



# Object Detection Using OpenVINO™ Toolkit

May 2018

Core and Visual Computing Group

# 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. 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

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](http://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/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.

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.

Copyright © 2018, Intel Corporation. All rights reserved.

# 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](http://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](http://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.

Copyright © 2018, Intel Corporation. All rights reserved.

# Smart Video Workshop Overview

## Introduction

1. Introduction to Intel technologies for deep learning inference
2. Hardware acceleration techniques

Each module contains a hands-on lab exercise that introduces various Intel technologies to accelerate computer vision application with hardware heterogeneity.

## OpenVINO™ 101

### Hardware Acceleration

### Optimization

### Application

2. Basic End-to-End Object Detection Example

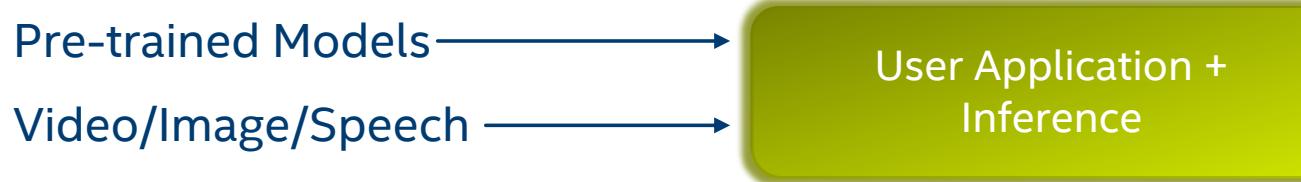
- 3./4./5. Hardware Acceleration with CPU, Integrated GPU, Intel® Movidius™ NCS, FPGA

6. Optimization Tools and Techniques

7. Advanced Video Analytics

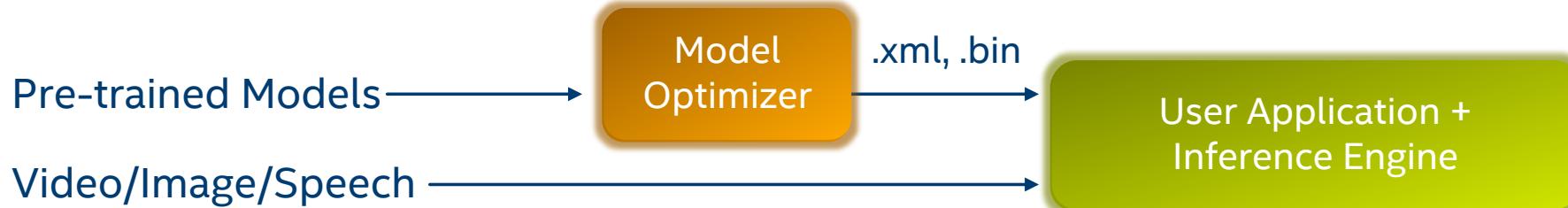
# Deep Learning Application Deployment

## Traditional



## With OpenVINO™ Toolkit

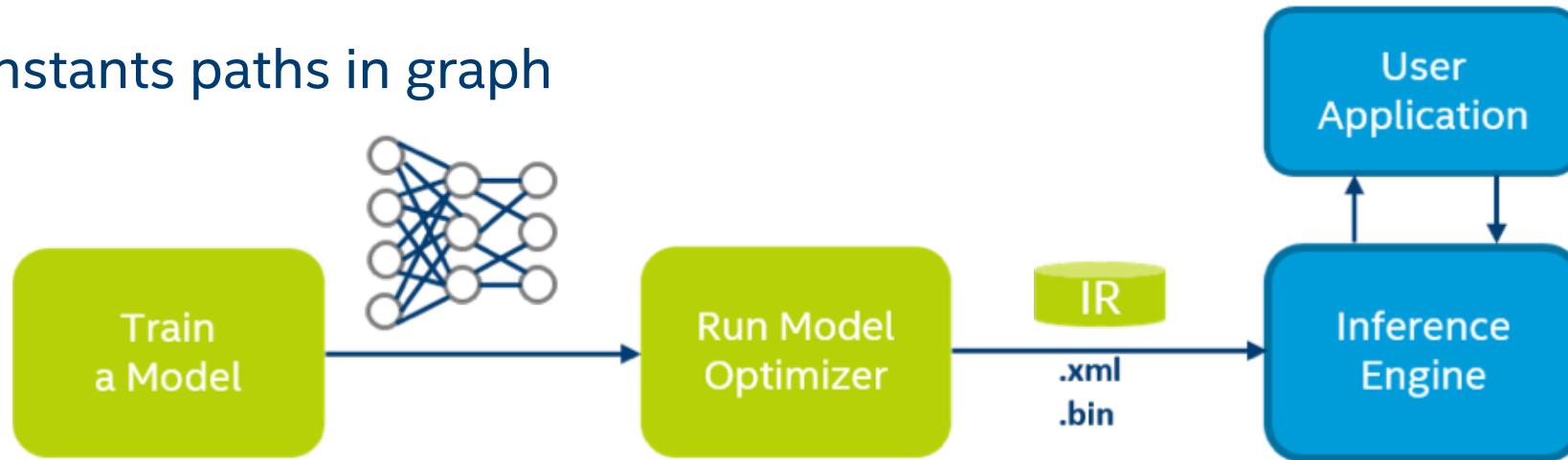
### One-Time Process



# Model Optimizer

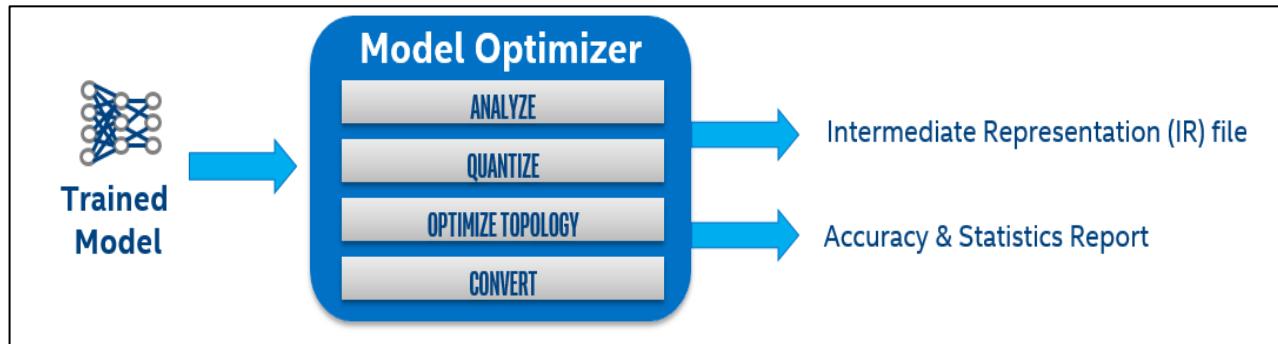
# Model Optimizer

- 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

# Intel® DL Deployment Toolkit Functionality

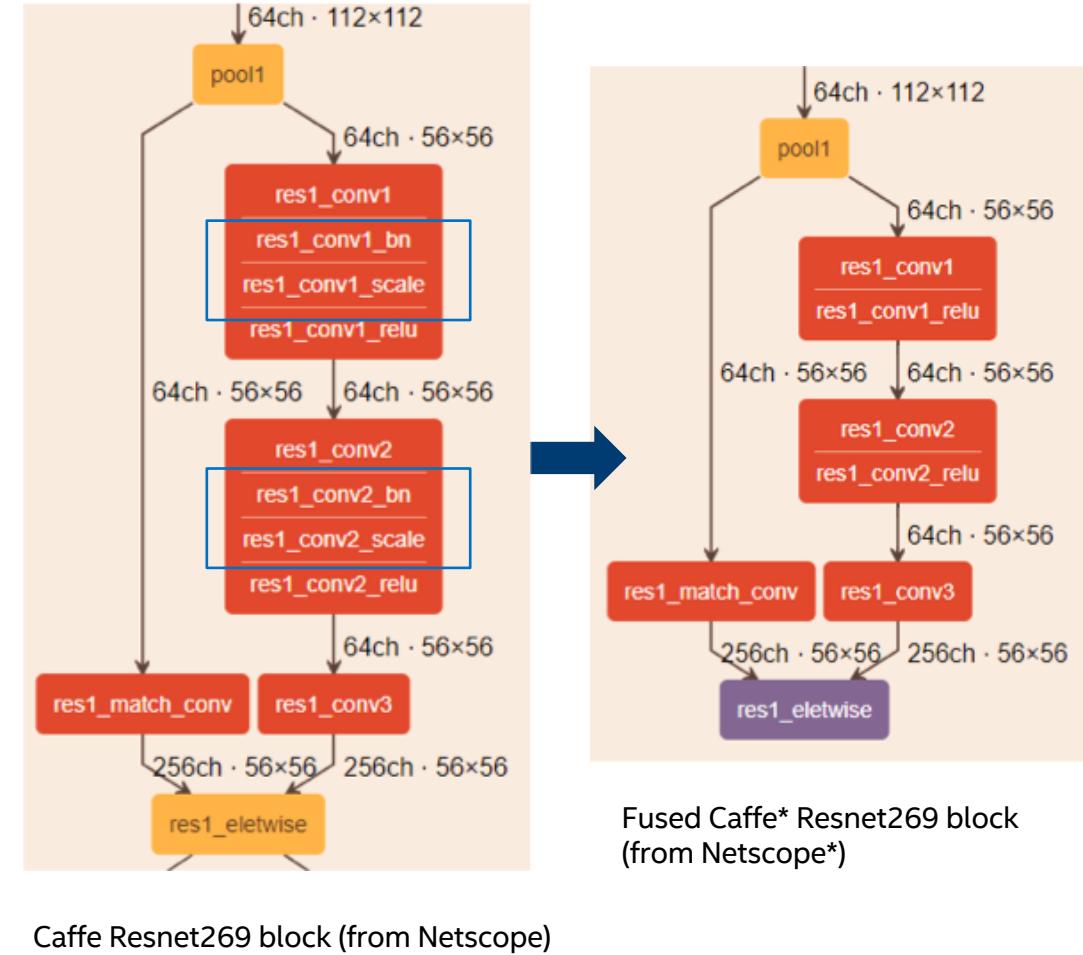
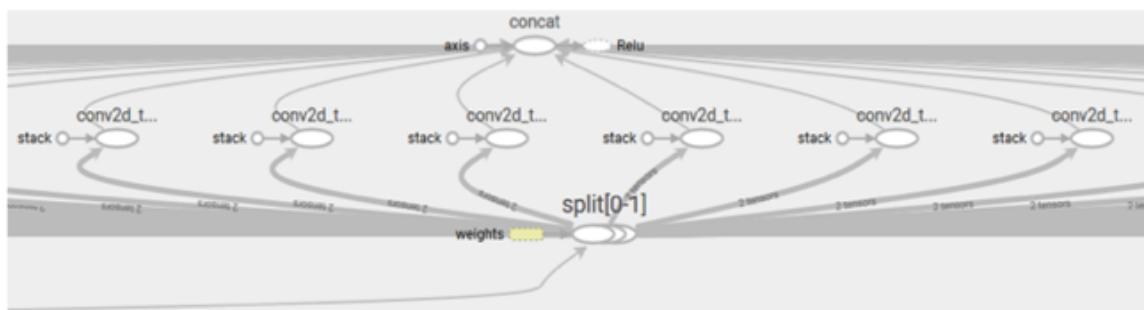
## Model Optimizer/Optimization Tech

### Linear Operation Fusing

- **BatchNorm and ScaleShift decomposition:** BN layers decomposes to *Mul->Add->Mul->Add* sequence; ScaleShift layers decomposes to *Mul->Add* sequence.
- **Linear operations merge:** Merges sequences of Mul and Add operations to the **single** *Mul->Add* instance.
- **Linear operations fusion:** Fuses Mul and Add operations to Convolution or FullyConnected layers.

### Grouped Convolutions Fusing

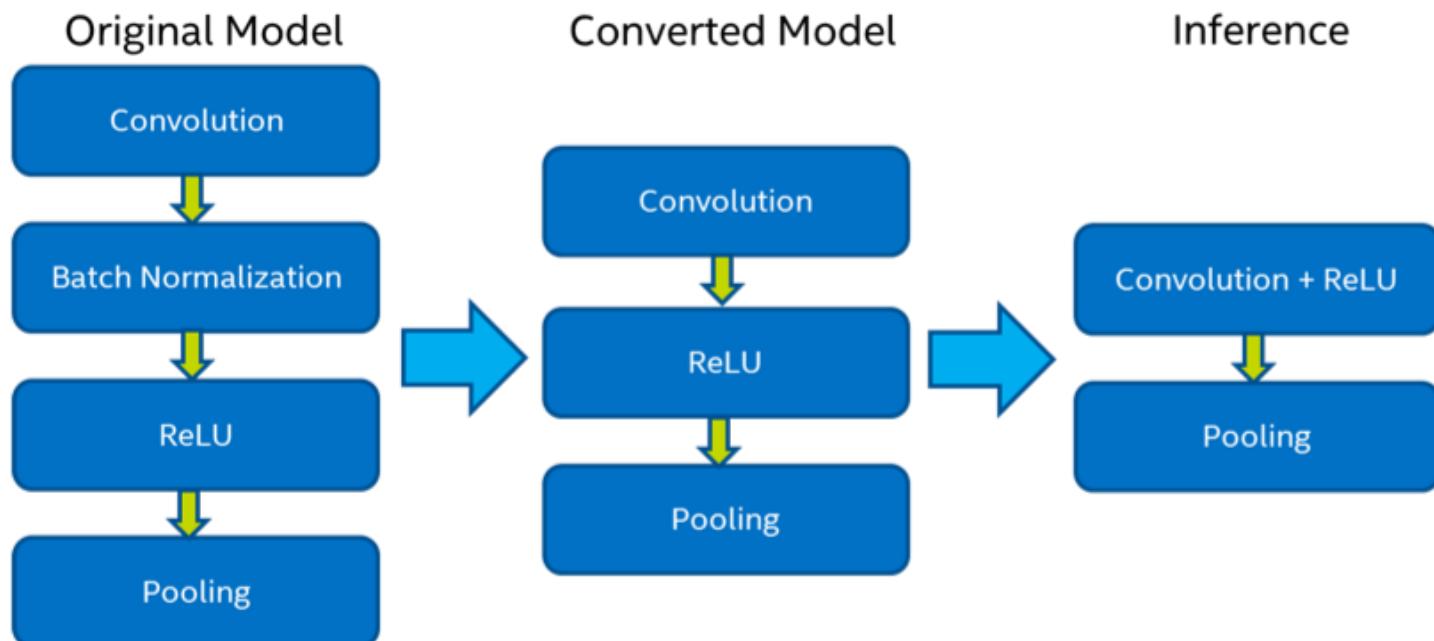
Specific optimization that applies for TensorFlow\* topologies. (Xception\*)



# 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.

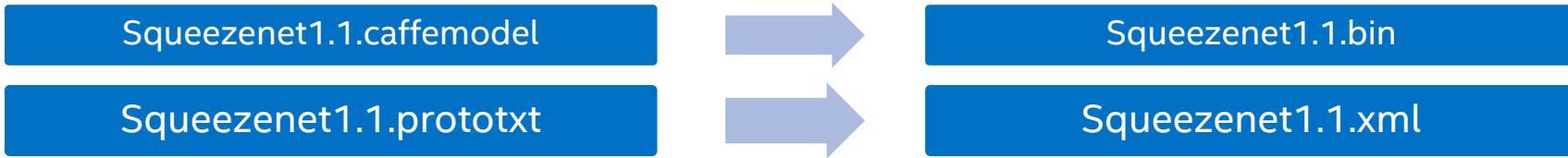
--mean\_values (104.006, 116.66, 122.67)

--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)



```
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" pad="0" type="NCHW"/>
<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>
..
```

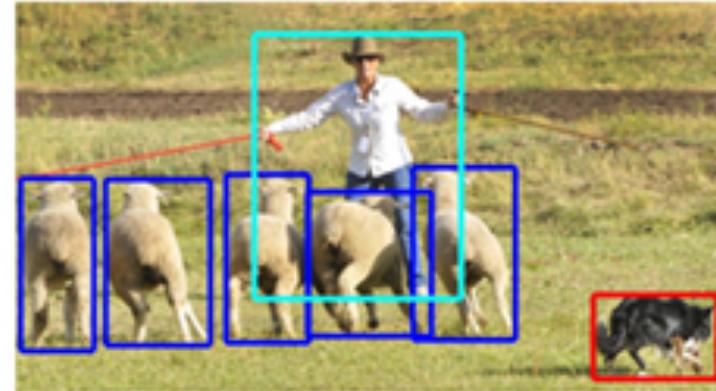
# Inference Engine

# Inference on an Intel® Edge Systems

Many deep learning networks are available—choose the one you need.



(a) classification



(b) detection

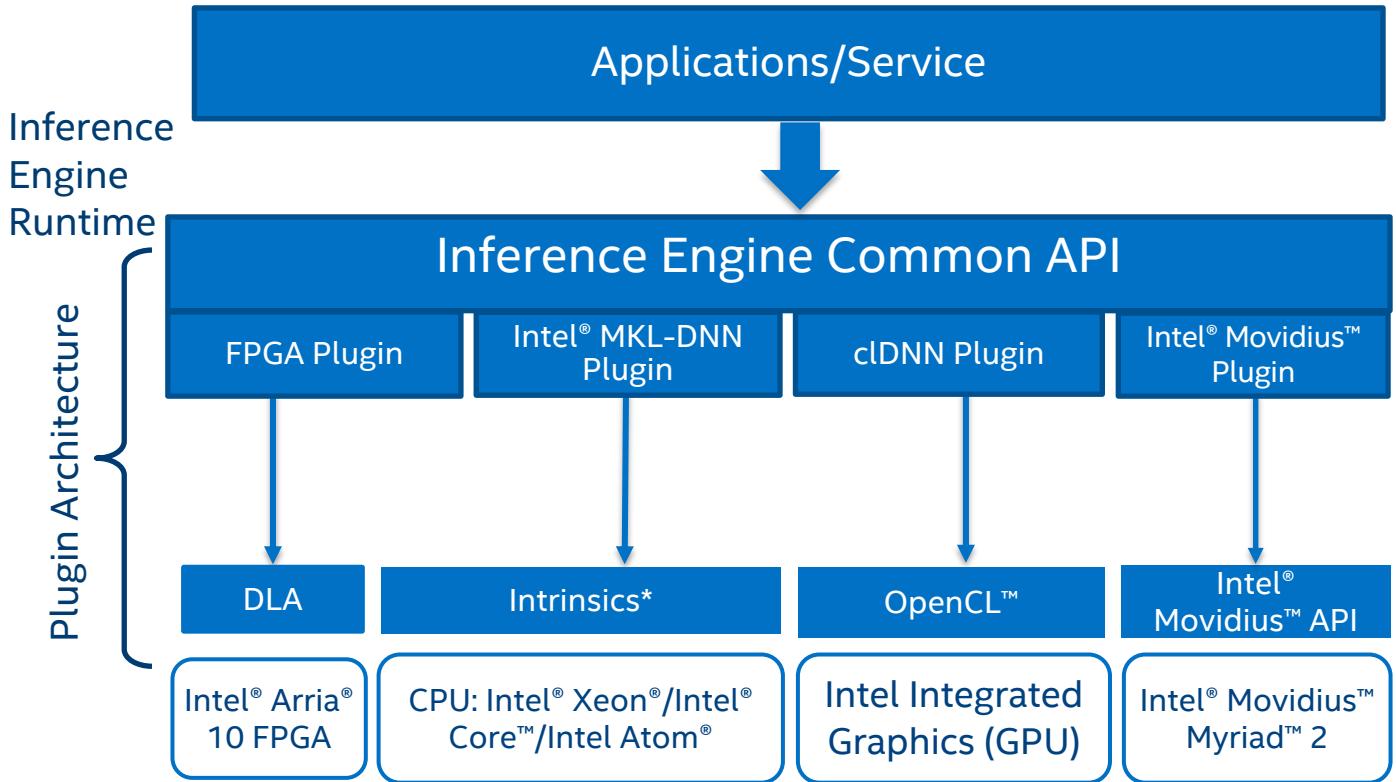


(c) segmentation

The complexity of the problem (data set) dictates the network structure. The more complex the problem, the more 'features' required, the deeper the network.

# 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

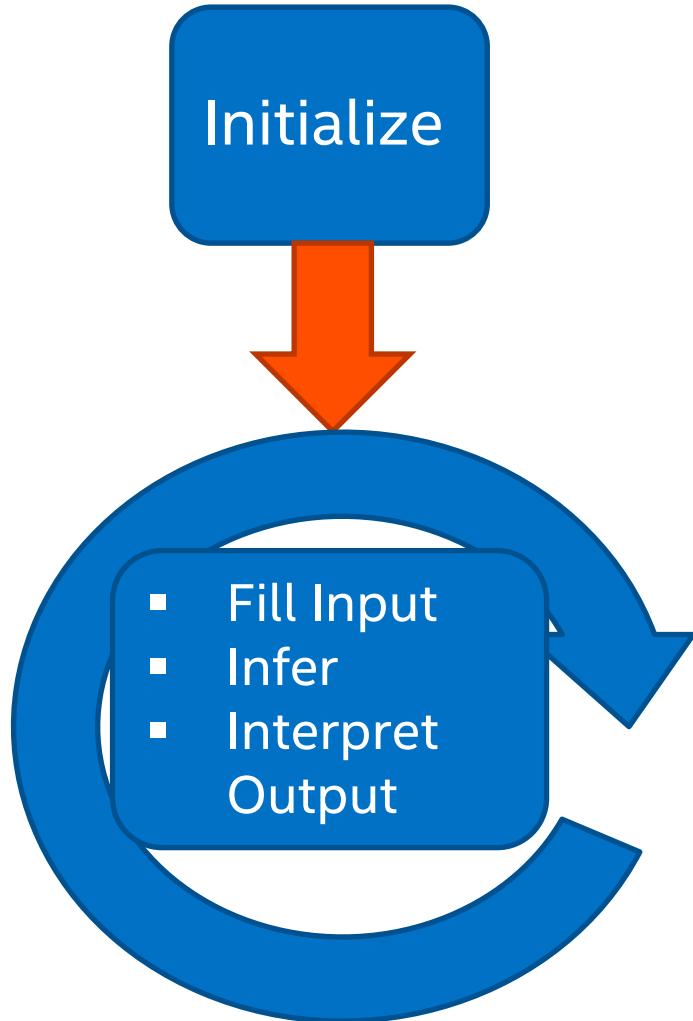
- Supports Intel® Arria® 10
- FP16 data types, FP11 is coming

## 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 Normalization	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 Obj	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

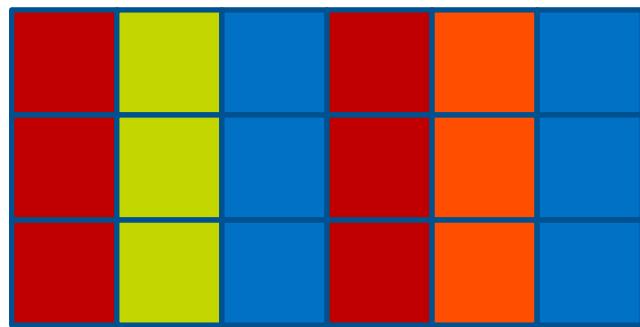
```
//-----  
// Run inference  
//-----  
sts = plugin->Infer(inputBlobs, outputBlobs, &dsc);
```

# Pre-processing

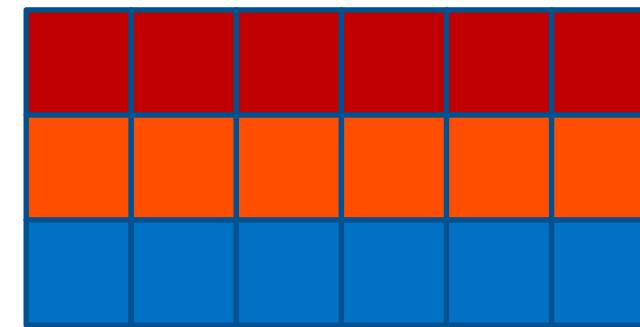
Most of the image formats are interleaved formats (RGB, BGR, BGRA, and so forth). Inference engine expects input to be in RGB planar format, such as:

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.

CPU/GPU/FPGA/Intel® Movidius™ Neural Compute Stick

# Hardware Heterogeneity

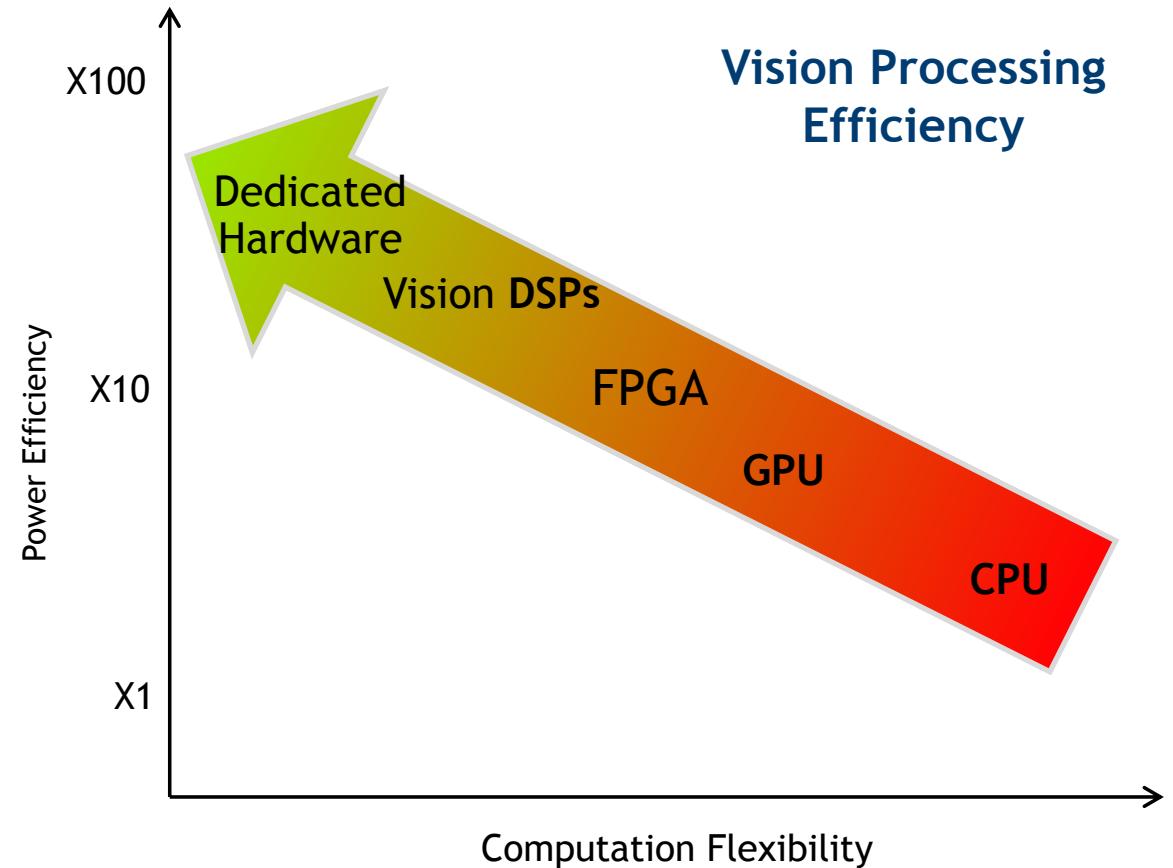
# Choosing the “Right” Hardware

## Power/Performance Efficiency Varies

- Running the right workload on the right piece of hardware → higher efficiency
- Hardware acceleration is a must
- Heterogeneous computing?

## Tradeoffs

- Power/performance
- Price
- Software flexibility, portability



# Heterogeneous Dichotomies

## Data vs. Task Parallelism

- Data parallelism between devices is usually explicit and extremely error-prone (like in OpenCL™).
- Inference engine heterogeneity support is inherently **task-oriented** (node-level).

## Dynamic vs. Static

- **User-defined work split** vs. adaptive schemes (like load-balancing).
- For both, there is a question on the granularity (communications costs, next).

***Heterogeneity is basically Fallback.***

# Apply Device Affinities to Layers Automatically Using Fallback Policy (1 of 2)

```
$ object_detection_sample_ssd -d HETERO:CPU,GPU -l lib/libicv_extension.so -m ssd.xml -i snake.bmp
```

All IE samples support loading CPU and GPU extensions as usual (“-l” and “-c”). The “priorities” defines a greedy behavior:

- Keeps all layers that can be executed on the device (FPGA)
- Carefully respects topological and other limitations
- Follows priorities when searching (for example, CPU)

# Apply Device Affinities to Layers Automatically Using Fallback Policy (2 of 2)

```
HeteroPluginPtr plugin(make_plugin_name("HeteroPlugin"));

CNNNetReader reader;

reader.ReadNetwork("Model.xml");

reader.ReadWeights("Model.bin");

CNNNetwork network = reader.getNetwork();

plugin->SetConfig({ { "TARGET_FALLBACK", "FPGA,CPU" } }, &response);

plugin->LoadNetwork(exeNetwork, network, {}, &response);
```

# Intel® Movidius™ Neural Compute Stick

# Intel® Movidius™ Neural Compute Stick

## Redefining the AI Developer Kit

- Neural network accelerator in USB stick form factor
- Tensorflow\* and Caffe\* and many popular frames are supported
- Prototype, tune, validate, and deploy deep neural networks at the edge
- Features the same Intel® Movidius™ Vision Processing Unit (VPU) used in drones, surveillance cameras, virtual reality headsets, and other low-power intelligent and autonomous products



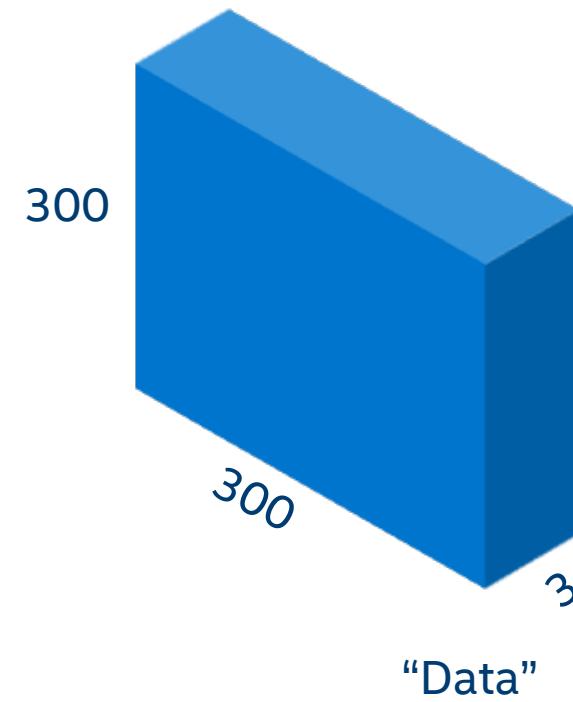
Car Detection

# Hands-On Example

# SSD\_GoogleNetV2\_caffe/SSD\_GoogleNetV2\_Deploy.prototxt

```
name: "SSD_300x300"
input: "data"
input_shape {
    dim: 1
    dim: 3
    dim: 300
    dim: 300
}
layer {
    bottom: "data"
    top: "conv1/7x7_s2"
    name: "conv1/7x7_s2"
    type: "Convolution"
    convolution_param { num_output: 64 pad: 3 }
}
layer {
    bottom: "conv1/7x7_s2"
    name: "conv1/7x7_s2/bn"
    top: "conv1/7x7_s2/bn"
    type: "BatchNorm"
}
```

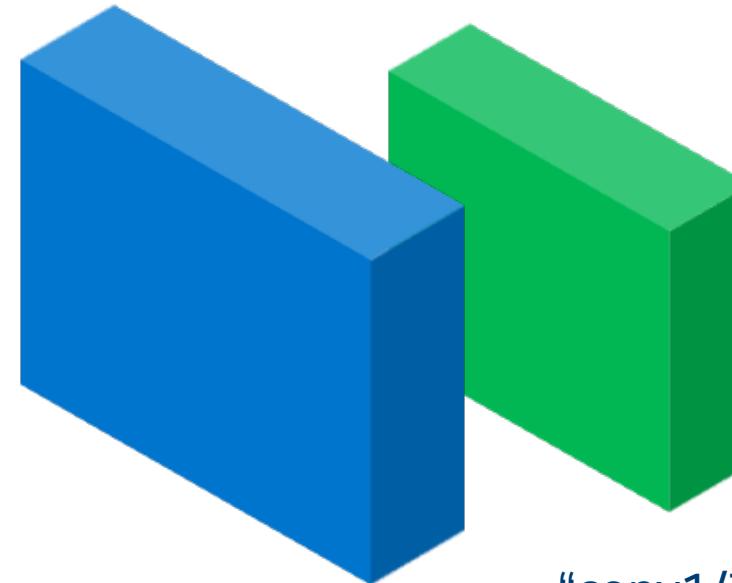
Define input "data" dimensions  
300 x 300, RGB



# SSD\_GoogleNetV2\_caffe/SSD\_GoogleNetV2\_Deploy.prototxt

```
name: "SSD_300x300"
input: "data"
input_shape {
    dim: 1
    dim: 3
    dim: 300
    dim: 300
}
layer {
    bottom: "data"
    top: "conv1/7x7_s2"
    name: "conv1/7x7_s2"
    type: "Convolution"
    convolution_param { num_output: 64 pad: 3 }
}
layer {
    bottom: "conv1/7x7_s2"
    name: "conv1/7x7_s2/bn"
    top: "conv1/7x7_s2/bn"
    type: "BatchNorm"
}
```

First Layer.. A convolution



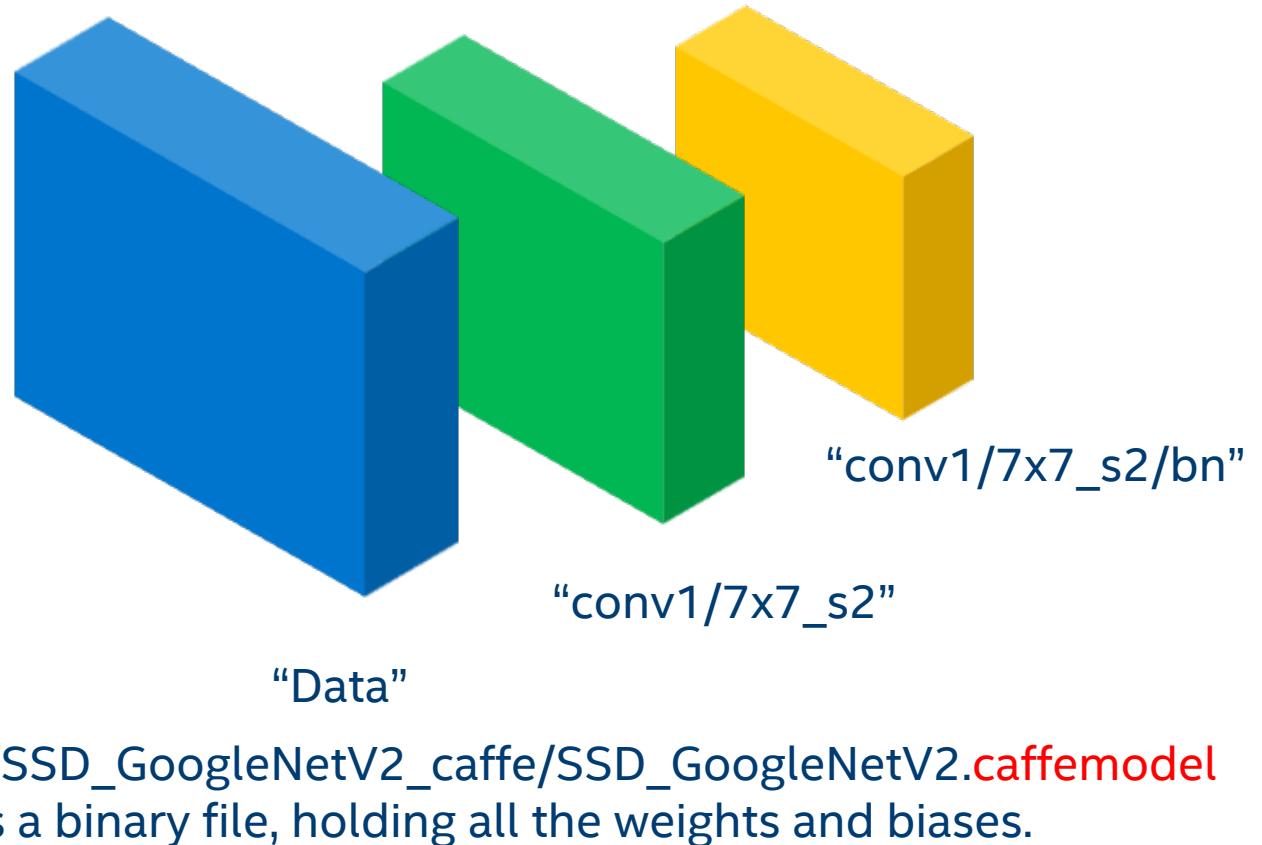
“conv1/7x7\_s2”

“Data”

# SSD\_GoogleNetV2\_caffe/SSD\_GoogleNetV2\_Deploy.prototxt

```
name: "SSD_300x300"
input: "data"
input_shape {
    dim: 1
    dim: 3
    dim: 300
    dim: 300
}
layer {
    bottom: "data"
    top: "conv1/7x7_s2"
    name: "conv1/7x7_s2"
    type: "Convolution"
    convolution_param { num_output: 64 pad: 3 }
}
layer {
    bottom: "conv1/7x7_s2"
    name: "conv1/7x7_s2/bn"
    top: "conv1/7x7_s2/bn"
    type: "BatchNorm"
}
```

Second Layer..  
Another convolution



# pascal\_voc\_classes.txt

plane  
bicycle  
bird  
boat  
bottle  
bus  
car  
cat  
chair  
cow  
table  
dog  
horse  
motorcycle  
person  
plant  
sheep



# Offline, One-Time Run of the Model Optimizer

- Integrate inference engine (IE) API into the application.
- Intel® MKL-DNN plugin for CPU, CI-DNN for GPU.
- Target device could be CPU/GPU/Intel® Movidius™ NCS/FPGA.

