WHITE PAPER

Artificial Intelligence
Image Feature Extraction
Intel AI® Builders

Building Large-Scale Image Feature Extraction with BigDL at JD.com

This article shares the experience and lessons learned from Intel and JD teams in building a large-scale image feature extraction framework using deep learning on Apache Spark* and BigDL*.

Authors

Jason Dai
Intel

Xianyan J.
Intel

Wang, Zhenhua

Table of Contents

Background

Image feature extraction is widely used in image-similarity search, picture deduplication, and so on. Before adopting BigDL[1], the JD team tried very hard to build the feature extraction application on both multi-graphics processing unit (GPU) servers and GPU cluster settings; however, our experience shows that there are many disadvantages in the above GPU solutions, including:

- The resource management and allocation of individual GPU cards in the GPU cluster is very complex and error-prone (for example, frequent out of memory (OOM) errors and program crashes due to insufficient GPU memory).
- In multi-GPU servers, developers need to put a lot of effort into manually managing data partitioning, task balancing, fault tolerance, and so on.
- Applications that are based on GPU solutions (for example, Caffe*) have many dependencies, such as CUDA*, which greatly increases the complexities in production deployment and operations; for instance, one often needs to rebuild the entire environment for different versions of operating systems or different versions of the GNU Compiler Collection (GCC).

As a result, there are many architectural and technical challenges in building the GPU application pipelines.

Let's examine the architecture of the image feature extraction application. As the background of many images can be very complex, and the main object in the image is often small, the main object needs to be separated from the picture's background for correct feature extraction. Naturally, the framework of image feature extraction can be divided into two steps. First, the object detection algorithm is used to detect the main object, and then the feature extraction algorithm is used to extract the features of the identified object. Here, we use the Single Shot MultiBox Detector* (SSD)[2] for object detection, and the DeepBit* model[3] for feature extraction.

JD has a massive number (hundreds of millions) of merchandise pictures, which are stored in mainstream open-source distributed database storage; therefore, efficient data retrieval and processing on this large-scale, distributed infrastructure is a key requirement of the image feature extraction pipeline. GPU-based solutions have some additional challenges to meeting the requirement:

- Reading out the image data takes a very long time, which cannot be easily optimized in the GPU solutions.
- The image preprocessing on distributed database storage can be very complex in the GPU solutions, as in this case no existing software frameworks can be used for resource management, data processing, fault tolerance, and so on.
- It is challenging to scale out the GPU solutions to analyze the massive number of pictures due to the software and hardware infrastructure constraints.

BigDL* Solutions

In the production environment, using existing software and hardware infrastructure can substantially improve efficiency (for example, time-to-product) and reduce cost. As the image data are already stored in the big data cluster (distributed database storage) in this case, the challenges mentioned above can be easily addressed if existing big data clusters (such as Hadoop* or Spark clusters) can be reused for deep learning applications.

BigDL[1], an open source, distributed, deep learning framework from Intel, provides comprehensive deep learning algorithm support on Apache Spark. Built on the highly scalable Apache Spark platform, BigDL can be easily scaled out to hundreds or thousands of servers. In addition, BigDL uses Intel® Math Kernel Library (Intel® MKL) and parallel computing techniques to achieve very high performance on Intel® Xeon® processor-based servers (comparable to mainstream GPU performance).

In this use case, BigDL can provide support for various deep learning models (for example, object detection, classification, and so on); in addition, it also lets us reuse and migrate pre-trained models (in Caffe, Torch*, TensorFlow*, and so on), which were previously tied to specific frameworks and platforms, to the general purpose big data analytics platform through BigDL. As a result, the entire application pipeline can be fully optimized to deliver significantly accelerated performance.

We built the end-to-end image feature extraction pipeline in Apache Spark and BigDL as follows (see Figure 1):

1. Read hundreds of millions of pictures from a distributed database in Spark as resilient distributed datasets (RDD) .
2. Preprocess the RDD of images (including resizing, normalization, and batching) in Spark.
3. Use BigDL to load the SSD model for large scale, distributed object detection on Spark, which will generate the coordinates and scores for the detected objects in the images.
4. Keep the object with highest score as the target, and crop the original picture based on the object coordinates to get the target picture.
5. Preprocess the RDD of target images (including resizing and batching).
6. Use BigDL to load the DeepBit model for distributed feature extraction of the target images on Spark, which will generate the corresponding features.
7. Store the result (RDD of extracted object features) in the Hadoop Distributed File System (HDFS).

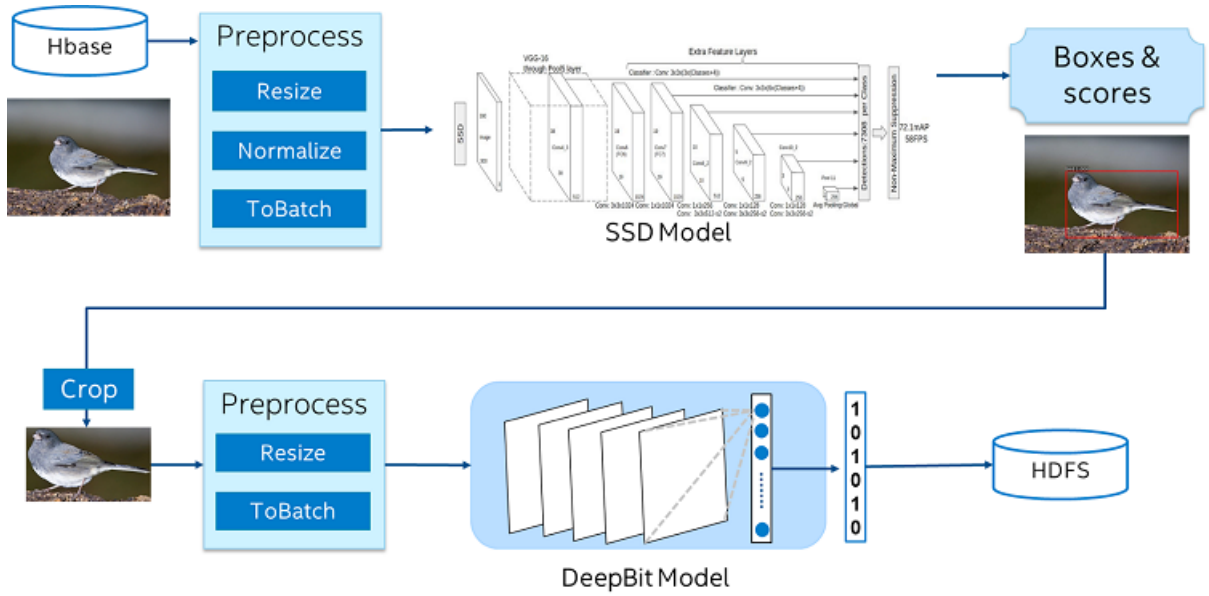


Figure 1: Image feature extraction pipeline based on BigDL.

The entire data analytics pipeline, including data loading, partitioning, preprocessing, prediction, and storing the results, can be easily implemented on Spark and BigDL. By using BigDL, users can directly use existing big data (Hadoop/Spark) clusters to run deep learning applications without any changes to the cluster; in addition, the BigDL applications can easily take

advantage of the scalability of the Spark platform to scale out to a large number of nodes and tasks, to greatly speed up the data analytics process.

In addition to distributed deep learning support, BigDL also provides a number of useful tools, such as image preprocessing libraries, model loading utilities (including loading models from third-party, deep learning frameworks), which make it convenient for users to build the entire deep learning pipeline.

Image Preprocessing

BigDL provides an image preprocessing library[4] built on top of OpenCV*[5], which provides support for common image transformation and augmentation operations, so that the users can easily construct an image-preprocessing pipeline using these operations. In addition, users can also build customized image transformation functions with OpenCV operations provided in the library.

```
val preProcessor =
   BytesToMat() ->
   Resize(300, 300) ->
   MatToFloats(meanRGB = Some(123, 117, 104)) ->
   RoiImageToBatch(10)

val transformed = preProcessor(dataRdd)
```

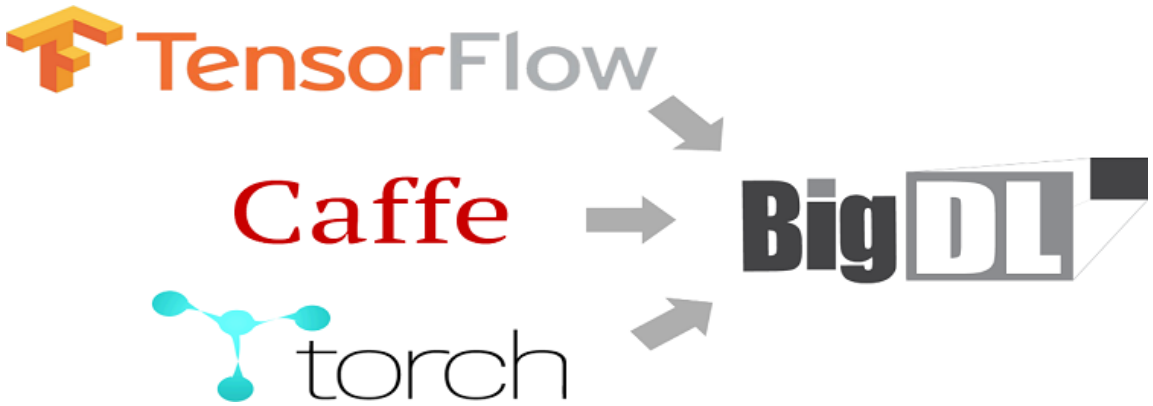
The sample image preprocessing pipeline above converts a raw RDD of byte arrays through a series of transformations. Among them, *ByteToMat* transforms the byte picture into *Mat* (the storage format used in OpenCV), *Resize* adjusts the picture size to 300 x 300, *MatToFloats* transforms the pixels of Mat into Float array format and subtracts the corresponding channel mean. Finally, *RoiImageToBatch* batches the data as the input to the model prediction or training.

Load the Model

Users can easily use BigDL to load pretrained models, which can then be directly used in the Spark program. Given a BigDL model file, one can call *Module.load* to load the model:

```
val model = Module.load[Float](bigdlModelPath)
```

In addition, BigDL also allows users to load models from third-party deep learning frameworks such as Caffe, Torch, and TensorFlow.



Users can easily load pretrained models for data prediction, feature extraction, fine-tuning, and so on. For instance, a Caffe model consists of two files, namely, the model *prototxt definition file* and the model *parameter file*. Users easily load pretrained Caffe models into the Spark and BigDL program as follows:

```
val model = Module.loadCaffeModel(caffeDefPath, caffeModelPath)
```

Performance

Performance benchmarking for both the Caffe-based GPU solutions and BigDL-based Intel Xeon processor solutions were conducted in JD's internal clusters.

Test Process

The end-to-end image processing and analytics pipeline includes:

1. Reading the pictures from the distributed database storage.
2. Inputting to the object detection model and feature extraction model for feature extraction.
3. Saving the results (image paths and features) to the distributed file system.

Note that the first step (reading the pictures from the distributed database storage) can take a lot of time in the end-to-end performance measurement. In this case, the first step takes about half of the total processing time (including image reading, object detection, and feature extraction) in GPU solutions, which cannot be easily optimized on multi-GPU servers or GPU clusters.

Test Environment

- GPU: 20 * NVIDIA Tesla* K40
- CPU: Intel® Xeon® processor E5-2650 v4 @ 2.20GHz, 1200 logical cores (each server has 24 physical cores with Intel® Hyper-Threading Technology (Intel® HT Technology) enabled, and is configured to support 50 logical cores in Yet Another Resource Negotiator (YARN).

Test Result

Figure 2 shows that the image processing throughput of Caffe on 20 * K40 is about 540 images per second, while the throughput of BigDL is about 2070 pictures per second in a YARN using an Intel Xeon processor cluster with 1200 logical cores. The throughput of BigDL on the Intel Xeon processor cluster is ~3.83X of the GPU cluster, which greatly speeds up the image processing and analytics tasks.

The test results show that BigDL provides much better support for the large-scale image feature extraction application. The high scalability, high performance, and ease of use of BigDL-based solutions make it easy for JD to deal with the massive and ever-growing number of images. As a result, JD is in the process of upgrading the GPU solution to the BigDL on the Intel Xeon processor solution, and deploying it to the production Spark cluster in JD.

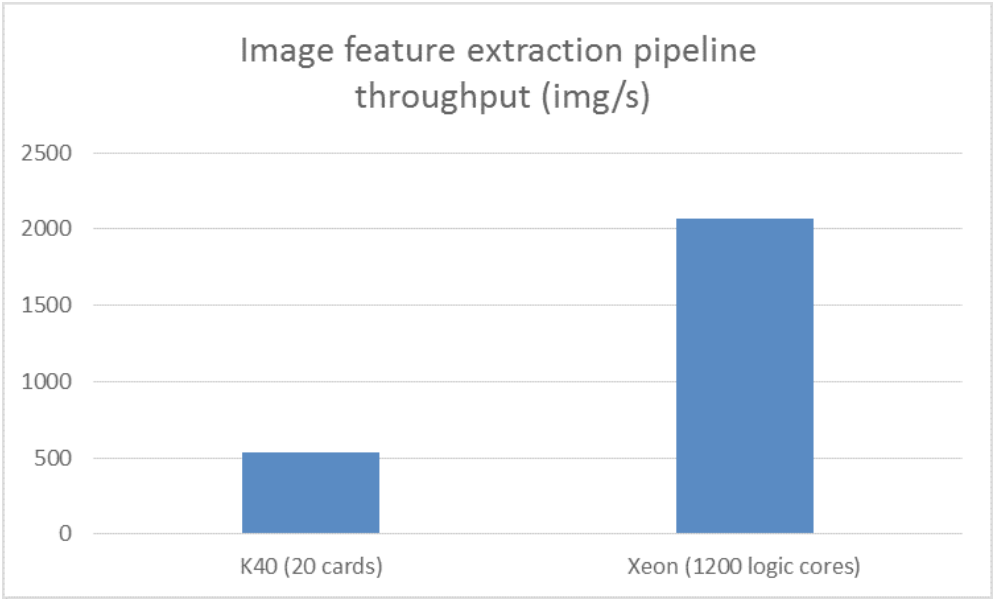


Figure 2: Compares the throughput of K40 and Intel® Xeon® processors in the image feature extraction pipeline.

Conclusion

The high scalability, high performance, and ease of use of BigDL make it easy for JD[6] to analyze the massive number of images using deep learning technologies. JD will continue to apply BigDL to a wider range of deep learning applications, including distributed model training.

References

1. BigDL, https://github.com/intel-analytics/BigDL
2. Liu, Wei, et al., *SSD: Single Shot MultiBox Detector*, European conference on computer vision. Springer, Cham, 2016.
3. Lin, Kevin, et al., *Learning compact binary descriptors with unsupervised deep neural networks*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016.
4. Vision: Image Augmentation, https://github.com/intel-analytics/analytics-zoo/tree/master/transform/vision
5. Open Source Computer Vision Library, http://opencv.org/
6. Same article (Chinese) published by JD.com