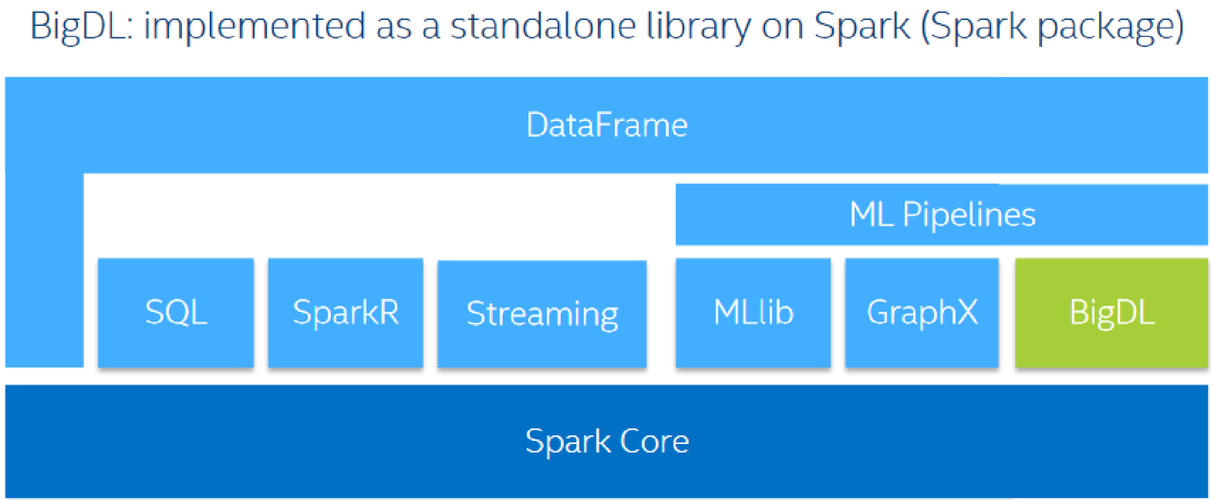
BIGDL: A DISTRIBUTED DEEP LEARNING LIBRARY ON SPARK

Yiheng Wang

Big Data Technology Team, Software and Service Group, Intel

What is BigDL?

BigDL is a distributed deep learning library for Apache Spark*



WHY BIGDL?

Why BigDL?

Production ML/DL system is **Complex**

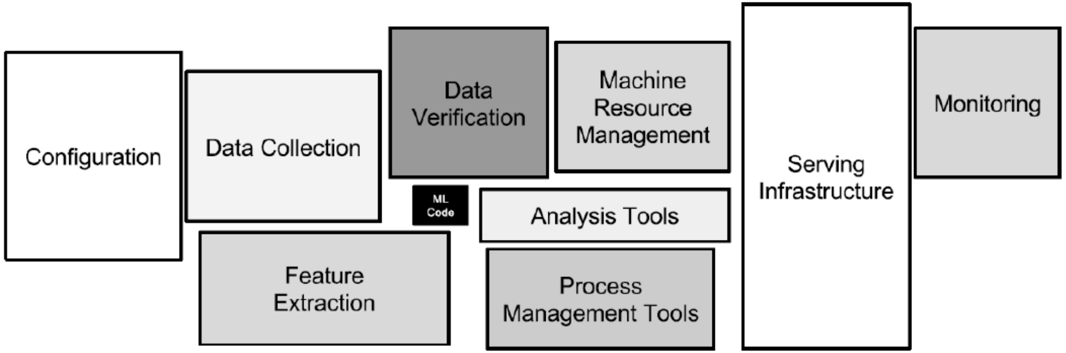


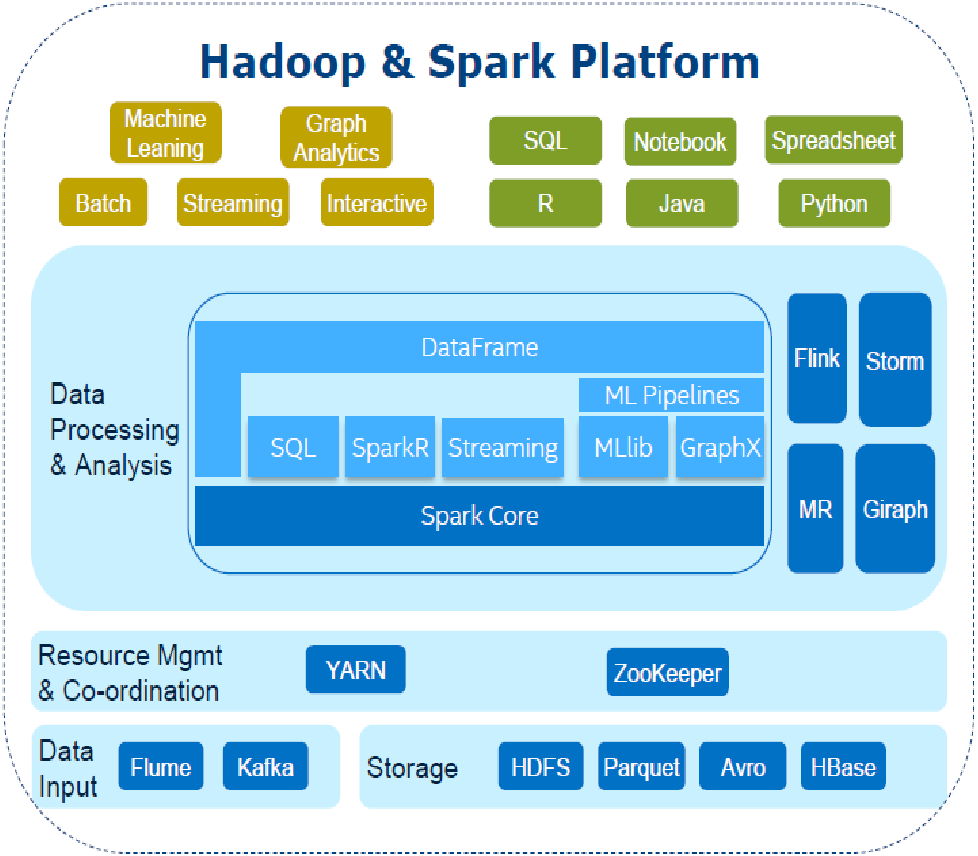
Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

"Hidden Technical Debt in Machine Learning Systems",
Google, NIPS 2015 Paper

Why BigDL?

How to Run Deep Learning Workloads Directly on Big Data Platform?

- Integrated with Big Data ecosystem
- Massively distributed, scale out
- Send compute to data
- Fault tolerance
- Elasticity
- Incremental scaling
- Dynamic resource sharing
- …



Why BigDL?

BigDL open sourced on Dec 30, 2016

https://github.com/intel-analytics/BigDL

- Write deep learning applications as standard Spark programs
- Run on top of existing Spark or Hadoop clusters(No change to the clusters)
- Rich deep learning support
- High performance powered by Intel MKL and multi-threaded programming
- Efficient scale-out with an all-reduce communications on Spark

Why BigDL?

You may want to write your deep learning programs using BigDL if:

- Analyze “big data” using deep learning on the same Hadoop/Spark cluster where the data are stored
- Add deep learning functionalities to the Big Data (Spark) programs and/or workflow
- Leverage existing Hadoop/Spark clusters to run deep learning applications, which are dynamically shared with other workloads (e.g., ETL, data warehouse, feature engineering, classical machine learning, graph analytics, etc.)
- Making deep learning more accessible for Big Data users and data scientists, who are usually not experts for deep learning

BIGDL FEATURES

BigDL Features

Tensor

- A powerful ndarray data structure
- Generic data type
- Data manipulate / math APIs, model after torch

```
scala> import com.intel.analytics.bigdl.tensor.Tensor
import com.intel.analytics.bigdl.tensor.Tensor

scala> val tensor = Tensor[Float](2, 3)
tensor: com.intel.analytics.bigdl.tensor.Tensor[Float] =
0.0     0.0     0.0
0.0     0.0     0.0
[com.intel.analytics.bigdl.tensor.DenseTensor of size 2x3]
```

BigDL Features

Layers

- 90+ Layers

Criterion

- 10+ loss functions

Optimization

- SGD, Adagrad, LBFGS

```
scala> import com.intel.analytics.bigdl.numeric.NumericFloat // import global float tensor n
import com.intel.analytics.bigdl.numeric.NumericFloat

scala> import com.intel.analytics.bigdl.nn._
import com.intel.analytics.bigdl.nn._

scala> val f = Linear(3,4) // create the module
mlp: com.intel.analytics.bigdl.nn.Linear[Float] = nn.Linear(3 -> 4)

// let's see what f's parameters were initialized to. ('nn' always inits to something reason
scala> f.weight
res5: com.intel.analytics.bigdl.tensor.Tensor[Float] =
-0.008662592    0.543819        -0.028795477
-0.30469555     -0.3909278      -0.10871882
0.114964925     0.1411745       0.35646403
-0.16590376     -0.19962183     -0.18782845
[com.intel.analytics.bigdl.tensor.DenseTensor of size 4x3]
```

BigDL Features

Build a simple model

```
scala> val g = Sum()
g: com.intel.analytics.bigdl.nn.Sum[Float] = nn.Sum

scala> val mlp = Sequential().add(f).add(g)
mlp: com.intel.analytics.bigdl.nn.Sequential[Float] =
nn.Sequential {
  [input -> (1) -> (2) -> output]
  (1): nn.Linear(3 -> 4)
  (2): nn.Sum
}
```

BigDL Features - A full example

```scala
val model = Sequential()
  .add(SpatialConvolution(3, 64, 11, 11, 4, 4, 2, 2, 1))
  .add(ReLU(true))
  .add(SpatialMaxPooling(3, 3, 2, 2))
  .add(SpatialConvolution(64, 192, 5, 5, 1, 1, 2, 2))
  .add(ReLU(true))
  .add(SpatialMaxPooling(3, 3, 2, 2))
  .add(SpatialConvolution(192, 384, 3, 3, 1, 1, 1, 1))
  .add(ReLU(true))
  .add(SpatialConvolution(384, 256, 3, 3, 1, 1, 1, 1))
  .add(ReLU(true))
  .add(SpatialConvolution(256, 256, 3, 3, 1, 1, 1, 1))
  .add(ReLU(true))
  .add(SpatialMaxPooling(3, 3, 2, 2))
  .add(View(256 * 6 * 6))
  .add(Linear(256 * 6 * 6, 4096))
  .add(ReLU(true))
  .add(Dropout(0.5))
  .add(Linear(4096, 4096))
  .add(ReLU(true))
  .add(Dropout(0.5))
  .add(Linear(4096, 1000))
  .add(LogSoftMax())
```

```scala
val optimizer = Optimizer(
  model = model,
  dataset = trainSet,
  criterion = new ClassNLLCriterion[Float]()
)
```

```scala
optimizer
  .setState(state)
  .setValidation(Trigger.severalIteration(620),
    valSet, Array(new Top1Accuracy[Float], new Top5Accuracy[Float]))
  .setEndWhen(Trigger.maxIteration(62000))
  .optimize()
```

BigDL Features

Start with tutorials

https://github.com/intel-analytics/BigDL/wiki/Tutorials

intel-analytics / **BigDL**

Code | Issues 51 | Pull requests 19 | Projects 0 | Wiki | Pulse

Tutorials

Shiqing Fan edited this page 24 days ago · 7 revisions

This page shows how to build simple deep learning programs using BigDL, including:

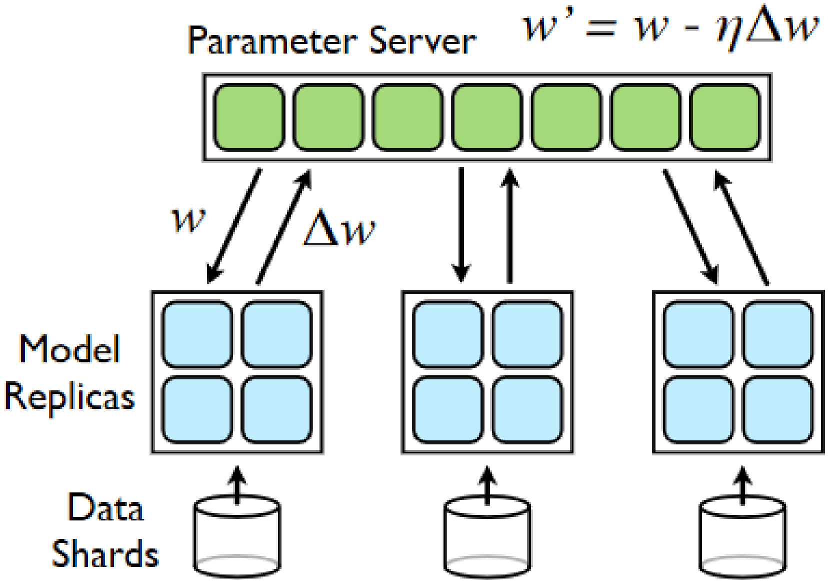
1. Training LeNet on MNIST - the "hello world" for deep learning
2. Text Classification - working with Spark RDD transformations
3. Image Classification - working with Spark DataFrame and ML pipeline

BigDL Features

Distributed Training

- Model Parallelism
- Data Parallelism

A distributed training example,
(Jeff Dean, NIPS 2012)

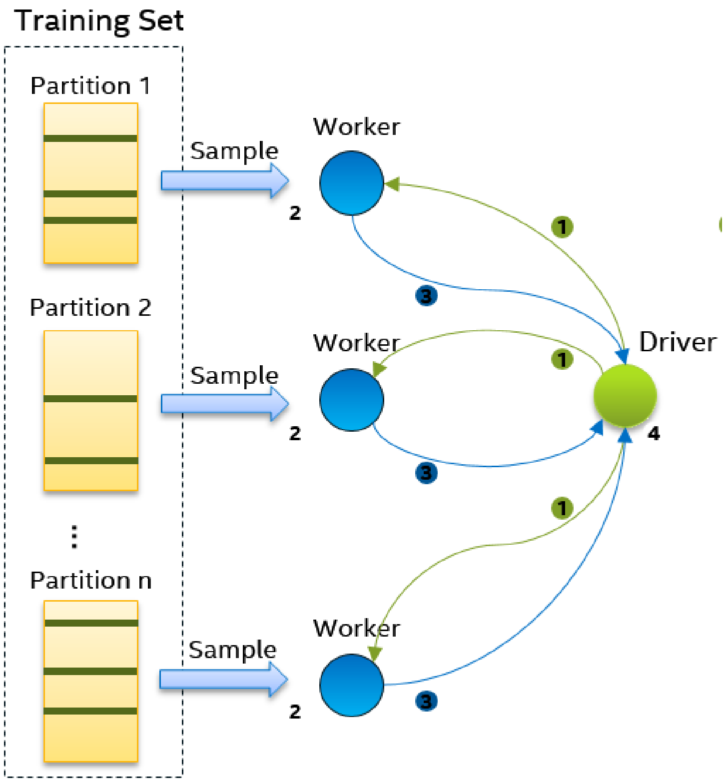


BigDL Features

“Canonical” implementation on Apache Spark

Driver become the bottleneck

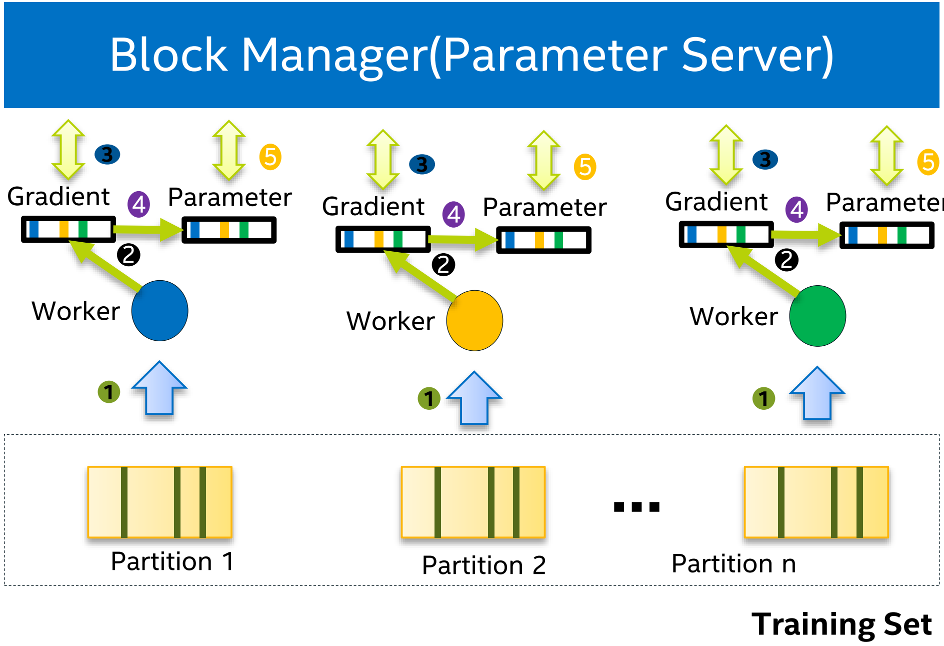
- RDD.reduce / aggregate
- RDD.treeAggregate (shuffle)



BigDL Features

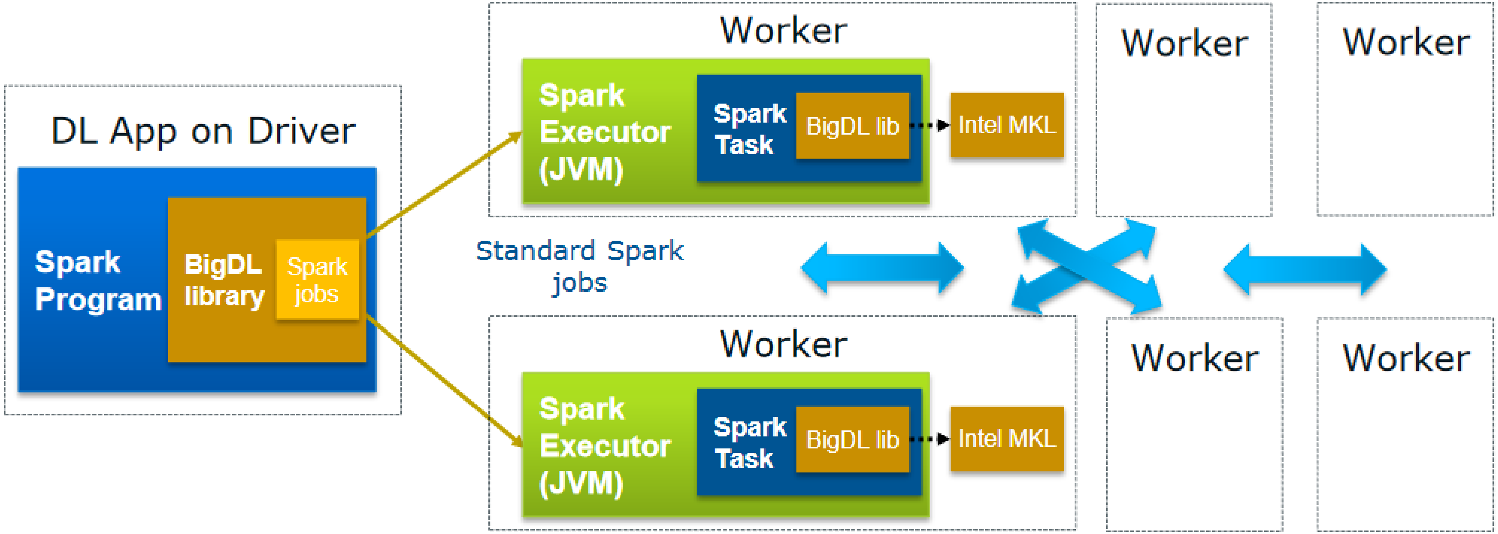
BigDL distributed training highlight

- Implement an P2P All Reduce Algorithm on Apache Spark
- Spark block manager as parameter server (handle different APIs of Spark 1.x/2.x)
- Compress float32 parameter to float16 parameter



BigDL Features

Train deep learning model on Apache Spark*



BigDL Features

Distributed training prefer large batch, need tune hyper parameter

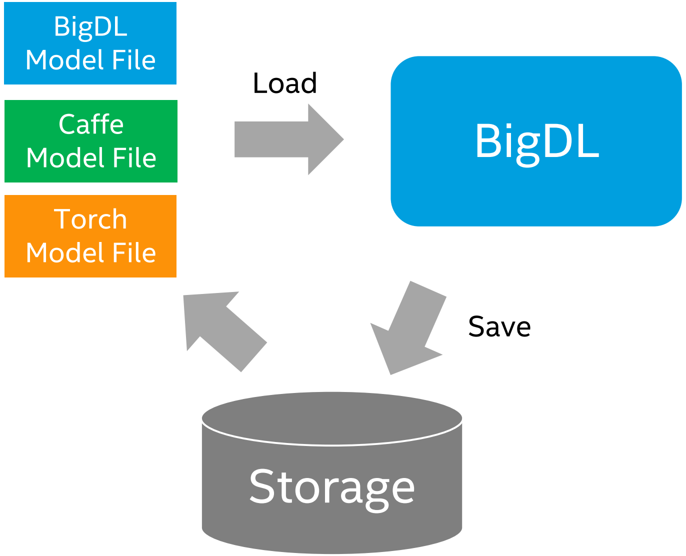
An example of Googlenet_v1 on ImageNet (batchsize = 1500+)

- learningRate -> 0.0896
- weightDecy -> 0.0001
- Momentum -> 0.9
- learningRateSchedule -> Poly(power = 0.5, iteration = 62000)

BigDL Features

- Model Snapshot
  - Useful in a long training
  - Used in inference later
  - Share your model with others
  - Fine-tune the model

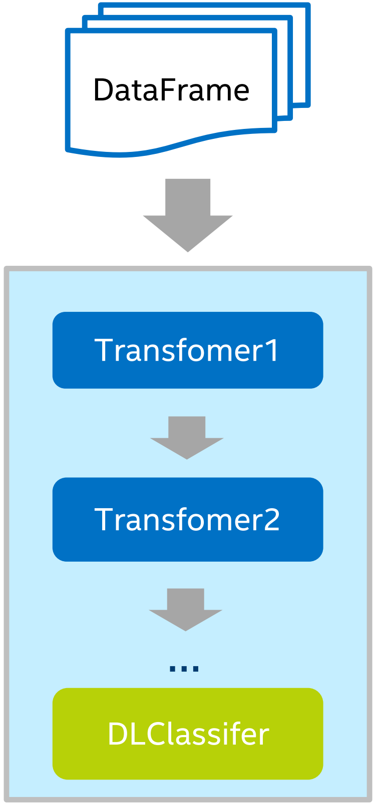
- Load Caffe/Torch Model
  - Leverage existed trained model



BigDL Features

Integrate with Spark-ML Pipeline

- Wrapper with Spark ML Transformer
- Plug into Spark ML pipeline
- Support 1.5/1.6/2.0



BigDL Features

BigDL provide examples to help developer play with bigdl and start with popular models.

https://github.com/intel-analytics/BigDL/wiki/Examples

Models (Train and Inference Example Code):

- LeNet, Inception, VGG, ResNet, RNN, Auto-encoder

Examples:

- Text Classification
- Image Classification
- Load Torch/Caffe model

Examples

Shiqing Fan edited this page 28 days ago · 4 revisions

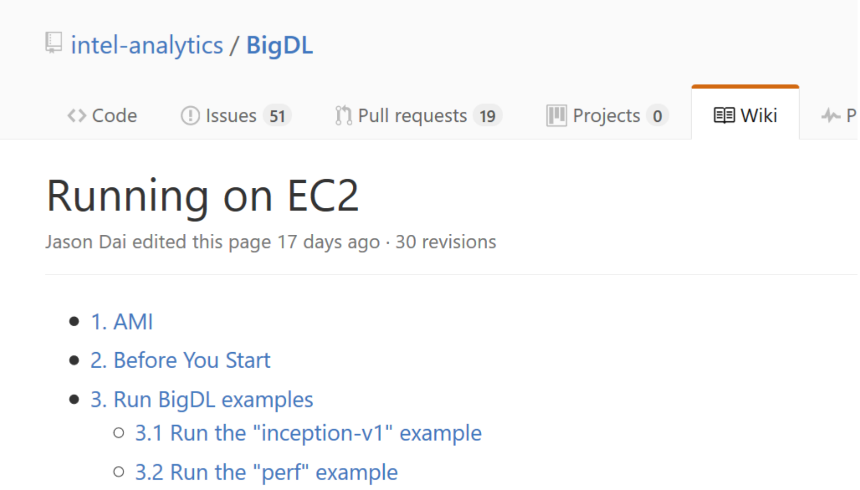
BigDL provides many popular neural network models and deep learning examples for Apache Spark, including:

- Models
  - LeNet: it demonstrates how to use BigDL to train and evaluate the LeNet-5 network on MNIST data.
  - Inception: it demonstrates how to use BigDL to train and evaluate Inception v1 and Inception v2 architecture on the ImageNet data.
  - VGG: it demonstrates how to use BigDL to train and evaluate a VGG-like network on CIFAR-10 data.
  - ResNet: it demonstrates how to use BigDL to train and evaluate the ResNet architecture on CIFAR-10 data.
  - RNN: it demonstrates how to use BigDL to build and train a simple recurrent neural network (RNN) for language model.
  - Auto-encoder: it demonstrates how to use BigDL to build and train a basic fully-connected autoencoder using MNIST data.
- Examples
  - text_classification: it demonstrates how to use BigDL to build a text classifier using a simple convolutional neural network (CNN) model.
  - image_classification: it demonstrates how to load a BigDL or Torch model trained on ImageNet data (e.g., Inception or ResNet), and then applies the loaded model to classify the contents of a set of images in Spark ML pipeline.
  - load_model: it demonstrates how to use BigDL to load a pre-trained Torch or Caffe model into Spark program for prediction.

BigDL Features

BigDL Out-of-box run scripts on AWS

https://github.com/intel-analytics/BigDL/wiki/Running-on-EC2



BigDL Features

Model on Data Set	Top-1 Accuracy
LeNet5 on MNIST	99%
Vgg on Cifar10	90%
AlexNet OWT on ImageNet	56%
GoogleNetV1 on ImageNet	68%

BigDL Features

- Single node Xeon performance
  - Benchmarked best on Xeon E5-26XX v3 or E5-26XX v4
  - Orders of magnitude speedup vs. out-of-box open source Caffe, Torch or TensorFlow

- Scaling-out
  - Efficiently scale out to several 10s of Xeon servers on Spark to distributed train some deep learning model on ImageNet dataset

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks

BIGDL – USE CASE

Fraud Transaction Detection

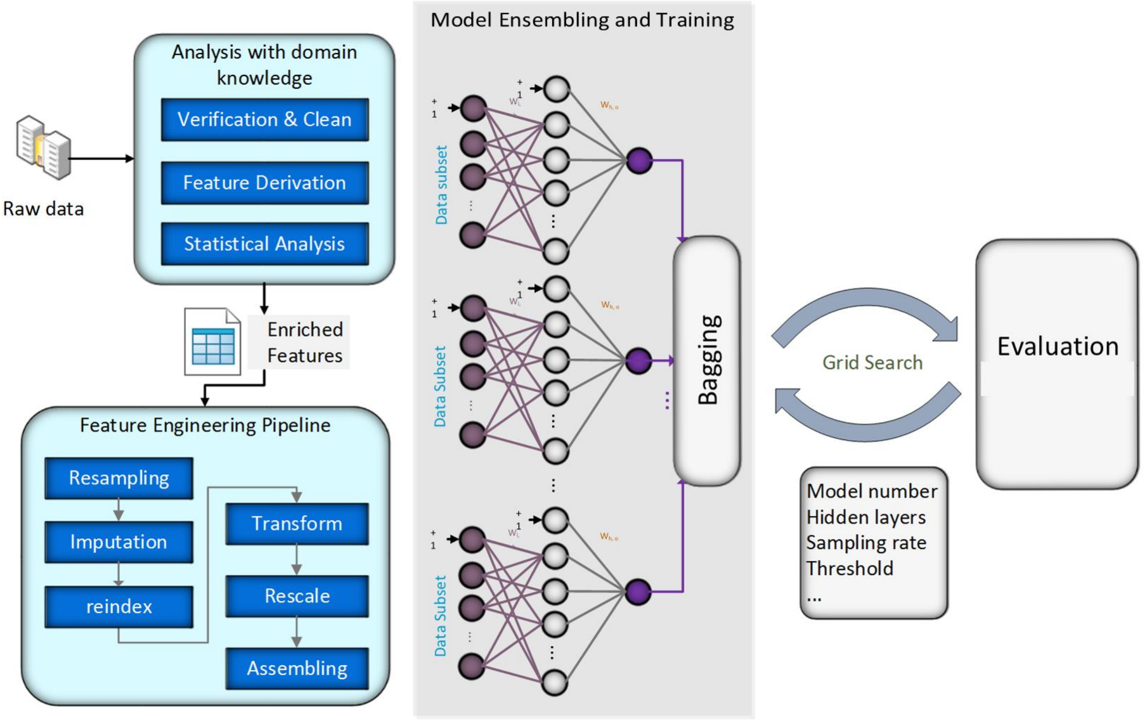
Fraud transaction detection is very import to finance companies. A good fraud detection solution can save a lot of money.

ML solution challenge

- Data cleaning
- Feature engineering
- Unbalanced data
- Hyper parameter

Fraud Transaction Detection

- History data is stored on Hive
- Easily data preprocess/cleaning with Spark-SQL
- Spark ML pipeline for complex feature engineering
- Under sample + Bagging solve unbalance problem
- Grid search for hyper parameter tunning



Powered by BigDL

Product Defect Detection and Classification

Data source

- Cameras installed on manufactory pipeline

Task

- Detect defect from the photos
- Classify the defect

Product Defect Detection and Classification

Big Data management

- High resolution images
- Large volume of data

Proposal Extraction

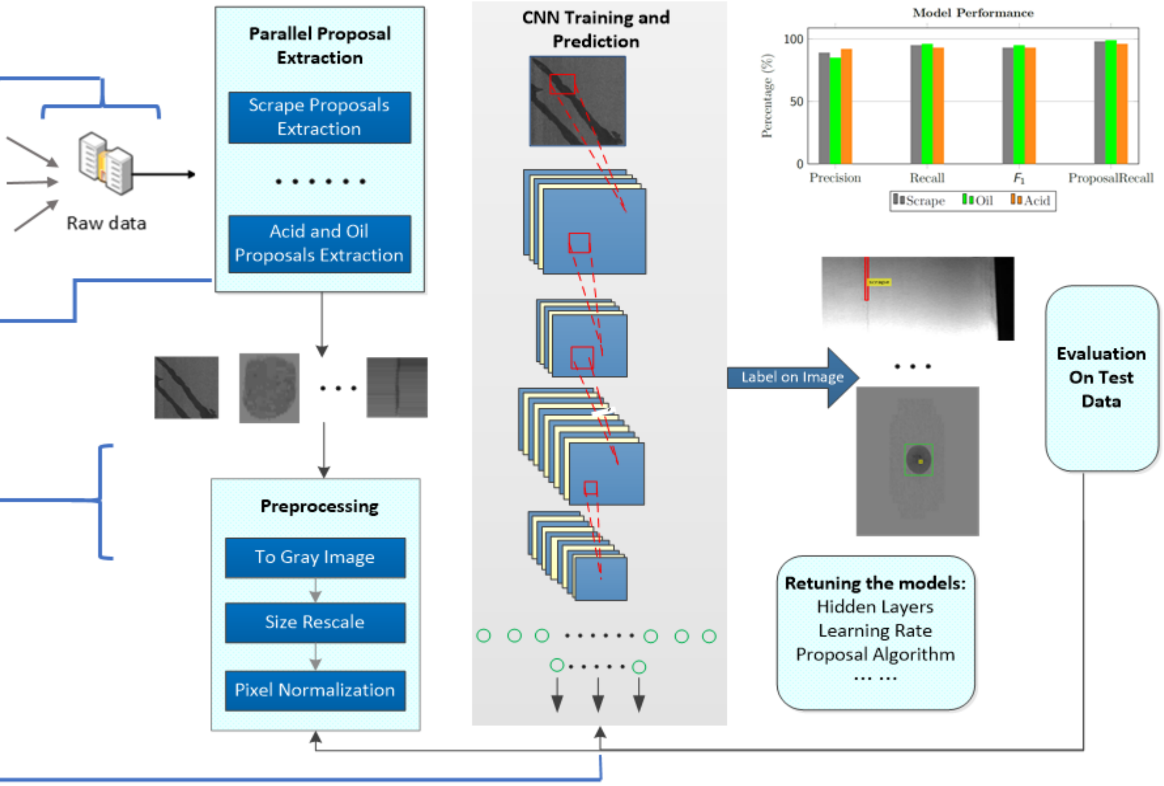
- Extract proposals for each defect
- Parallel pipeline **(KeyStone ML Pipeline)**
- Running **on Spark**

Preprocessing

- Preprocess the proposals before model training or testing

Model training pipeline

- Train ***Convolutional Neural Networks*** **on Spark**
- Parameter tuning, optimize proposal extraction algorithms



BIGDL ON GITHUB

https://github.com/intel-analytics/BigDL

Feature Requests

Some feature requests from community

- Mac support
- Python
- LSTM
- ...

Feedback or feature requests or PRs are welcome

Community

- Mail List

bigdl-user-group+subscribe@googlegroups.com

- Report bugs and feature request

https://github.com/intel-analytics/BigDL/issues

BigDL Contacts

Radhika Rangarajan (Spark Sr. Program Manager, Big Data Technologies, Intel)

Jason Dai (Sr. PE and Chief Architect, Big Data Technologies, Intel)

Ziya Ma (VP, SSG and Director, Big data Technologies, Intel)

Legal Disclaimer

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.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

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.

The products described in this document 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.

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 go to: http://www.intel.com/design/literature.htm

Intel, Quark, VTune, Xeon, Cilk, Atom, Look Inside and the Intel logo 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 ©2015 Intel Corporation.

Risk Factors

The above statements and any others in this document that refer to plans and expectations for the first quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Words such as “anticipates,” “expects,” “intends,” “plans,” “believes,” “seeks,” “estimates,” “may,” “will,” “should” and their variations identify forward-looking statements. Statements that refer to or are based on projections, uncertain events or assumptions also identify forward-looking statements. Many factors could affect Intel’s actual results, and variances from Intel’s current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be the important factors that could cause actual results to differ materially from the company’s expectations. Demand could be different from Intel's expectations due to factors including changes in business and economic conditions; customer acceptance of Intel’s and competitors’ products; supply constraints and other disruptions affecting customers; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Uncertainty in global economic and financial conditions poses a risk that consumers and businesses may defer purchases in response to negative financial events, which could negatively affect product demand and other related matters. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of Intel product introductions and the demand for and market acceptance of Intel's products; actions taken by Intel's competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel’s response to such actions; and Intel’s ability to respond quickly to technological developments and to incorporate new features into its products. The gross margin percentage could vary significantly from expectations based on capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; changes in revenue levels; segment product mix; the timing and execution of the manufacturing ramp and associated costs; start-up costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; product manufacturing quality/yields; and impairments of long-lived assets, including manufacturing, assembly/test and intangible assets. Intel's results could be affected by adverse economic, social, political and physical/infrastructure conditions in countries where Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Expenses, particularly certain marketing and compensation expenses, as well as restructuring and asset impairment charges, vary depending on the level of demand for Intel's products and the level of revenue and profits. Intel’s results could be affected by the timing of closing of acquisitions and divestitures. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust, disclosure and other issues, such as the litigation and regulatory matters described in Intel's SEC reports. An unfavorable ruling could include monetary damages or an injunction prohibiting Intel from manufacturing or selling one or more products, precluding particular business practices, impacting Intel’s ability to design its products, or requiring other remedies such as compulsory licensing of intellectual property. A detailed discussion of these and other factors that could affect Intel’s results is included in Intel’s SEC filings, including the company’s most recent reports on Form 10-Q, Form 10-K and earnings release.

intel®

Software