



# DEPLOYING INTEL<sup>®</sup> FPGA FOR DEEP LEARNING INFERENCE WITH OPENVINO<sup>™</sup> TOOLKIT

Class 3

# What's Been Discussed So Far

## In class 1:

- Refresher on deep learning CNN algorithms
- What FPGAs are and why they are excellent accelerators that provide a flexible, deterministic low-latency, high-throughput, and energy-efficient solution for accelerating the constantly changing networks and precisions for Deep Learning (DL) inference

## In class 2:

- The components that make up a computer vision application that includes deep learning inference
- Intel is providing software that abstracts away the hardware platforms allowing computer vision applications to run on heterogeneous systems
- Common programming languages and libraries for computer vision applications

# Agenda

## Introduction to the Open Visual Inference Neural Network Optimization (OpenVINO™) Toolkit

- Overview
- Model Optimizer
- Inference Engine

# Objectives

Explain the components of the Intel® Distribution of OpenVINO™ toolkit.

Explain how to optimize a model from frameworks such as Caffe\* or TensorFlow\*, into a format that the inference engine requires.

Use the inference engine to target the CPU or FPGA accelerator.

# Intel® AI Portfolio

## EXPERIENCES



## FRAMEWORKS



## TOOLS



Intel® Deep Learning  
Deployment Toolkit

Intel® Distribution of  
OpenVINO™ toolkit

## LIBRARIES



Intel Python  
Distribution

Intel® DAAL

Intel® FPGA DL  
Acceleration  
Suite

Intel® Nervana™ Graph  
Intel® Math Kernel Library  
(MKL, MKL-DNN)

Associative  
Memory Base

## HARDWARE



Compute



Memory & Storage



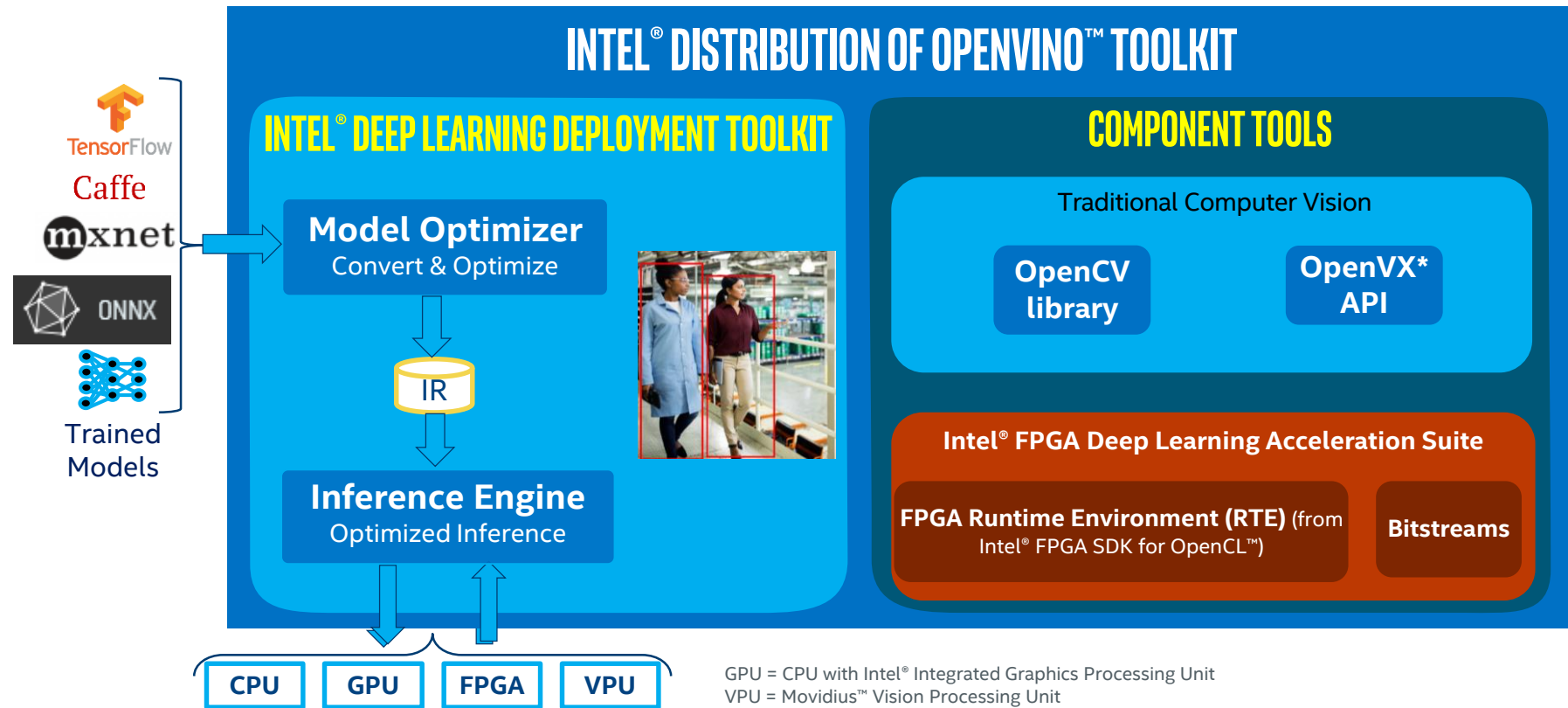
Networking



Visual Intelligence

UNLEASH  
**FULL**  
POTENTIAL

# Intel® Distribution of OpenVINO™ Toolkit



# Deep Learning Deployment Toolkit

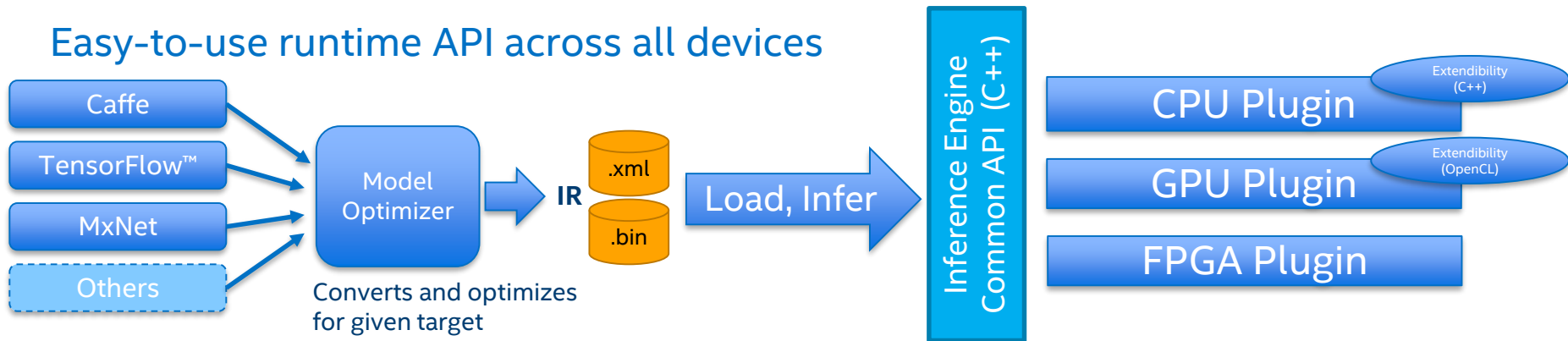
Enable deployment of trained model on all Intel® architectures

- CPU, GPU, VPU, FPGA, ...

Optimize for best execution

Enable user to validate and tune

Easy-to-use runtime API across all devices



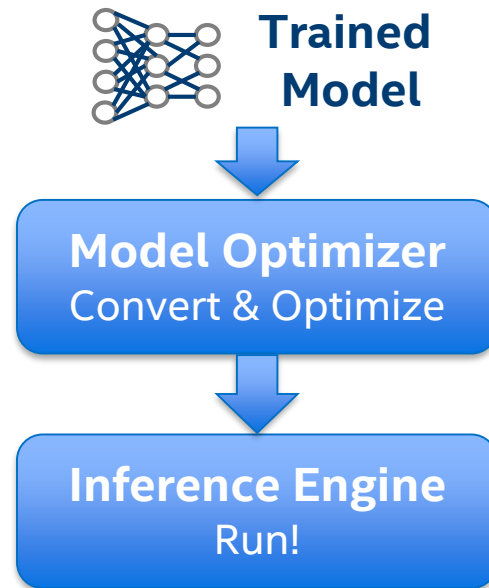
# Deep Learning Deployment Toolkit Details

## Model Optimizer

- Imports trained models from popular deep learning frameworks regardless of training hardware
- Conservative topology transformations
- Converts to a range of data types (Matched to HW)

## Inference Engine

- Optimizes Inference execution for target hardware (computational graph analysis, scheduling, model compression, quantization)
- Enables seamless integration with application logic





# Deployment Toolkit Benefits

To **Speed up deployment** by



**adjusting trained model**



for **target device**



& providing **unified optimized inference runtime**



Easy-to-use Tools:  
Model Optimizer, Inference Engine,  
Validation Application

Model Optimizer Quantization,  
Batch-Normalization Merging

CPU, GPU, FPGA & more

Inference Engine: Simple to use unified  
API of Inference Runtime

- API independent of training framework & target device
- Lightweight to run on IoT devices

# End-to-End Machine Learning

1  
Train

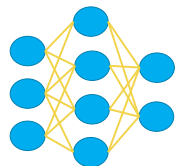


2  
Prepare  
model

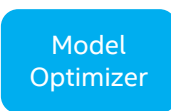


3  
Inference

Caffe\*  
TensorFlow\*  
MXNet\*



Trained Model



Optimized Model

Real-time Data



Inference Engine

MKLDNN

cIDNN

DLA  
Runtime

CPU

GPU

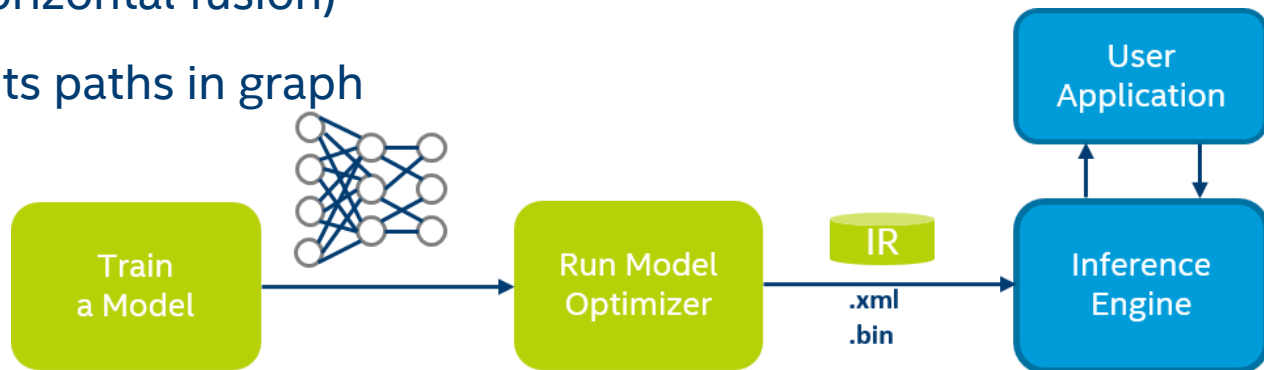
FPGA

# Agenda

- Introduction to Deep Learning Inference on FPGAs
- **Model Optimizer**
- Inference Engine

# Model Optimizer

- Convert models from frameworks (Caffe\*, TensorFlow\*, MXNet\*, ONNX\*)
  - Caffe2, PyTorch, and others via ONNX format
- Converts to a unified Model (IR, later n-graph)
- Optimizes topologies (Node merging, batch normalization elimination, performing horizontal fusion)
- Folds constants paths in graph



# Model Optimizer Performed Optimizations

- 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)

# Model Optimizer Options

Python script: `$MO_DIR/mo.py`

Option for Deployment	Description
<code>--input_model</code>	Network binary weights file TensorFlow* .pb Caffe* .caffemodel MXNet* .params
<code>--input_proto</code>	Caffe .prototxt file
<code>--data_type</code>	IP Precision (i.e., FP16)
<code>--scale</code>	Network normalization factor (Optional)
<code>--output_dir</code>	Output directory path (Optional)

Full Model Optimizer options covered in Model Optimizer documentations

# Run Model Optimizer Caffe\*

## To generate IR .xml and .bin files for Inference Engine

```
$ source $MO_DIR/venv/bin/activate  
$ cd $MO_DIR/
```

```
$ python mo.py \  
--input_model <model_dir>/<weights>.caffemodel \  
--scale 1 \  
--data_type FP16 \  
--output_dir <output_dir>
```

Start working...

```
Framework plugin: CAFFE  
Network type: CLASSIFICATION  
Batch size: 1  
Precision: FP16  
Layer fusion: false  
Horizontal layer fusion: NONE  
Output directory: /home/student/work  
Custom kernels directory:  
Network input normalization: 1  
Writing binary data to:  
/.../GoogLeNet/GoogLeNet.bin
```

# Configure Model Optimizer for TensorFlow\*

Location `$MO_DIR/mo.py`

## Configure MO for TensorFlow\*

1. Install Prerequisites (Python\*, Bazel\*)
2. Install TensorFlow
  1. Clone TensorFlow source, checkout appropriate branch, prepare environment, build TensorFlow, Install TensorFlow wheel
3. Build the Graph Transform Tool via `bazel`
4. Run `model_optimizer_tensorflow/configure.py` script
5. Install Model Optimizer as Python package (`setup.py`)

*Details of each step described in the Model Optimizer documentation*



# Convert TensorFlow\* Models using Model Optimizer

## Generate protobuf binary file (.pb )

- Clone Repository with Models
- Checkout specific revision
- Go to slim directory and modify downloading logic of synset files
- Generate inference graph for model with `export_inference_graph.py`
- Build tool for freezing inference graph
- Freeze inference graph with `freeze_graph`

## Get input and output layer names for the model using `summarize_graph`

- Build and run `summarize_graph`

## Run Model Optimizer (`mo.py`)

# Run Model Optimizer for TensorFlow\*

To generate IR .xml and .bin files for Inference Engine

```
$ cd $MO_DIR

$ python3 mo.py \
  --input_model=$MODEL_DIR/<model>.pb \
  --input=<name of input layer> \
  --output=<name of output layer> \
  --data_type=FP16 \
  --input_shape 1,244,244,3 \
  --model_name <Model Name>
```

Batch size, height, width, number of channels

Output File Name

# Agenda

- Introduction to Deep Learning Inference on FPGAs
- Model Optimizer
- **Inference Engine**

# Inference Engine Structure

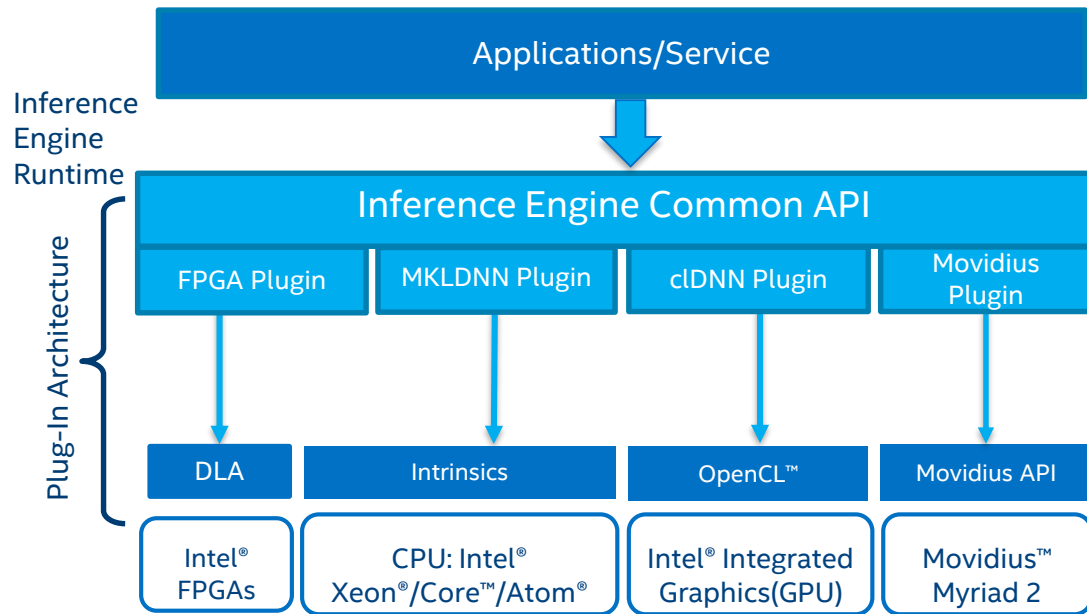
Simple & Unified API for Inference across all Intel® architecture (IA)

Optimized inference on large IA hardware targets (CPU/GPU/FPGA)

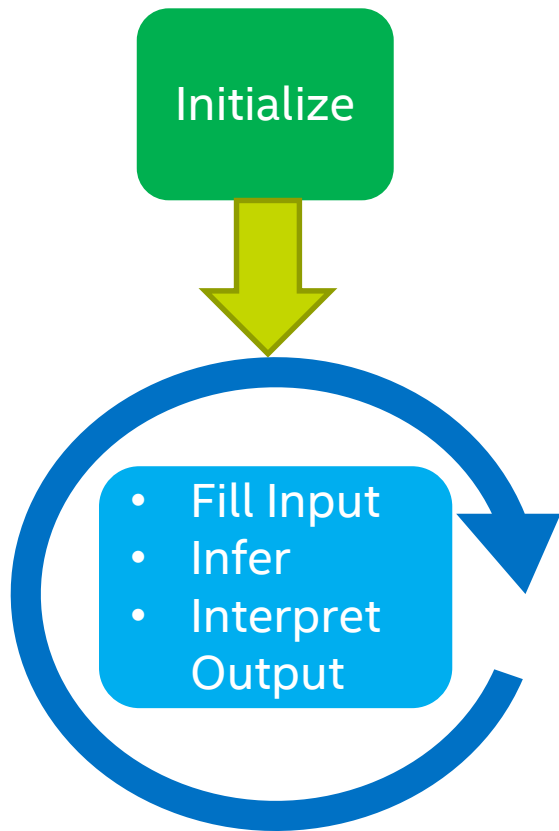
Heterogeneity support allows execution of layers across hardware types

Asynchronous execution improves performance

Futureproof/scale your development for future Intel® processors



# Inference Engine 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

# Inference Engine Details

A runtime with a unified API for integrating inference with application logic

- Delivers optimized inference solution with reduced footprint on inference execution for target hardware

`libinference_engine.so` library implements core functions

- Loading and parsing of model IR
- Preparation of input and output blobs
- Triggers inference using specified plug-in

Include file: `inference_engine.hpp`

# Inference Engine Plugins

## CPU MKLDNN Plugin (Intel® Math Kernel Library for Deep Neural Networks)

- Supports Intel® Xeon®/Core®/Atom® platform
- Widest set of network classes supported, easiest way to enable topology

## GPU cLDNN Plugin (Compute Library for Deep Neural Networks)

- Supports 9<sup>th</sup> generation and above Intel® HD and Iris graphics processors
- Extensibility mechanism to develop custom layers through OpenCL™

## FPGA DLA Plugin

- Supports Intel® Arria 10 GX and above devices
- Basic set of layers are supported on FPGA, non-supported layers inferred through other plugins

# Inference Engine Classes

Class	Details
InferencePlugin, InferenceEnginePluginPtr	Main plugin interface
PluginDispatcher	Finds suitable plug-in for specified device
CNNNetReader	Build and parse a network from given IR
CNNNetwork	Neural Network and binary information
Blob, TBlob, BlobMap	Container object representing a tensor
InputInfo, InputsDataMap	Information about input of the network



# Inference Engine API Usage (1)

## 1. Load Plugin

- FPGA Plugin: `libdliaPlugin.so`
  - Others: `libclDNNPlugin.so` (GPU), `libMKLDNNPlugin.so` (CPU)
- Plugin Dir: `<OpenVINO install dir>/inference_engine/lib/<OS>/intel64`

```
InferenceEngine::PluginDispatcher dispatcher(<pluginDir>);  
InferenceEngine::InferenceEnginePluginPtr enginePtr;  
enginePtr = dispatcher.getSuitablePlugin(TargetDevice::eFPGA);
```

## 2. Load Network

```
InferenceEngine::CNNNetReader netBuilder  
netBuilder.ReadNetwork("<Model>.xml");  
netBuilder.ReadWeights("<Model>.bin");
```

# Inference Engine API Usage (2)

## 3. Prepare Input and Output Blobs

- For Input Blobs
  - Allocate based on the size of the input, number of channels, batch size, etc.
  - Set input precision
  - Fill in data (i.e., from RGB value of image)
- For Output Blobs
  - Set output precision
  - Allocate based on output format

# Inference Engine API Usage (3)

## 4. Load the model to the plugin

```
InferenceEngine::StatusCode status=enginePtr->LoadNetwork(netBuilder.getNetwork(), &resp);
```

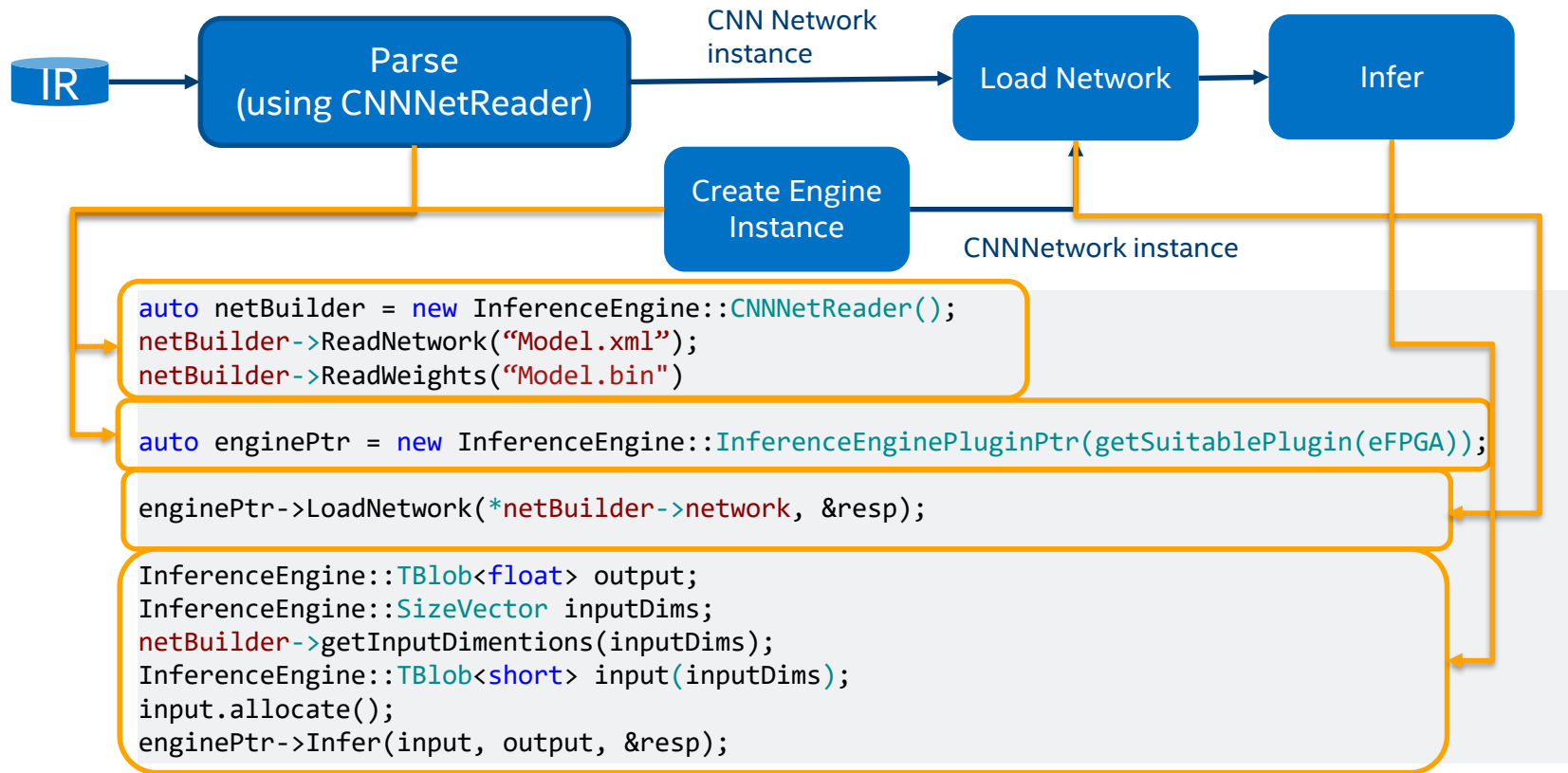
## 5. Perform inference

```
status= enginePtr->Infer(inputBlobs, outputBlobs, &resp);
```

## 6. Process output blobs

```
const TBlob<float>::Ptr fOutput =  
std::dynamic_pointer_cast<TBlob<float>>(outputBlobs.begin()->second);
```

# Using the Inference Engine API



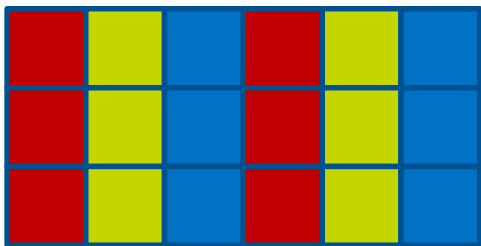
# Pre-processing

Most image formats are interleaved (RGB, BGR, BGRA, etc.)

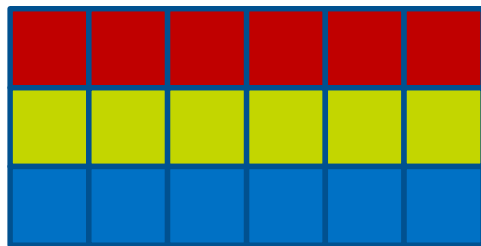
Models usually expect RGB planar format:

- R-plane, G-plane, B-plane

Interleaved



Planar



# Batch Execution

For better performance, using a larger batch size will likely help

Allocate input and output Blob according to batch size

Set the Batch size on the network

```
netBuilder.getNetwork().setBatchSize(<size>);
```

# Automatic Fallback with Hetero Plugin

```
$ classification_sample -d HETERO:FPGA,CPU ...
```

The “priorities” define search order

Keeps all layers that can be executed on the device (FPGA)

Carefully respecting the topological and other limitations

Then follows priorities when searching ( e.g. CPU)

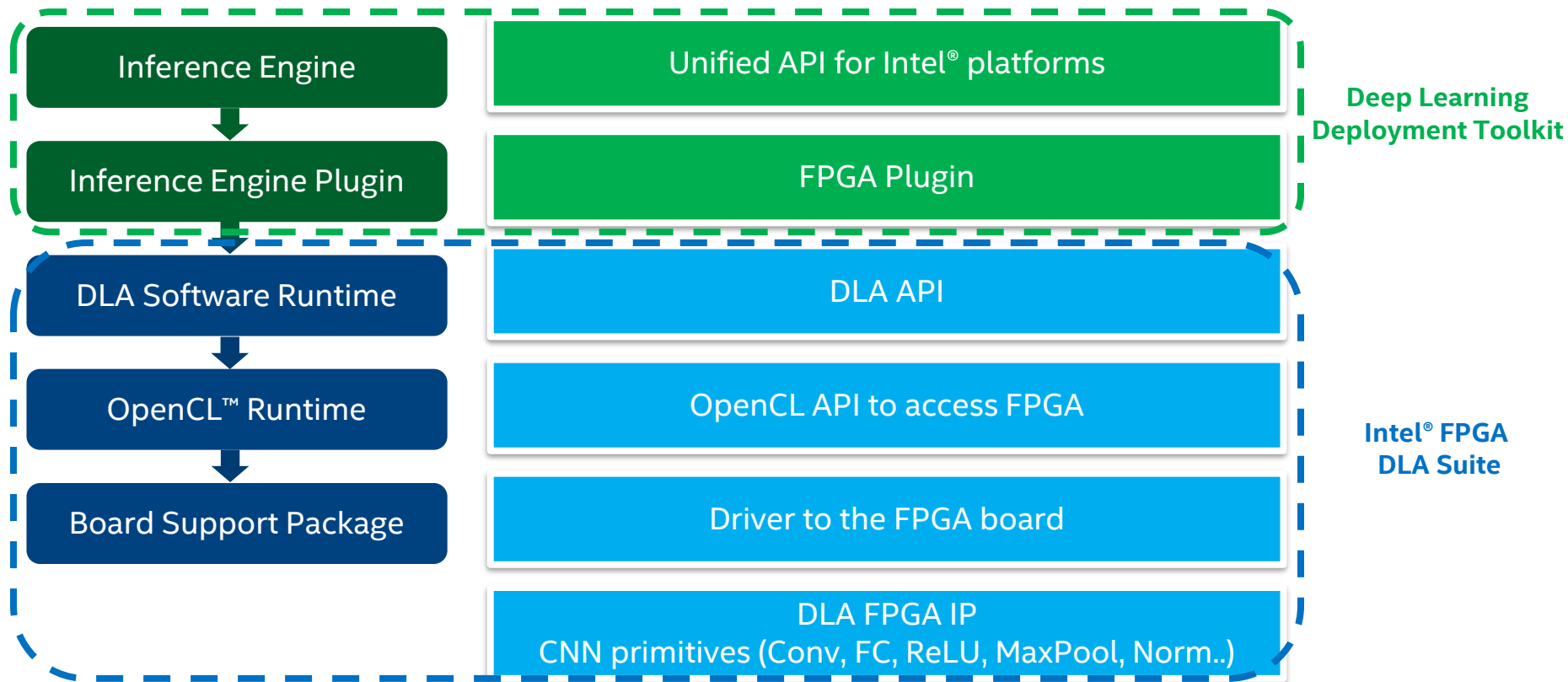
# Inference Engine Example Applications

Execute samples and demos with FPGA support

- Shipped with the Intel® Distribution of OpenVINO™ toolkit
- `classification_sample`
- `object_detection_sample_ssd`
- and many others



# IE Software Architecture with FPGAs



# Prepare FPGA Environment for Inference

Intel® FPGA Runtime Environment for OpenCL™

Prepare FPGA Board for OpenCL

Set environment

- Use script to ensure DLA and OpenCL libraries part of LD\_LIBRARY\_PATH

```
[xkqi@centos-z620 ~]$ aocl diagnose
aocl diagnose: Running diagnose from /opt/intelFPGA_pro/17.0/hld/board/a10_ref/linux64/libexec

----- acl0 -----
Vendor: Intel(R) Corporation

Phys Dev Name  Status  Information
acla10_ref0    Passed  Arria 10 Reference Platform (acla10_ref0)
                                   PCIe dev_id = 2494, bus:slot.func = 04:00.00, Gen3 x8
                                   FPGA temperature = 68.5547 degrees C.

DIAGNOSTIC_PASSED
-----
```

# Load FPGA Image and Execute IE Application

FPGAs needs to be preconfigured with primitives prior to application execution

Choose FPGA bitstream from the DLA suite

- Based on topology needs and data type requirements
- Option to create custom FPGA bitstream based on requirements

```
[xkqi@centos-z620 ~]$ aocl program acl0 $DLA_AOCX
aocl program: Running program from /opt/intelFPGA_pro/17.0/hld/board/a10_ref/linux64/libexec
Programming device: a10gx : Arria 10 Reference Platform (acla10_ref0)
Reprogramming device [0] with handle 1
Program succeed.
[xkqi@centos-z620 ~]$
```

Execute User or Example Application

```
[xkqi@centos-z620 work]$ classification_sample -i mypic.BMP -m GoogleNet/GoogleNet.xml -d FPGA -ni 100
```

# FPGA Image Selection

Precompiled FPGA image available with the toolkit

Choose image based on

- Primitives needed (architecture)
- Data type support for accuracy/performance trade-off
- K (filter) and C (channel depth) vectorization for performance
- Data path width
- On-chip stream buffer depth

May also generate customized FPGA image to meet your needs

# Heterogeneous FPGA + CPU Execution

Use HETERO plugin

- Functions the FPGA does not support falls back to the CPU
- Fallback happens automatically

Manual splitting of original network may be required

- One network fully accelerated on the FPGA device
- Second network (consumes output of the first) executed on CPU or other devices
- Easier to split in original framework model
  - Create separate IR and load to different Inference Engine devices

# Summary

Intel® Distribution of OpenVINO™ toolkit provides a simple to use tool flow with a common front end to take models from common frameworks and target different hardware platforms easily

# Legal Disclaimers/Acknowledgements

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 [www.intel.com](http://www.intel.com).

Intel, the Intel logo, Intel Inside, the Intel Inside logo, MAX, Stratix, Cyclone, Arria, Quartus, HyperFlex, Intel Atom, Intel Xeon and Enpirion are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

OpenCL is the trademark of Apple Inc. used by permission by Khronos

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

© Intel Corporation

