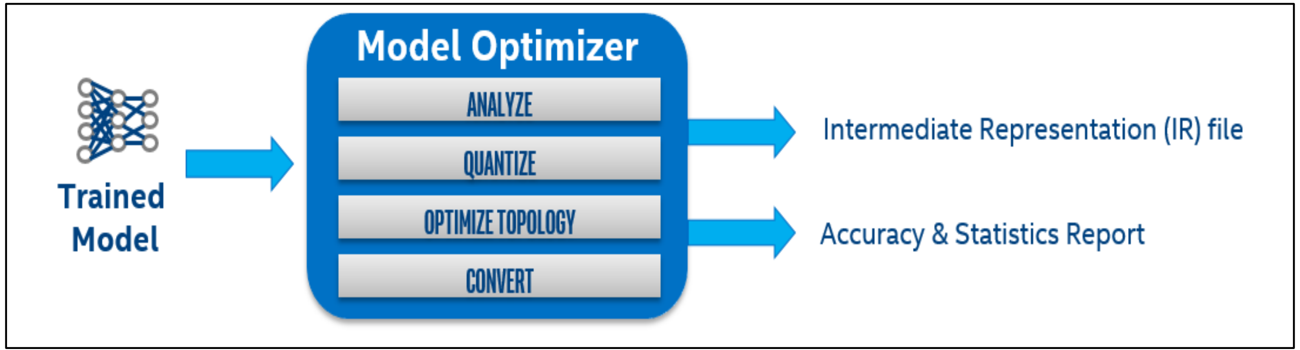
Model Optimizer

- Converts models from various frameworks (Intel® Optimization for Caffe*, Intel® Optimization for TensorFlow*, Apache* MXNet*)
- Converts to a unified model (IR, later n-graph)
- Optimizes topologies (node merging, batch normalization elimination, performing horizontal fusion)
- Folds constants paths in graph



Improve Performance with Model Optimizer

- Easy to use, Python*-based workflow does not require rebuilding frameworks.
- Imports models from various frameworks (Caffe*, TensorFlow*, MXNet*, and more planned).
- More than 100 models for Caffe, MXNet, and TensorFlow validated.
- 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.



DLDT supports a wide range of DL topologies:

- Classification models:
  - AlexNet*
  - VGG-16*, VGG-19
  - SqueezeNet* v1.0/v1.1
  - ResNet*-50/101/152
  - Inception* v1/v2/v3/v4
  - CaffeNet*
  - MobileNet*
- Object detection models:
  - SSD300/500-VGG16
  - Faster-RCNN*
  - SSD-MobileNet v1, SSD-Inception v2
  - Yolo* Full v1/Tiny v1
  - ResidualNet*-50/101/152, v1/v2
  - DenseNet* 121/161/169/201
- Face detection models:
  - VGG Face*
- Semantic segmentation models:
  - FCN8*

Model Optimizer

Model optimizer performs generic optimization:

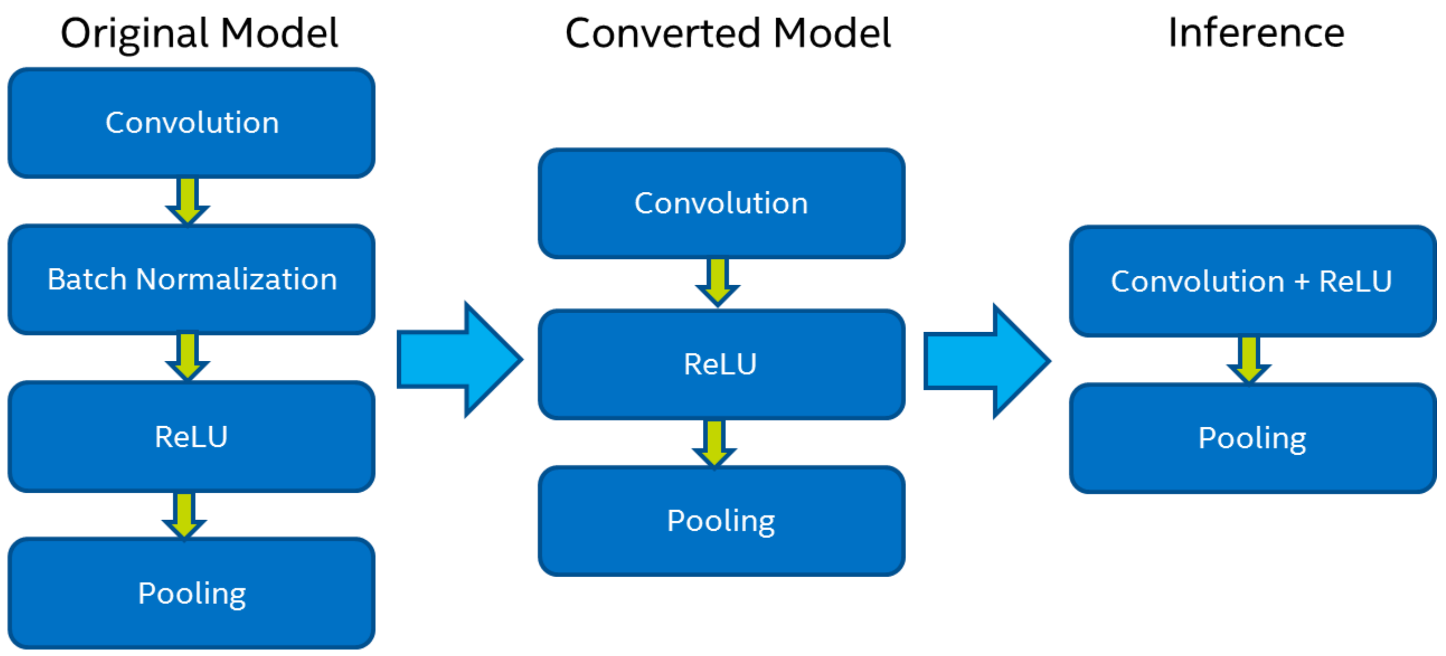
- Node merging
- Horizontal fusion
- Batch normalization to scale shift
- Fold scale shift with convolution
- Drop unused layers (dropout)
- FP16/Int8 quantization
- Model optimizer can add normalization and mean operations, so some preprocessing is 'added' to the deep learning model
- --mean_values (104.006, 116.66, 122.67)
- --scale_values (0.07, 0.075, 0.084)

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

Model Optimizer (1 of 2)

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 (2 of 2)

Model optimizer can change the topology:

- Model optimizer can add normalization and mean operations, so some pre-processing is ‘added’ to the deep learning model.
- --mean_values (104.006, 116.66, 122.67)
- --scale_values (0.07, 0.075, 0.084)

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.
- Use the --input and --output options.

Intermediate Representation (IR)

Squeezenet1.1.caffemodel → Squeezenet1.1.bin

Squeezenet1.1.prototxt → Squeezenet1.1.xml

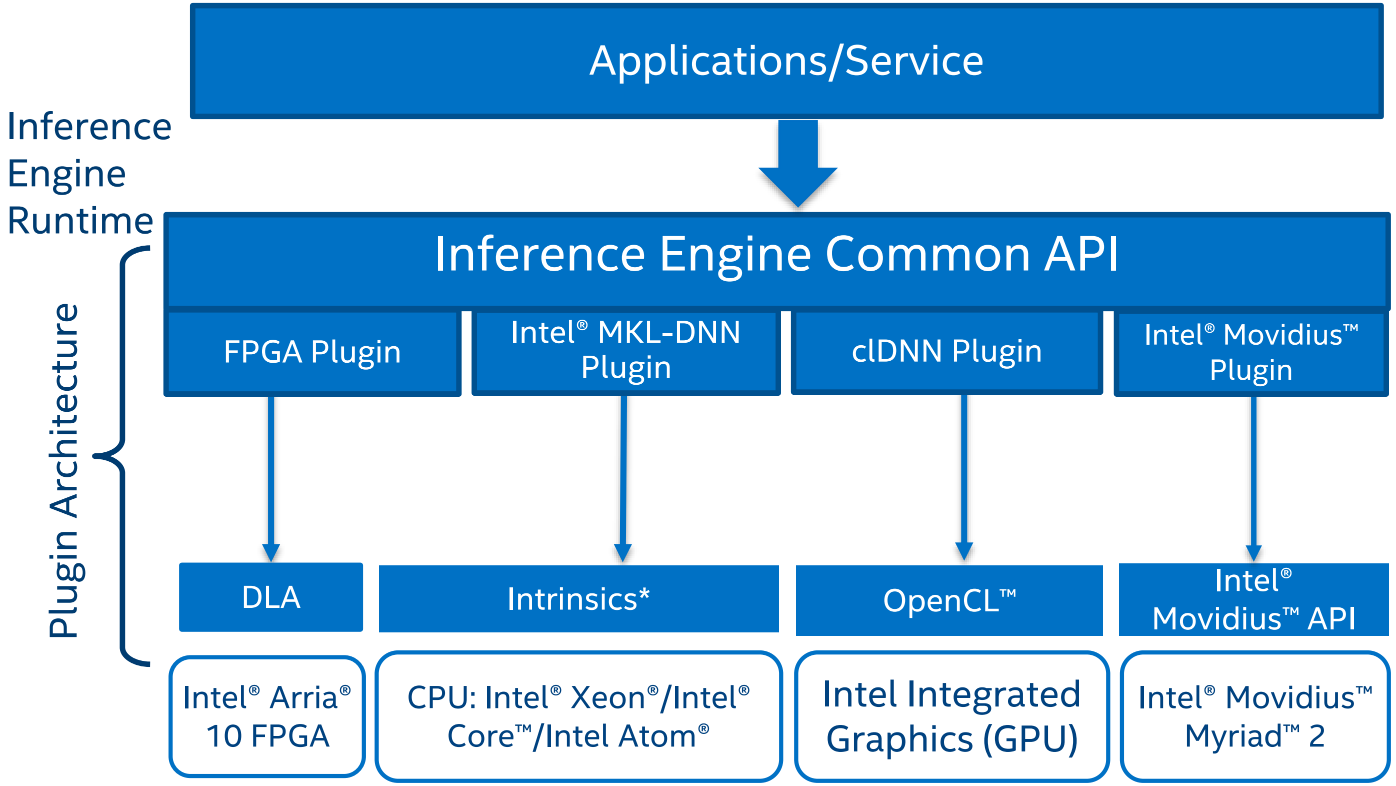
```
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param { shape: { dim: 1 dim: 3 dim: 227 dim: 227 } }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 64
    kernel_size: 3
    stride: 2
  }
}
```

```
<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 Engine

- Simple and unified API for inference across all Intel® architecture
- Optimized inference on large Intel® architecture hardware targets (CPU/GEN/FPGA)
- Heterogeneous support allows execution of layers across hardware types
- Asynchronous execution improves performance
- Futureproof/scale development for future Intel® architecture processors



Layers Supported by Inference Engine Plugins

CPU – Intel® MKL-DNN Plugin

- Supports FP32, INT8 (planned)
- Supports Intel® Xeon®/Intel® Core™/Intel Atom® platforms (https://github.com/01org/mkl-dnn)

GPU – clDNN Plugin

- Supports FP32 and FP16 (recommended for most topologies)
- Supports Gen9 and above graphics architectures (https://github.com/01org/clDNN)

FPGA – DLA Plugin

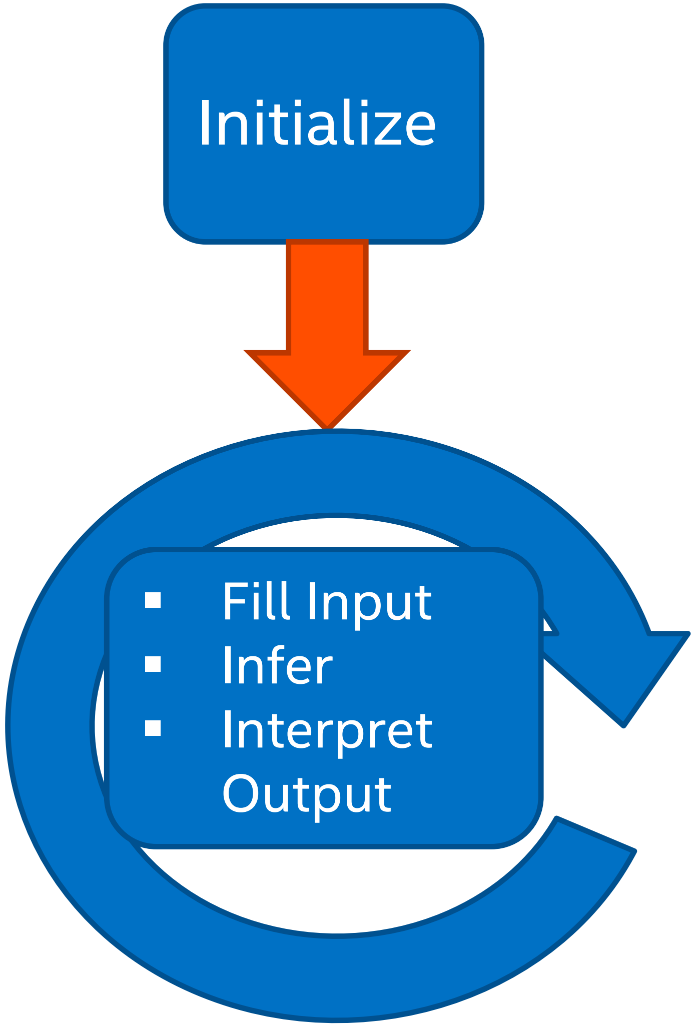
- Supports Intel® Arria® 10
- FP16 data types, FP11 is coming

Intel® Movidius™ – Intel® Movidius™ Myriad™ X Plugin

- Set of layers are supported on Intel® Movidius™ Myriad™ X (28 layers), non-supported layers must be inferred through other inference engine (IE) plugins
- Support FP16

Layer Type	CPU	FPGA	GPU	MyriadX
Convolution	Yes	Yes	Yes	Yes
Fully Connected	Yes	Yes	Yes	Yes
Deconvolution	Yes	Yes	Yes	Yes
Pooling	Yes	Yes	Yes	Yes
ROI Pooling	Yes		Yes	
ReLU	Yes	Yes	Yes	Yes
PReLU	Yes		Yes	Yes
Sigmoid			Yes	Yes
Tanh			Yes	Yes
Clamp	Yes		Yes	
LRN	Yes	Yes	Yes	Yes
Normalize	Yes		Yes	Yes
Mul & Add	Yes		Yes	Yes
Scale & Bias	Yes	Yes	Yes	Yes
Batch Normalizatio	Yes		Yes	Yes
SoftMax	Yes		Yes	Yes
Split	Yes		Yes	Yes
Concat	Yes	Yes	Yes	Yes
Flatten	Yes		Yes	Yes
Reshape	Yes		Yes	Yes
Crop	Yes		Yes	Yes
Mul	Yes		Yes	Yes
Add	Yes	Yes	Yes	Yes
Permute	Yes		Yes	Yes
PriorBox	Yes		Yes	Yes
SimplerNMS	Yes		Yes	
Detection Output	Yes		Yes	Yes
Memory / Delay Ok	Yes			
Tile	Yes			Yes

Workflow



Initialization

- Load model and weights
- Set batch size (if needed)
- Load inference plugin (CPU, GPU, FPGA)
- Load network to plugin
- Allocate input, output buffers

Main Loop

- Fill input buffer with data
- Run inference
- Interpret output results

Load Model

```
//----------------------------------------------------------------------------
// Read network information from XML file
//----------------------------------------------------------------------------

InferenceEngine::CNNNetReader network;
network.ReadNetwork(FLAGS_m);

//----------------------------------------------------------------------------
// Read network parameters from BIN file
//----------------------------------------------------------------------------

std::string binFileName = fileNameNoExt(FLAGS_m) + ".bin";
network.ReadWeights(binFileName.c_str());
```

Load Plugin

```
//------------------------------------------------------------------------------
// Set batch size
//------------------------------------------------------------------------------

network.getNetwork().setBatchSize(FLAGS_batch);

//------------------------------------------------------------------------------
// Load plugin
//------------------------------------------------------------------------------

PluginDispatcher dispatcher({""});
InferenceEnginePluginPtr plugin = dispatcher.getSuitablePlugin(TargetDevice::eCPU);
```

Input

```
//----------------------------------------------------------------------------
// Setting up input
//----------------------------------------------------------------------------

InputsDataMap inputs = network.getNetwork().getInputsInfo();

InputInfo::Ptr inputInfo = inputs.begin()->second;
SizeVector inputDims = inputInfo->getDims();
DataPtr imageData = inputs.begin()->second->getInputData();

//----------------------------------------------------------------------------
// Allocate input blobs
//----------------------------------------------------------------------------

InferenceEngine::BlobMap inputBlobs;
InferenceEngine::TBlob<float>::Ptr input =
        InferenceEngine::make_shared_blob <float, const SizeVector>(Precision::FP32, inputDims);
input->allocate();

inputBlobs[inputs.begin()->first] = input;
```

Output

```
//------------------------------------------------------------------------------
//  Inference engine output setup
//------------------------------------------------------------------------------

OutputsDataMap outputInfo = network.getNetwork().getOutputsInfo();
BlobMap outputBlobs;
SizeVector outputDims = outputInfo.begin()->second->getDims();

TBlob<float>::Ptr output = make_shared_blob<float, const SizeVector>(Precision::FP32, outputDims);
output->allocate();

outputBlobs[outputInfo.begin()->first] = output;

size_t outputSize = outputBlobs.cbegin()->second->size() / batchSize;
```

Prepare Input Data

```
//--------------------------------------------------------------------------
// PREPROCESS STAGE:
// Convert image to format expected by inference engine
// IE expects planar: R plain, G plain, B plain, convert from packed RGBRGB...
//--------------------------------------------------------------------------
// imgIdx: image pixel counter
// channelSize: size of a channel, computed as image width * image height
// inputPtr: a pointer to pre-allocated input buffer
for (size_t i = 0, imgIdx = 0, idx = 0; i < channelSize; i++, idx++) {
    for (size_t ch = 0; ch < inputChannels; ch++, imgIdx++) {
        inputPtr[idx + ch * channelSize] = resized[mb].data[imgIdx];
    }
}
```

Infer

```
//--------------------------------------------------------------------------
// Run inference
//--------------------------------------------------------------------------
sts = plugin->Infer(inputBlobs, outputBlobs, &dsc);
```

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.

intel®