Software

OPEN VISUAL INFERENCE & NEURAL NETWORK OPTIMIZATION (OPENVINO™) TOOLKIT

Legal Notices and Disclaimers

This presentation is for informational purposes only. INTEL MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at intel.com.

This sample source code is released under the Intel Sample Source Code License Agreement.

Intel, the Intel logo, neon, and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

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

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

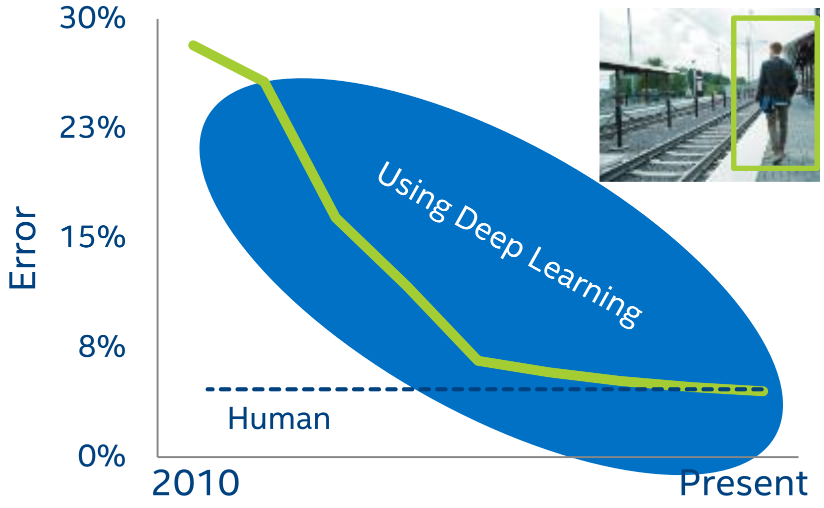
Outline: What students are expected to learn

- Introducing the OpenVINO™ toolkit
- Usage Model
- Tutorial: Model Optimizer
- Tutorial: Inference Engine API and Code Walkthrough
- Summary

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

Open Visual Inference & Neural network Optimization (OpenVINO™) toolkit

Accelerate Computer Vision Solutions

Free Download

https://software.intel.com/en-us/openvino-toolkit

What it is

A toolkit to fast-track development of **high performance computer vision** and **deep learning into vision applications.** It enables deep learning on hardware accelerators and easy **heterogeneous** execution across Intel® platforms. Components include:

- Intel® Deep Learning Deployment Toolkit (model optimizer, inference engine)
- Optimized functions for OpenCV* and OpenVX*

Why important

Demand is growing for intelligent vision solutions. **Deep learning revenue** is estimated to grow from $655M in 2016 to **$35B by 2025**[1]. This requires **developer tools** to integrate computer vision, deep learning, and analytics processing capabilities into applications, so they can help **turn data into insights that fuel artificial intelligence**.



Users: Software developers, data scientists working on vision solutions for surveillance, robotics, healthcare, office automation, autonomous vehicles, & more.

OpenVINO™ version is 2018 R1

[1]Tractica 2Q 2017

What's Inside the OpenVINO™ toolkit

INTEL® DEEP LEARNING DEPLOYMENT TOOLKIT

Model Optimizer
Convert & Optimize

IR

Inference Engine
Optimized Inference

Code Samples & 10 Pre-trained Models

TRADITIONAL COMPUTER VISION TOOLS & LIBRARIES

Optimized Computer Vision Libraries

OpenCV* **OpenVX*** **Photography Vision**

Code Samples

For Intel® CPU & CPU with integrated graphics

Increase Media/Video/Graphics Performance

Intel® Media SDK
Open Source version

OpenCL™ Drivers & Runtimes

For CPU with integrated graphics

Optimize Intel® FPGA

FPGA RunTime Environment
(from Intel® FPGA SDK for OpenCL™)

Bitstreams

FPGA – Linux* only

OS Support: CentOS* 7.4 (64 bit), Microsoft Windows* 10 (64 bit), Ubuntu* 16.04.3 LTS (64 bit), Yocto Project* version Poky Jethro v2.0.3 (64 bit)

Intel® Architecture-Based Platforms Support

IR = Intermediate Representation file

Benefits of the OpenVINO™ toolkit

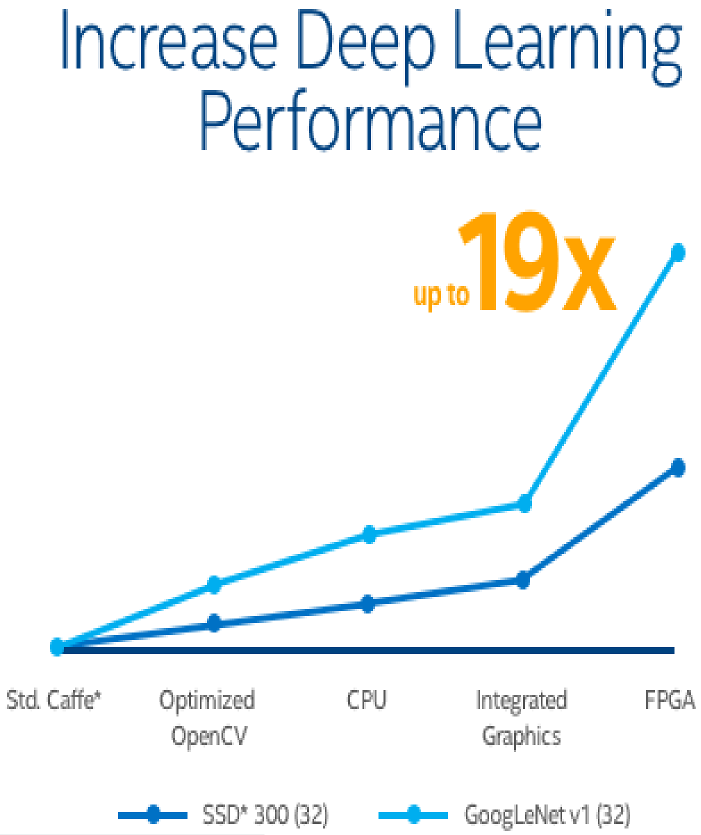
Harness the Power of Intel® Processors: CPU, CPU with Integrated Graphics, FPGA,VPU

ACCELERATE PERFORMANCE	INTEGRATE DEEP LEARNING
Access Intel computer vision accelerators. Speed code performance. Supports heterogeneous processing & Asynchronous execution.	Unleash convolutional neural network (CNN) based deep learning inference across using a common API & 10 trained models. **Up to 10x increase***
SPEED DEVELOPMENT	**INNOVATE & CUSTOMIZE**
Reduce time using a library of optimized OpenCV* & OpenVX* functions, 15+ samples. Develop once, deploy for current & future Intel-based devices.	Use the increasing repository of OpenCL™ starting points in OpenCV* to add your own unique code.

[1]10x performance increase comparing certain standard framework models vs. Intel-optimized models in the Intel®Deep Learning Deployment Toolkit. See Benchmarks slides.
Benchmark results 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. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Performance optimization

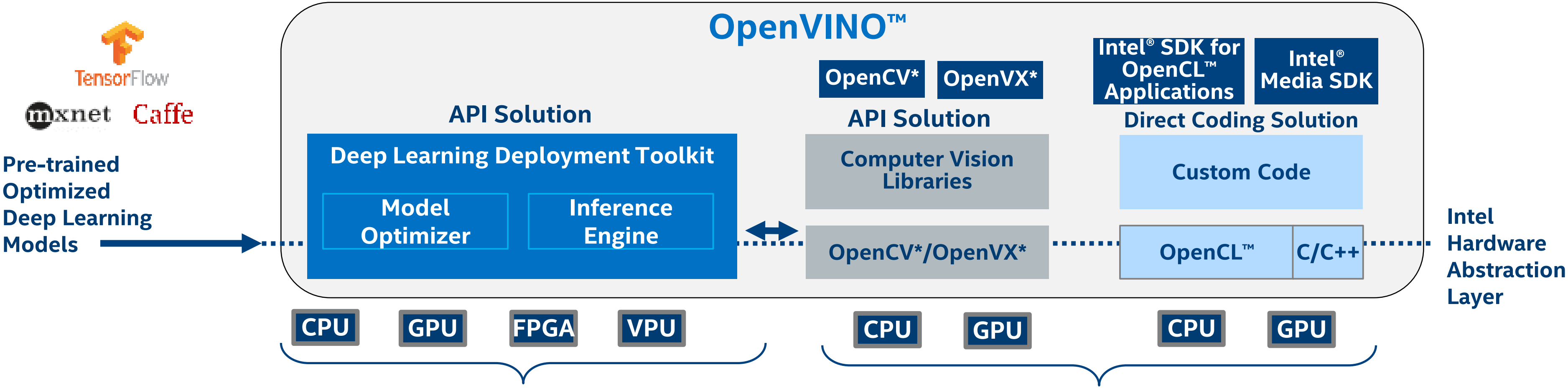
- Increase deep learning workload performance up to 19x.
- Unleash convolutional neural network (CNN)-based deep learning inference using a common API.
- Speed development using optimized OpenCV* and OpenVX* functions.



https://software.intel.com/en-us/openvino-toolkit/

Deep Learning vs. Traditional Computer Vision

OpenVINO™ has tools for an end to end vision pipeline



DEEP LEARNING COMPUTER VISION

- Based on 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

USAGE MODEL

Intel® Deep Learning Deployment Toolkit

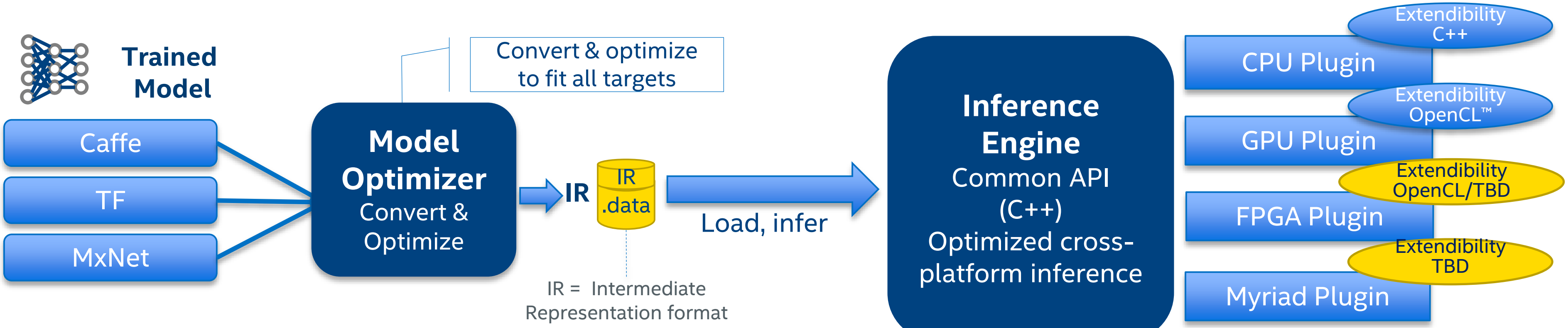
Take Full Advantage of the Power of Intel® Architecture

Model Optimizer

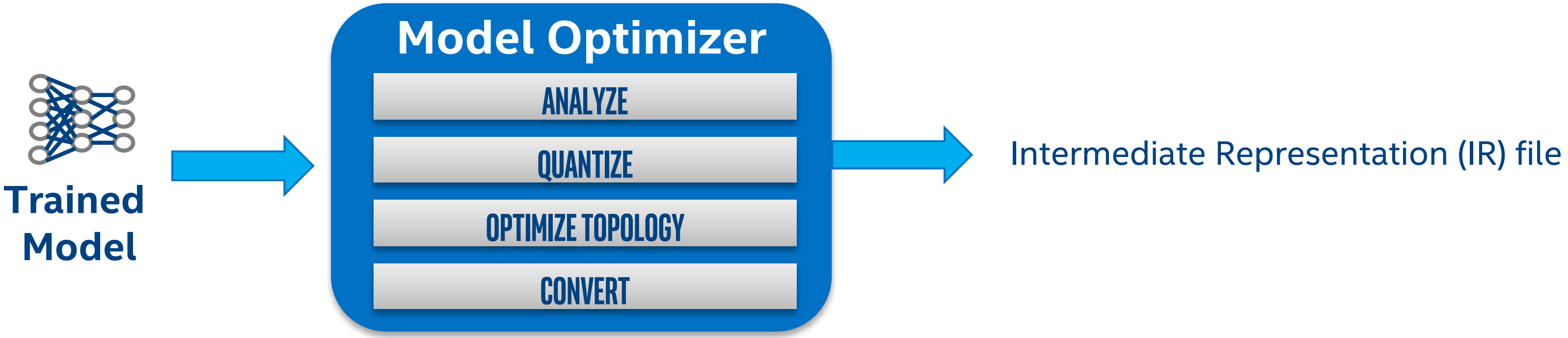
- **What it is**: Preparation step -> imports trained models
- **Why important**: Optimizes for performance/space with conservative topology transformations; biggest boost is from conversion to data types matching hardware.

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.



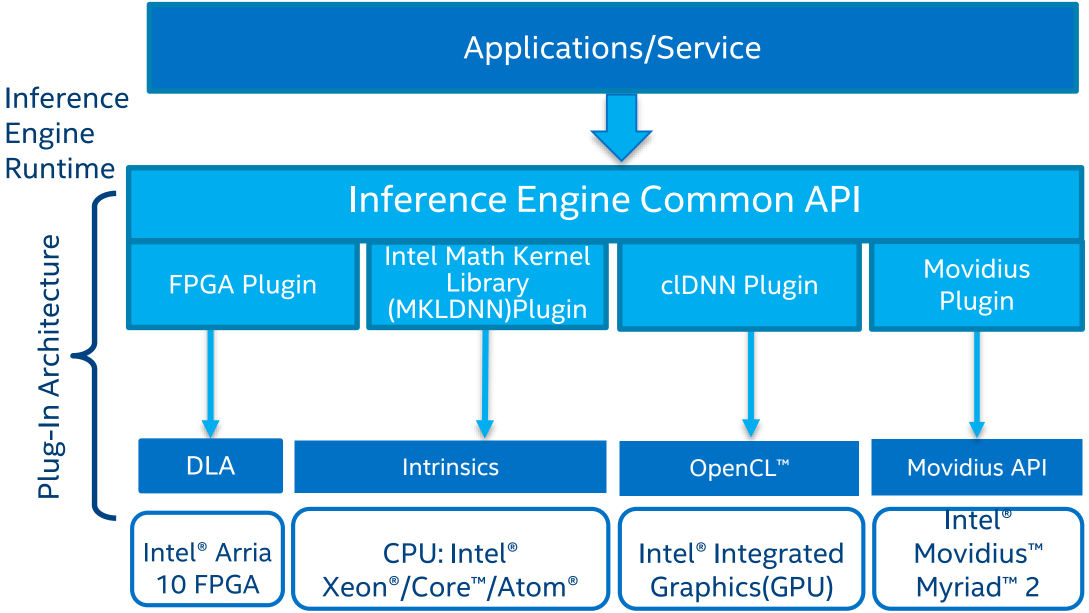
Improve Performance with Model Optimizer



- Easy to use, Python*-based workflow does not require rebuilding frameworks.
- Import Models from various frameworks (Caffe*, TensorFlow*, MXNet*, more are 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

Optimal Model Performance Using the Inference Engine

- Simple & Unified API for Inference across all Intel® architecture (IA)
- 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



Transform Models & Data into Results & Intelligence

Save Time with Deep Learning Samples & Models

Samples

- Image Classification
- Image Segmentation
- Object Detection
- Object Detection for Single Shot Multibox Detector (SSD)
- Neural Style Transfer
- Validation Application

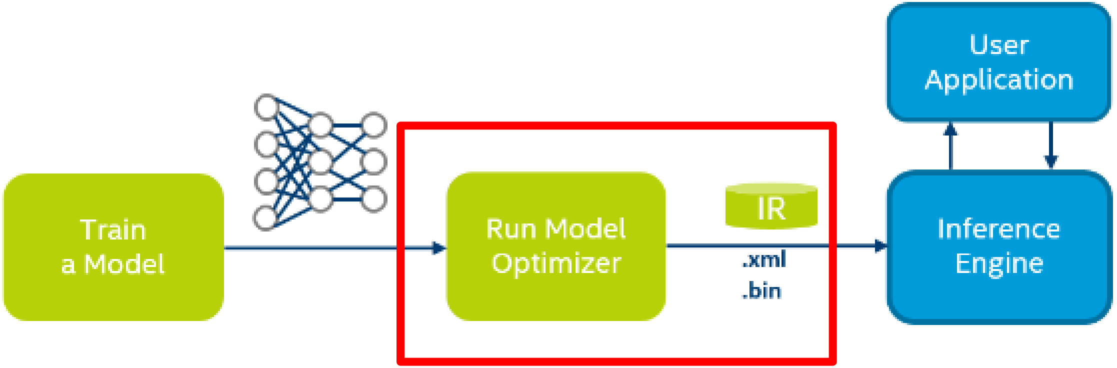
Pre-Trained Models

- Age – Gender
- Security barrier
- Crossroad
- Headpose
- Mobilenet SSD
- Face mobilenet reduced SSD with shared weights
- Face detect with SQ Light SSD
- Vehicle Attributes

TUTORIAL

CREATING INTERMEDIATE REPRESENTATION (IR) FILES TO DEPLOY ON CPU/GPU/NCS

Creating IR files to deploy on CPU/GPU/ NCS



1. Provide as input a trained network that contains the certain topology, and the adjusted weights and biases.
2. Convert the TensorFlow* model to an optimized Intermediate Representation.

Model Optimizer produces as output an Intermediate Representation (IR) of the network, which can be read, loaded, and inferred with the Inference Engine. The Intermediate Representation is a pair of files that describe the whole model:

- **.xml**: Describes the network topology
- **.bin**: Contains the weights and biases binary data

Deploying a model trained with the TensorFlow* framework

A summary of the steps for optimizing and deploying a model that was trained with the TensorFlow* framework:

1. Configure the Model Optimizer for TensorFlow* (TensorFlow was used to train your model).
2. Freeze the TensorFlow* model if your model is not already frozen.
3. Convert a TensorFlow* model to produce an optimized Intermediate Representation (IR) of the model based on the trained network topology, weights, and biases values.
4. Test the model in the Intermediate Representation format using the Inference Engine in the target environment via provided Inference Engine validation application or sample applications.
5. Integrate the Inference Engine in your application to deploy the model in the target environment.

1. Configure the Model Optimizer for TensorFlow*

Configure the Model Optimizer for the TensorFlow* framework running the configuration bash script (Linux* OS) or batch file (Windows* OS) from:

```
<INSTALL_DIR>/deployment_tools/model_optimizer/install_prerequisites folder:

     install_prerequisites_tf.sh

     install_prerequisites_tf.bat
```

2. Freeze the TensorFlow* model with TensorFlow* slim

1. Download the repository, including the models:

```
git clone https://github.com/tensorflow/models/
```

2. Export the inference graph for a model. This example uses Inception V1:

```
python models/research/slim/export_inference_graph.py \
  --alsologtostderr \
  --model_name=inception_v1 \
  --output_file=train_dir/inception_v1_inf_graph.p
```

3. Freeze the graph; use the script freeze_graph.py:

```
python tensorflow/tensorflow/python/tools/freeze_graph.py \
    --input_graph=train_dir/inception_v1_inf_graph.pb \
    --input_checkpoint=train_dir/model.ckpt \
    --input_binary=true \
    --output_graph=train_dir/frozen_inception_v1.pb \
    --output_node_names=InceptionV1/Logits/Predictions/Reshape_1
```

3. Converting a TensorFlow* Model to produce an optimized IR

To convert a TensorFlow* model:

```
Go to the <INSTALL_DIR>/deployment_tools/model_optimizer directory
```

Use the mo_tf.py script to simply convert a model with the path to the input model .pb file with the output Intermediate Representation called result.xml and result.bin that are placed in the specified ../../models/:

```
python mo_tf.py --input_model <TRAIN_DIR>/frozen_inception_v1.pb
--model_name result \
--output_dir ../../models/
```

Launching the Model Optimizer for model .pb file, with reversing channels order between RGB and BGR, specifying mean values for the input and the precision of the Intermediate Representation to be FP16:

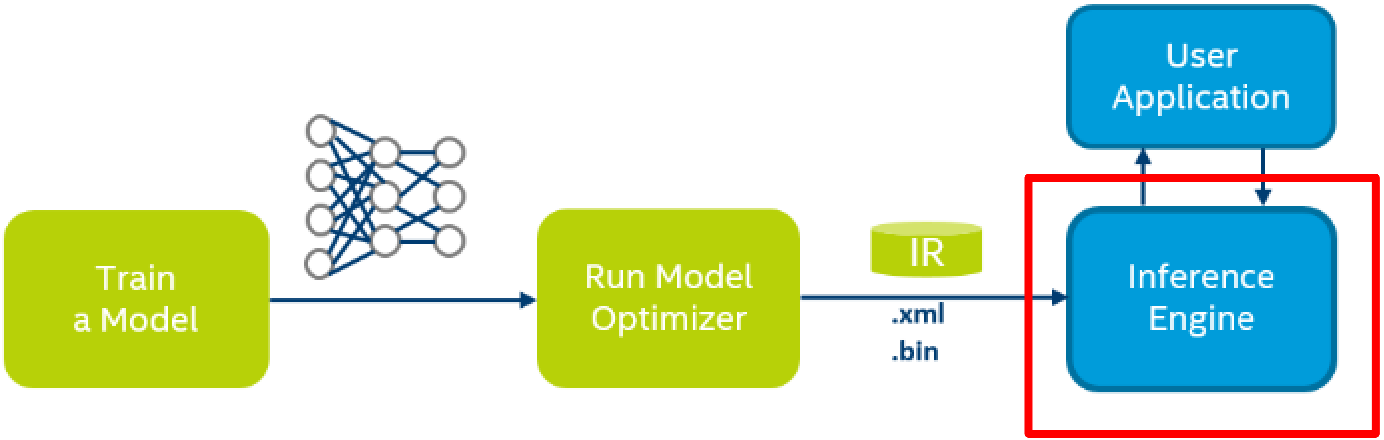
```
python mo_tf.py --input_model <TRAIN_DIR>/frozen_inception_v1.pb \
--reverse_input_channels \
--mean_values [255,255,255] \
--data_type FP16
. . . . . .
```

Software

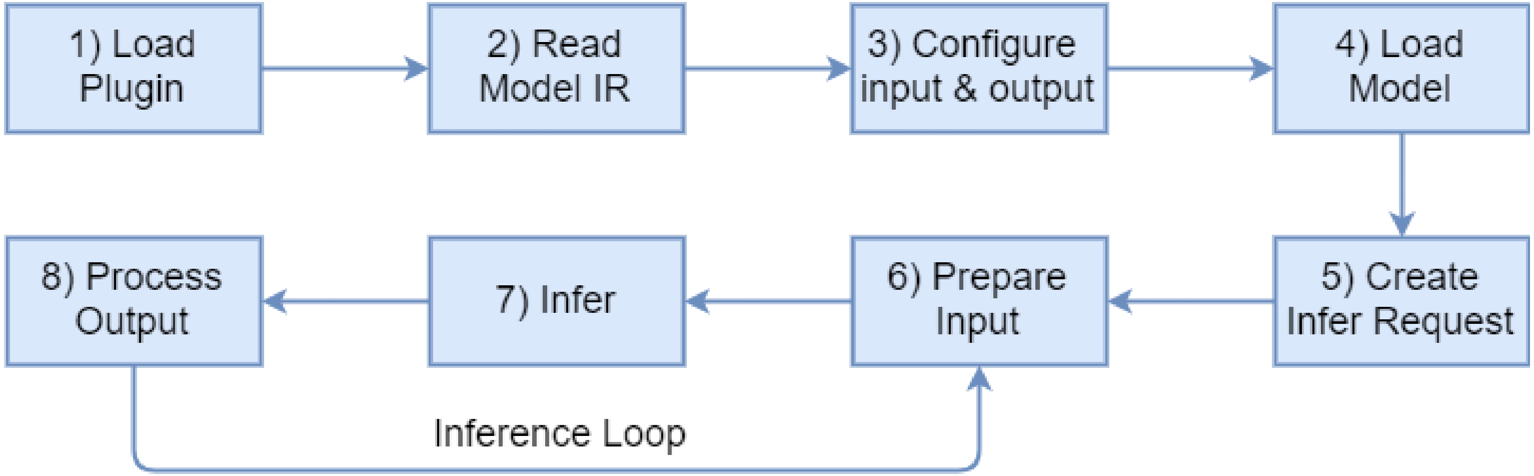
OPENVINO TOOLKIT: INFERENCE ENGINE API WALKTHROUGH

OpenVINO™ Inference Engine

- The Inference Engine is a C++ library with a set of C++ classes that application developers use in their application to infer input data (images) and get the result.
- The library provides you an API to read the IR, set input and output formats and execute the model on various devices.



OpenVINO™ App Execution flow



Steps in OpenVINO™ App Execution

Below are the steps to take when Using the Inference Engine API in Your Code.

1. Load Plugin – e.g. Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN), clDNN
2. Read IR Generated by ModelOptimizer
3. Configure Input/Output Buffers and Set Target Device – e.g. CPU, GPU, Myriad/VPU, FPGA.
4. Load Model into Plugin
5. Create the Infer Request
6. Prepare Input - Capture Frame from Camera and Pass to Inference Engine Input Buffer
7. Do Inference - Execution of Inference Request
8. Post-process Results

1. Load the Plugin

```
InferenceEngine::PluginDispatcher dispatcher({""});

InferenceEngine::InferenceEnginePluginPtr
_plugin(dispatcher.getPluginByDevice(“CPU”));

InferencePlugin plugin(_plugin);
```

2. Read IR Generated by ModelOptimizer

```
CNNNetReader network_reader;
network_reader.ReadNetwork(<<model name>>);
network_reader.ReadWeights(<<model name>> + ".bin");
network_reader.getNetwork().setBatchSize(1);
CNNNetwork network = network_reader.getNetwork();
```

3a. Set Input Buffer

```
/** Get information about topology inputs **/
InferenceEngine::InputsDataMap input_info(network.getInputsInfo());
InferenceEngine::SizeVector inputDims;
for (auto &item : input_info) {
    auto input_data = item.second;
    input_data->setPrecision(Precision::U8);
    input_data->setLayout(Layout::NCHW);
    inputDims=input_data->getDims();
}
```

3b. Set Output Buffer

```
/** Get information about topology outputs **/
output_mode_type output_mode;
InferenceEngine::OutputsDataMap
output_info(network.getOutputsInfo());
InferenceEngine::SizeVector outputDims;
for (auto &item : output_info) {
    auto output_data = item.second;
    output_data->setPrecision(Precision::FP32);
        outputDims=output_data->getDims();
}
```

4. Load Model to Plugin

```
auto executable_network = plugin.LoadNetwork(network, {});
```

5. Create ASYNC Infer Request

```
auto async_infer_request =
executable_network.CreateInferRequest();
```

6. Prepare Input - Capture Frame from Camera and Set to Inference Engine Input Buffer

```
ofImage img;

if (vidGrabber._grabber.isFrameNew()) {

    img.setFromPixels(vidGrabber._grabber.getPixels());

    img.getPixels().resize(inputWidth, inputHeight);

    img.getPixels().swapRgb();

    img.update();

}

ConvertBGR(img.getWidth(), img.getHeight(), { img.getPixels().getData() },
(float*)inputPtr, mean, scale);
```

7. Do Inference - Execution of Inference Request

```
async_infer_request.StartAsync();

async_infer_request.Wait(IInferRequest::WaitMode::RESULT_READY);
```

8. Post-Processing of Inference Results

```
ParseClassificationResults(output, 1, classes); // only top 1

for (size_t i = 0; i < classes.size(); i++) {
     szBreed = labels[classes[i].classId];
     sprintf_s(str, sizeof(str), "%3.f infers/sec", 1.0f / diff.count());
     szFramerate = str;
     sprintf_s(str, sizeof(str), "%3.1f%% confidence", classes[i].confidence * 100.f);
     szConfidencePct = str;
}
```

OpenVINO™ Technical Specifications

	Intel® Platforms	Compatible Operating Systems
Target Solution Platforms	**CPU** ▪ 6th-8th generation Intel® Xeon® and Core™ processors	▪ Ubuntu* 16.04.3 LTS (64 bit) ▪ Microsoft Windows* 10 (64 bit) ▪ CentOS* 7.4 (64 bit)
	▪ Intel® Pentium® processor N4200/5, N3350/5, N3450/5 with Intel® HD Graphics	▪ Yocto Project* Poky Jethro v2.0.3 (64 bit)
	Iris® Pro & Intel® HD Graphics ▪ 6th-8th generation Intel® Core™ processor with Intel® Iris™ Pro graphics and Intel® HD Graphics ▪ 6th-8th generation Intel® Xeon® processor with Intel® Iris™ Pro Graphics and Intel® HD Graphics (excluding e5 product family, which does not have graphics[1])	▪ Ubuntu 16.04.3 LTS (64 bit) ▪ Windows 10 (64 bit) ▪ CentOS 7.4 (64 bit)
	FPGA ▪ Intel® Arria® FPGA 10 GX development kit ▪ Intel® Programmable Acceleration Card with Intel® Arria® 10 GX FPGA operating systems ▪ OpenCV* and OpenVX* functions must be run against the CPU or Intel® Processor Graphics (GPU)	▪ Ubuntu 16.04.3 LTS (64 bit) ▪ CentOS 7.4 (64 bit)
	VPU ▪ Intel Movidius™ Neural Compute Stick	
Development Platforms	6th-8th generation Intel® Core™ and Intel® Xeon® processors	▪ Ubuntu* 16.04.3 LTS (64 bit) ▪ Windows® 10 (64 bit) ▪ CentOS* 7.4 (64 bit)
Additional Software Requirements	Linux* build environment required components ▪ OpenCV 3.4 or higher ▪ CMake* 2.8 or higher ▪ GNU Compiler Collection (GCC) 3.4 or higher ▪ Python* 3.4 or higher	
	Microsoft Windows* build environment required components ▪ Intel® HD Graphics Driver (latest version)† ▪ Intel® C++ Compiler 2017 Update 4 ▪ Python 3.4 or higher ▪ OpenCV 3.4 or higher ▪ CMake 2.8 or higher ▪ Microsoft Visual Studio* 2015	
External Dependencies/Additional Software		View Product Site, detailed System Requirements

[1]Graphics drivers are required only if you use Intel® Processor Graphics (GPU).

Legal Disclaimer & Optimization Notice

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.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

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 of 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. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.
Notice revision #20110804

SUMMARY

Summary

- The **OpenVINO** toolkit provides fast-track development of high performance computer vision and deep learning into vision applications on the PC. It deploys a Model Optimizer and an Inference Engine.
- The **Model Optimizer** is a Python*-based workflow that imports trained models and optimizes for performance/space with conservative topology transformations. Produces as output an Intermediate Representation (IR) of the network, which can be read, loaded, and inferred with the Inference Engine.
- The **Inference Engine** is a high-level inference API with C++ and Python bindings. It presents an interface that is implemented as dynamically loaded plugins for each hardware type.
- **Benefits** of the OpenVINO toolkit are development of highly performant vision applications, decreased time to market, and a robust development ecosystem for cross-platform applications.
- For download, please go to: https://software.intel.com/en-us/openvino-toolkit

experience
what's inside™

Intel as Best Choice for Vision Solutions

Deliver High Performance Computer Vision & Deep Learning – Transform Data & Results into Artificial Intelligence

Intel offers the broadest portfolio of hardware and software that help you

- Accelerate workloads for a wide range of solutions and vertical use cases
- Increase application performance through Intel accelerators and flexible heterogeneous architectures[1] (CPU, CPU w/integrated graphics, Vision Processing Units (VPU) and FPGA)
- Drive power, cost and development efficiencies to designs and applications for cameras, gateways, network video recorders (NVR), and servers
- Enable deep learning capabilities for smarter, faster analytics – transform data into artificial intelligence for competitive advantage

OpenVINO™ + additional Intel Software Tools

[1]Certain technical specifications and select processors/skus apply. See product site for more details.