Machine Learning: Your Path to Deeper Insight

Driving increasing innovation and competitive advantage across industries

![Solutions](image)

strategy provides the foundation for success using AI

**Solutions**
for reference across industries

**Tools/Platforms**
to accelerate deployment

**Optimized Frameworks**
to simplify development

**Libraries/Languages**
featuring optimized building blocks

**Hardware Technology**
portfolio that is broad and cross-compatible

---

Intel® Deep Learning SDK for Training & Deployment

Intel® Data Analytics Acceleration Library (Intel® DAAL)

Intel® Math Kernel Library (Intel® MKL & MKL-DNN)

Intel® Integrated Performance Primitives (Intel® IPP)

Intel® Distribution for Python*

---

+ Network
+ Memory
+ Storage

---

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Problem Statement

Big data frameworks: Hadoop, Spark, Cassandra, etc.

Data mining
Recommendation engines
Customer behavior modeling
BI analytics
Real time analytics

- Limited performance
- Many layers of dependencies
- Low ROI on HW investment

Data sources
- SQL stores
- NoSQL stores
- In-memory stores

Connectors
- Spark* MLLib
- Breeze
- Netlib-Java
- JVM
- JNI
- Netlib BLAS

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Desired Solution

Big data analytics

Desired practice

- Run on state-of-art hardware
- Single library to cover all stages of data analytics
- Fully optimized for underlying hardware

Data sources

SQL stores

NoSQL stores

In-memory stores

Big data frameworks: Hadoop, Spark, Cassandra, etc.

Optimized performance

Simpler development & deployment

High ROI on HW investment
Intel® Data Analytics Acceleration Library (Intel® DAAL)

An IA-optimized library that provides building blocks for all data analytics stages, from data preparation to data mining & machine learning

- C++ and Java APIs in the first release
- Python API – second Release
- Can be used with many platforms (Hadoop*, Spark*, R*, Matlab*, ...) but not tied to any of them
- Flexible interface to connect to different data sources (CSV, SQL, HDFS, ...)
- Windows*, Linux*, and OS X*
- IA-32 and Intel64 support
- Supports static and dynamic linking
- Developed by same team as industry-leading Intel® Math Kernel Library
- Commercial and Free Community editions
- Also included in Parallel Studio XE suites

*Other names and brands may be claimed as the property of others.
Intel® Data Analytic Acceleration Library

- Targets both data centers (Intel® Xeon® and Intel® Xeon Phi™) and edge-devices (Intel® Atom)
- Perform analysis close to data source (sensor/client/server) to optimize response latency, decrease network bandwidth utilization, and maximize security
- Offload data to server/cluster for complex and large-scale analytics
Overview of Big Data Attributes and Solution:

<table>
<thead>
<tr>
<th>Big Data Attributes</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Volume:</strong></td>
<td>• Distributed computing (e.g., communication-avoiding algorithms), streaming algorithms</td>
</tr>
<tr>
<td>• Huge data not fitting into node/device memory/distributed across nodes</td>
<td></td>
</tr>
<tr>
<td><strong>Velocity:</strong></td>
<td>• Data buffering, streaming algorithms</td>
</tr>
<tr>
<td>• Data arriving in time</td>
<td></td>
</tr>
<tr>
<td><strong>Variety:</strong></td>
<td>• Categorical → Numeric (counters, histograms, etc)</td>
</tr>
<tr>
<td>• Non-homogeneous/sparse/missing/noisy data</td>
<td>• Homogeneous numeric data kernels</td>
</tr>
<tr>
<td>• Conversions, Indexing, Repacking</td>
<td>• Sparse data algorithms</td>
</tr>
<tr>
<td>• Recovery methods (bootstrapping, outlier correction)</td>
<td></td>
</tr>
</tbody>
</table>

**Distributed Computing**
- Volume: $R_1, R_2, \ldots, R_k$
- Velocity: $D_1, D_2, \ldots, D_k$
- Variety: $S_{i+1} = T(S_i, D_i)$
- $R_{i+1} = F(S_{i+1})$

**Streaming Computing**
- Volume: $D_1, D_2, \ldots, D_k$
- Velocity: $S_i, R_i$
- Variety: $D_{i+1} = \text{Append}(D_i, D_{i+1})$

**Batch Computing**
- Volume: $D_1, D_2, \ldots, D_k$
- Velocity: $R = F(D_1, \ldots, D_k)$
- Variety: $R_{i+1} = F(S_{i+1})$

**Converts, Indexing, Repacking**
- Volume: $D_1, D_2, \ldots, D_k$
- Velocity: $R = F(D_1, \ldots, D_k)$
- Variety: $R_{i+1} = F(S_{i+1})$

**Data Recovery**
- Volume: $D_1, D_2, \ldots, D_k$
- Velocity: $R = F(D_1, \ldots, D_k)$
- Variety: $R_{i+1} = F(S_{i+1})$

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
## Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.

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

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Batch</th>
<th>Distributed</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Descriptive statistics</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low order moments</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Quantiles/sorting</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Statistical relationships</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correlation / Variance-Covariance</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(Cosine, Correlation) distance matrices</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Matrix decomposition</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cholesky</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Regression</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear/ridge regression</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Classification</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multinomial Naïve Bayes</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SVM (two-class and multi-class)</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boosting (Ada, Brown, Logit)</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unsupervised learning</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Association rules mining (Apriori)</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anomoly detection (uni-/multi-variate)</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KMeans</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>EM for GMM</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Recommender systems</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALS</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Deep learning</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fully connected, convolution, normalization, activation layers, model, NN, optimization solvers,</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Intel® DAAL Components

**Data Management**
Interfaces for data representation and access. Connectors to a variety of data sources and data formats, such as HDFS, SQL, CSV, and user-defined data source/format

- **Data Sources**
- **Numeric Tables**
- **Compression / Decompression**
- **Serialization / Deserialization**

**Data Processing Algorithms**
Optimized analytics building blocks for all data analysis stages, from data acquisition to data mining and machine learning

**Data Modeling Algorithms**
Data structures for model representation, and operations to derive model-based predictions and conclusions
Data Sets and Numeric Tables

Feature (a.k.a. Variable)
- Measurement of an individual property of the observed object.
- Categorical (nominal) – Qualitative classifications.
- Ordinal – Qualitative, but can be ranked/ordered.
- Continuous (interval) – Quantitative measurements.

Observation (a.k.a. Feature Vector)
- A vector contains values of features of an observed object.
- Homogeneous or Heterogeneous.

Data sets
- A collection of observations.

Data dictionary
- Metadata, mapping features to their meanings.
Numeric Tables: In-Memory Data Representation

Heterogeneous – AOS
- Observations are stored in contiguous memory buffers.

Heterogeneous – SOA
- Features are stored in contiguous memory buffers.

Homogeneous – Dense matrix
- 2D matrix: $n$ rows (observations), $p$ columns (features)

Homogeneous – Sparse matrix (CSR)
- Support both 0-based indexing and 1-based indexing.

Tensors (since v.2017)
- in-memory numeric multidimensional data
Intel® DAAL - technical details

Underlying dependencies:

- Intel® C/C++ Compiler, Intel® Integrative Performance Primitives (IPP), Intel® Math Kernel Libraries (MKL)
- Threading Building Blocks (TBB), Intel® OpenMP® Runtime Library

Compiling/Linking:

- Single-threaded/multithreaded
- Static, Dynamic Linking
- IDE Integration
- Compiler Options (/Qdaal: parallel, sequential. Or –daal=parallel, sequential)

```bash
icc my_first_daal_program.cpp -o my_first_daal_program $DAALROOT/lib/intel64_lin/libdaal_core.a
$DAALROOT/lib/intel64_lin/libdaal_thread.a -liomp5 -ltbb -lpthread -ldl
```

CPU Dispatching

Customization
Intel® Data Analytics Acceleration Library 2017

• **Major new functionality Python APIs**
  - Python APIs (in addition to existing C++ and Java APIs)

• **Support building neural networks for deep learning applications**
  - Layers: convolution, pooling, fully connected, locally connected, dropout, etc.
  - Activation functions: logistic regression, hyperbolic tangent, ReLU, pReLU, soft ReLU, etc.
  - Function optimizations: SGD, L-BFGS, minibatch, Adagrad, etc.

• **Support more types of data sources** (Attribute-Relation File Format, PostgreSQL, asynchronous)

• **Open source version under Apache 2.0 license**
Moments of Low Order
Quantile
Correlation and Variance-Covariance Matrices
Cosine Distance Matrix
k-Means Clustering
Principal Component Analysis
Cholesky Decomposition
Singular Value Decomposition
Association Rules
Qf Decomposition
Expectation-Maximization
Multivariate Outlier Detection
Univariate Outlier Detection
Kernel Functions
Linear Kernel
Radial Basis Function Kernel
Quality Metrics
Quality Metrics for Binary Classification Algorithms
Quality Metrics for Multi-class Classification Algorithms
Quality Metrics for Linear Regression
Working with User-defined Quality Metrics
Sorting
Normalization
Z-score
Min-max
Math Functions
Logistic
Softmax
Hyperbolic Tangent
Absolute Value (abs)
Rectifier Linear Unit (ReLU)
Smooth Rectifier Linear Unit (SmoothReLU)
Optimization Solvers
Objective Function
Iterative Solver
Training and Prediction
Regressors
Regression methods. These methods predict the values of dependent variables (responses) by observing independent variables.
Classification
Naive Bayes Classifier
Boosting
Support Vector Machine Classifier
Multi-class Classifier
k-Nearest Neighbors (kNN) Classifier
Recommendation Systems
Implicit Alternating Least Squares

Intel® DAAL Content

Intel DAAL analysis algorithms

Intel DAAL Training and prediction algorithms

Layers: Intel DAAL provides the following types of layers:

Fully-connected layers, which compute the inner product of all weighed inputs plus bias.

Fully-connected Layers Forward | Backward

Activation layers, which apply a transform to the input data.

Abs Value Layers Forward | Backward
Logistic Layers Forward | Backward
Parametric Rectifier Linear Unit (pReLU) Layers Forward | Backward
Rectifier Linear Unit (ReLU) Layers Forward | Backward
Smooth Rectifier Linear Unit (SmoothReLU) Layers Forward | Backward
Hyperbolic Tangent Layers Forward | Backward

Normalization layers, which normalize the input data.

Batch Normalization layers Forward | Backward
Local Contrast Normalization Layers Forward | Backward

Anti-overfitting layers, which prevent the neural network from overfitting.

Dropout Layers Forward | Backward

Pooling layers, which apply a form of non-linear downsampling to input data.

1D Max Pooling Layers Forward | Backward
2D Max Pooling Layers Forward | Backward
3D Max Pooling Layers Forward | Backward
1D Average Pooling Layers Forward | Backward
2D Average Pooling Layers Forward | Backward
3D Average Pooling Layers Forward | Backward
2D Stochastic Pooling Layers Forward | Backward
2D Spatial Pyramid Pooling Layers Forward | Backward

Convolutional and locally-connected layers, which apply filters to input data.

2D Convolution Layers Forward | Backward
2D Transposed Convolution Layers Forward | Backward
2D Locally-connected Layers Forward | Backward

Service layers, which apply service operations to the input tensors.

Reshape Layers Forward | Backward

Concat Layers Forward | Backward
Split Layers Forward | Backward

Softmax layers, which measure confidence of the output of the neural network.

Softmax Layers Forward | Backward

Loss layers, which measure the difference between the output of the neural network and ground truth.

Loss Layers Forward | Backward
Loss Cross-entropy Layers Forward | Backward
Loss Logistic Cross-entropy Layers
• Targets both data centers (Intel® Xeon® and Intel® Xeon Phi™) and edge-devices (Intel® Atom)
• Perform analysis close to data source (sensor/client/server) to optimize response latency, decrease network bandwidth utilization, and maximize security
• Offload data to server/cluster for complex and large-scale analytics
<table>
<thead>
<tr>
<th>Image Processing</th>
<th>Computer Vision</th>
<th>Signal Processing</th>
</tr>
</thead>
</table>
| • Geometry transformations  
• Linear and non-linear filtering  
• Linear transforms  
• Statistics and analysis  
• Color models | • Feature detection  
• Objects tracking  
• Pyramids functions  
• Segmentation, enhancement  
• Camera functions  
• And more | • Transforms  
• Convolution, Cross-Correlation  
• Signal generation  
• Digital filtering  
• Statistical |
| Data Compression | Cryptography | String Processing |
| • LZSS  
• LZ77(ZLIB)  
• LZO  
• Bzip2 | • Symmetric cryptography  
• Hash functions  
• Data authentication  
• Public key | • String Functions: Find, Insert, Remove, Compare, etc.  
• Regular expression |
Intel® IPP v.2017 update 2, LZO optimization

**Optimization Notice**

Copyright © 2017, Intel Corporation. All rights reserved.

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

---

**Configuration Info** - Versions: Intel® Integrated Performance Primitives Library 2017 update 2;
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. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

---

**LZO, Compression, Mb/s, Calgary**

- Lzo (MB/s)
- ipp_lzo (MB/s)

**LZO, Compression Ratio, Calgary**

- Ratio: Compression - ipp_lzo/lzo
- Ratio: Performance - ipp_lzo/lzo

---

Intel® IPP v.2017 update 2, LZO optimization

---

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.
Intel® DAAL - Handwritten Digit Recognition
/* Create a numeric table from a CSV file */
const std::string datasetFileName = "../data/batch/em_gmm.csv";
FileDataSource<CSVFeatureManager> data_source(
    datasetFileName,
    DataSource::doAllocateNumericTable,
    DataSource::doDictionaryFromContext);

/* Make the data available for algorithms */
data_source.loadDataBlock(NUMBER_OF_OBSERVATIONS);
SharedPtr<NumericTable>& numTable = data_source.getNumericTable();

/* Retrieve the data from the input file */
dataSource.loadDataBlock();
void trainModel()
{
    /* Initialize FileDataSource<CSVFeatureManager> to retrieve input data from .csv file */
    FileDataSource<CSVFeatureManager> trainDataSource(trainDatasetFileName,
        DataSource::doAllocateNumericTable, DataSource::doDictionaryFromContext);

    /* Load data from the data files */
    trainDataSource.loadDataBlock(nTrainObservations);

    /* Create algorithm object for multi-class SVM training */
    multi_class_classifier::training::Batch<> algorithm;
    algorithm.parameter.nClasses = nClasses;
    algorithm.parameter.training = training;

    /* Pass training dataset and dependent values to the algorithm */
    algorithm.input.set(classifier::training::data, trainDataSource.getNumericTable());

    /* Build multi-class SVM model */
    algorithm.compute();

    /* Retrieve algorithm results */
    trainingResult = algorithm.getResult();

    /* Serialize the learned model into a disk file */
    ModelFileWriter writer("./model");
    writer.serializeToFile(trainingResult->get(classifier::training::model));
}
void testDigit()
{
    /* Initialize FileDataSource<CSVFeatureManager> to retrieve the test data from .csv file */
    FileDataSource<CSVFeatureManager> testDataSource(testDatasetFileName,
        DataSource::doAllocateNumericTable, DataSource::doDictionaryFromContext);
    testDataSource.loadDataBlock(1);

    /* Create algorithm object for prediction of multi-class SVM values */
    multi_class_classifier::prediction::Batch<> algorithm;

    algorithm.parameter.prediction = prediction;

    /* Deserialize a model from a disk file */
    ModelFileReader reader("./model");
    services::SharedPtr<multi_class_classifier::Model> pModel(new multi_class_classifier::Model());
    reader.deserializeFromFile(pModel);

    /* Pass testing dataset and trained model to the algorithm */
    algorithm.input.set(classifier::prediction::data,
        testDataSource.getNumericTable());
    algorithm.input.set(classifier::prediction::model, pModel);

    /* Predict multi-class SVM values */
    algorithm.compute();

    /* Retrieve algorithm results */
    predictionResult = algorithm.getResult();

    /* Retrieve predicted labels */
    predictedLabels = predictionResult->get(classifier::prediction::prediction);
}
Training multi-class SVM for 10 digits recognition.

3,823 pre-processed training data.


99.6% accuracy with 1,797 test data from the same data provider.

Confusion matrix:

<table>
<thead>
<tr>
<th></th>
<th>0.000</th>
<th>0.000</th>
<th>0.000</th>
<th>0.000</th>
<th>0.000</th>
<th>1.000</th>
<th>0.000</th>
<th>0.000</th>
<th>0.000</th>
<th>0.000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>177.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>181.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>2.000</td>
<td>173.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>176.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>3.000</td>
</tr>
<tr>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>179.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>180.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>2.000</td>
</tr>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>180.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>180.000</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>170.000</td>
<td>8.000</td>
</tr>
<tr>
<td>0.000</td>
<td>3.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>166.000</td>
</tr>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>2.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>2.000</td>
</tr>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>175.000</td>
</tr>
</tbody>
</table>

Average accuracy: 0.996
Error rate: 0.004
Micro precision: 0.978
Micro recall: 0.978
Micro F-score: 0.978
Macro precision: 0.978
Macro recall: 0.978
Macro F-score: 0.978
**Intel® DAAL-PCA Performance Boosts Using Intel® DAAL vs. Spark**

Mllib on an Eight-node Cluster

---

**Configuration Info - Versions:** Intel® Data Analytics Acceleration Library 2017, Spark 1.2; Hardware: Intel® Xeon® Processor E5-2699 v3, 2 Eighteen-core CPUs (45MB LLC, 2.3GHz), 128GB of RAM per node; Operating System: CentOS 6.6 x86_64.

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. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

**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

• Intel® DAAL+Intel® MKL+Intel® IPP = Complementary Big Data Libraries Solution

• Intel MKL, Intel DAAL and Intel IPP provide high performance and optimized building blocks for data analytics and machine learning algorithms on Intel platforms.
Intel® DAAL Resources

Intel® Machine Learning


Intel® DAAL website


Intel® DAAL forum


Intel® DAAL blogs

Intel® MKL Resources

Intel® MKL website, forum, benchmarks


Intel® MKL link line advisor


Intel® MKL-DNN

- https://01.org/mkl-dnn

Intel® IA optimized frameworks

- https://github.com/intel/caffe
- https://github.com/intel/theano
Legal Disclaimer and Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

All products, platforms, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit Intel Performance Benchmark Limitations.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See www.intel.com/products/processor_number for details.

The Intel Core and Itanium processor families 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.

The code names Arrandale, Bloomfield, Boazman, Boulder Creek, Calpella, Chief River, Clarkdale, Cliffside, Cougar Point, Gulftown, Huron River, Ivy Bridge, Kilmer Peak, King's Creek, Lewissille, Lynnfield, Maho Bay, Montevina, Montevina Plus, Nehalem, Penryn, Puma Peak, Rainbow Peak, Sandy Bridge, Sugar Bay, Tylersburg, and Westmere presented in this document are only for use by Intel to identify a product, technology, or service in development, that has not been made commercially available to the public, i.e., announced, launched or shipped. It is not a "commercial" name for products or services and is not intended to function as a trademark.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

Intel, Intel Core, Core Inside, Itanium, and the Intel Logo are trademarks of Intel Corporation in the U.S. and other countries.