# Leveraging Spark for Large Scale Deep Learning: Data Preparation and Inference

## *Difficult to understand what’s going on Here*

## While it has been ~~been~~ known that training a Deep Learning model requires lots of data to produce good result, rapidly growing business data often requires deployed Deep Learning model to be able to process larger and larger dataset. It is not uncommon nowadays that Deep Learning practitioners face the challenge of operating in a big data world.

To solve the problem with large dataset in training, distributed Deep Learning frameworks were introduced. At the inference side, machine learning models ~~and~~, particularly deep learning models ~~in particular~~ are usually deployed as Rest API end point and scalability is implemented by replicating the deployment across multiple nodes in framework such as Kubernetes.

These mechanisms usually require~~s~~ a lot of engineering effort to setup correctly and is not always efficient, especially in very big data volume.

In this article, I’d like to present two technical approaches to address the two challenges of Deep Learning in Big data:

1. Parallelize large volume data preprocessing for structured and unstructured data

2. Deploy Deep Learning Model for high performance batch scoring in big data pipeline with Spark.

The approaches leverages latest features and enhancements in Spark Framework and Tensorflow 2.0.

# 1. Introduction

Deep Learning is usually trained with large amount of unstructured data such as images, audio, text etc. or structured data with thousands of features. It is not uncommon to see hundreds of GB dataset is provided as input for training. If a single node architecture is used, it may take ~~very long time~~ extremely long to finish digesting and training ~~with this large dataset.~~ There are distributed Deep Learning Frameworks out there such as Horovod and Distributed Tensorflow that helps scale out Deep Learning training in overall. But these frameworks focus on distributing core training tasks of calculating gradient on a shard of data from a replica of the model while do not do a good job of parallelizing other general computation steps. Apache Spark, is an [open-source](https://analyticsindiamag.com/10-open-source-data-science-big-data-applications-well-supported-linux/) cluster computing framework that provides an interface for programming an entire cluster with implicit data parallelism and fault tolerance, can be an excellent tool to offload general computation steps in a simple manner.

By using Apache Spark, we can offload preprocessing steps including data cleaning, feature engineering and produce a readily consumable dataset that Deep Learning framework can then be used. This will save ~~a lot of~~ time and possibly cost as expensive resources such as GPU can be utilized to focus on where it shines: computing gradient of model from large tensor metrices. (matrices?)

On the other end, while traditionally it is common that a deployed ML model in general and a deep learning model in particular have been deployed as Rest APIs.(don’t understand this) While this approach is good for real time scoring scenario, it may not provide best throughput when ~~the scenario is~~ data comes in as very large volume in batches.

Again, Spark shines here as an excellent choice for large volume, batch style scoring for Deep Learning Model.

While Spark has been around for long and there have been multiple initiatives that attempted to leverage Spark to scale Deep Learning, how are these approaches different from the others?

The approaches in this paper utilize two very important features in Spark that addresses current weakness from above other frameworks:

* Support for binary file format to deal with virtually any types of unstructured data in Deep Learning
* Use the latest Pandas UDF, including Scalar Iterator that allows for caching and reusing model in memory for multiple batches without having to reload from storage

**2. Scaling Data Preprocessing For Any Type of Data**

Data preprocessing includes data cleaning and feature engineering that produce data ready for Deep Learning training and inference.

Offloading these logics to Spark distributed framework can significantly improve performance of Deep Learning training.

There several features in Spark that allow efficient distributed data preprocessing:

1. Spark’s support for binary data input and tfrecords output
2. Spark support for Deep Learning & Python libraries at worker node and use of UDF to perform complex feature engineering

First, it is important for Spark to be able to work with not only structured data and text data but also binary data as Deep Learning usually deal with images, audio and other non-structured data. This simplifies the data loading and preprocessing step as we do not need to write a custom reader for the tasks.

Recent Spark’s support for binary data deliver this capability. Spark’s Binary Files dataset provides ability to read binary data into following Dataframe’s representation

* File path
* modificationTime
* length
* Binary content

Second, Spark provides support to output to a format native to Deep Learning. For Tensorflow, there is a library called Spark Tensorflow connector that allow reading data in TFRecords (what is TFRecords?)format to Spark Dataframe.

The preprocessing of data and feature engineering may involve the use of certain custom libraries such as Pillow for image, Scipy Librosa for audio and of course tensorflow itself. Luckily, commercial Spark framework make it easy to install custom libraries across nodes in a cluster.

The strategy now is to split data using Spark Framework and have a mechanism to preprocess data. Here we will use UDF, in particular Pandas UDF to efficiently do this.

In the following code illustration that is part a of Speech Recognition exercise, I designed an UDF function together with needed libraries to derive spectrogram features from an Audio file.

An algorithm for preprocessing data

* Reading binary data using Binary Files type Spark Dataframe
* Select binary content column into an UDF function to extract feature
* Select other columns such as file path into another UDF as needed (for example to create a label column from file name)
* Inside features extraction UDF function, import needed libraries to extract features from binary data

For structured data where Spark has native support such as csv and parquet file, the process is much simpler. You can perform direct transformation on columns value and in the end convert the dataform to TFRecords.

Here is an example code to preprocess audio data for Deep Learning training

# 3. Efficient Large Scale Inference

Spark has better and better support as a deployment environment for Deep Learning by allowing to intall and interoperate with Python and Deep Learning libraries across worker nodes in a cluster. Recent advancements allow Spark to load and cache large size Deep Learning model in memory to be reused for multiple batches scoring (Spark Pandas UDF) . Data conversion of Spark data representation and that of Deep Learning framework (Python object…) has better support with Pyarrow. These feature are implemented in Pandas UDFs (a.k.a. Vectorized UDFs) feature from Apache Spark 2.3 release that substantially improves the performance and usability of user-defined functions (UDFs) in Python. The Pandas UDF addresses problem with regular UDF operate one-row-at-a-time, and thus suffer from high serialization and invocation overhead

To further support Deep Learning Large Scale inference, there is a new version of Pandas Scalar iterator Pandas UDF, which is the same as the scalar Pandas UDF above except that the underlying Python function takes an iterator of batches as input instead of a single batch and, instead of returning a single output batch, it yields output batches or returns an iterator of output batches. A scalar iterator Pandas UDF is useful when the UDF execution requires initializing some state, e.g., loading a machine learning model file to apply inference to every input batch.

Here is an example implementation of batch scoring from a persisted Keras model using Scalar Iterator UDF. Raw data is loaded using again binaryFile Spark Dataframe then logics of preprocessing and model scoring can be combined together efficiently. Data is splited into batches of Pandas series. Notice that model loading and initialization is only performed once for multiple batches as this is an expensive operation.

# 4. Conclusions

Using Spark’s can substantially improve productivity in Deep Learning development and scoring performance with big data. The new efficient implementations of binary files and Pandas UDF make Spark a viable solution in almost any type of Deep Learning development and deployment scenarios regardless of data types and model complexity. In Training, Spark can be used to offload data preprocessing tasks. In production pipeline where Spark is already popular, Deep Learning models can be deployed directly to leverage Spark’s powerful scalability and with much less complexity.

# 5. Data

The examples in this paper is performed on the Speech command