



Software

# DISTRIBUTED DEEP LEARNING AT SCALE ON APACHE SPARK WITH BIGDL

Yao Zhang

Big Data Technologies, Software and Service Group, Intel

# Intel Big Data Technology Team



## Closely collaborating with Spark open source community

- Started in 2012 when Spark is still a research project, and is among top 3 Spark contributors
- Many critical contributions (e.g., FairScheduler, Netty-Shuffle, YARN-Client, etc.)
- Library contribution: BigDL, Sparse-ML

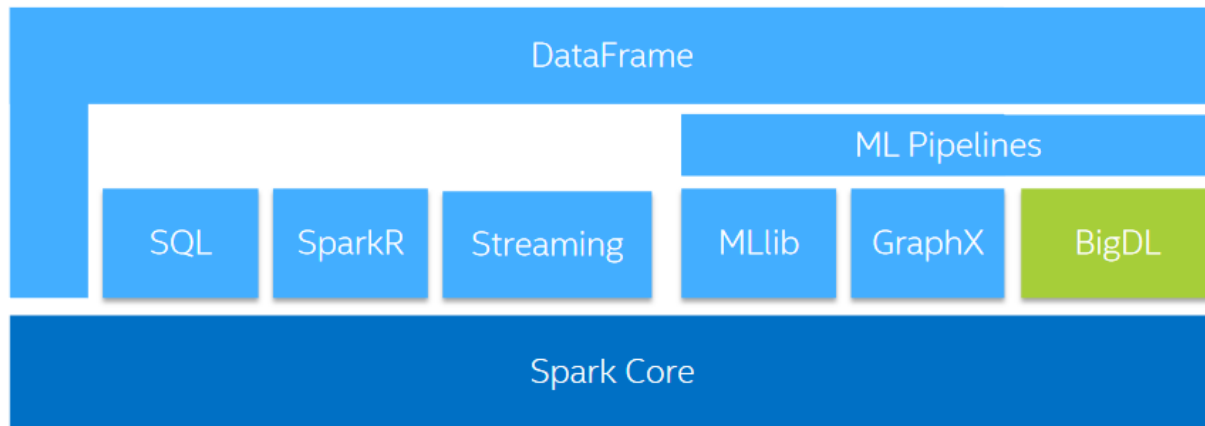
## Bridging advanced research and real-world applications

- Build advance big data analytics solutions with customer

# What is BigDL?

BigDL is a distributed deep learning library for Apache Spark\*

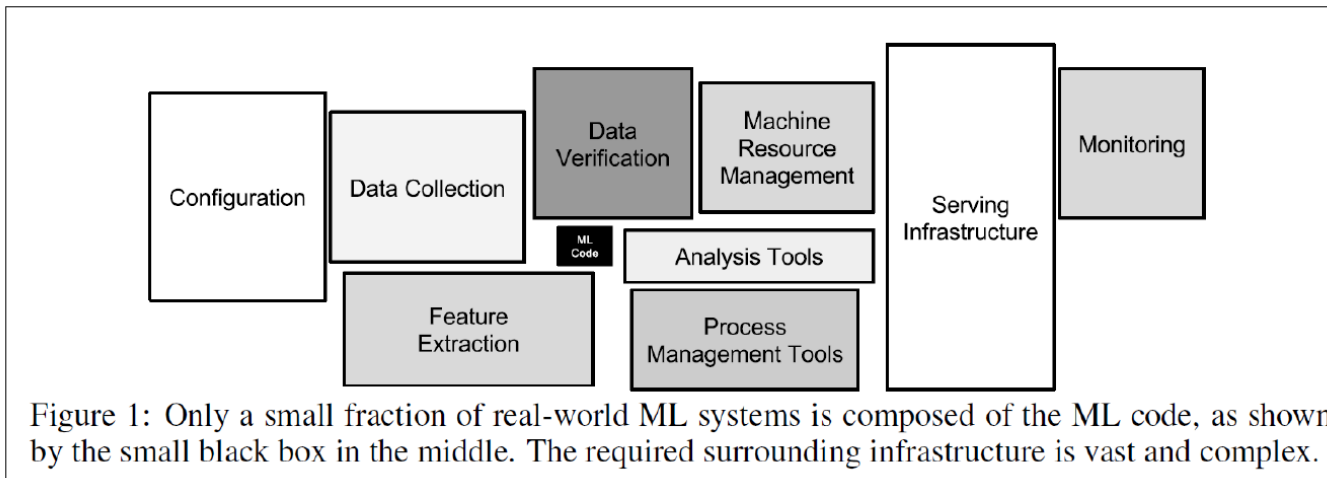
BigDL: implemented as a standalone library on Spark (Spark package)



# WHY BIGDL?

# Why BigDL?

Production ML/DL system is **Complex**

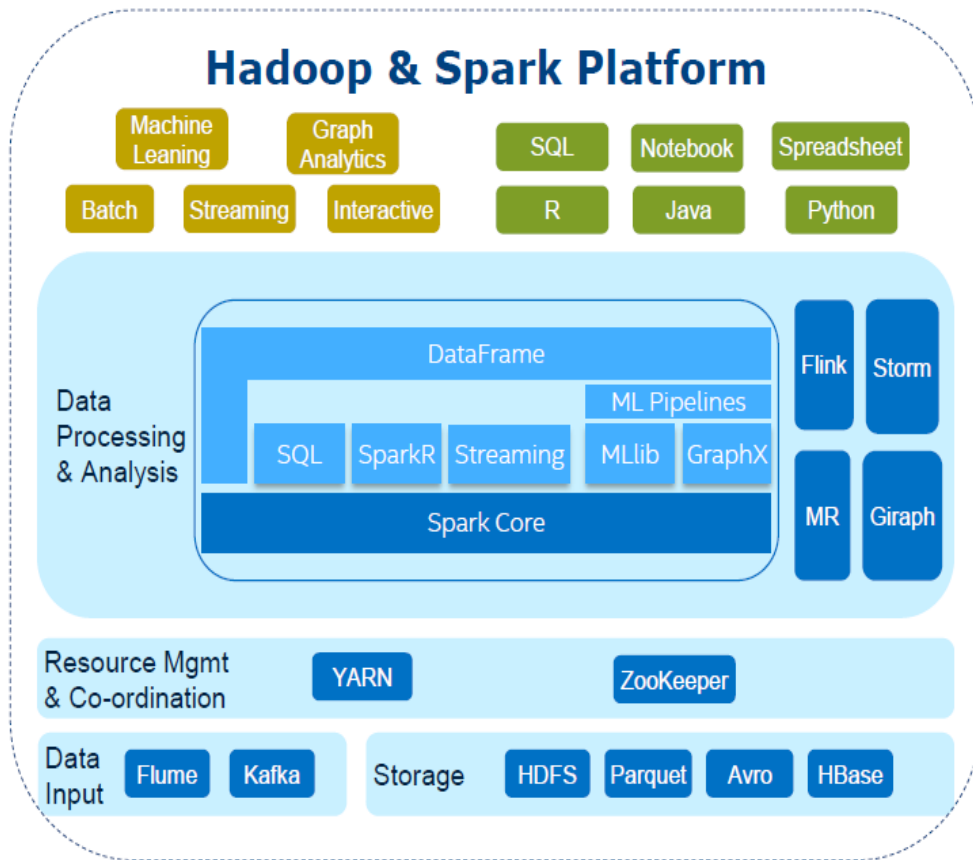


“Hidden Technical Debt in Machine Learning Systems”,  
Google, NIPS 2015 Paper

# Why BigDL?

End-to-end Big Data Analytics w/ Deep Learning Functionalities Directly on Spark

- Natively integrated with Big Data (Hadoop/Spark) ecosystem
- Massively distributed, scale out
- Send compute to data
- Fault tolerance
- Elasticity
- Incremental scaling
- Dynamic resource sharing



# Why BigDL?

- Write deep learning applications as **Standard Spark Programs (Scala and Python)**
- Run on top of **Existing** Spark or Hadoop clusters. **Easy to deploy, easy to manage your cluster and better cluster usage**
- **Rich** deep learning support, **close integrate** with other big data work load
- **High performance** powered by Intel MKL and multi-threaded programming
- **Efficient scale-out** with an all-reduce communications on Spark

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

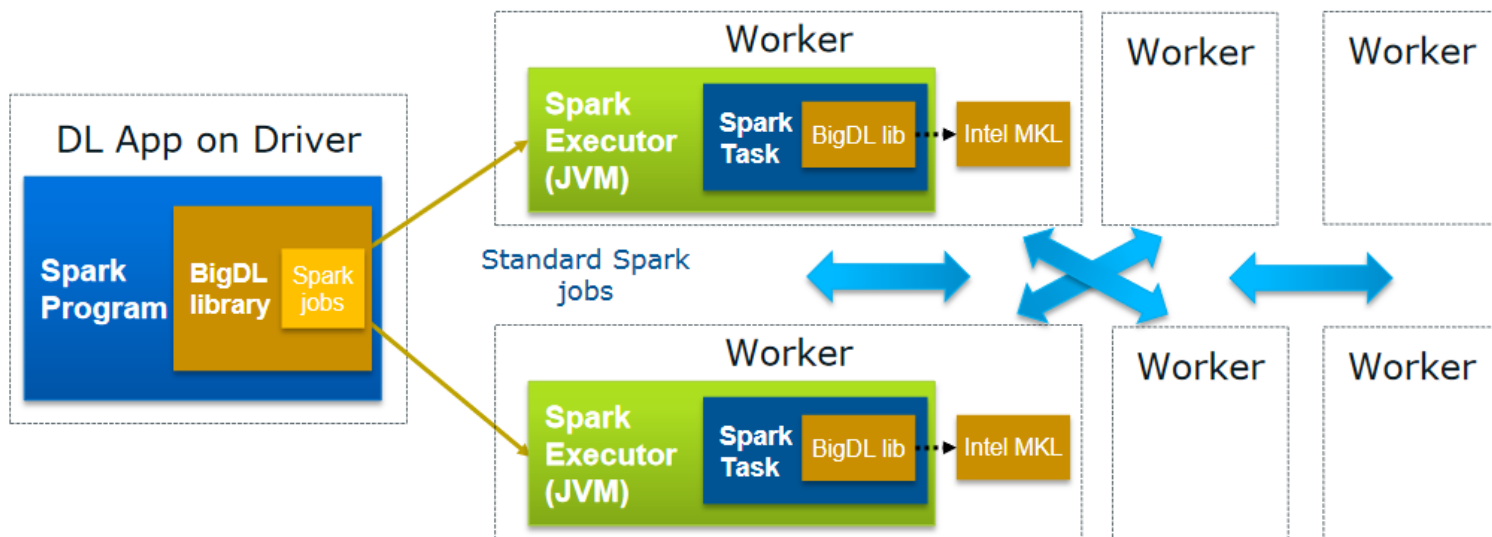
# BIGDL FEATURES



# BigDL Features

Distributed Deep learning applications (training, fine-tuning & prediction) on Apache Spark\*

- No changes to the existing Hadoop/Spark clusters needed



# BigDL Features

## Tensor

- ND-array data structure
- Generic data type
- Rich and fast math operations (powered by Intel MKL)

## Layers

- 113 layers (Conv, Pooling, FC...)

## Criterion

- 23 criterions

## Optimization

- SGD, Adagrad, LBFGS
- **Community contribution(PASA Lab):** Adam, Adadelata, RMSprop, Adamx

# 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

## 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

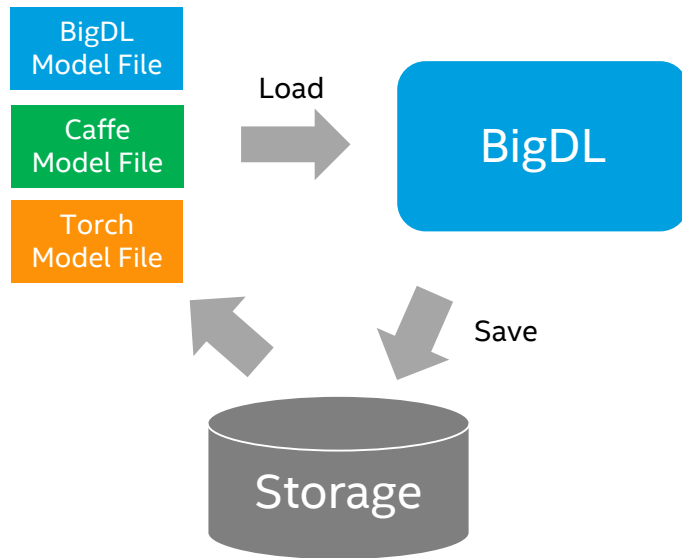
```
val model = Sequential()  
  .add(SpatialConvolution(3, 64, 11, 11, 4, 4, 2, 2, 1))  
  .add(ReLU(true))  
  .add(SpatialMaxPooling(3, 3, 2, 2))  
  .add(SpatialConvolution(64, 192, 5, 5, 1, 1, 2, 2))  
  .add(ReLU(true))  
  .add(SpatialMaxPooling(3, 3, 2, 2))  
  .add(SpatialConvolution(192, 384, 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())
```

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

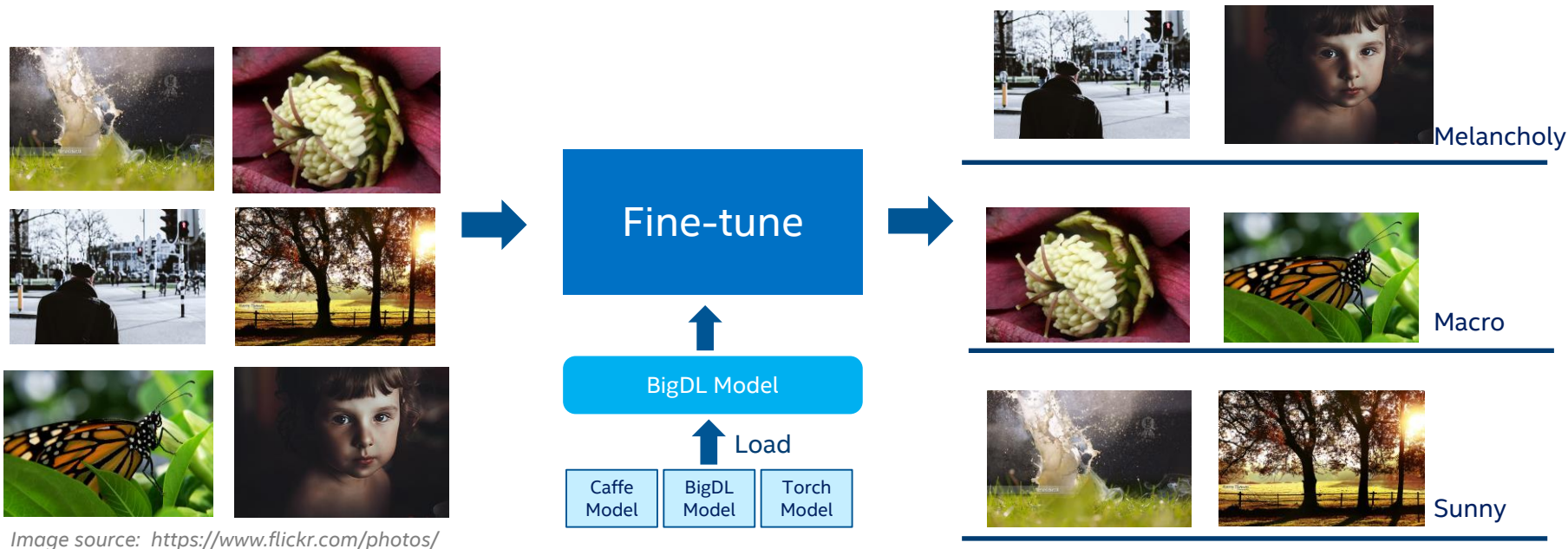
```
optimizer  
  .setState(state)  
  .setValidation(Trigger.severalIteration(620),  
    valSet, Array(new Top1Accuracy[Float], new Top5Accuracy[Float]))  
  .setEndWhen(Trigger.maxIteration(62000))  
  .optimize()
```

# 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



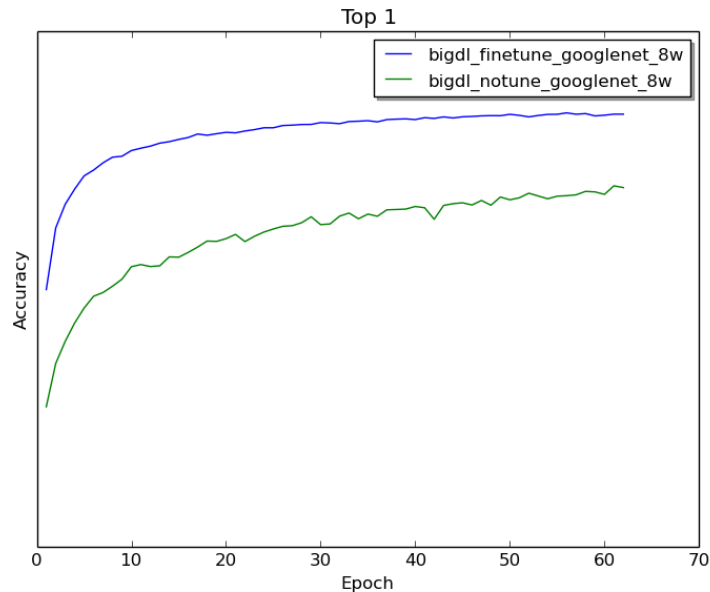
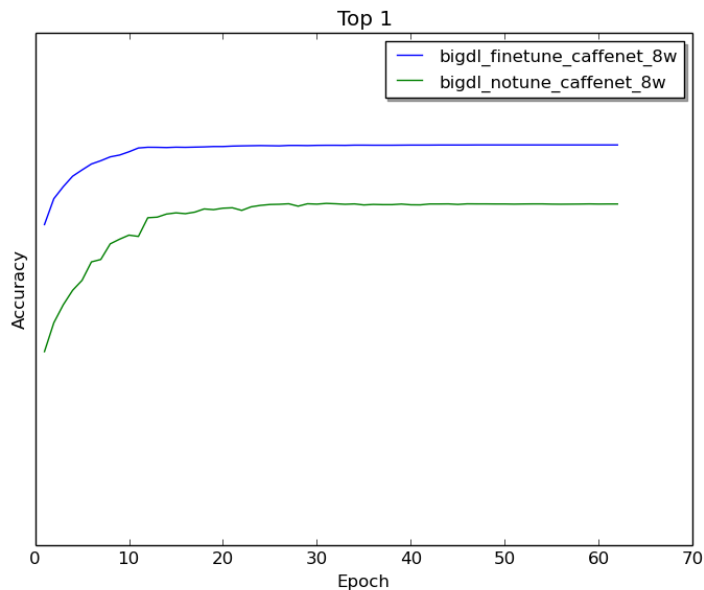
# BigDL Features



- Train on different dataset based on pre-trained model
- Predict image style instead of type
- Save training time and improve accuracy

# BigDL Features

## Fine-tune Caffe/Torch Model on Spark



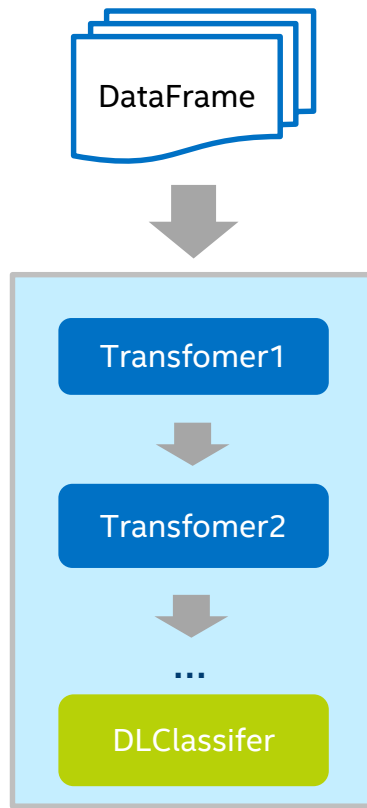
Accuracy increases 10% and converge time decreases.



# BigDL Features

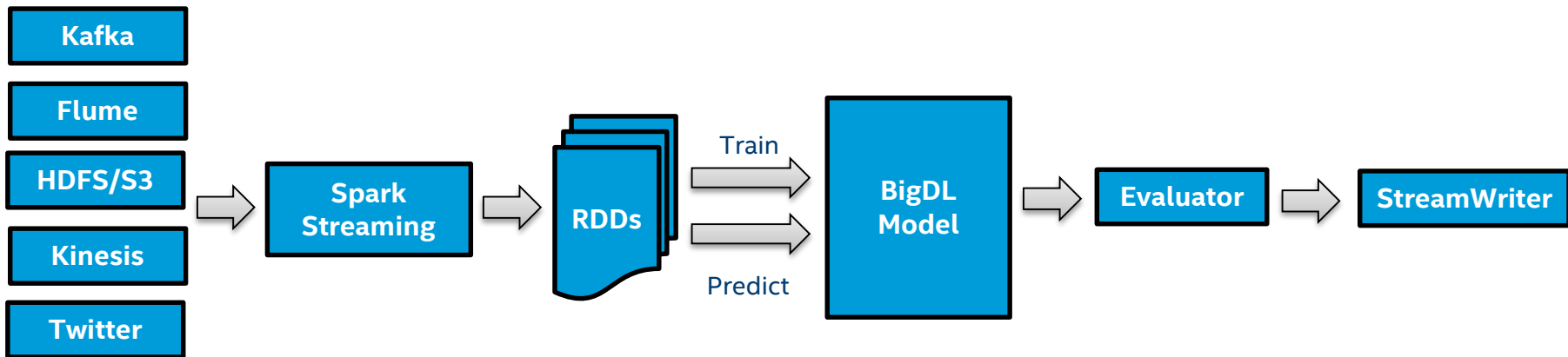
## Integrate with Spark-ML Pipeline

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



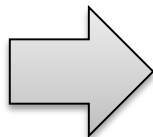
# BigDL Features

Integrations with Spark Streaming for runtime training and prediction



# BigDL Features

Tight Integrations with Spark SQL, DataFrames and Structured Streaming



```
df.select($"image")  
  .withColumn(  
    "image_type",  
    ImgClassifier("image"))  
  .filter($"image_type" == 'dog')  
  .show()
```

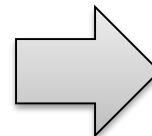


Image classification on ImageNet(<http://www.image-net.org>)

# Python API Support

Based on PySpark, **Python API** in BigDL allows use of existing Python libs:

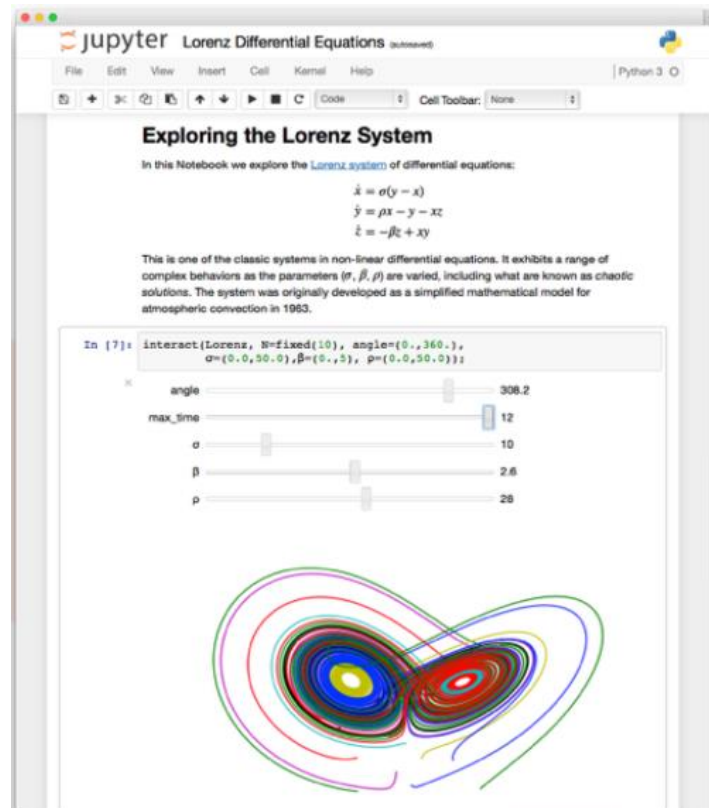
- Numpy
- Scipy
- Pandas
- Scikit-learn
- NLTK
- Matplotlib
- ...

```
train_data = get_minst("train").map(  
    normalizer(mnist.TRAIN_MEAN, mnist.TRAIN_STD))  
test_data = get_minst("test").map(  
    normalizer(mnist.TEST_MEAN, mnist.TEST_STD))  
state = {"batchSize": int(options.batchSize),  
        "learningRate": 0.01,  
        "learningRateDecay": 0.0002}  
optimizer = Optimizer(  
    model=build_model(10),  
    training_rdd=train_data,  
    criterion=ClassNLLCriterion(),  
    optim_method="SGD",  
    state=state,  
    end_trigger=MaxEpoch(100))  
optimizer.setvalidation(  
    batch_size=32,  
    val_rdd=test_data,  
    trigger=EveryEpoch(),  
    val_method=["top1"]  
)  
optimizer.setcheckpoint(EveryEpoch(), "/tmp/lenet5/")  
trained_model = optimizer.optimize()
```

# Working with Jupyter Notebook

## Running BigDL applications directly in Jupyter notebooks

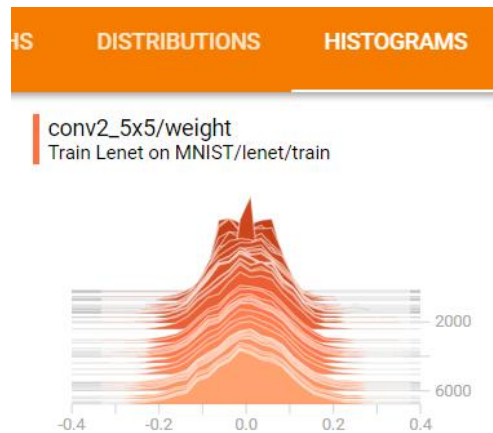
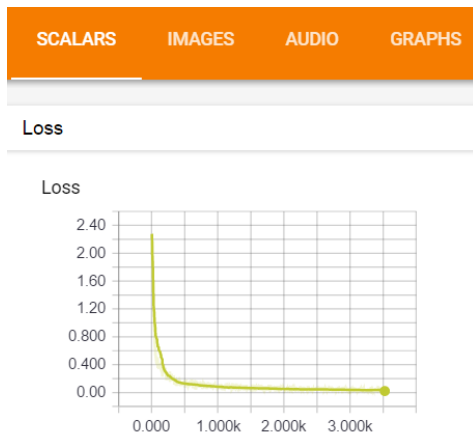
- Share and Reproduce
  - Notebooks can be shared with others
  - Easy to reproduce and track
- Rich Content
  - Texts, images, videos, LaTeX and JavaScript.
  - Code can also produce rich contents.
- Rich toolbox
  - Apache Spark, from Python, R and Scala.
  - Pandas, scikit-learn, ggplot2, dplyr, etc.



# Visualization for Learning

## BigDL integration with TensorBoard

- TensorBoard is a suite of web applications for inspecting and understanding your deep learning applications from Google



# 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

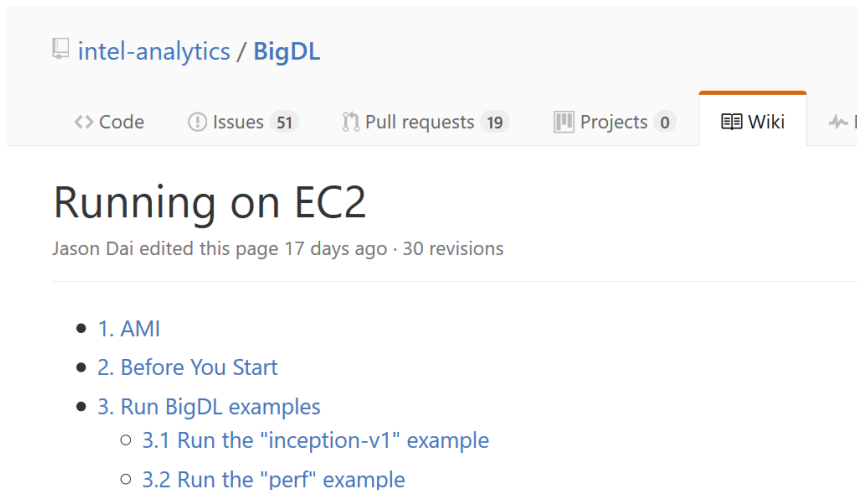
BigDL provides many popular [neural network models](#) and [deep learning examples](#) for Apache Spark, including:

- 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>



The screenshot shows the GitHub interface for the repository 'intel-analytics / BigDL'. The navigation bar includes links for Code, Issues (51), Pull requests (19), Projects (0), and Wiki (selected). The main heading is 'Running on EC2', with a note that 'Jason Dai edited this page 17 days ago · 30 revisions'. Below the heading is a bulleted list of steps:

- 1. AMI
- 2. Before You Start
- 3. Run BigDL examples
  - 3.1 Run the "inception-v1" example
  - 3.2 Run the "perf" example



# BigDL Features

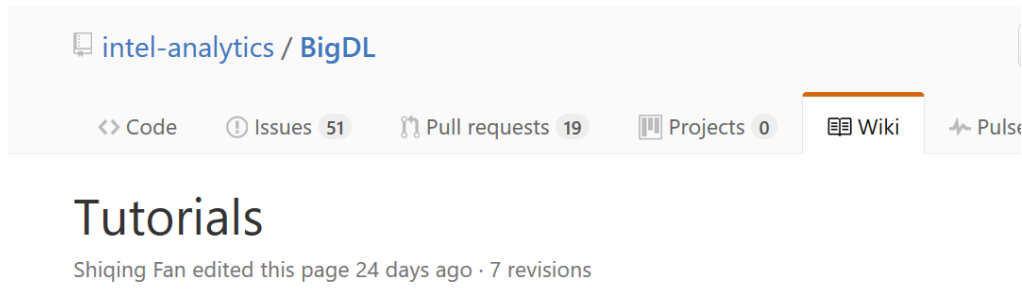
- Single node Xeon performance
  - Benchmarked best on Xeon E5-26XX v3 or E5-26XX v4
  - Orders of magnitude speedup vs. out-of-box open source Caffe, Torch or TensorFlow
- Scaling-out
  - Efficiently scale out to several 10s~100s of Xeon servers on Spark

For more complete information about performance and benchmark results, visit [www.intel.com/benchmarks](http://www.intel.com/benchmarks)

# BigDL Features

Start with tutorials

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



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 Tutorials

- “Intel’s BigDL on Databricks”

<https://databricks.com/blog/2017/02/09/intels-bigdl-databricks.html>

- “How to use BigDL on Apache Spark for Azure HDInsight”

<https://blogs.msdn.microsoft.com/azuredatalake/2017/03/17/how-to-use-bigdl-on-apache-spark-for-azure-hdinsight/>

<https://azure.microsoft.com/en-us/blog/use-bigdl-on-hdinsight-spark-for-distributed-deep-learning/>

- Jupyter Version 0.1 tutorial (very technical):

[https://nbviewer.jupyter.org/github/intel-analytics/BigDL/blob/branch-0.1/pyspark/dl/example/tutorial/simple\\_text\\_classification/text\\_classification.ipynb](https://nbviewer.jupyter.org/github/intel-analytics/BigDL/blob/branch-0.1/pyspark/dl/example/tutorial/simple_text_classification/text_classification.ipynb)

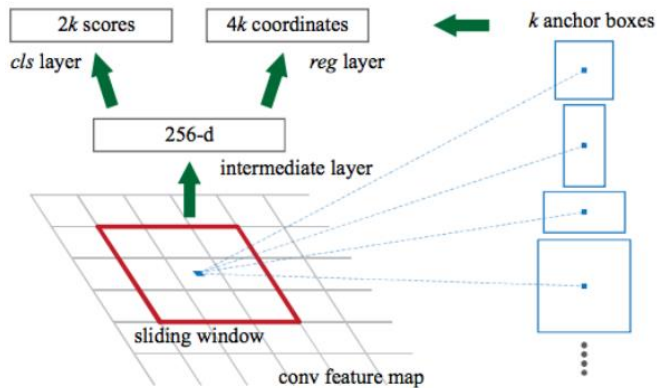
- Blog: BigDL on Ali EMR cloud:

<https://yq.aliyun.com/articles/73347>

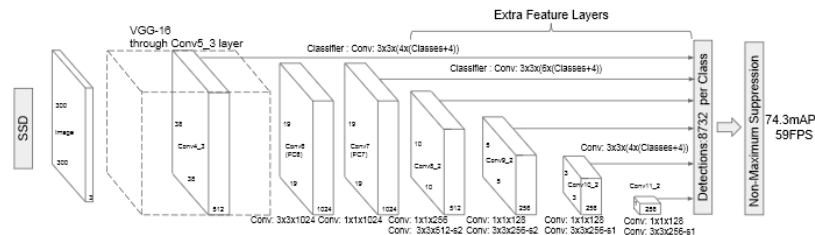
# BIGDL – USE CASE

# Object Detection

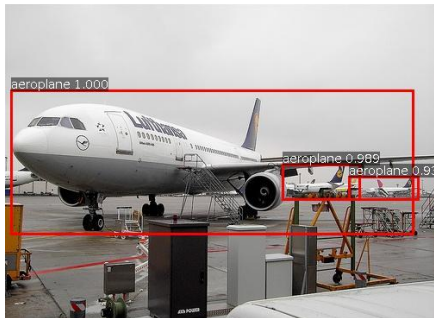
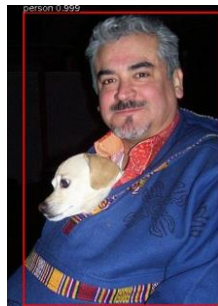
## Faster-RCNN



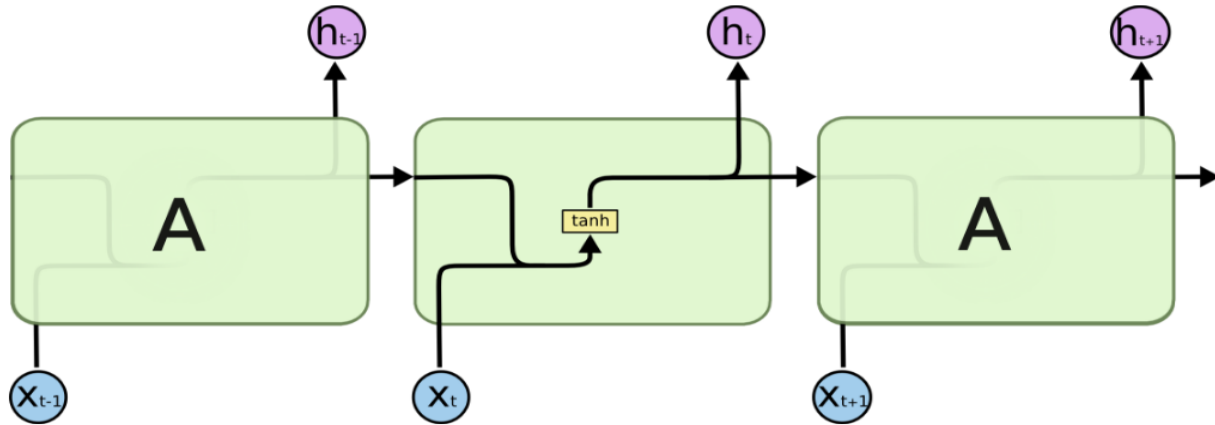
## SSD: Single Shot MultiBox Detector



# Object Detection on PASCAL(<http://host.robots.ox.ac.uk/pascal/VOC/>)



# Language Model - RNN



The repeating module in a standard RNN contains a single layer.

Source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# Learn from Shakespeare Poems

Output of RNN:

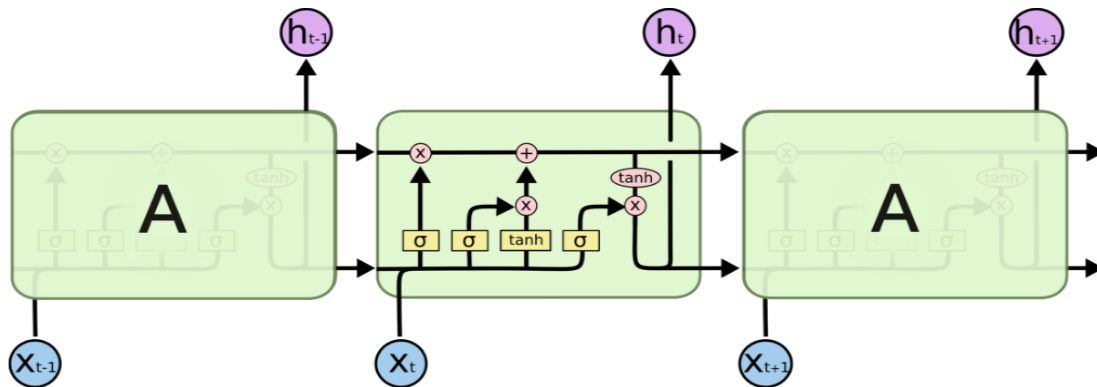
Long live the King . The King and Queen , and the Strange of the Veils of the rhapsodic . and grapple, and the entreatments of the pressure .

Upon her head , and in the world ? `` Oh, the gods ! O Jove ! To whom the king :  
`` O friends !

Her hair, nor loose ! If , my lord , and the groundlings of the skies . jocund and Tasso in the Staggering of the Mankind . and



# More RNN Support: LSTM



The repeating module in an LSTM contains four interacting layers.

Source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

BigDL also supports LSTM variants such as GRU and LSTM with peepholes

# Fraud Transaction Detection

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

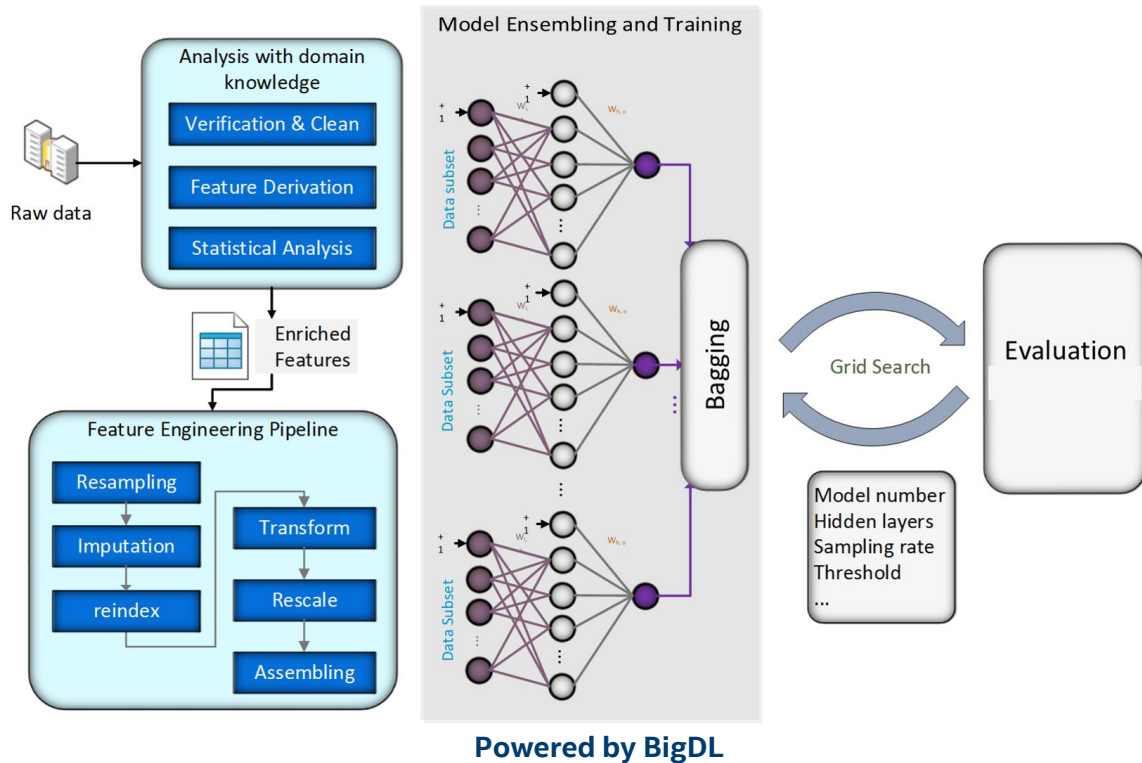
## ML solution challenge

- 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 tuning



# 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

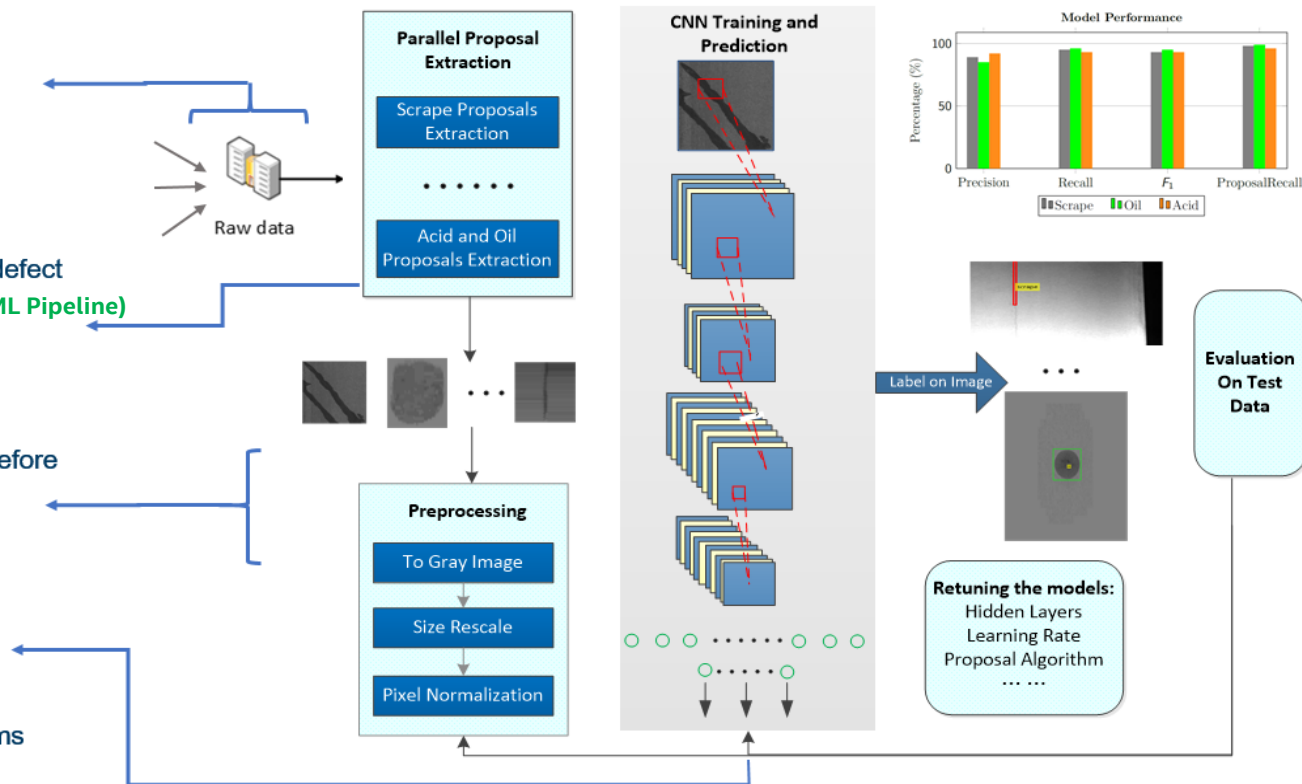
- Extract proposals for each defect
- Parallel pipeline (**KeyStone ML Pipeline**)
- Running on **Spark**

## Preprocessing

- Preprocess the proposals before model training or testing

## Model training pipeline

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



# What we're working on

- End2End voice recognition (Deep Speech)
- Large scale object detection(SSD, Faster-RCNN)
- Neural network based recommendation(MLP MF, RNN)
- TensorFlow model read/write
- Compactable with Keras
- Reinforcement Learning
- Recursive NN / Tree LSTM
- ...

# BIGDL ON GITHUB

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

# Community

- Mail List

[bigdl-user-group+subscribe@googlegroups.com](mailto: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)

# Demo

# 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>

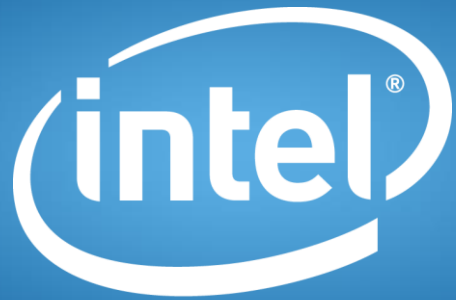
Intel, Quark, VTune, Xeon, Cilk, Atom, Look Inside and the Intel logo are trademarks of Intel Corporation in the United States and other countries.

\*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.



Software