INTEL® OPTIMIZED AI FRAMEWORKS

Dr. Fabio Baruffa & Shailen Sobhee
Technical Consulting Engineers, Intel IAGS
SPEED UP DEVELOPMENT

using open AI software

TOOLKITS

App developers

Open source platform for building E2E Analytics & AI applications on Apache Spark* with distributed TensorFlow®, Keras®, BigDL

Deep learning inference deployment on CPU/GPU/FPGA/VPU for Caffe®, TensorFlow®, MXNet®, ONNX®, Kaldi®

Open source, scalable, and extensible distributed deep learning platform built on Kubernetes (BETA)

LIBRARIES

Data scientists

Python
- Scikit-learn
- Pandas
- NumPy

R
- Cart
- Random Forest
e1071

Distributed
- MLlib (on Spark)
- Mahout

Intel-optimized Frameworks

And more framework optimizations underway including PaddlePaddle®, Chainer®, CNTK® & others

KERNELS

Library developers

Intel® Distribution for Python®
Intel distribution optimized for machine learning

Intel® Data Analytics Acceleration Library (Intel® DAAL)
High performance machine learning & data analytics library

Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN)
Open source DNN functions for CPU / integrated graphics

Open source compiler for deep learning model computations optimized for multiple devices (CPU, GPU, NNP) from multiple frameworks (TF, MXNet, ONNX)

Visit: www.intel.ai/technology
SPEED UP DEVELOPMENT
using open AI software

Visit: www.intel.ai/technology
INTRODUCTION TO NEURAL NETWORK
Linear classifier can solve the **AND** problem.

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

We need to train the network to compute the ‘unknown’ weights and threshold.
Linear classifier can solve the **AND** problem.

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
x_1 \times 1 + x_2 \times 1 = z
\]

if \( z > 1.5 \) Output = 1
Linear classifier can solve the **OR** problem.

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$X_1 \times 1 + X_2 \times 1 = Z$

if $(Z > 0.5)$ Output = 1
A single linear classifier cannot solve the XOR problem.

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

XOR
The counter example to all models

We need two straight line for separation
**XOR = (X1 and not X2) OR (Not X1 and X2)**

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Threshold to 0 or 1

\[
\text{XOR} = (X1 \text{ and not } X2) \text{ OR (Not } X1 \text{ and } X2)
\]
**MULTILAYER PERCEPTRON**

**XOR = (X1 and not X2) OR (Not X1 and X2)**

<table>
<thead>
<tr>
<th></th>
<th>X1</th>
<th>X2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Threshold to 0 or 1**

\[
(1 \times 1) + (1 \times -2) + (1 \times 1) = 0 < .5 = 0
\]

\[
(1 \times 1) + (1 \times 1) = 2 > 1.5
\]
MOTIVATION FOR NEURAL NETS

- Use biology as inspiration for mathematical model
- Get signals from previous neurons
- Generate signals (or not) according to inputs
- Pass signals on to next neurons
- By layering many neurons, can create complex model

\[
\begin{align*}
\text{Input} & \rightarrow 1.5 \rightarrow 0.5 \rightarrow \text{Output} \\
+1 & \rightarrow \rightarrow -2 \\
\end{align*}
\]
Deep Learning Neural Network

The Neuron

Activation Function

ReLU (rectified linear unit)

\[ F(x) = \max(0, x) \]

\( F \): a non-linear differentiable function

Input

Hidden

Output

https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Print_Version
DEEP LEARNING NEURAL NETWORK

Simple Neural Network

Deep Learning Neural Network

Input Layer

Hidden Layer

Output Layer
• Training:
  • Use neural network techniques to gather common information among one category objects. ➔ Model

• Inference:
  • Use the gathered common information (model) for prediction or/and generation.
TRAINING TECHNIQUE

• Techniques used for training and inference
  • Forward propagation
    • Go through the network in forward direction
    • Yield result of the model
  • Backward propagation
    • Go through the network in backward direction
    • Check how different results of the model is against correct answers
    • Update model parameters based on the differences
Training and Inference

Training:
- Input
  - Forward Propagation
  - Deep Learning Neural Network
  - Result
  - Difference (Loss)
  - Answer

Inference:
- Input
  - Forward Propagation
  - Deep Learning Neural Network
  - Result
NEURAL NETWORK PLAYGROUND

DATA
Which dataset do you want to use?

FEATURES
Which properties do you want to feed in?

OUTPUT
Test loss 0.498
Training loss 0.506

https://playground.tensorflow.org
Deep Neural Network (DNN)

<table>
<thead>
<tr>
<th>Convolution Neural Network (CNN)</th>
<th>Recurrent Neural Network (RNN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Recognition, Outline of object in an image, Video processing</td>
<td>Image captioning, Text generation, Language translation</td>
</tr>
<tr>
<td>Hierarchical</td>
<td>Sequential</td>
</tr>
<tr>
<td>extract position invariant features</td>
<td>model units in sequence</td>
</tr>
<tr>
<td>Learns to recognize patterns across space</td>
<td>Learns to recognize patterns across time</td>
</tr>
<tr>
<td>The lines and curves I saw will help me recognize the faces and objects</td>
<td>What I spoke last will impact what I will speak next</td>
</tr>
</tbody>
</table>

The number of neurons in a layer (hidden size) and batch size can make DNN performance vary dramatically. This suggests that optimization of these two parameters is crucial to good performance of both CNNs and RNNs (1).

### Types of RNN

<table>
<thead>
<tr>
<th>Long Short-Term Memory (LSTM)</th>
<th>Gated Recurrent Unit (GRU)</th>
</tr>
</thead>
</table>

In empirical evaluations, Chung et al. (2014) and Jozefowicz et al. (2015) found there is no clear winner between GRU and LSTM. In many tasks, they yield comparable performance and tuning hyperparameters like layer size is often more important than picking the ideal architecture (1).


Image Source: https://www.slideshare.net/sheemap/convolutional-neural-networks
DEEP LEARNING OPERATIONS

- Fully Connected
- Convolution 2D
- Convolution 3D
- Batch Normalization
- ReLU
- Dropout
- RNN
- ...

Figure 2. **Normalization methods.** Each subplot shows a feature map tensor, with $N$ as the batch axis, $C$ as the channel axis, and $(H, W)$ as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.
Why do we need deep learning frameworks?

Questions:
1. How long will it take to write code to implement those processes?
2. Are you willing to write code to implement those processes every time when you would like to develop a new topology?

Deep learning frameworks implemented these complicated operations for you:
1. Easy to use
2. High efficiency for development of topologies
3. Even if you don’t understand mathematic theories underneath
INTEL® AI OPTIMIZED FRAMEWORKS

Popular DL Frameworks are now optimized for CPU!

See installation guides at ai.intel.com/framework-optimizations/

More under optimization:

SEE ALSO: Machine Learning Libraries for Python (Scikit-learn, Pandas, NumPy), R (Caret, randomForest, e1071), Distributed (MILib on Spark, Mahout)

*Limited availability today
*Other names and brands may be claimed as the property of others.
Deep learning and AI ecosystem includes edge and datacenter applications.
- Open source frameworks (Tensorflow*, MXNet*, PyTorch*, PaddlePaddle*)
- Intel deep learning products (BigDL, OpenVINO™ toolkit)
- In-house user applications

Intel® MKL and Intel® MKL-DNN optimize deep learning and machine learning applications for Intel® processors:
- Through the collaboration with framework maintainers to upstream changes (Tensorflow*, MXNet*, PyTorch, PaddlePaddle*)
- Through Intel-optimized forks (Caffe*)
- By partnering to enable proprietary solutions

Intel® MKL-DNN is an open source performance library for deep learning applications (available at https://github.com/intel/mkl-dnn)
- Fast open source implementations for wide range of DNN functions
- Early access to new and experimental functionality
- Open for community contributions

Intel® MKL is a proprietary performance library for wide range of math and science applications
Distribution: Intel Registration Center, package repositories (apt, yum, conda, pip), Intel® Parallel Studio XE, Intel® System Studio
INTRODUCTION TO TENSORFLOW*
WHAT IS TENSORFLOW?

• Framework for math using Computation Graphs
• Has features specifically for machine learning
• Primary interface is Python, integrates NumPy
• Designed to be flexible, scalable and deployable

• Easy to install including the Intel® optimization via using conda
  • conda install tensorflow -c intel

COMPUTATION GRAPH

Nodes represent computations
COMPUTATION GRAPH

Edges represent numerical data flowing through the graph.
DATA FLOW

input

mul

add

input

add

7

3

7

7

3

3

21

10

31
TWO-STEP PROGRAMMING PATTERN

1. Define a computation graph

2. Run the graph

- data
- Predictions
Define a computational graph

```python
>>> a = tf.placeholder(tf.float32, name="input1")
>>> b = tf.placeholder(tf.float32, name="input2")
>>> c = tf.add(a, b, name="my_add_op")
>>> d = tf.multiply(a, c, name="my_mul_op")
```
We use a `Session` object to execute graphs. Each `Session` is dedicated to a single graph.

```python
>>> sess = tf.Session()
```
SETUP THE CONFIGURATION (OPTIONAL)

```python
>>> config = tf.ConfigProto(inter_op_parallelism_threads=2,
                           intra_op_parallelism_threads=44)

>>> tf.Session(config=config)
```

Session: 

Graph: `default`

Variable values:
**INITIALIZE THE VALUES**

*placeholders* require data to fill them in when the graph is run.

We do this by creating a dictionary mapping *Tensor* keys to numeric values.

```python
>>> feed_dict = {a: 3.0, b: 2.0}
```

```
Session  sess

Graph: default

Variable values:

feed_dict: {a: 3.0, b: 2.0}
```

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.*
We execute the graph with `sess.run(fetches, feed_dict)`.

`sess.run` returns the fetched values as a NumPy array.

```python
>>> out = sess.run(d, feed_dict=feed_dict)
```

**Session**

- **Graph:** `default`
- **Variable values:**
  - `run()`
  - `fetches: d`
  - `feed_dict: feed_dict`

**Graph:**

- **Nodes:**
  - `my_add_op`
    - Values: `a`, `c`
  - `my_mul_op`
    - Values: `b`, `d`

- **Edges:**
  - `my_add_op` connected to `input1`, `input2`
  - `my_mul_op` connected to `input2`

**feed_dict:** `{a: 3.0, b: 2.0}`
MAIN TENSORFLOW API

Graph
- Container for operations and tensors

Operation
- Nodes in the graph
- Represent computations

Tensor
- Edges in the graph
- Represent data

TENSORFLOW* GRAPH EXAMPLE:
TensorFlowBasic.ipynb
NEURAL NETWORK WITH TENSORFLOW*
Use biology as inspiration for math model

Neurons:
- Get signals from previous neurons
- Generate signal (or not) according to inputs
- Pass that signal on to future neurons

By layering many neurons, can create complex model
Data flows into neuron from previous layers

Some form of computation transforms the inputs

The neuron outputs the transformed data

activation function

READS ROUGHLY THE SAME AS A TENSORFLOW GRAPH
READS ROUGHLY THE SAME AS A TENSORFLOW GRAPH

\[ z = x_1w_1 + x_2w_2 + x_3w_3 + b \]

activation function

\[ f(z) \]
TYPES OF ACTIVATION FUNCTIONS

▪ Sigmoid function
  – Smooth transition in output between (0,1)

▪ Tanh function
  – Smooth transition in output between (-1,1)

▪ ReLU function
  – \( f(x) = \max(x,0) \)

▪ Step function
  – \( f(x) = (0,1) \)
INSIDE A SINGLE NEURON (TENSORFLOW GRAPH)

Represents the function $z = W^t X + b$
The activation function applies a non-linear transformation and passes it along to the next layer.
TENSORFLOW* NEURON EXAMPLE:
TensorNeuronBasic.ipynb
INTEL® TENSORFLOW* OPTIMIZATION
INTEL® TENSORFLOW® OPTIMIZATIONS

1. **Operator optimizations**: Replace default (Eigen) kernels by highly-optimized kernels (using Intel® MKL-DNN)

2. **Graph optimizations**: Fusion, Layout Propagation

3. **System optimizations**: Threading model
BEFORE GIVING MORE DETAILS
RUN: TENSORFLOW* BENCHMARK
In TensorFlow, computation graph is a data-flow graph.

Examples

Weights

MatMul

Add

ReLU

Bias
Replace default (Eigen) kernels by highly-optimized kernels (using Intel® MKL-DNN)

Intel® MKL-DNN has optimized a set of TensorFlow operations.

Library is open-source (https://github.com/intel/mkl-dnn) and downloaded automatically when building TensorFlow.

<table>
<thead>
<tr>
<th>Forward</th>
<th>Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv2D</td>
<td>Conv2DGrad</td>
</tr>
<tr>
<td>ReLU, TanH, ELU</td>
<td>ReLUGrad, TanHGrad, ELUGrad</td>
</tr>
<tr>
<td>MaxPooling</td>
<td>MaxPoolingGrad</td>
</tr>
<tr>
<td>AvgPooling</td>
<td>AvgPoolingGrad</td>
</tr>
<tr>
<td>BatchNorm</td>
<td>BatchNormGrad</td>
</tr>
<tr>
<td>LRN</td>
<td>LRNGrad</td>
</tr>
<tr>
<td>MatMul, Concat</td>
<td></td>
</tr>
</tbody>
</table>
On Intel processors a high % of time is typically spent in BW-limited ops

- ~40% of ResNet-50, even higher for inference

The solution is to fuse BW-limited ops with convolutions or one with another to reduce the # of memory accesses

- Conv+ReLU+Sum, BatchNorm+ReLU, etc
- Done for inference, WIP for training

The FWKs are expected to be able to detect fusion opportunities

- IntelCaffe already supports this

Major impact on implementation

- All the impls. must be made aware of the fusion to get max performance
- Intel MKL-DNN team is looking for scalable solutions to this problem
GRAPH OPTIMIZATIONS: FUSION

Before Merge

After Merge
**Graph Optimizations: Layout Propagation**

- All MKL-DNN operators use highly-optimized layouts for TensorFlow tensors.
More on memory channels: Memory layouts

Most popular memory layouts for image recognition are **nhwc** and **nchw**

- Challenging for Intel processors either for vectorization or for memory accesses (cache thrashing)

Intel MKL-DNN convolutions use blocked layouts

- Example: **nhwc** with channels blocked by 16 – **nChw16c**
- Convolutions define which layouts are to be used by other primitives
- Optimized frameworks track memory layouts and perform reorders **only** when necessary
**DATA LAYOUT HAS A BIG IMPACT**

- Continuous access to avoid gather/scatter
- Have iterations in inner most loop to ensure high vector utilization
- Maximize data reuse; e.g. weights in a convolution layer

Overhead of layout conversion is sometimes negligible, compared with operating on unoptimized layout

<table>
<thead>
<tr>
<th></th>
<th>21</th>
<th>18</th>
<th>6</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>92</td>
<td>37</td>
<td>29</td>
</tr>
<tr>
<td>40</td>
<td>11</td>
<td>9</td>
<td>22</td>
<td>3</td>
</tr>
<tr>
<td>23</td>
<td>3</td>
<td>47</td>
<td>29</td>
<td>88</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>16</td>
<td>22</td>
<td>46</td>
</tr>
<tr>
<td>29</td>
<td>9</td>
<td>13</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

for \( i = 1 \) to \( N \) # batch size
for \( j = 1 \) to \( C \) # number of channels, image RGB = 3 channels
for \( k = 1 \) to \( H \) # height
for \( l = 1 \) to \( W \) # width

dot_product( ...)

<table>
<thead>
<tr>
<th>21</th>
<th>18</th>
<th>...</th>
<th>1</th>
<th>8</th>
<th>92</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>8</td>
<td>18</td>
<td>92</td>
<td>..</td>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>
Proven only for certain CNNs by IntelCaffe at the moment

A trained float32 model quantized to int8

Some operations still run in float32 to preserve accuracy
SYSTEM OPTIMIZATIONS: LOAD BALANCING

TensorFlow graphs offer opportunities for parallel execution.

Threading model

1. `inter_op_parallelism_threads` = max number of operators that can be executed in parallel

2. `intra_op_parallelism_threads` = max number of threads to use for executing an operator

3. `OMP_NUM_THREADS` = MKL-DNN equivalent of `intra_op_parallelism_threads`
tf.ConfigProto is used to set the `inter_op_parallelism_threads` and `intra_op_parallelism_threads` configurations of the Session object.

```python
>>> config = tf.ConfigProto()
>>> config.intra_op_parallelism_threads = 56
>>> config.inter_op_parallelism_threads = 2
>>> tf.Session(config=config)
```

https://www.tensorflow.org/performance/performance_guide#tensorflow_with_intel_mkl_dnn

Example setting MKL variables with python `os.environ`:

```python
os.environ["KMP_BLOCKTIME"] = "1"
os.environ["KMP_AFFINITY"] = "granularity=fine,compact,1,0"
os.environ["KMP_SETTINGS"] = "0"
os.environ["OMP_NUM_THREADS"] = "56"
```

Tuning MKL for the best performance

This section details the different configurations and environment variables that can be used to tune the MKL to get optimal performance. Before tweaking various environment variables make sure the model is using the NCHW (channels_first) data format. The MKL is optimized for NCHW and Intel is working to get near performance parity when using NHWC.

MKL uses the following environment variables to tune performance:

- **KMP_BLOCKTIME** - Sets the time, in milliseconds, that a thread should wait, after completing the execution of a parallel region, before sleeping.
- **KMP_AFFINITY** - Enables the run-time library to bind threads to physical processing units.
- **KMP_SETTINGS** - Enables (true) or disables (false) the printing of OpenMP* run-time library environment variables during program execution.
- **OMP_NUM_THREADS** - Specifies the number of threads to use.

Tensorflow with Intel® MKL-DNN

### Executive Summary

<table>
<thead>
<tr>
<th>Environment</th>
<th>Network</th>
<th>Batch Size</th>
<th>Images/Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>resnet50</td>
<td>16</td>
<td>3.84</td>
</tr>
<tr>
<td>Optimized</td>
<td>resnet50</td>
<td>16</td>
<td>17.69</td>
</tr>
</tbody>
</table>

Average Intel Optimized speedup = 5X
CONVOLUTIONAL NEURAL NETWORK (CNN)
Complex Networks with billions of parameters can take days to train on a modern processor.

Hence, the need to reduce time-to-train using a cluster of processing nodes.
CONVOLUTION NEURAL NETWORK LAYERS

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

CONVOLUTION = MULTIPLY – ADD OP.

Image

Convolved Feature

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
DEEP LEARNING TRAINING & INFERENCE

Training

Input

Deep Learning Neural Network

Forward Propagation

Result

Difference (Loss)

Answer

Backward Propagation

Inference

Input

Deep Learning Neural Network

Forward Propagation

Result
HANDS-ON
IMAGE CLASSIFICATION OF THE MNIST DATASET

• Implementation of a simple Convolutional Neural Network in TensorFlow with two convolutional layers, followed by two fully-connected layers at the end

Source: https://www.easy-tensorflow.com/tf-tutorials/convolutional-neural-nets-cnns
TENSORFLOW: CNN_MNIST.IPYNB

Let's try to run this example a observe the performance

Standard Python and Tensorflow installation

- source activate python-3.6
- pip show tensorflow | grep Location
  - useful to locate the TF installation for see the library linked: ldd $Location/tensorflow,
- rm -rf mnist_convnet_model/*
- Run the sample: time python cnn_mnist.py

Intel Python and Optimized Tensorflow

- source activate intel-py
- pip show tensorflow | grep Location
  - useful to locate the TF installation for see the library linked: ldd $Location/tensorflow,
- rm -rf mnist_convnet_model/*
- export export MKLDNN VERBOSE=1
What it is
A toolkit to accelerate development of high performance computer vision & deep learning inference into vision/Artificial Intelligence applications used from edge to cloud. It enables deep learning on hardware accelerators and easy deployment across multiple types of Intel® platforms.

Who needs this product?
- Computer vision, deep learning software developers
- Data scientists
- OEMs, ISVs, System Integrators

Usages
Security surveillance, robotics, retail, healthcare, AI, office automation, transportation, non-vision use cases (speech, text) & more.

Free Download ➤ software.intel.com/openvino-toolkit
Open Source version ➤ 01.org/openvinotoolkit
What’s Inside Intel® Distribution of OpenVINO™ toolkit

### Deep Learning

**Intel® Deep Learning Deployment Toolkit**

- **Model Optimizer**: Convert & Optimize
- **Inference Engine**: Optimized Inference

**Open Model Zoo**

- **40+ Pretrained Models**
- **Samples**
- **Model Downloader**

**Deep Learning Workbench**

- Calibration Tool
- Model Analyzer
- Benchmark App
- Accuracy Checker
- Aux. Capabilities

### Traditional Computer Vision

**Optimized Libraries & Code Samples**

- **OpenCV**
- **OpenVX**
- **Samples**

For Intel® CPU & GPU/Intel® Processor Graphics

### Tools & Libraries

**Increase Media/Video/Graphics Performance**

- **Intel® Media SDK**: Open Source version
- **OpenCL™ Drivers & Runtimes**: For GPU/Intel® Processor Graphics

**Optimize Intel® FPGA (Linux® only)**

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

### OS Support:

- CentOS* 7.4 (64 bit), Ubuntu* 16.04.3 LTS (64 bit), Microsoft Windows* 10 (64 bit), Yocto Project* version Poky Jethro v2.0.3 (64 bit), macOS* 10.13 & 10.14 (64 bit)

### Other Information:

- An open source version is available at [01.org/openvinotoolkit](http://01.org/openvinotoolkit) (some deep learning functions support Intel CPU/GPU only).

- OpenVX and the OpenVX logo are trademarks of the Khronos Group Inc.

- OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

- Intel® Architecture-Based Platforms Support

- Intel® Vision Accelerator Design Products & AI in Production/Developer Kits

*Other names and brands may be claimed as the property of others.*
Intel® Distribution of OpenVINO™ in a nutshell

1. Trained DL model
2. Model Optimizer
3. IR
4. IR

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Free Download ▶
software.intel.com/openvino-toolkit
Open Source version ▶
01.org/openvinotoolkit
Legal Disclaimer & Optimization Notice

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.

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Copyright © 2019, Intel Corporation. All rights reserved. Intel, the Intel logo, Pentium, Xeon, Core, VTune, OpenVINO, Cilk, are trademarks of Intel Corporation or its subsidiaries in the U.S. and other countries.

**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