Building Deep Learning Applications on Big Data Platform

*An Introduction to **Analytics Zoo** for Apache Spark and BigDL*

Jason Dai

CVPR 2018

Agenda

- Motivation (10 minutes)
  - Trends, real-world scenarios
- DL frameworks on Apache Spark (20 minutes)
  - BigDL, TensorFlowOnSpark, DL Pipelines, SparkNet
- Analytics Zoo for Spark and BigDL (15 minutes)
  - High level pipeline APIs, feature engineering, built-in models, reference use cases
- Analytics Zoo Examples (15 minutes)
  - Dogs vs. cats, object detection, TFNet
- Break (30 minutes)

Agenda

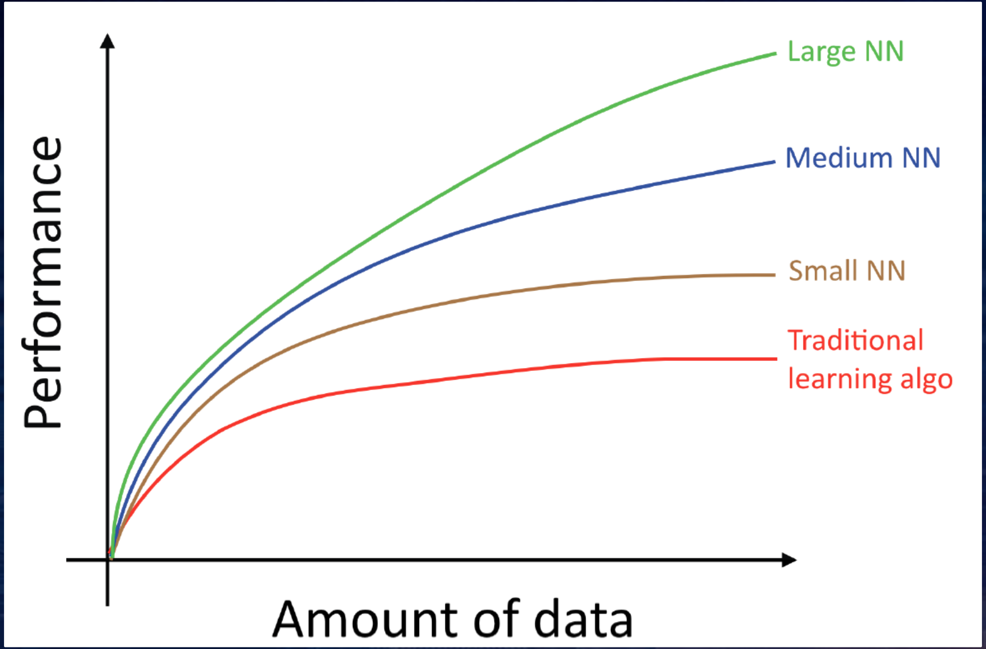
- **Distributed training in BigDL (30 minutes)**
  - Data parallel training, parameter synchronization, scaling & convergence, task scheduling, etc.
- **Advanced applications (15 minutes)**
  - Variational autoencoder, movie recommendations
- **Real-world applications (30 minutes)**
  - Object detection and image feature extraction at *JD.com*
  - Image similarity based house recommendation for *MLSlistings*
  - Transfer learning based image classifications for *World Bank*
  - Fraud detection for payment transactions for *UnionPay*
- **Conclusion and Q&A (15 minutes)**

Motivations

Technology and Industry Trends

Real World Scenarios

Trend #1: Data Scale Driving Deep Learning Process



"Machine Learning Yearning", Andrew Ng, 2016

Trend #2: Hadoop Becoming the Center of Data Gravity

Why an Enterprise Data Hub ?

- Single place for all enterprise data... (unedited hi-resolution history of everything)
- Reduces Application Integration Costs
  - Connect once to Hub (N vs N^2 connections)
- Lowest unit cost data processing & storage platform
  - Open source S/W on commodity H/W (reliability in S/W not H/W)
  - Can mix H/W vendors means every expansion is competitively tendered
- Fast Standardised Provision
  - No custom design task, re-use Active Directory account/password processes
  - Reduces Shadow IT
- Secure (audited, E2E visibility/auditing, encryption)
  - Eliminate need for one off extracts

#StrataHadoop

Strata + Hadoop WORLD

Phillip Radley, BT Group
Strata + Hadoop World 2016 San Jose

Everyone is building Data Lakes

- Universal data acquisition makes all big data analytics and reporting easier
- Hadoop provides a scalable storage with HDFS
- How will we scale consumption and curation of all this data?

7

Matthew Glickman, Goldman Sachs
Spark Summit East 2015

Trend #3: Real-World ML/DL Systems Are Complex Big Data Analytics Pipelines

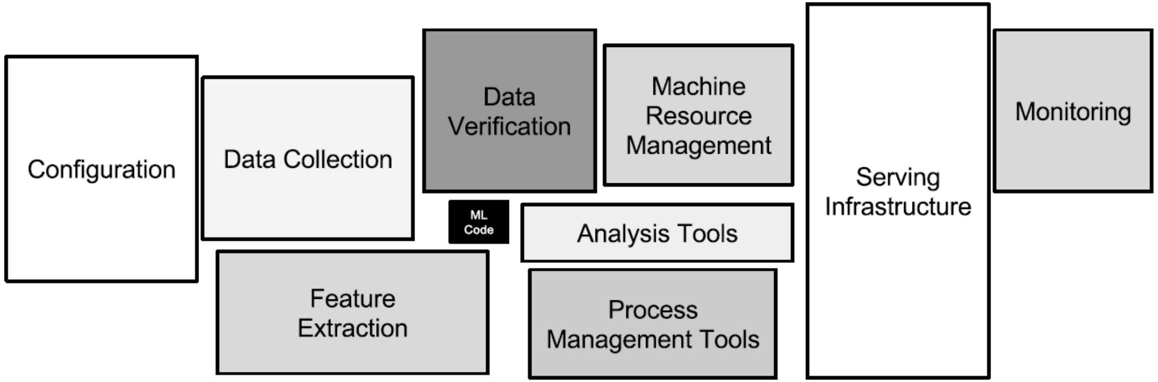


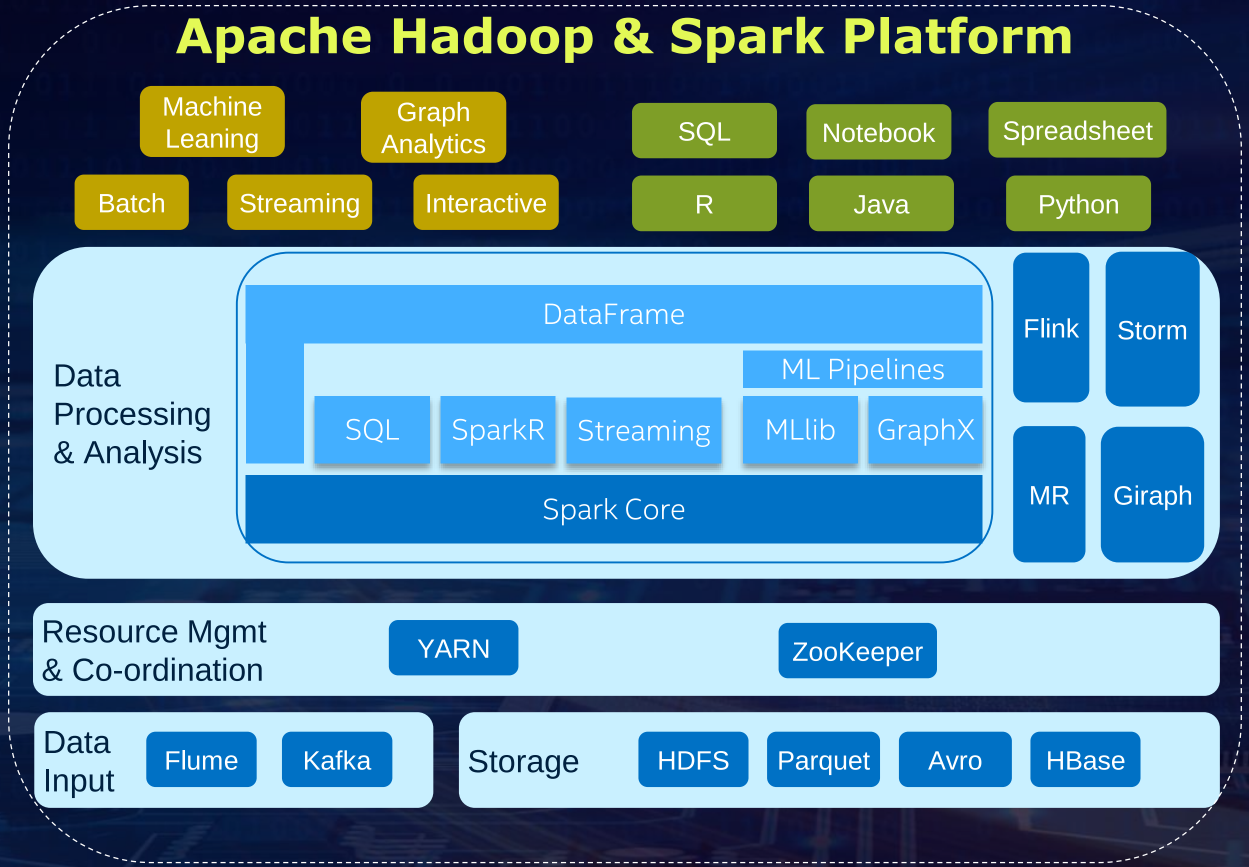
Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

“Hidden Technical Debt in Machine Learning Systems”,
Sculley et al., Google, NIPS 2015 Paper

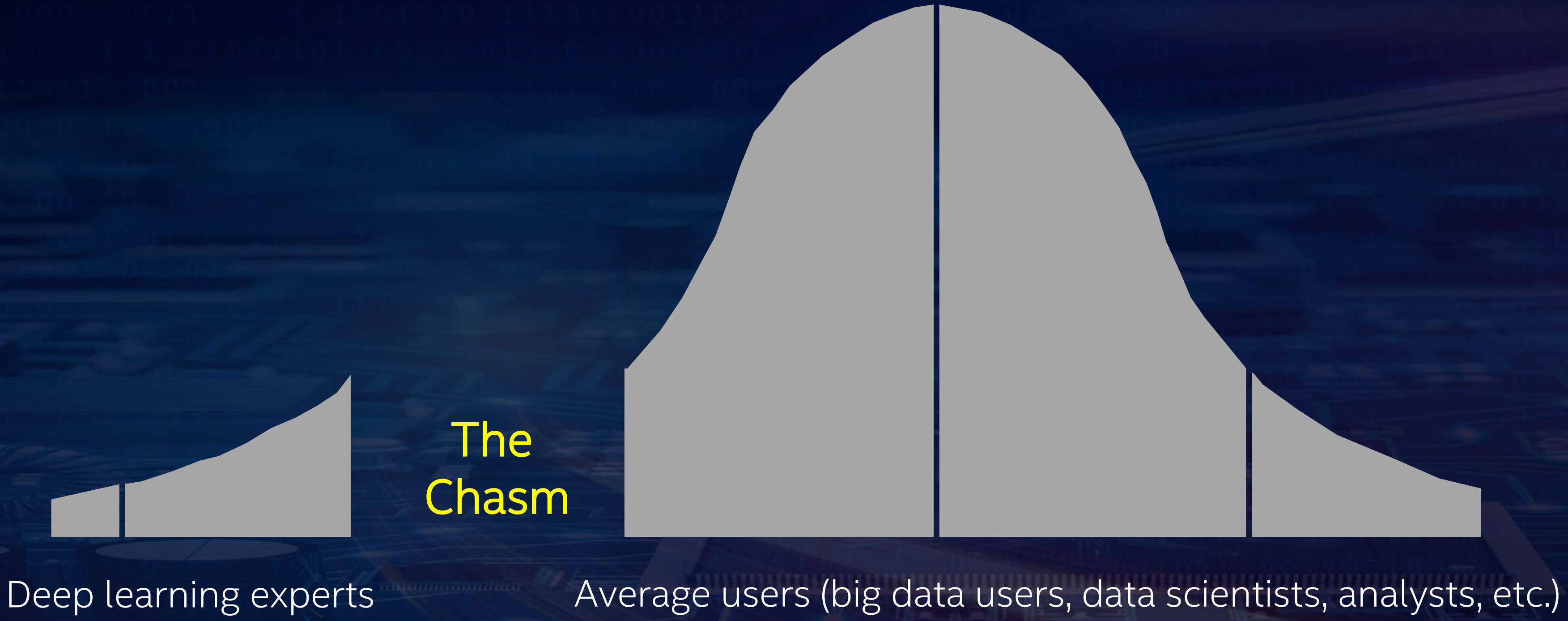
Trend #4: Unified Big Data Platform Driving Analytics & Data Science

Ion Stoica, UC Berkeley, Spark Summit 2013 Keynote

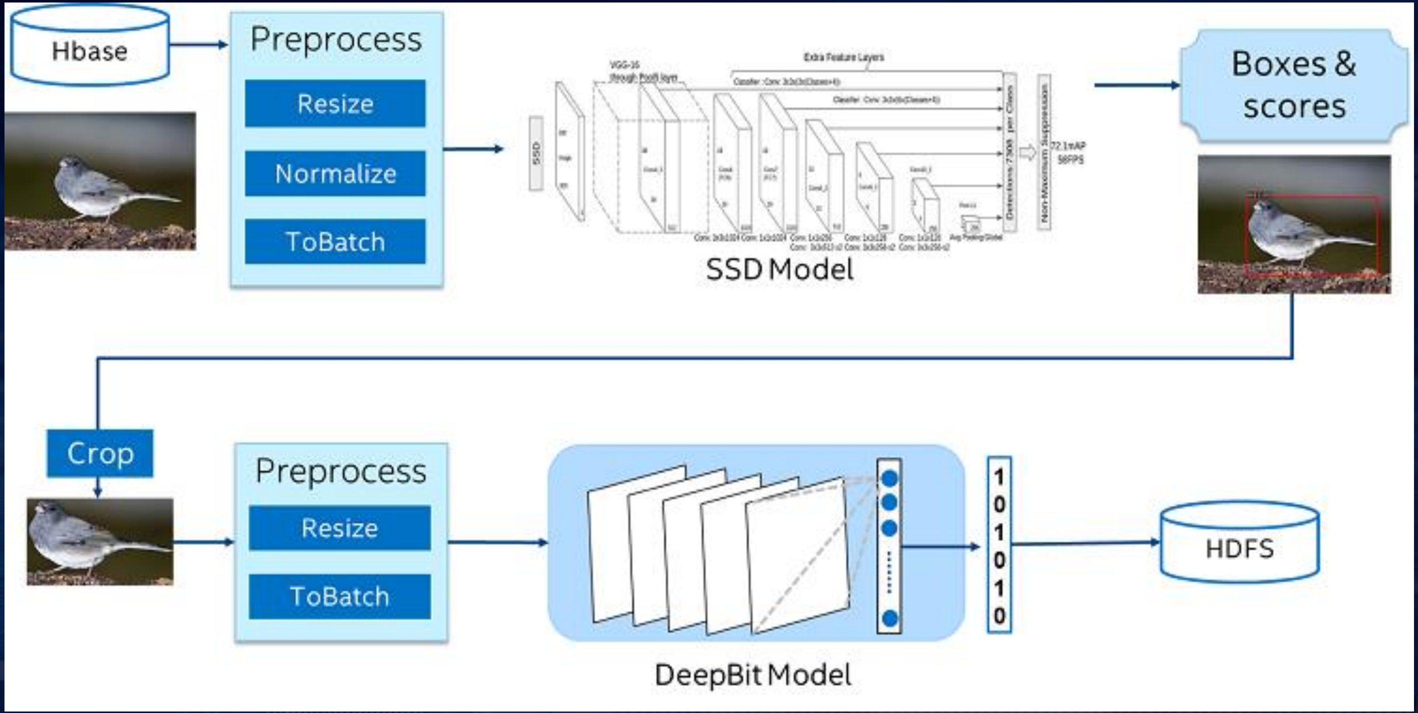
Unified Big Data Analytics Platform



Chasm b/w Deep Learning and Big Data Communities



Large-Scale Image Recognition at JD.com



Bridging the Chasm

Make deep learning more accessible to big data and data science communities

- Continue the use of familiar SW tools and HW infrastructure to build deep learning applications
- Analyze "big data" using deep learning on the same Hadoop/Spark cluster where the data are stored
- Add deep learning functionalities to large-scale big data programs and/or workflow
- Leverage existing Hadoop/Spark clusters to run deep learning applications
    - Shared, monitored and managed with other workloads (*e.g., ETL, data warehouse, feature engineering, traditional ML, graph analytics, etc.*) in a dynamic and elastic fashion

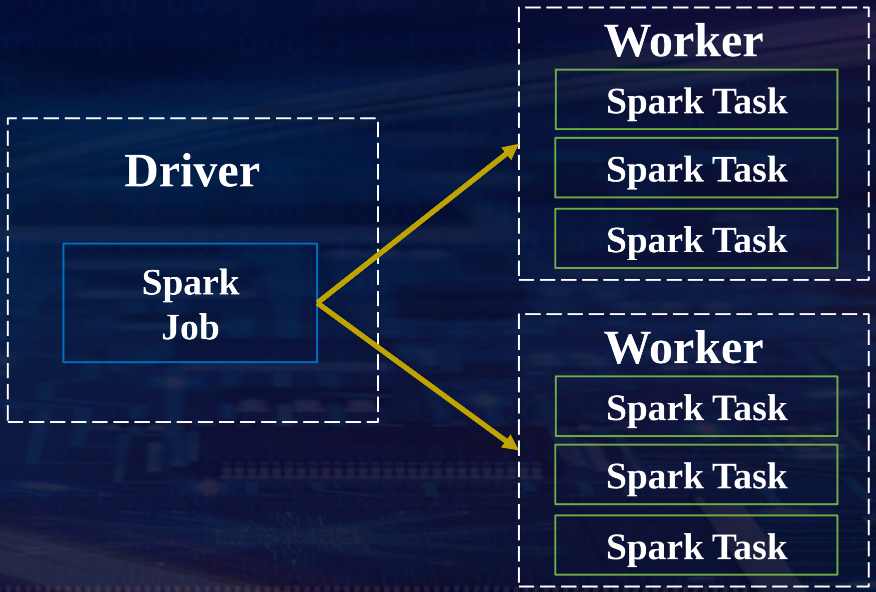
DL Frameworks on Apache Spark

BigDL, DL Pipelines for Spark, TensorflowOnSpark, SparkNet, etc.

Apache Spark

Low Latency, Distributed Data Processing Framework

- A Spark cluster consists of a single *driver* node and multiple *worker* nodes
- A Spark *job* contains many Spark *tasks*, each working on a data *partition*
- Driver is responsible for scheduling and dispatching the tasks to workers, which runs the actual Spark tasks

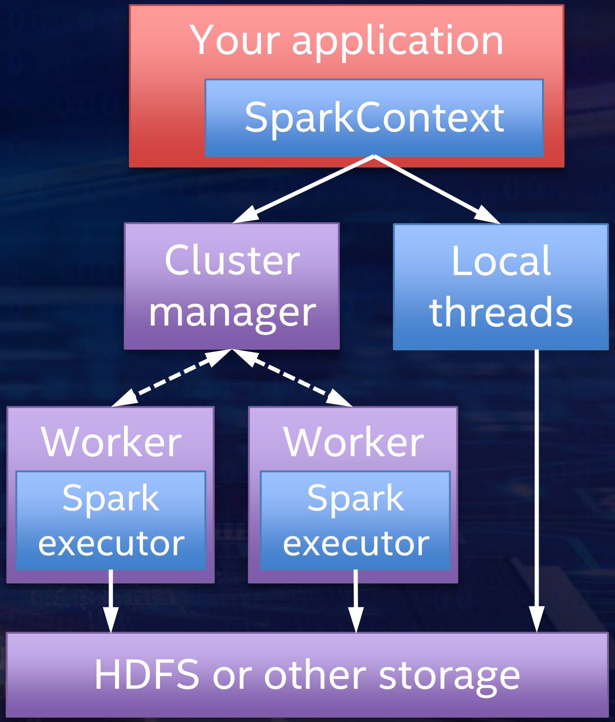


https://spark.apache.org

Apache Spark

Spark Program

- Spark runs as a library in your program (1 instance per app)
- Runs tasks locally or on cluster
  - K8s, YARN, Mesos or standalone mode
- Accesses storage systems via Hadoop InputFormat API
  - Can use HBase, HDFS, S3, …

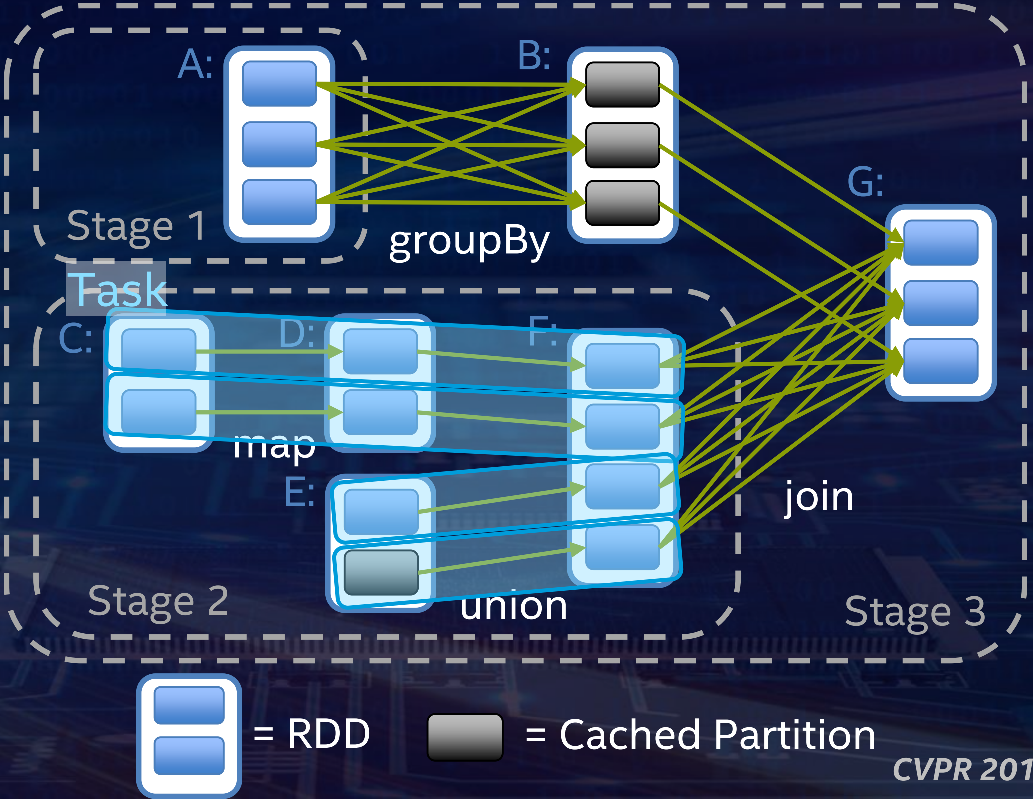


Source: “Parallel programming with Spark”, Matei Zaharia, AMPCamp 3

Apache Spark

Distributed Task Execution

- General task graphs
- Automatically pipelines functions
- Data locality aware
- Partitioning aware to avoid shuffles

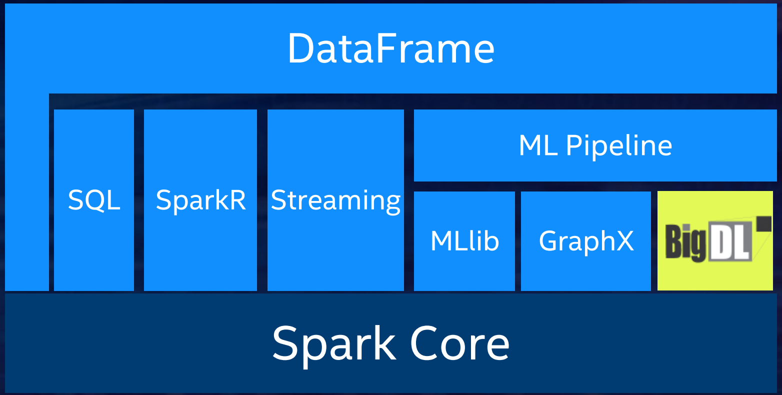


Source: "Parallel programming with Spark", Matei Zaharia, AMPCamp 3

BigDL

Bringing Deep Learning To Big Data Platform

- Distributed deep learning framework for Apache Spark*
- Make deep learning more accessible to big data users and data scientists
  - Write deep learning applications as ***standard Spark programs***
  - Run on existing Spark/Hadoop clusters (***no changes needed***)
- Feature parity with popular deep learning frameworks
  - E.g., Caffe, Torch, Tensorflow, etc.
- High performance (on CPU)
  - Powered by Intel MKL and multi-threaded programming
- Efficient scale-out
  - Leveraging Spark for distributed training & inference



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

https://bigdl-project.github.io/

BigDL Run as Standard Spark Programs

Standard Spark jobs

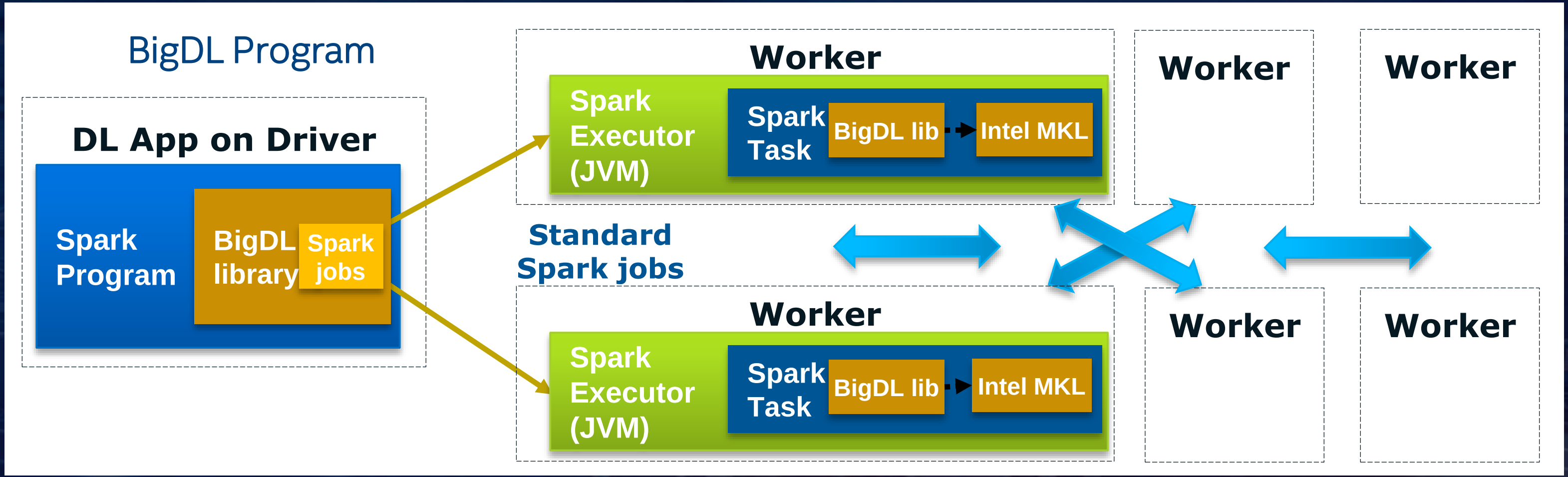
- No changes to the Spark or Hadoop clusters needed

Iterative

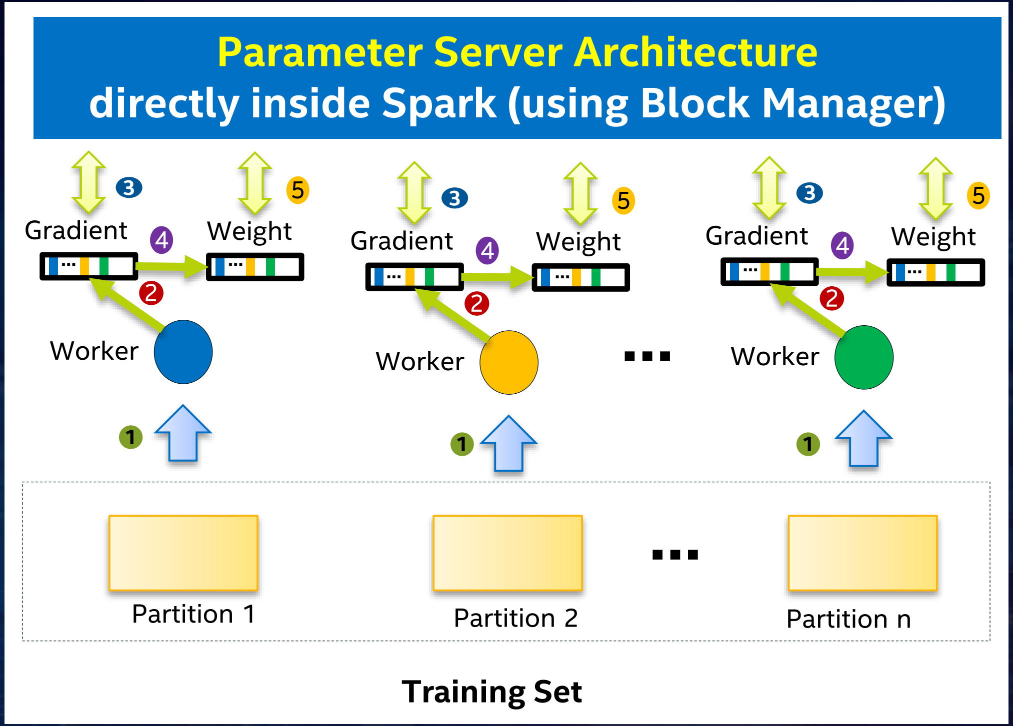
- Each iteration of the training runs as a Spark job

Data parallel

- Each Spark task runs the same model on a subset of the data (batch)



Distributed Training in BigDL

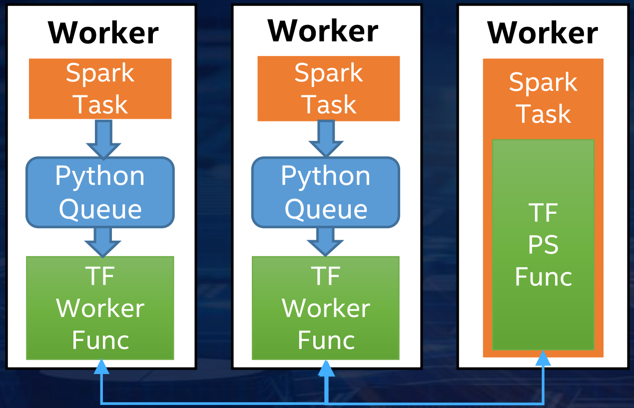


Peer-2-Peer All-Reduce synchronization

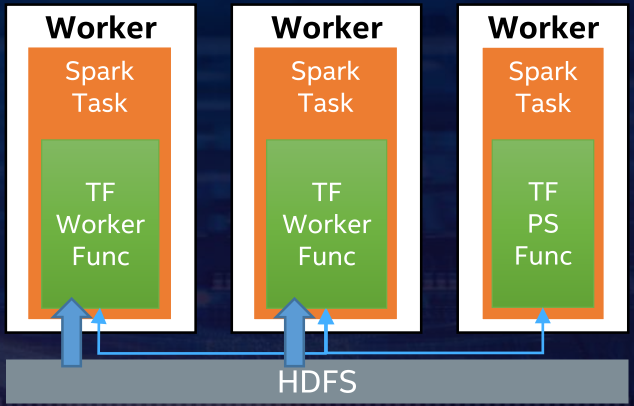
TensoflowOnSpark

Standalone TF jobs on Spark cluster

- Use Spark as the orchestration layer to allocate resources
- Launch distributed TensorFlow job on the allocated resources
- Coarse-grained integration of two independent frameworks
  - Memory overheads, no gang scheduling, limited interactions with data pipelines, etc.



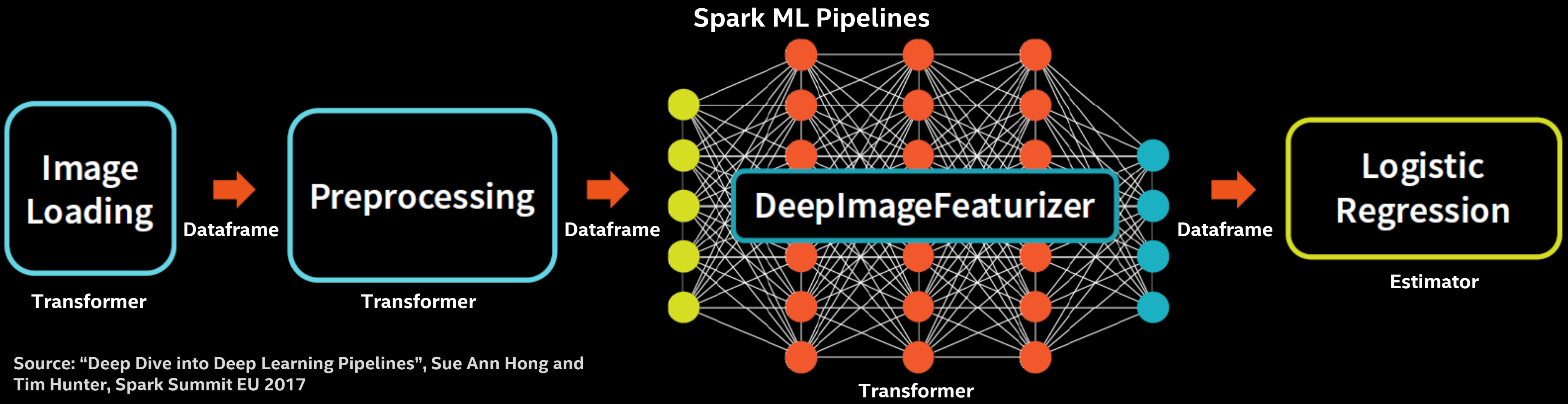
feed_dict: TF worker func runs as independent process in background, reading data from Python queue



queue_runner: direct HDFS access from TF work func

https://github.com/yahoo/TensorFlowOnSpark

DL Pipelines for Spark



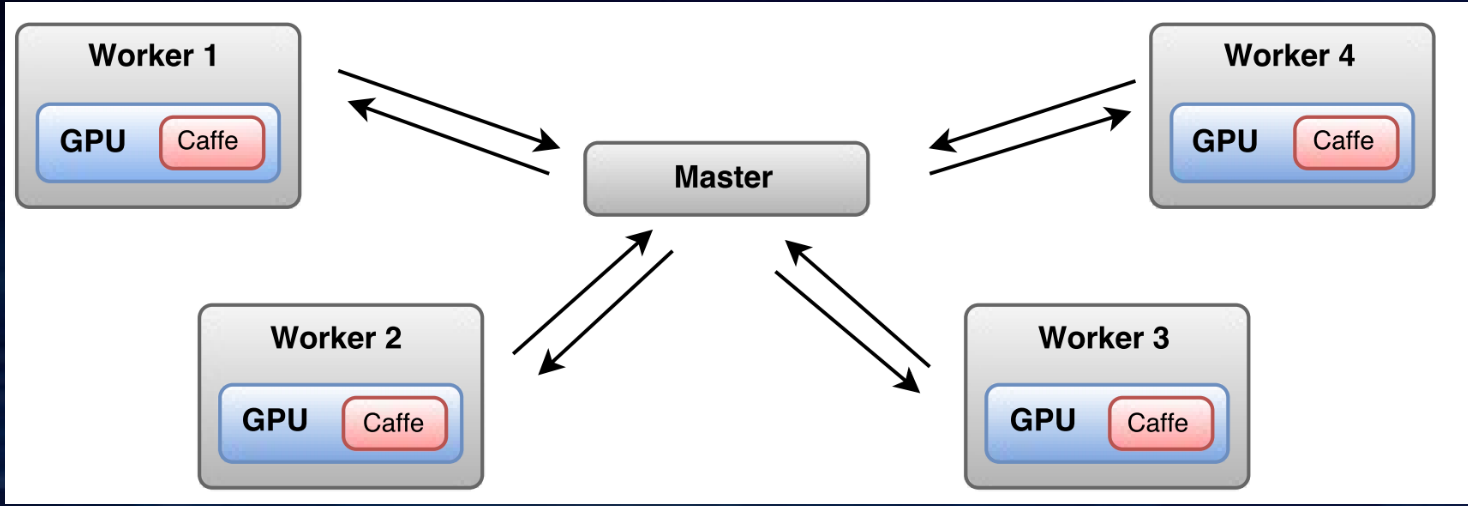
Source: "Deep Dive into Deep Learning Pipelines", Sue Ann Hong and Tim Hunter, Spark Summit EU 2017

Load existing TF or Keras models in Spark ML pipelines

- Load into transformer: inference only
- Load into estimator: single node training/tuning only

https://github.com/databricks/spark-deep-learning

SparkNet



Source: "SparkNet: Training Deep Networks in Spark", Philipp Moritz, et al., ICLR 2016

Distributed DL training by running Caffe in each worker

- Asynchronous parameter synchronization through master (driver) mode
  - Very inefficient (~20 seconds with just 5 workers)

Analytics Zoo

Analytics + AI Platform for Spark and BigDL

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

Analytics Zoo

Build and Productionize Deep Learning Apps for Big Data at Scale

Reference Use Cases	• Anomaly detection • Sentiment analysis • Fraud detection • Chatbot, sequence prediction, etc.
Built-In Deep Learning Models	• Image classification • Object detection • Text classification • Recommendations • Sequence-to-sequence, GAN, etc.
Feature Engineering	Feature transformations for • Image, text, 3D imaging, time series, speech, etc.
High-Level Pipeline APIs	• Native deep learning support in Spark DataFrames and ML Pipelines • Autograd, Keras and transfer learning APIs for model definition • Model serving API for model serving/inference pipelines
Backbends	Spark, BigDL, TensorFlow, etc.

https://github.com/intel-analytics/analytics-zoo/ https://analytics-zoo.github.io/

Analytics Zoo

Build end-to-end deep learning applications for big data

- E2E analytics + AI pipelines (natively in Spark DataFrames and ML Pipelines) using ***nnframes***
- Flexible model definition using ***autograd, Keras-style & transfer learning APIs***
- Data preprocessing using built-in ***feature engineering operations***
- Out-of-the-box solutions for a variety of problem types using ***built-in deep learning models and reference use cases***

Productionize deep learning applications for big data at scale

- Serving models in web services and big data frameworks (e.g., Storm or Kafka) using POJO ***model serving APIs***
- Large-scale distributed TensorFlow model inference using ***TFNet***

Analytics Zoo

Build end-to-end deep learning applications for big data

- **E2E analytics + AI pipelines (natively in Spark DataFrames and ML Pipelines) using *nnframes***
- *Flexible model definition using autograd, Keras-style & transfer learning APIs*
- *Data preprocessing using built-in feature engineering operations*
- *Out-of-the-box solutions for a variety of problem types using built-in deep learning models and reference use cases*

Productionize deep learning applications at scale for big data

- *Serving models in web services and big data frameworks (e.g., Storm or Kafka) using POJO model serving APIs*
- *Large-scale distributed TensorFlow model inference using TFNet*

nnframes

Native DL support in Spark DataFrames and ML Pipelines

1. **Initialize *NNContext* and load images into *DataFrames* using `NNImageReader`**

```
from zoo.common.nncontext import *
from zoo.pipeline.nnframes import *
sc = init_nncontext()
imageDF = NNImageReader.readImages(image_path, sc)
```

2. **Process loaded data using `DataFrame` transformations**

```
getName = udf(lambda row: ...)
df = imageDF.withColumn("name", getName(col("image")))
```

3. **Processing image using built-in *feature engineering* operations**

```
from zoo.feature.image import *
transformer = ChainedPreprocessing(
        [RowToImageFeature(), ImageChannelNormalize(123.0, 117.0, 104.0),
         ImageMatToTensor(), ImageFeatureToTensor()])
```

nnframes

Native DL support in Spark DataFrames and ML Pipelines

4. **Define model using *Keras-style API***

```
from zoo.pipeline.api.keras.layers import *
from zoo.pipeline.api.keras.models import *
model = Sequential()
          .add(Convolution2D(32, 3, 3, activation='relu', input_shape=(1, 28, 28))) \
          .add(MaxPooling2D(pool_size=(2, 2))) \
          .add(Flatten()).add(Dense(10, activation='softmax')))
```

5. **Train model using *Spark ML Pipelines***

```
Estimater = NNEstimater(model, CrossEntropyCriterion(), transformer) \
                .setLearningRate(0.003).setBatchSize(40).setMaxEpoch(1) \
                .setFeaturesCol("image").setCachingSample(False)
nnModel = estimater.fit(df)
```

Analytics Zoo

Build end-to-end deep learning applications for big data

- *E2E analytics + AI pipelines (natively in Spark DataFrames and ML Pipelines) using nnframes*
- **Flexible model definition using *autograd, Keras-style & transfer learning APIs***
- *Data preprocessing using built-in feature engineering operations*
- *Out-of-the-box solutions for a variety of problem types using built-in deep learning models and reference use cases*

Productionize deep learning applications at scale for big data

- *Serving models in web services and big data frameworks (e.g., Storm or Kafka) using POJO model serving APIs*
- *Large-scale distributed TensorFlow model inference using TFNet*

Autograd, Keras &Transfer Learning APIs

1. Use transfer learning APIs to
   - Load an existing Caffe model
   - Remove last few layers
   - Freeze first few layers
   - Append a few layers

```
from zoo.pipeline.api.net import *
full_model = Net.load_caffe(def_path, model_path)
# Remove layers after pool5
model = full_model.new_graph(outputs=["pool5"]).to_keras())
# freeze layers from input to res4f inclusive
model.freeze_up_to(["res4f"])
# append a few layers
image = Input(name="input", shape=(3, 224, 224))
resnet = model.to_keras()(image)
resnet50 = Flatten()(resnet)
```

Build Siamese Network Using Transfer Learning

Autograd, Keras &Transfer Learning APIs

2. Use *autograd and Keras-style* APIs to build the Siamese Network

```
import zoo.pipeline.api.autograd as A
from zoo.pipeline.api.keras.layers import *
from zoo.pipeline.api.keras.models import *

input = Input(shape=[2, 3, 226, 226])
features = TimeDistributed(layer=resnet50)(input)
f1 = features.index_select(1, 0) #image1
f2 = features.index_select(1, 1) #image2
diff = A.abs(f1 - f2)
fc = Dense(1)(diff)
output = Activation("sigmoid")(fc)
model = Model(input, output)
```

Build Siamese Network Using Transfer Learning

Analytics Zoo

Build end-to-end deep learning applications for big data

- *E2E analytics + AI pipelines (natively in Spark DataFrames and ML Pipelines) using nnframes*
- *Flexible model definition using autograd, Keras-style & transfer learning APIs*
- **Data preprocessing using built-in *feature engineering operations***
- *Out-of-the-box solutions for a variety of problem types using built-in deep learning models and reference use cases*

Productionize deep learning applications at scale for big data

- *Serving models in web services and big data frameworks (e.g., Storm or Kafka) using POJO model serving APIs*
- *Large-scale distributed TensorFlow model inference using TFNet*

Feature Engineering

1. **Read images into local or distributed *`ImageSet`***

```
from zoo.common.nncontext import *
from zoo.feature.image import *
spark = init_nncontext()
local_image_set = ImageSet.read(image_path)
distributed_image_set = ImageSet.read(image_path, spark, 2)
```

2. **Image augmentations using built-in *`ImageProcessing`* operations**

```
transformer = ChainedPreprocessing([ImageBytesToMat(),
                    ImageColorJitter(),
                    ImageExpand(max_expand_ratio=2.0),
                    ImageResize(300, 300, -1),
                    ImageHFlip()])
new_local_image_set = transformer(local_image_set)
new_distributed_image_set = transformer(distributed_image_set)
```

Image Augmentations Using Built-in Image Transformations (w/ OpenCV on Spark)

Analytics Zoo

Build end-to-end deep learning applications for big data

- *E2E analytics + AI pipelines (natively in Spark DataFrames and ML Pipelines) using nnframes*
- *Flexible model definition using autograd, Keras-style & transfer learning APIs*
- *Data preprocessing using built-in feature engineering operations*
- **Out-of-the-box solutions for a variety of problem types using *built-in deep learning models and reference use cases***

Productionize deep learning applications at scale for big data

- *Serving models in web services and big data frameworks (e.g., Storm or Kafka) using POJO model serving APIs*
- *Large-scale distributed TensorFlow model inference using TFNet*

Built-in Deep Learning Models

- *Object detection API*
  - High-level API and pretrained models (e.g., SSD, Faster-RCNN, etc.) for object detection
- *Image classification API*
  - High-level API and pretrained models (e.g., VGG, Inception, ResNet, MobileNet, etc.) for image classification
- *Text classification API*
  - High-level API and pre-defined models (using CNN, LSTM, etc.) for text classification
- *Recommendation API*
  - High-level API and pre-defined models (e.g., Neural Collaborative Filtering, Wide and Deep Learning, etc.) for recommendation

Object Detection API

1. **Load pretrained model in *Detection Model Zoo***

```
from zoo.common.nncontext import *
from zoo.models.image.objectdetection import *
spark = init_nncontext()
model = ObjectDetector.load_model(model_path)
```

2. **Off-the-shell inference using the loaded model**

```
image_set = ImageSet.read(img_path, spark)
output = model.predict_image_set(image_set)
```

3. **Visualize the results using utility methods**

```
config = model.get_config()
visualizer = Visualizer(config.label_map(), encoding="jpg")
visualized = visualizer(output).get_image(to_chw=False).collect()
```

Off-the-shell Inference Using Analytics Zoo Object Detection API

https://github.com/intel-analytics/analytics-zoo/tree/master/pyzoo/zoo/examples/objectdetection

Reference Use Cases

- *Anomaly Detection*
  - Using LSTM network to detect anomalies in time series data
- *Fraud Detection*
  - Using feed-forward neural network to detect frauds in credit card transaction data
- *Recommendation*
  - Use Analytics Zoo Recommendation API (i.e., Neural Collaborative Filtering, Wide and Deep Learning) for recommendations on data with explicit feedback.
- *Sentiment Analysis*
  - Sentiment analysis using neural network models (e.g. CNN, LSTM, GRU, Bi-LSTM)
- *Variational Autoencoder (VAE)*
  - Use VAE to generate faces and digital numbers

https://github.com/intel-analytics/analytics-zoo/tree/master/apps

Analytics Zoo

Build end-to-end deep learning applications for big data

- *E2E analytics + AI pipelines (natively in Spark DataFrames and ML Pipelines) using nnframes*
- *Flexible model definition using autograd, Keras-style & transfer learning APIs*
- *Data preprocessing using built-in feature engineering operations*
- *Out-of-the-box solutions for a variety of problem types using built-in deep learning models and reference use cases*

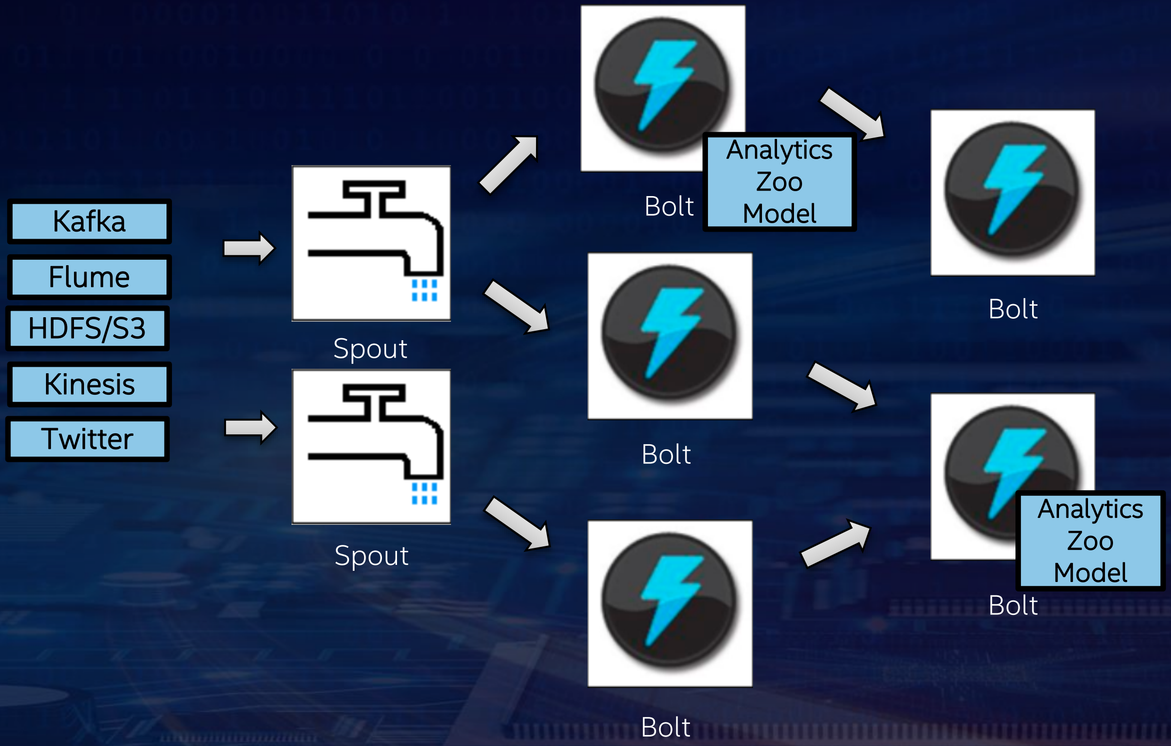
Productionize deep learning applications at scale for big data

- **Serving models in web services and big data frameworks (e.g., Storm or Kafka) using POJO *model serving APIs***
- *Large-scale distributed TensorFlow model inference using TFNet*

POJO Model Serving API

```
import com.intel.analytics.zoo.pipeline.inference.AbstractInferenceModel;

public class TextClassification extends AbstractInferenceModel {
  public RankerInferenceModel(int concurrentNum) {
    super(concurrentNum);
  }
  ...
}

public class ServingExample {
  public static void main(String[] args) throws IOException {
    TextClassification model = new TextClassification();
    model.load(modelPath, weightPath);

    texts = …
    List<JTensor> inputs = preprocess(texts);
    for (JTensor input : inputs) {
      List<Float> result = model.predict(input.getData(), input.getShape());
      ...
  }
}
```

Model Serving & Inference



Seamless integration in Web Services, Storm, Flink, Kafka, etc. (using POJO *local Java APIs*)

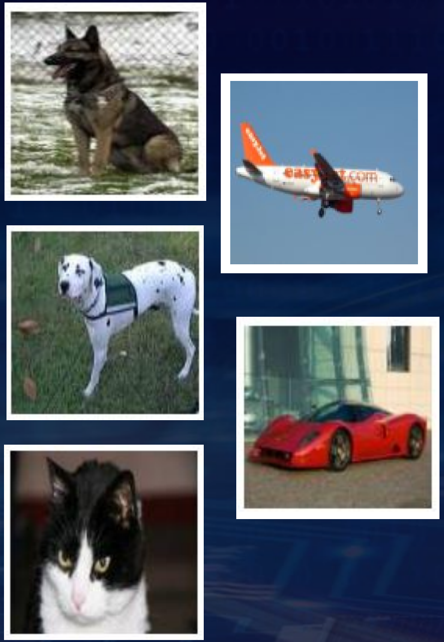
Model Serving in Spark DataFrames

Kafka | **File**

Data Frame
(Batch/Stream)

Analytics Zoo
UDF

Filtered Data
Frame
(Batch/Stream)



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

ImageNet dataset (http://www.image-net.org)

Seamless support of DL functionalities in Spark *SQL* queries, *Dataframe* operation and *stream* processing

Analytics Zoo

Build end-to-end deep learning applications for big data

- *E2E analytics + AI pipelines (natively in Spark DataFrames and ML Pipelines) using nnframes*
- *Flexible model definition using autograd, Keras-style & transfer learning APIs*
- *Data preprocessing using built-in feature engineering operations*
- *Out-of-the-box solutions for a variety of problem types using built-in deep learning models and reference use cases*

Productionize deep learning applications at scale for big data

- *Serving models in web services and big data frameworks (e.g., Storm or Kafka) using POJO model serving APIs*
- **Large-scale distributed TensorFlow model inference & fine tuning using *TFNet***

Distributed TensorFlow Model Inference

1. **Export TensorFlow models (in your TensorFlow program)**

```
import tensorflow as tf

batch_size_tensor = tf.placeholder_with_default(128, shape=[])
x_batch, y_batch = tf.train.shuffle_batch (..., batch_size=batch_size_tensor, ...)
cnn = cnn_model_fn(x_batch, y_batch)

sess = tf.Session()
init_op = tf.group(tf.global_variables_initializer(), tf.local_variables_initializer())
sess.run(init_op)
for step in range(600):
     _, loss = sess.run([cnn.train_op, cnn.loss], ...)

from zoo.utils.tf import *
export_tf(sess, folder_path, [x_batch], [cnn.prediction])
```

Distributed TensorFlow Model Inference

2. Load exported TensorFlow model into Analytics Zoo

```
from zoo.pipeline.api.net import *
model = TFNet.from_export_folder(folder_path)

# Alternatively, you may directly load a frozen TensorFlow model as follows
# model = TFNet(model_path, ["image_tensor:0"], ["output_tensor:0"])
```

3. Add a few layers and run distributed model inference

```
import zoo.pipeline.api.autograd as A
from zoo.pipeline.api.keras.layers import *
from zoo.pipeline.api.keras.models import *

input = Input(shape=[2, 3, 226, 226])
features = TimeDistributed(layer=model)(input)
f1 = features.index_select(1, 0)
f2 = features.index_select(1, 1)
diff = A.abs(f1 - f2)
result = Model(input, diff)
result.predict_image_set(...)
```

Analytics Zoo Examples

Dogs vs. cats, object detections, TFNet

Dogs vs. Cats

Notebook:

https://github.com/intel-analytics/analytics-zoo/blob/master/apps/dogs-vs-cats/transfer-learning.ipynb

Object Detection API

Notebook:

https://github.com/intel-analytics/analytics-zoo/blob/master/apps/object-detection/object-detection.ipynb

Image Classification Using TFNet

Notebook:

https://github.com/intel-analytics/analytics-zoo/blob/master/apps/tfnet/image_classification_inference.ipynb

Break

Distributed Training In BigDL

Data parallel training

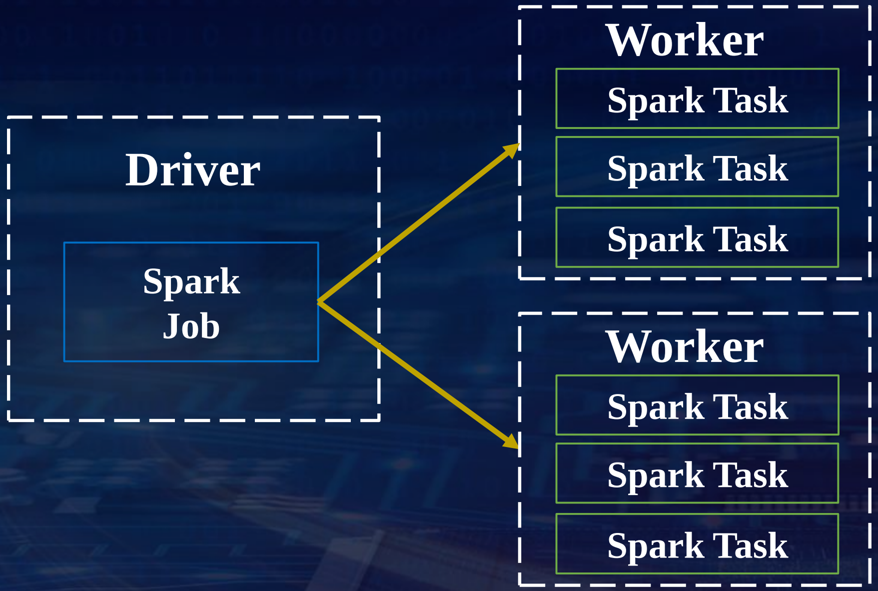
Parameter synchronization

Scaling and Convergence

Task scheduling

"BigDL: A Distributed Deep Learning Framework for Big Data", https://arxiv.org/abs/1804.05839

Apache Spark

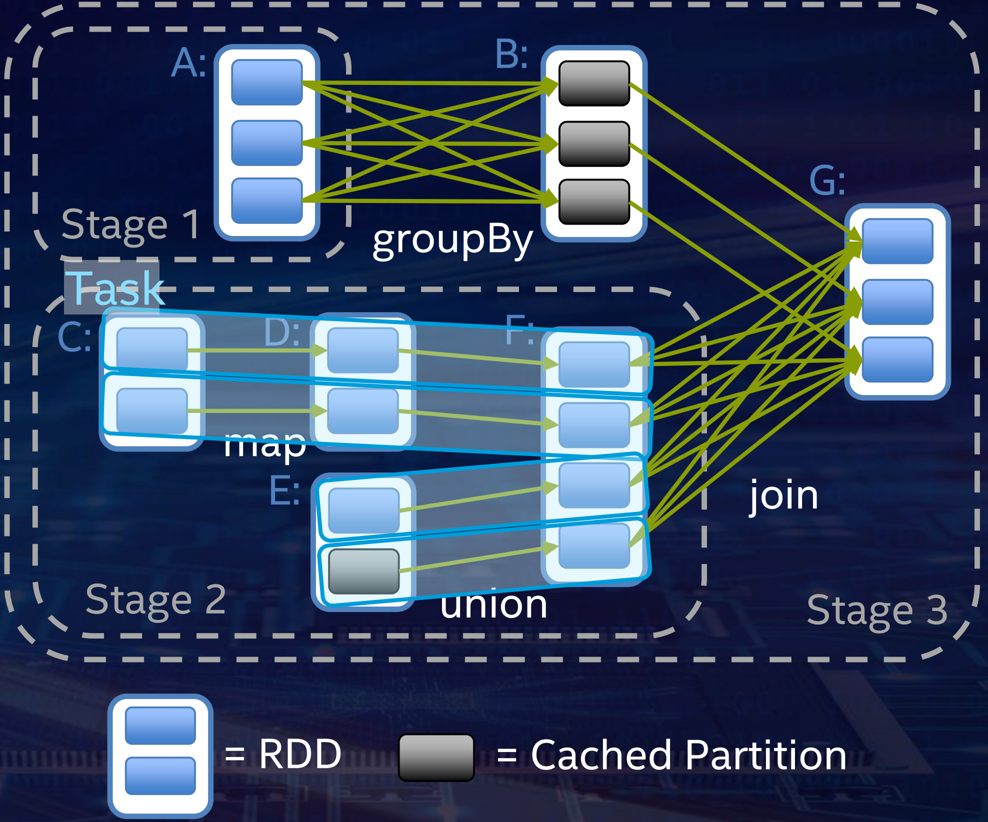


Single master (driver), multiple workers

Apache Spark

Spark compute model

- Data parallel
- Functional, coarse-grained operators
  - Immutable RDDs
  - Applying the same operation (e.g., `map`, `filter`, etc.) to all data items



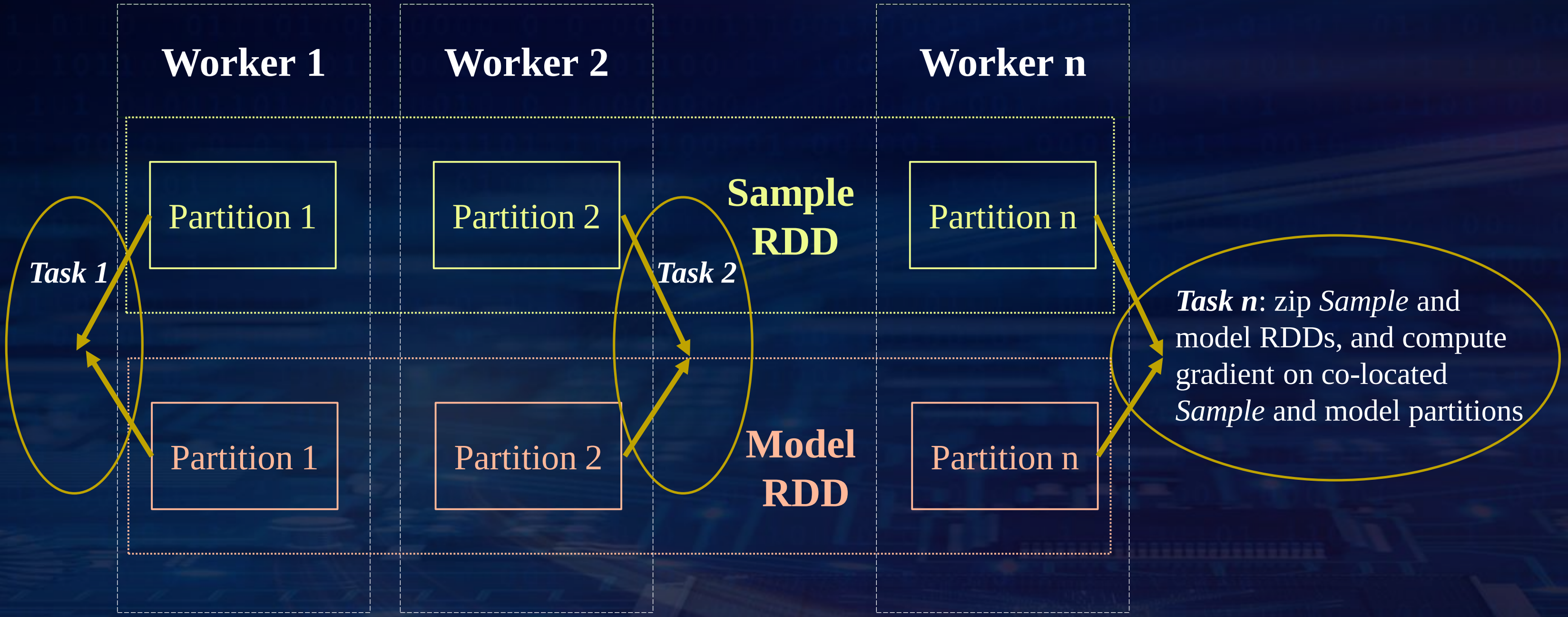
Source: "Parallel programming with Spark", Matei Zaharia, AMPCamp 3

Distributed Training in BigDL

Data Parallel, Synchronous Mini-Batch SGD

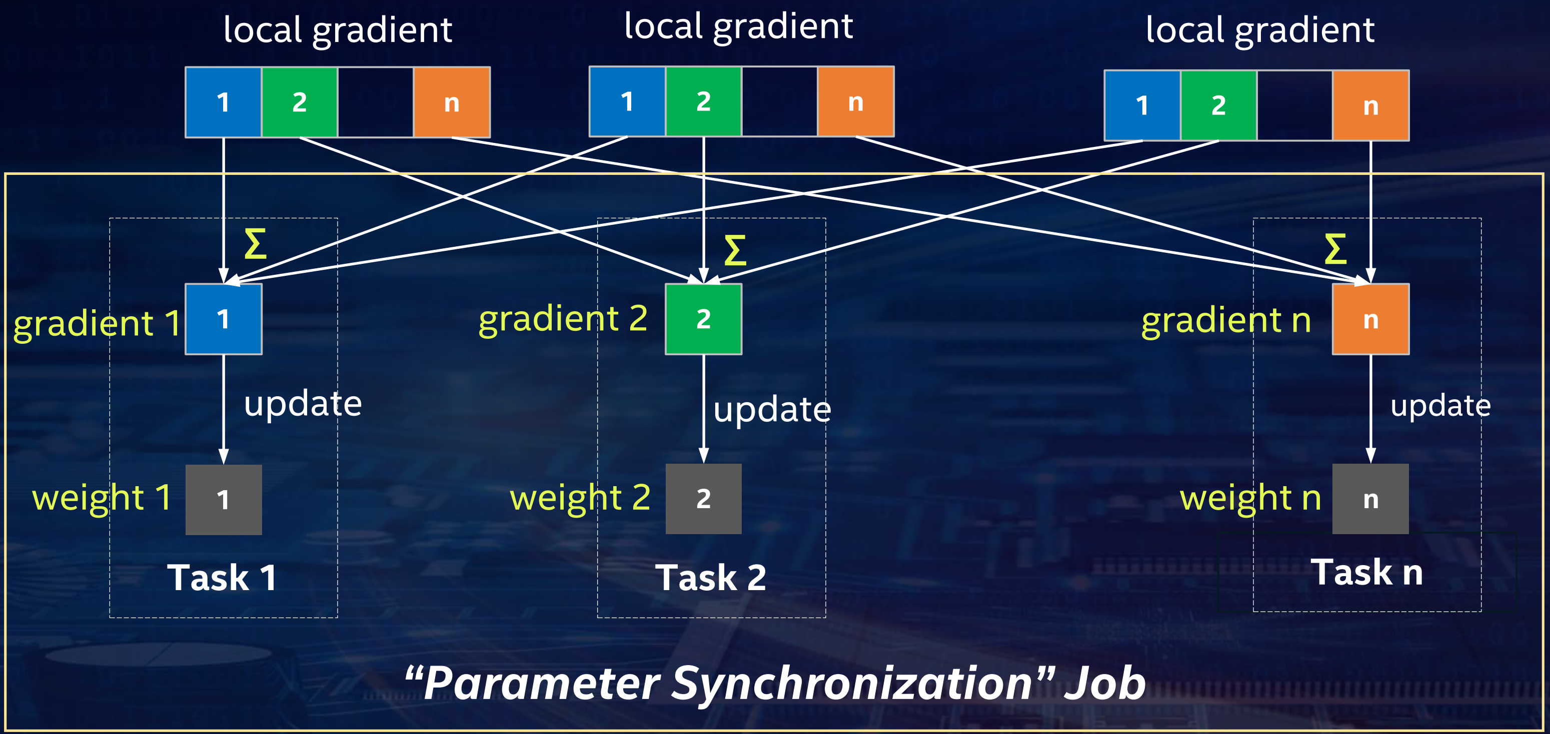
```
Prepare training data as an RDD of Samples
Construct an RDD of models (each being a replica of the original model)

for (i <- 1 to N) {
  //"model forward-backward" job
  for each task in the Spark job:
     read the latest weights
     get a random batch of data from local Sample partition
     compute errors (forward on local model replica)
     compute gradients (backward on local model replica)

  //"parameter synchronization" job
  aggregate (sum) all the gradients
  update the weights per specified optimization method
}
```

Data Parallel Training



"Model Forward-Backward" Job

Parameter Synchronization



Parameter Synchronization

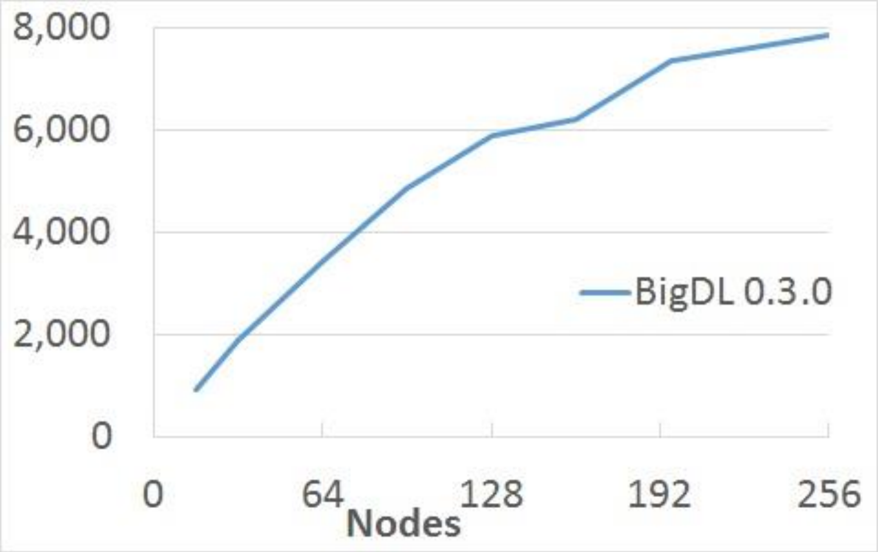
```
For each task n in the "parameter synchronization" job {
    shuffle the nth partition of all gradients to this task
    aggregate (sum) the gradients
    updates the nth partition of the weights
    broadcast the nth partition of the updated weights
}
```

"Parameter Synchronization" Job
(managing n^{th} partition of the parameters - similar to a parameter server)

"Parameter Server" style architecture (directly on top of primitives in Spark)

- Gradient aggregation: *shuffle*
- Weight sync: *task-side broadcast*
- *In-memory* persistence

Training Scalability



Throughput of ImageNet Inception v1 training (w/ BigDL 0.3.0 and dual-socket Intel Broadwell 2.1 GHz); the throughput scales almost linear up to 128 nodes (and continue to scale reasonably up to 256 nodes).

Source: Scalable Deep Learning with BigDL on the Urika-XC Software Suite
(https://www.cray.com/blog/scalable-deep-learning-bigdl-urika-xc-software-suite/)

Increased Mini-Batch Size

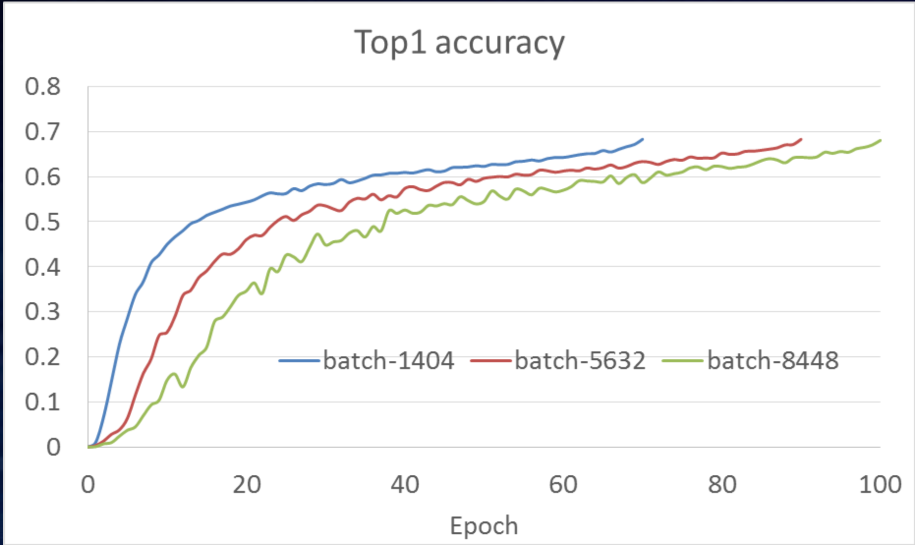
- Distributed synchronous mini-batch SGD
  - Increased mini-batch size

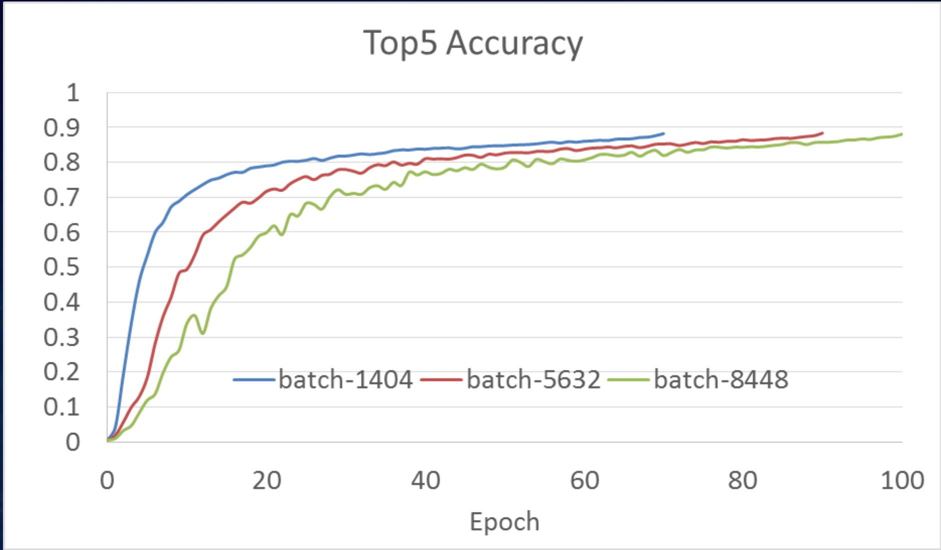
    *total_batch_size = batch_size_per_worker * num_of_workers*
  - Can lead to loss in test accuracy

- State-of-art method for scaling mini-batch size*
  - Linear scaling rule
  - Warm-up strategy
  - Layer-wise adaptive rate scaling
  - Adding batch normalization

*Source: "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour", Priya Goyal, et at. https://arxiv.org/abs/1706.02677
*Source: "ImageNet Training in Minutes", Yang You, et at. https://arxiv.org/abs/1709.05011

Training Convergence: Inception v1



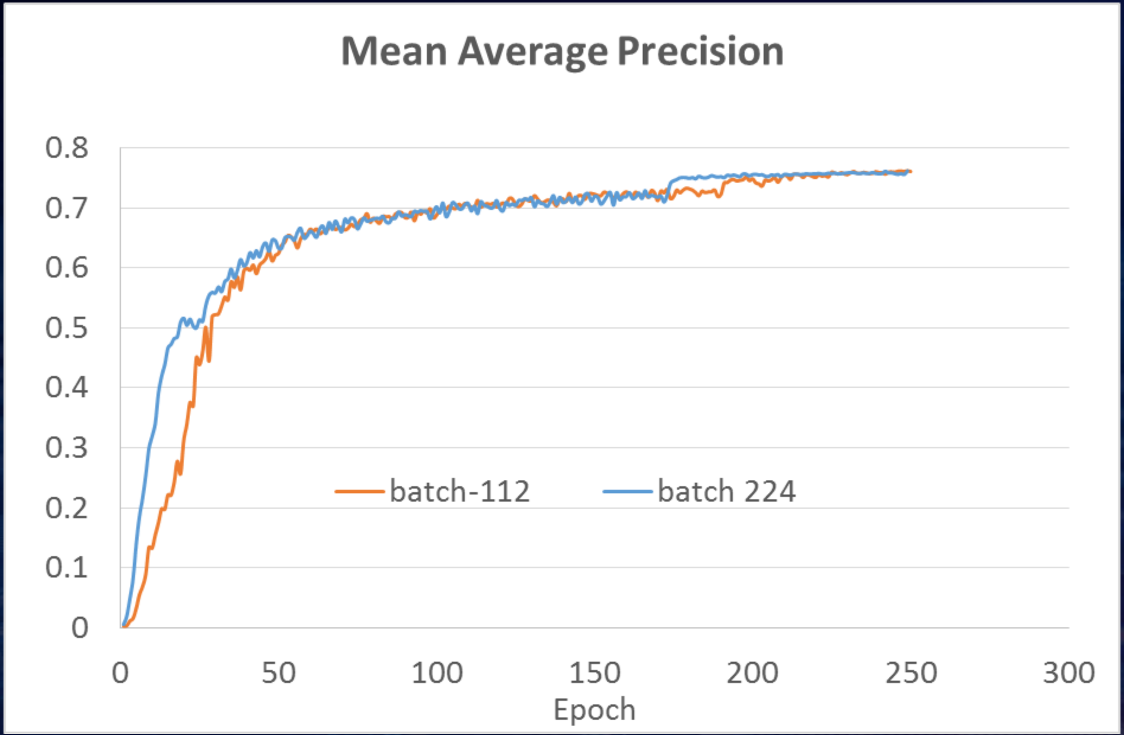


Strategies

- Warm-up
- Linear scaling
- Gradient clipping
- TODO: adding batch normalization

Source: Very large-scale distributed deep learning with BigDL, Jason Dai and Ding Ding. O'Reilly AI Conference 2017

Training Convergence: SSD



Strategies

- Warm-up
- Linear scaling
- Gradient clipping

Source: Very large-scale distributed deep learning with BigDL, Jason Dai and Ding Ding. O'Reilly AI Conference 2017

Difference vs. Classical PS Architecture

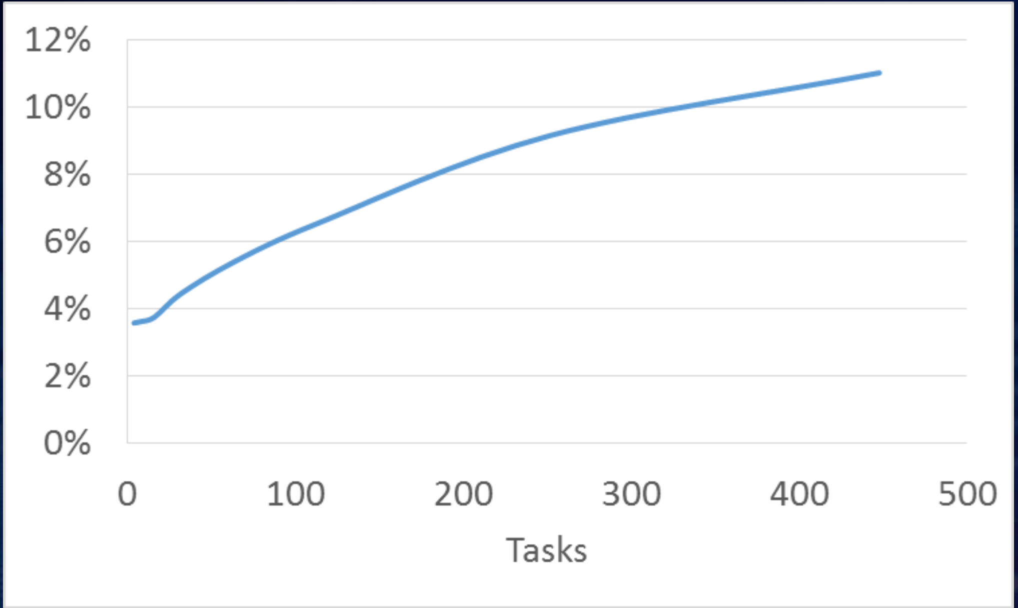
Classical PS architecture

- Multiple long-running, potentially stateful tasks
- Interact with each other (in a blocking fashion for synchronization)
- Require fine-grained data access and in-place data mutation
- Not directly supported by existing big data systems

BigDL implementations

- Run a series of short-lived Spark jobs (e.g., two jobs per mini-batch)
- Each task in the job is stateless and non-blocking
- Automatically adapt to the dynamic resource changes (e.g., *preemption*, *failures*, *resource sharing*, etc.)
- Built on top of existing primitives in Spark (e.g., *shuffle*, *broadcast*, and *in-memory data persistence*)

Task Scheduling Overheads



Spark overheads (task scheduling & task dispatch) as a fraction of average compute time for Inception v1 training

BigDL implementations

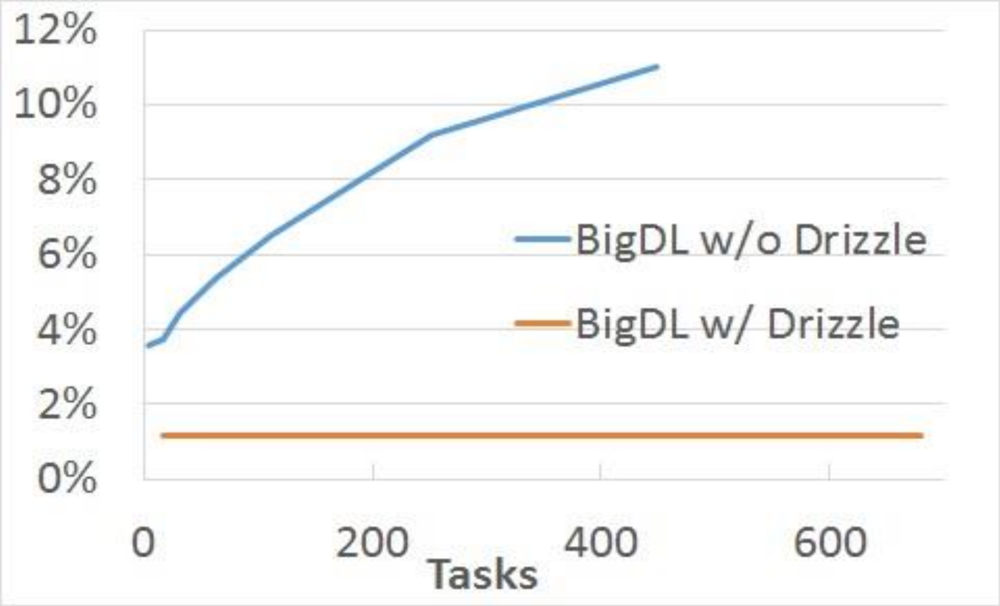
- Run a single, multi-threaded task on each worker
- Achieve high scalability on large clusters (e.g., up to 256 servers

Source: Accelerating Deep Learning Training with BigDL and Drizzle on Apache Spark, Shivaram Venkataraman, Ding Ding, and Sergey Ermolin. (https://rise.cs.berkeley.edu/blog/accelerating-deep-learning-training-with-bigdl-and-drizzle-on-apache-spark/)

Reducing Scheduling Overheads Using Drizzle

Scaling to even larger (>500) workers

- Iterative model training
  - Same operations run repeatedly
- Drizzle
  - A low latency execution engine for Spark
  - *Group scheduling* for multiple iterations of computations at once



Source: Accelerating Deep Learning Training with BigDL and Drizzle on Apache Spark, Shivaram Venkataraman, Ding Ding, and Sergey Ermolin. (https://rise.cs.berkeley.edu/blog/accelerating-deep-learning-training-with-bigdl-and-drizzle-on-apache-spark/)

Advanced Analytics Zoo Applications

Variational autoencoder, movie recommendations

Variational AutoEncoder

Notebook:

https://github.com/intel-analytics/analytics-zoo/blob/master/apps/variational-autoencoder/using_variational_autoencoder_to_generate_digital_numbers.ipynb

https://github.com/intel-analytics/analytics-zoo/blob/master/apps/variational-autoencoder/using_variational_autoencoder_to_generate_faces.ipynb

Movie Recommendations

Notebook:

https://github.com/intel-analytics/analytics-zoo/blob/master/apps/recommendation/ncf-explicit-feedback.ipynb

Real-World Applications

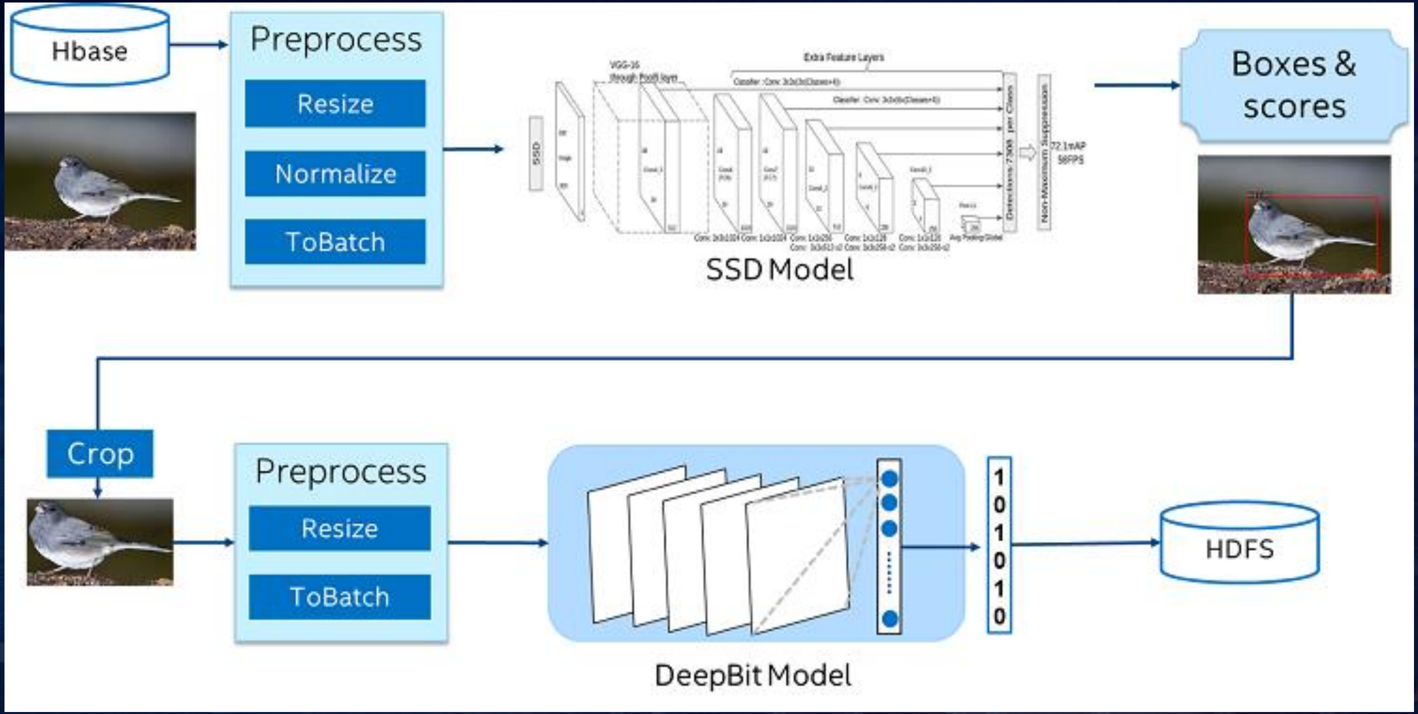
Object detection and image feature extraction at JD.com

Image similarity based house recommendation for MLSlisting

Transfer learning based image classifications for World Bank

Fraud detection for payment transactions for UnionPay

Object Detection and Image Feature Extraction at JD.com



Applications

Large-scale image feature extraction

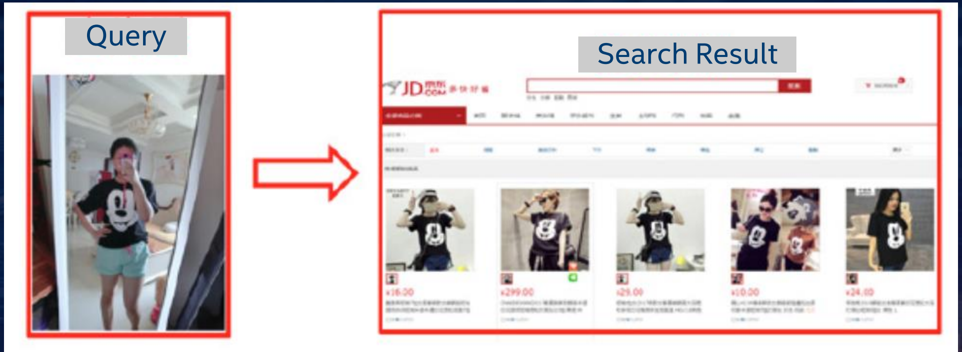
- Object detect (remove background, optional)
- Feature extraction

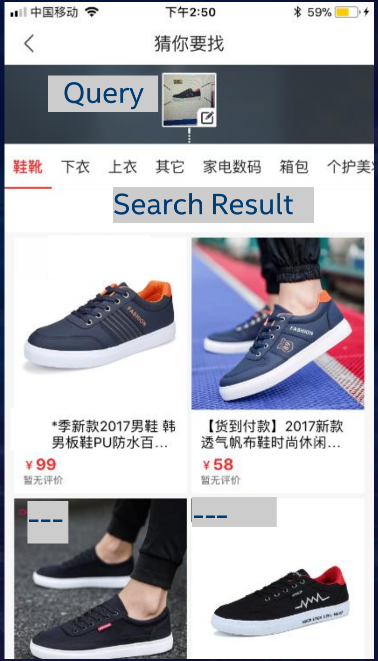
Application

- Similar image search
- Image Deduplication
    - Competitive price monitoring
    - IP (image copyright) protection system

Source: “Bringing deep learning into big data analytics using BigDL”, Xianyan Jia and Zhenhua Wang, Strata Data Conference Singapore 2017

Similar Image Search





Source: “Bringing deep learning into big data analytics using BigDL”, Xianyan Jia and Zhenhua Wang, Strata Data Conference Singapore 2017

Challenges of Productionizing Large-Scale Deep Learning Solutions

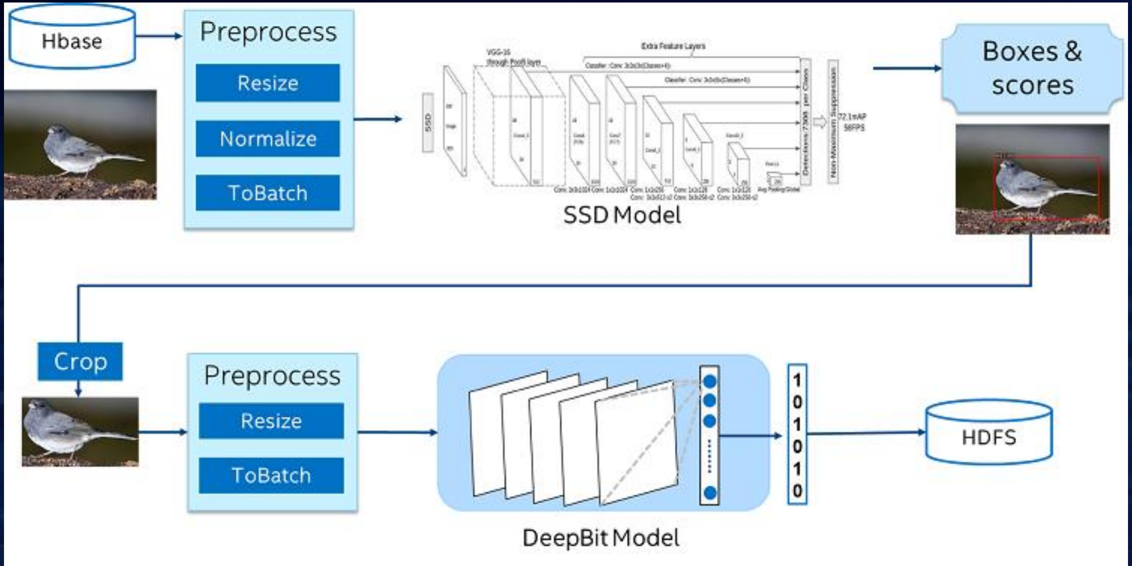
Productionizing large-scale seep learning solutions is challenging

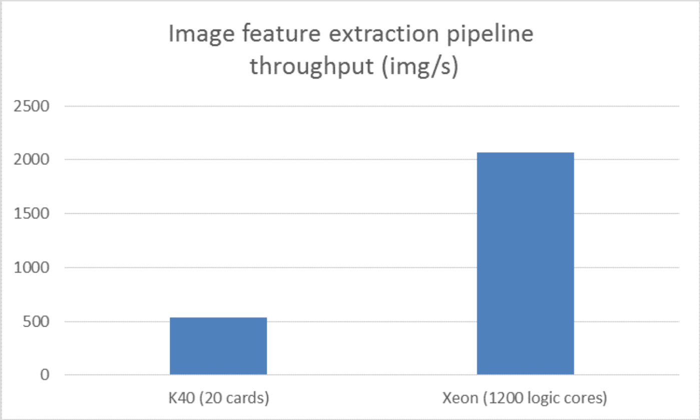
- Very complex and error-prone in managing large-scale distributed systems
  - E.g., resource management and allocation, data partitioning, task balance, fault tolerance, model deployment, etc.

- Low end-to-end performance in GPU solutions
  - E.g., reading images out from HBase takes about half of the total time

- Very inefficient to develop the end-to-end processing pipeline
  - E.g., image pre-processing on HBase can be very complex

Production Deployment with Analytics Zoo for Spark and BigDL





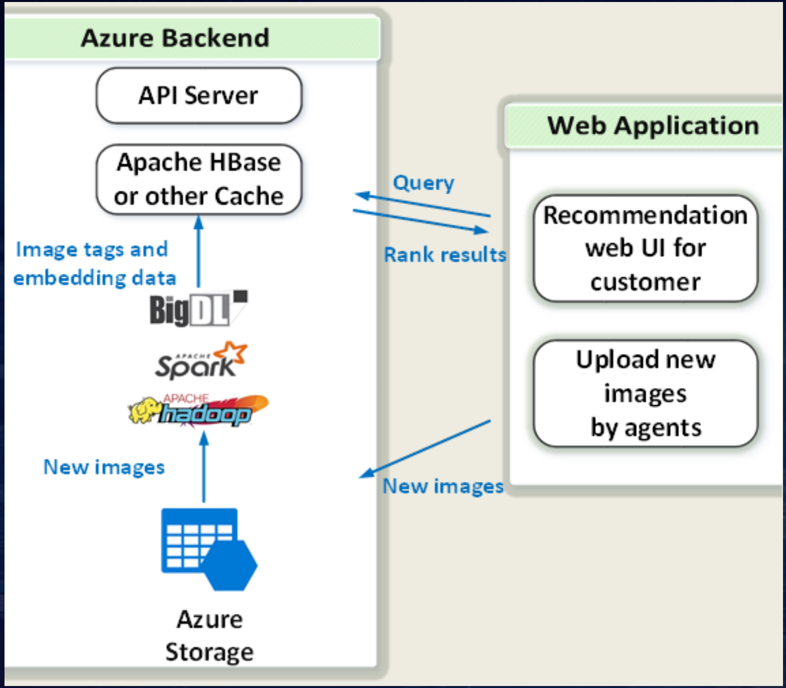
- Reuse existing Hadoop/Spark clusters for deep learning with no changes (image search, IP protection, etc.)
- Efficiently scale out on Spark with superior performance (***3.83x*** speed-up vs. GPU severs) as benchmarked by JD

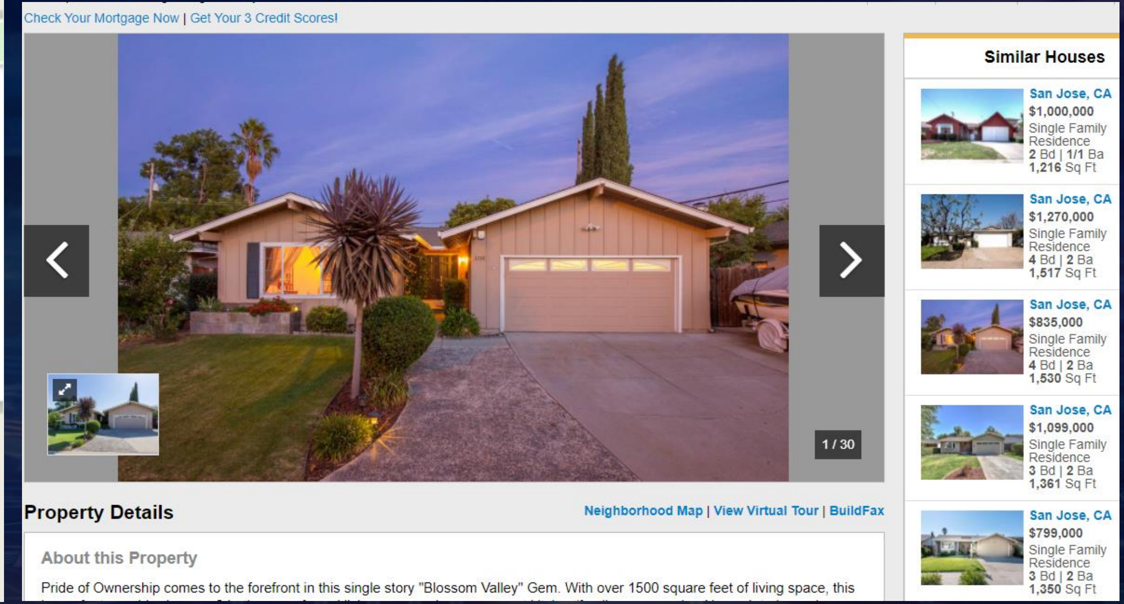
  http://mp.weixin.qq.com/s/xUCkzbHK4K06-v5qUsaNQQ
  https://software.intel.com/en-us/articles/building-large-scale-image-feature-extraction-with-bigdl-at-jdcom

CVPR 2018

Image Similarity Based House Recommendation for MLSlistings

MLSlistings built image-similarity based house recommendations using BigDL on Microsoft Azure





https://software.intel.com/en-us/articles/using-bigdl-to-build-image-similarity-based-house-recommendations

Image Similarity Based House Recommendation for MLSlistings

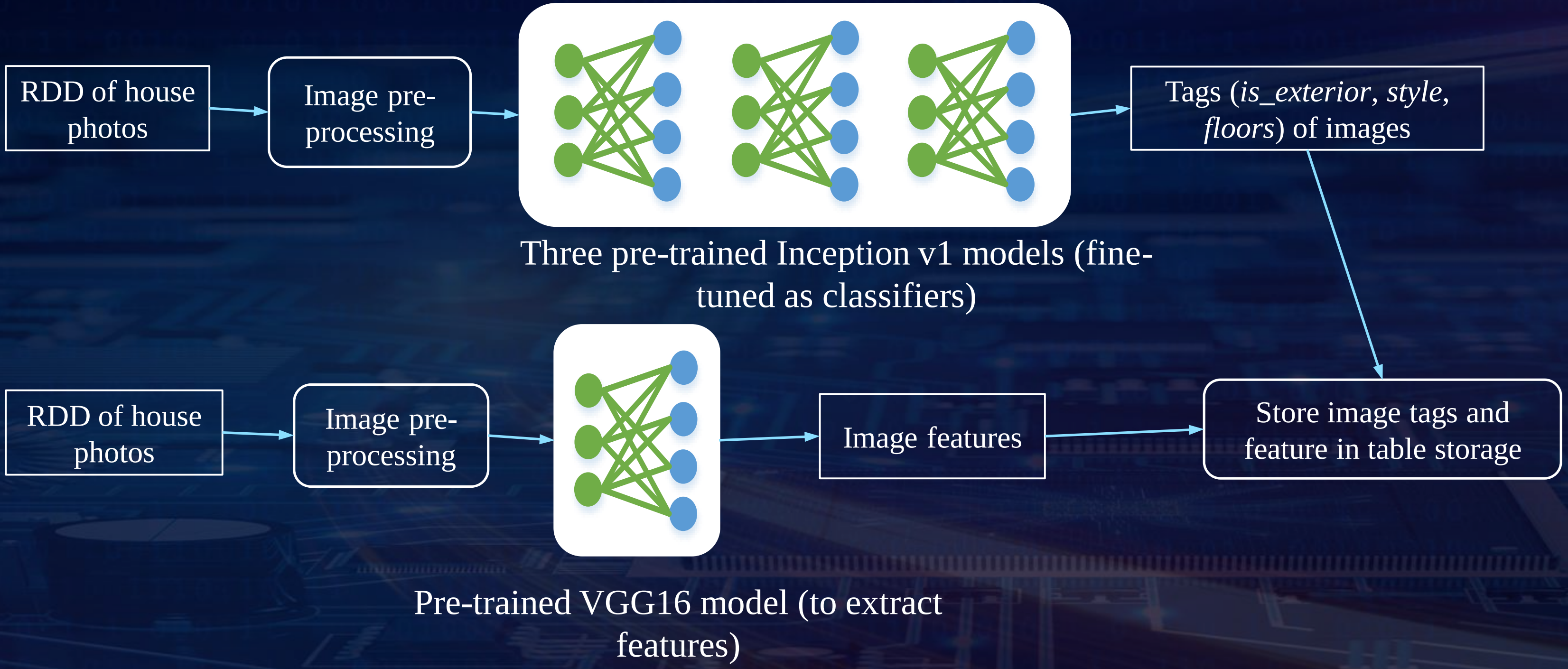


Image Similarity Based House Recommendation for MLSlistings

Notebook:

https://github.com/intel-analytics/analytics-zoo/blob/master/apps/image-similarity/Image%20similarity.ipynb

Transfer Learning Based Image Classifications for World Bank

Classifying Real Food Images is not a Cat vs. Dog Problem

Source: Using Crowdsourced Images to Create Image Recognition Models with Analytics Zoo using BigDL, Maurice Nsabimana and Jiao Wang, Spark Summit 2018

Project Layout

Phase 1:

- Image preprocessing (eliminate poor quality images and invalid images)
- Classify images (by food type) to validate existing labels

Phase 2:

- Identify texts in the image and make bounding box around them
- Text recognition (words/sentences in the image text)
- Determine whether text contains PII (personal identifiable information)
- Blur areas with PII text

Source: Using Crowdsourced Images to Create Image Recognition Models with Analytics Zoo using BigDL, Maurice Nsabimana and Jiao Wang, Spark Summit 2018

Code – Phase 1

Fine-tuning Training

Cmd 32

```
# get model
pretrained_model_path = path.join(MODEL_ROOT,"bigdl_inception_v1_imagenet_0_4_0.model")
n_classes = len(label_dict)# label categories
full_model = Net.load_bigdl("dbfs:" + pretrained_model_path)
# create a new model by remove layers after pool5/drop_7x7_s1
model = full_model.new_graph(["pool5/drop_7x7_s1"])

inputNode = Input(name="input", shape=(3, 224, 224))
inception = model.to_keras()(inputNode)
flatten = Flatten()(inception)
logits = Dense(n_classes)(flatten)

lrModel = Model(inputNode, logits)
```

```
creating: createZooKerasInput
creating: createZooKerasFlatten
creating: createZooKerasDense
creating: createZooKerasModel
```

Command took 4.74 seconds -- by Jiao.Wang@intel.com at 6/2/2018, 8:03:46 PM on 20-node-cluster

Cmd 33

```
# train model
classifier = NNClassifier(lrModel, CrossEntropyCriterion(), train_transformer) \
         .setLearningRate(learning_rate)\
         .setBatchSize(batch_size)\
         .setMaxEpoch(no_epochs)\
         .setFeaturesCol("image")\
         .setValidation(EveryEpoch(), val_image, [Top1Accuracy()], batch_size)
start = time.time()
trained_model = classifier.fit(train_image)
end = time.time()
print("Optimization Done.")
print("Training time is: %s seconds" % str(end-start))
# + dt.datetime.now().strftime("%Y%m%d-%H%M%S")
```

Prediction and Evaluation

```
#predict
predict_model = trained_model.setBatchSize(batch_size)
predictionDF = predict_model.transform(test_image)
predictionDF.cache()
```

```
'''
Measure Test Accuracy w/Test Set
'''
evaluator = MulticlassClassificationEvaluator(labelCol="label",
                                              predictionCol="prediction",
                                              metricName="accuracy")
accuracy = evaluator.evaluate(predictionDF)
# expected error should be less than 10%
print("Accuracy = %g " % accuracy)
predictionDF.unpersist()
```

Source: Using Crowdsourced Images to Create Image Recognition Models with Analytics Zoo using BigDL, Maurice Nsabimana and Jiao Wang, Spark Summit 2018

Result – Phase 1

- Fine tune with Inception v1 on a full dataset
- Dataset: 994325 images, 69 categories

Nodes	Cores	Batch Size	Epochs	Training Time (sec)	Throughput (images/sec)	Accuracy (%)
20	30	1200	12	61125	170	81.7

* This model training was performed using multinode cluster on AWS R4.8xlarge instance with 20 nodes

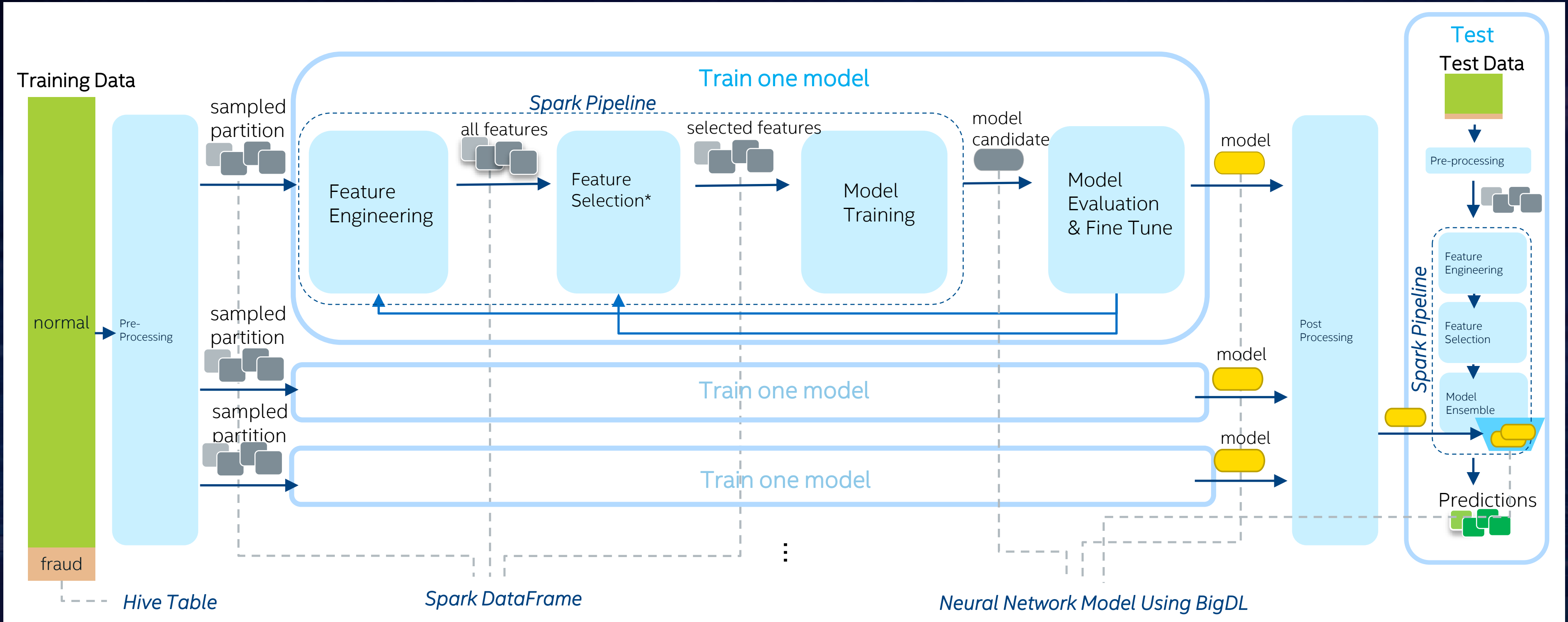
Source: Using Crowdsourced Images to Create Image Recognition Models with Analytics Zoo using BigDL, Maurice Nsabimana and Jiao Wang, Spark Summit 2018

Next Steps – Phase 2

- Image Quality Preprocessing
  - Filter with print text only
  - Rescaling, Binarisation, Noise Removal, Rotation / Deskewing (OpenCV, Python, etc.)
- Detect text and bounding box circle text
- Recognize text
- Determine whether text contains PII (personal identifiable information)
  - Recognize PII with leading words
- Blur areas with PII text
  - Image tools

Source: Using Crowdsourced Images to Create Image Recognition Models with Analytics Zoo using BigDL, Maurice Nsabimana and Jiao Wang, Spark Summit 2018

Fraud Detection for Payment Transactions for UnionPay



https://mp.weixin.qq.com/s?__biz=MzI3NDAwNDUwNg==&mid=2648307335&idx=1&sn=8eb9f63eaf2e40e24a90601b9cc03d1f

CVPR 2018

Fraud Detection for Payment Transactions for UnionPay

Notebook:

https://github.com/intel-analytics/analytics-zoo/blob/master/apps/fraud-detection/fraud-detection.ipynb

Summary

Make deep learning more accessible to big data and data science communities

- Analyze "big data" using deep learning on the same Hadoop/Spark cluster where the data are stored
- Add deep learning functionalities to large-scale big data programs and/or workflow
- Leverage existing Hadoop/Spark clusters to run deep learning applications
  - Shared, managed and monitored with other workloads (*ETL, data warehouse, traditional ML, etc.*)

Analytics Zoo: https://github.com/intel-analytics/analytics-zoo

- End-to-end Analytics + AI platform for Apache Spark and BigDL
- Build and productionize deep learning application for Big Data at Scale

LEGAL DISCLAIMERS

- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.
- No computer system can be absolutely secure.
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.