



# INTEL FPGA WORKSHOP

# Today you'll learn

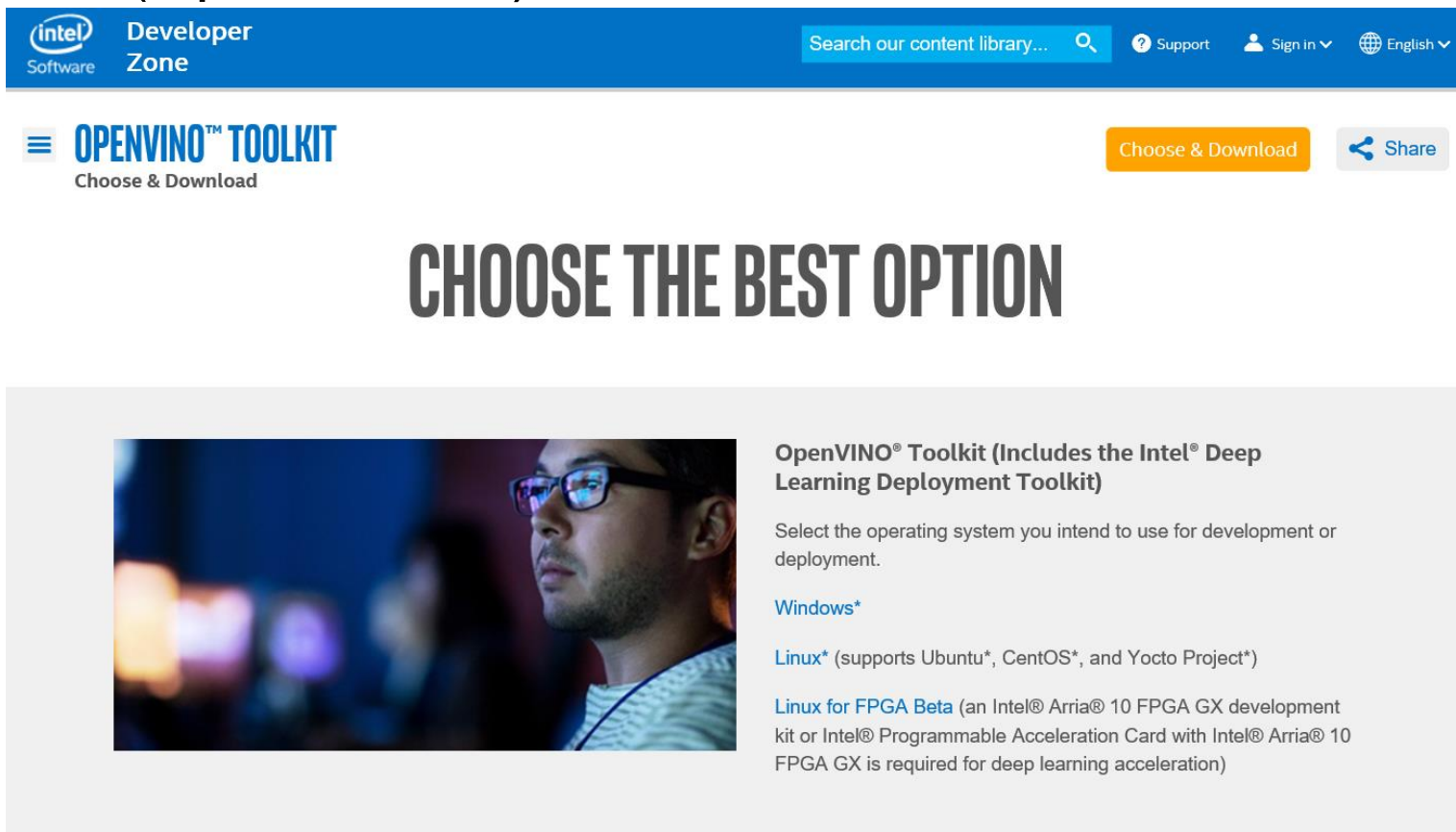
- How to get started with Intel® OpenVINO toolkit
- How to run samples from package
- Deployment workflow
- What's Inference Engine Async API

# OpenVINO toolkit Prerequisites

- System requirements:
  - **Intel® 6<sup>th</sup>, 7<sup>th</sup> or 8<sup>th</sup> generation Intel® Core™/Xeon processors**
  - **Ubuntu 16.04**, CentOS 7.4, Windows 10 (CV SDK SW requirements for needed OS ***for CPU only***)
- Download Intel OpenVINO toolkit:
  - <https://software.intel.com/en-us/openvino-toolkit/choose-download>

# Install the OpenVINO™

Follow the instructions to install Open Visual Inference & Neural network Optimization (OpenVINO™)



The screenshot shows the Intel Developer Zone website for the OpenVINO Toolkit. The header includes the Intel Software logo, 'Developer Zone' text, a search bar, and links for Support, Sign in, and English. Below the header, the 'OPENVINO™ TOOLKIT' is highlighted with a 'Choose & Download' button and a 'Share' button. The main heading is 'CHOOSE THE BEST OPTION'. Below this, there is a section for the 'OpenVINO® Toolkit (Includes the Intel® Deep Learning Deployment Toolkit)'. This section includes a photo of a person wearing glasses and a blue shirt, and text describing the toolkit and its compatibility with Windows, Linux, and Linux for FPGA Beta.

**OpenVINO™ Toolkit**  
Choose & Download

**CHOOSE THE BEST OPTION**

**OpenVINO® Toolkit (Includes the Intel® Deep Learning Deployment Toolkit)**

Select the operating system you intend to use for development or deployment.

[Windows\\*](#)

[Linux\\*](#) (supports Ubuntu\*, CentOS\*, and Yocto Project\*)

[Linux for FPGA Beta](#) (an Intel® Arria® 10 FPGA GX development kit or Intel® Programmable Acceleration Card with Intel® Arria® 10 FPGA GX is required for deep learning acceleration)

For more complete information about compiler optimizations, see our [Optimization Notice](#).

## [Optimization Notice](#)

Copyright © 2018, Intel Corporation. All rights reserved.

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



# Security Barrier Camera Demo

```
$ sudo ./demo_security_barrier_camera.sh
```

- Builds the Inference Engine security barrier camera sample (inference\_engine\samples\security\_barrier\_camera\_sample)
- Runs the sample with the security\_barrier\_input\_example.jpg picture located in the demo folder.

# Security Barrier Camera Demo


## Targeting vehicle detection and classification workloads to CPU only


```
$ /opt/intel/computer_vision_sdk_fpga/deployment_tools/inference_
engine/samples/build/intel64/Release/security_barrier_camera_sample
-d CPU
-i /opt/intel/computer_vision_sdk_fpga/deployment_tools/demo/
  test_video.mp4
-m /opt/intel/computer_vision_sdk_fpga/deployment_tools/demo/
  intel_models/vehicle-license-plate-detection-barrier-0007/FP32/vehicle-
  license-plate-detection-barrier-0007.xml
-m_va opt/intel/computer_vision_sdk_fpga/deployment_tools/demo/
  intel_models/vehicle-attributes-recognition-barrier-0010/FP32/
  vehicle-attributes-recognition-barrier-0010.xml
```


\$ █


# Install the OpenVINO™


Follow the instructions to install Open Visual Inference & Neural network Optimization (OpenVINO™)


**Developer  
Zone**

Search our content library... 

 Support

 Sign in


 English

**OPENVINO™ TOOLKIT**  
Choose & Download

Choose & Download

Share

## CHOOSE THE BEST OPTION



**OpenVINO® Toolkit (Includes the Intel® Deep Learning Deployment Toolkit)**

Select the operating system you intend to use for development or deployment.

[Windows\\*](#)

[Linux\\*](#) (supports Ubuntu\*, CentOS\*, and Yocto Project\*)

[Linux for FPGA Beta](#) (an Intel® Arria® 10 FPGA GX development kit or Intel® Programmable Acceleration Card with Intel® Arria® 10 FPGA GX is required for deep learning acceleration)

For more complete information about compiler optimizations, see our [Optimization Notice](#).

## Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.

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



# Setup the FPGA Card

*You have three options, more are coming up...*

Intel® Arria® 10 GX FPGA Development Kit



PCIe Gen3 x 8, full size card  
(We are using this card in this session)



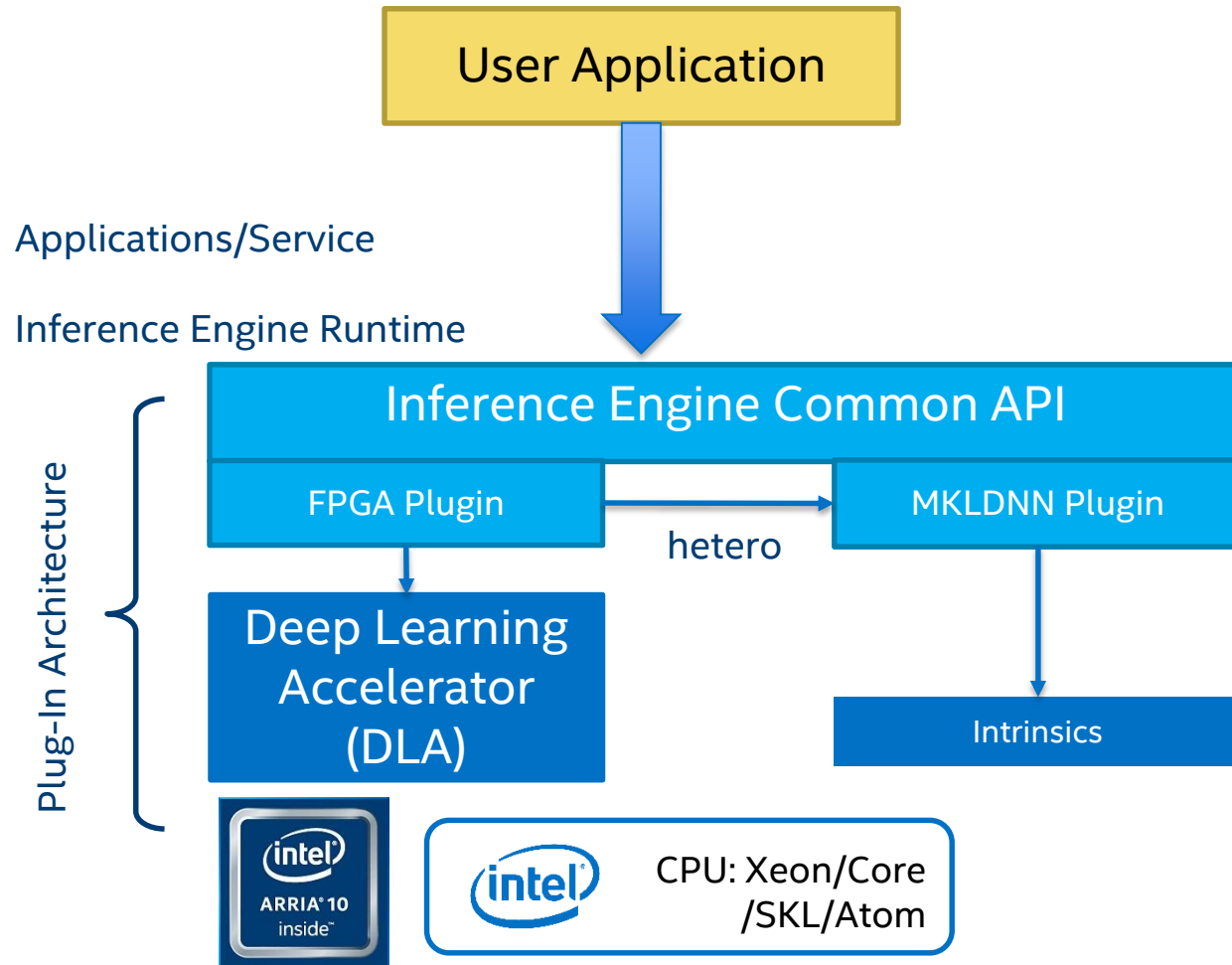
Intel® Programmable Acceleration Card with  
Intel Arria® 10 GX FPGA  
PCIe Gen3 x 8, Half-height half-length card



Deep Learning Acceleration Card with  
Intel Arria® 10 GX FPGA  
PCIe Gen3 x 8, Half-height half-length card



# Software stack



FPGA implementation is “Deep Learning Accelerator” (DLA)

Additional step: FPGA RTE (aocl) loads bitstream with desired version of DLA to FPGA before running

# Using Plugins, Depending on the Target

Target	Linux Library Name	Linux Dependency Libraries	Windows Library Name	Windows Dependency Libraries
CPU	libMKLDNNPlugin.so	libmklml_tiny.so, libiomp5md.so	MKLDNNPlugin.dll	mklml_tiny.dll, libiomp5md.dll
Intel® Integrated Graphics	libclDNNPlugin.so	libclDNN64.so	clDNNPlugin.dll	clDNN64.dll
<b>FPGA</b>	<b>libdliaPlugin.so</b>	<b>libdla.so</b>	<b>Not supported</b>	<b>Not supported</b>
Intel® Movidius™ Myriad™ 2 Vision Processing Unit (VPU)	libmyriadPlugin.so	No dependencies	Not supported	Not supported
<b>Heterogeneous</b>	<b>libHeteroPlugin.so</b>	<b>Same as selected plugins</b>	<b>HeteroPlugin.dll</b>	<b>Same as selected plugins</b>

## Supported layers

The following layers are supported by the plugin:

- Batch\_norm (being converted by Model Optimizer to ScaleShift layer)
- Concat
- Convolution (dilated convolutions are supported, depthwise are not supported)
- Eltwise (operation sum is supported)
- Fully Connected
- LRN Normalization
- Pooling
- Power (scale and offset parameters are supported)
- ReLu (with negative slope)
- ScaleShift

**Note** Support is limited to the specific parameters (depending on the bitstream).

## Heterogeneous execution

In case when topology contains layers not supported on FPGA, you need to use [Heterogeneous plugin](#) with dedicated fallback device.

# Adding Your Own Kernels in the Inference Engine

1. Implement your NN primitive (kernel). See Extension library in Samples directory, that comes with few real example of CPU-targeted kernels, like DetectionOutput (used in SSD\*), etc.
2. Plug your kernel implementations into the Inference Engine and map them to the layers in the original framework. See the [Model Optimizer Developer Guide](#) for information about how a mapping between framework's layers and Inference Engine kernels is registered.

# Bitstream - topology

Network	Bitstreams (Intel® Arria® 10 GX DevKit)	Bitstreams (Programmable Acceleration Card with Intel® Arria® 10 GX FPGA)
AlexNet	arch4, arch6, arch7, arch8, arch16	arch2, arch3, arch10, arch11, arch12, arch16, arch17, arch23, arch25, arch26
GoogleNet v1	arch6, arch7, arch8, arch16	arch2, arch3, arch11, arch12, arch16, arch17, arch23, arch25, arch26
VGG-16	arch6, arch16	arch2, arch3, arch16
VGG-19	arch6, arch16	arch2, arch3, arch16
SqueezeNet v 1.0	arch6, arch7, arch8, arch14	arch2, arch3, arch9, arch11, arch12, arch16, arch17, arch18, arch20, arch23, arch25, arch26
SqueezeNet v 1.1	arch6, arch7, arch8, arch14	arch2, arch3, arch9, arch11, arch12, arch16, arch17, arch18, arch20, arch23, arch25, arch26

# Bitstream - topology

Network	Bitstreams (Intel® Arria® 10 GX DevKit)	Bitstreams (Programmable Acceleration Card with Intel® Arria® 10 GX FPGA)
ResNet-18	arch8, arch9, arch19	arch2, arch3, arch9, arch16, arch20
ResNet-50	arch9, arch19	arch3, arch9, arch20
ResNet-101	arch9, arch19	arch3, arch9, arch20
ResNet-18	arch8, arch9, arch19	arch2, arch3, arch9, arch16, arch20
ResNet-152	arch9, arch19	N/A
SqueezeNet-based variant of the SSD*	arch9	N/A
GoogLeNet-based variant of SSD	arch6, arch7, arch8, arch9	N/A
VGG-based variant of SSD	arch6	N/A

# Setup the Environment

Before start deploying your workloads to FPGA, setup the environment...

```
$ source /opt/intel/computer_vision_sdk_fpga/bin/setupvars.sh
$ export LD_LIBRARY_PATH=/opt/intel/computer_vision_sdk_fpga/deployment_
  tools/inference_engine/lib/ubuntu_16.04/intel64:$LD_LIBRARY_PATH
$ export LD_LIBRARY_PATH=/opt/altera/aocl-pro-rte/aclrte-linux64/host/
  linux64/lib:$LD_LIBRARY_PATH
$ █
```

# Program the Bitstream to the FPGA

Setup bitstream variables, and load the bitstream to FPGA...

```
$ export CL_CONTEXT_COMPILER_MODE_INTELFPGA=3
$ export DLA_AOCX=/opt/intel/computer_vision_sdk_fpga/a10_devkit_
  bitstreams/0-8-1_a10dk_fp16_8x48_arch06.aocx
$ source /opt/altera/aocl-pro-rte/aclrte-linux64/init_openc1.sh
$ aocl diagnose
$ aocl program acl0 $DLA_AOCX
Programming device: a10gx : Arria 10 Reference Platform (acla10_ref0)
Reprogramming device [0] with handle 1
Program succeed.
$ █
```



# Inference Engine, few FPGA tips

- First iteration with FPGA is always significantly slower than the rest, make sure you run multiple iterations (“-ni”)
- If CPU utilization of concern, minimizes the busy wait time when OMP threads loop in between parallel regions:
  - set KMP\_BLOCKTIME=1
  - consider playing OMP\_NUM\_THREADS
- FPGA performance heavily depends on the bitstream

# Run the Demo in the Heterogeneous Mode

Setup the environment properly for heterogeneous mode  
[CPU: Intel Core i7]

```
$ export KMP_BLOCKTIME=1
$ export OMP_NUM_THREADS=4
$ █
```

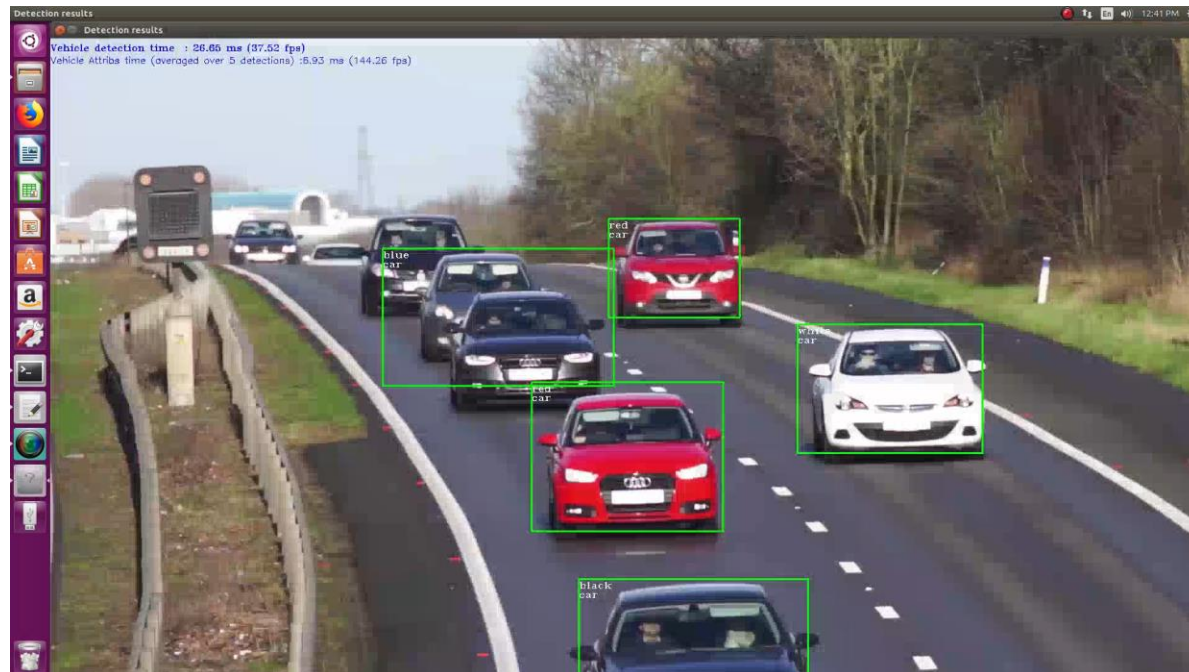
# Run the Demo on FPGA!

## Targeting vehicle detection and classification workloads to FPGA and CPU

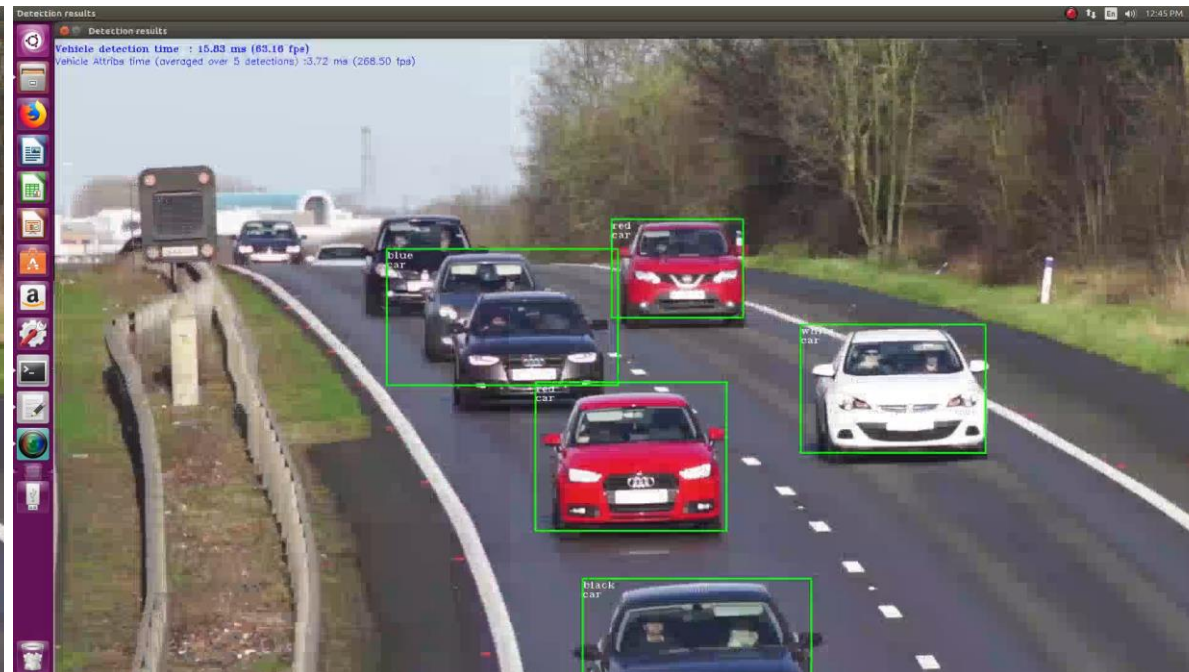
```
$ /opt/intel/computer_vision_sdk_fpga_2018.0.234/deployment_tools/inference_
engine/samples/build/intel64/Release/security_barrier_camera_sample
-d HETERO:FPGA,CPU
-i /opt/intel/computer_vision_sdk_fpga_2018.0.234/deployment_tools/demo/
  test_video.mp4
-m /opt/intel/computer_vision_sdk_fpga_2018.0.234/deployment_tools/demo/
  intel_models/vehicle-license-plate-detection-barrier-0007/FP32/vehicle-
  license-plate-detection-barrier-0007.xml
-m_va opt/intel/computer_vision_sdk_fpga_2018.0.234/deployment_tools/demo/
  intel_models/vehicle-attributes-recognition-barrier-0010/FP32/
  vehicle-attributes-recognition-barrier-0010.xml
```

\$ █

# Contrast



CPU only without OpenMP tweak



Heterogeneous mode (FPGA + CPU) with OpenMP tweak