YouTube Video Classification Using Neural Networks on Different Distributed Frameworks

Prashant K. Thakur, Paahuni Khandelwal, Heting Wang
Colorado State University

April 11, 2019

1 Problem Description

As nowadays analysis on big data becomes a hot spot in this information explosion era, every academic researcher in this field needs to have the ability to extract and process a huge amount of data. However large dataset in real world is difficult to refine and analyze valuable information manually on a single machine, fortunately, with the help of existing deep learning frameworks, by passing large dataset in terms of dataflow to clusters which consists of one chef node and multiple worker nodes, training time can be reduced to one day or even less, the time consuming problem and programming difficulty can be easily solved. And the following parameter tuning and model optimization procedure become simpler.

Our objective of this project is to benchmark the performance and efficiency of two most popular distributed deep learning frameworks - Distributed TensorFlow and Apache Spark. TensorFlow [1] is an open-source distributed software library from Google which increases the scalability and parallelisms while working on big datasets. Similarly, Apache Spark MLLib [2] has a rich collection of different machine learning algorithms involving multiple iterations which have been optimized for better performance than classical Hadoop MapReduce [3] programming framework.

In our project we deployed these two frameworks and evaluated the performance of Neural Network-based classification of large-scale YouTube-8M Video dataset [4] on clusters. We measured the performance of the classifier based on the time taken to train the model for different number of nodes, a given error rate or loss during training on different nodes (cluster number).

Overall, considering at present more and more developers enter the field of big data and the purposes of different research groups vary a lot, it is always hard to find a most appropriate data processing tool from mixed types resources all at once. We expect this paper would guide users to select a classification framework for large dataset.

2 Problem Solution

2.1 Frameworks

In this section, we first introduce two frameworks being compared in our project - Distributed TensorFlow, Apache Spark.

2.1.1 TensorFlow

TensorFlow is a framework released by Google for numerical computation using data flow graphs. This framework was developed by the Google Brain Team to facilitate implementing machine learning and deep neural networks research. The data flow graphs comprise of nodes which represent the mathematical operations, and the graph edges representing the flow of data (tensor or multidimensional arrays) in-between the nodes. The framework has C/C++ engine that improves the performance along with a very rich Python APIs.

Distributed TensorFlow provides the flexibility to scale up the system by computing certain portions of the graph on different chef/worker servers. Because workers calculate gradients during training, they are typically placed on a GPU. One of the workers works as a parameter server, it only needs to aggregate gradients and broadcast updates, so it is typically placed on CPU. The chef worker coordinates model training, initializes the model, counts the number of training steps completed, monitors the session, saves

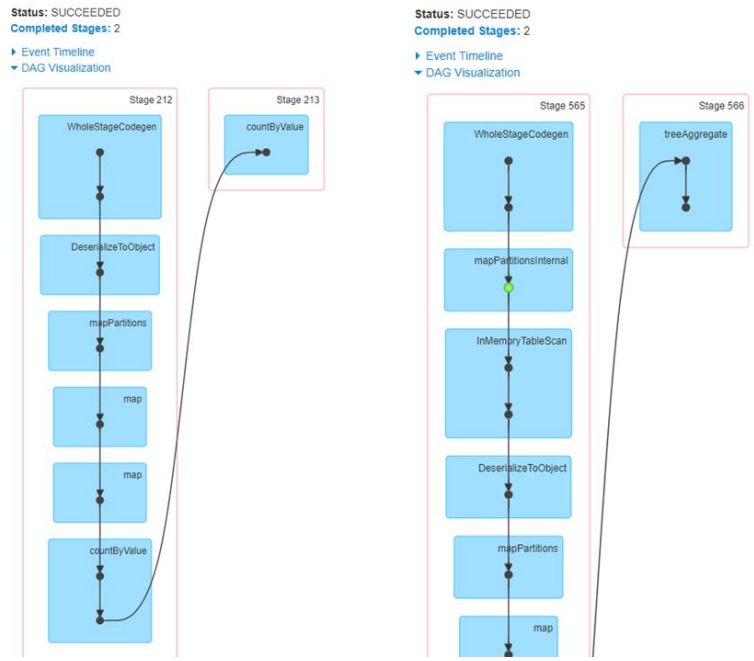

Figure 1: DAG Visualization in Apache Spark

logs for TensorBoard, and saves and restores model checkpoints to recover from failures. The chef also manages failures, ensuring fault tolerance if a worker or parameter server fails. If the chef worker itself dies, training will need to be restarted from the most recent checkpoint [5].

2.1.2 Apache Spark

Apache Spark is an open source framework for cluster-computing. The framework builds a lineage graph and the computation is carried out on different clusters based on the actions. The performance of the framework has been optimized 100 times than Hadoop by using in-memory computation and other optimization. There are two types of operations in Spark: transformations, which perform operations on an input RDD to produce one or more new RDDs, and actions, which launch the series of transformations to return the final result.

A logical execution plan, or lineage graph, provides the series of transformations to be performed on an input data set once an action has been called. This plan determines the dependencies between the RDDs in the set of transformations. The DAGScheduler then converts this logical plan into a physical execution plan: a directed acyclic graph of stages. Each stage is a set of tasks based on partitions of the input data which can be run in parallel.[8]

Spark has different packages such as support for SQL queries (*Spark SQL*), data streaming (*Spark Streaming*), graph processing (*GraphX*) and machine learning (*Spark Mllib*). Spark Mllib works on top of Spark Core and has several machine learning algorithms libraries (like - sampling, classification, filtering, clustering, dimensionality reduction, feature extraction etc).This library is designed for distributed, scalable and in-memory computation. For our classification problem, we wanted to use Neural Networks which involve lots of learning, updating weights of input parameters, repetitive steps to train. Therefore, our algorithms require many iterations, which is well-supported in Spark.

In summary, it could be said that TensorFlow is used for custom deep learning and neural network design, whereas Apache Spark is a data processing framework [6].

2.2 Methodologies and Algorithms

2.2.1 Preprocessing

The total size of the dataset is 31 Gigabytes, and the dataset also contains many attributes unrelated to the classification which is dirty and forms barriers for our evaluation running on existing machines. Therefore, we randomly sampled 7G data from the entire dataset. Now each video in the sampled dataset contains id, audio and RGB features, along with one or multiple labels. Since our training model only

considered the condition one label per video, we defined a certain range of labels to apply the actual classification. In a CSV file provided in this dataset, each label is represented by an index, such as 0 represents label category 'sports', we formulated a rule which was, if a video only contains labels from 0 to 9, then we select it to our training sample set, otherwise we ignore it. After this process, we had a dataset which each video labels from 0 to 9. Then we randomly selected a label in all its labels of each video and treated it as the label of the video. Finally, our dataset consists of single label videos. In other words, the later processing was focused on 10 different target label video-level feature sets. In this way, the complex computation on a huge amount of data is prevented.

The dimension of RGB and audio feature in each video are already reduced to 1024 and 128 by PCA technology, respectively. For processing convenience, before passing parameters through the command line to our training model, we combined these two features in an array with 1152 dimensions.

2.2.2 Data Transformation

While the tfrecords can be read directly to TensorFlow, we must transform the dataset format before input it to Spark for following processing purpose .The selection process is the same as in TensorFlow. After that, tfrecords are converted to libsvm files. This format is compatible with Spark framework. It stores the target class value along with each input parameter with its index. So, in libsvm data stored is reduced by not storing the values which are equal to zero. Only non-zero values for any parameters are stored.

2.2.3 System Architecture

The pre-processing of the data is same in both Spark Mllib and TensorFlow. The data was then transformed into numpy array in TensorFlow, which was then feed into the neural network in batches. The model was designed such that the data is distributed into a different chunk logically into a different batch size and different worker work on different chuck. One iteration finishes when the entire dataset was used for the training purpose. During the training phase, we validated the model with our validation dataset after the certain number of iterations (2 iterations). Different hidden layers were tested for analyzing the accuracy and later one model was adopted for the use in the cluster. Additionally, no cluster manager was used for the distributed TensorFlow. The workers and the parameter servers were manually configured in the code to configure the clustering and the selection of the machine was based on the resources. While working on the configuration, some machines were causing the bottleneck in the computation so those machines were either removed or kept as parameter servers instead of being deployed as the worker node.

In Spark, after converting the dataset into libsvm format. Training and testing dataset was stored in dataframe, where we performed controlled partitioning by distributing data uniformly across all nodes. The partitioning was done based on the 'label' column which represented the target class. Based on the value of this column data was parallel stored on different nodes. Then we fit the training data and tested and collected different evaluation metrics. To get best results from the MLPC classifier, the Mllib library gives us the ability to set the hidden layer structures, tolerance of iteration, block size of the learning, seed size and maximum number of iterations. After performing different combinations of all these parameters we performed hyperparameter tuning to give better accuracy.

Artificial neural nets is an iterative process. Iteration number may vary from 25 to 2000 as well. So, in Spark lineage graphs can become quite large. Spark saves data objects and DAG information associated with each stage to the JVM heap on the driver node. Once the heap is filled, the JVM is forced to do full garbage collection which increases in frequency as lineage graphs expand with more iterations. For fault tolerance as well, recomputing results from a large lineage graph in case of node failure may extend execution time significantly[8]. So, for further optimization, we performed manual check-pointing by caching the intermediate results which helped us improved the execution time. The application JAR file was submitted to Spark cluster manager.

2.2.4 Training and Testing

We worked upon classification models provided by both the frameworks. Therefore, in TensorFlow we developed Neural Network for classifying videos, and to perform classification on Spark we applied Multilayer perceptron classifier using Neural Network provided in Mllib. Furthermore, we implemented stratified partitioning which took into account equal percentage of samples from all the target classes while training and testing the network. Our model was trained on dataset provided in training set and tested on 1 GB of records from testing set.

```
features: {
  feature: {
    key  : "video_id"
    value: {
      bytes_list: {
        value: [YouTube video id string]
      }
    }
  }
  feature: {
    key  : "labels"
    value: {
      int64_list: {
        value: [1, 522, 11, 172] # The meaning of the labels can be found here.
      }
    }
  }
  feature: {
    key  : "mean_rgb" # Average of all 'rgb' features for the video
    value: {
      float_list: {
        value: [1024 float features]
      }
    }
  }
  feature: {
    key  : "mean_audio" # Average of all 'audio' features for the video
    value: {
      float_list: {
        value: [128 float features]
      }
    }
  }
}
```

Figure 2: Video-level feature in TensorFlow.Example protocol

2.2.5 Multilayer Perceptron Classifier - MLPC

Multilayer perceptron classifier consists of multiple layers of units, each layer is fully connected to the next layer in the network. We passed input data to the first layer, and the other layers mapped the inputs to outputs by a linear combination of the inputs with the node's weights w and bias b and applying an activation function. Units in intermediate layers used sigmoid (logistic) function, units in the output layer used softmax function. The number of units in the output layer corresponded to the number of target classes. The MLPC employs backpropagation for learning the model and uses the logistic loss function for optimization and L-BFGS(Limited-memory Broyden-Fletcher-Goldfarb-Shanno) as an optimization routine used when there is limited main memory.

3 Dataset

We worked on a labeled dataset from YouTube. We only considered input attributes video-level features instead of frame features because we processed data on video levels and the video features were the mean values of all the frames a video contained. For example, the RGB values in a video feature are the global mean of the RGB values in all frames of that video. Our model classified these videos into different class labels, such as politics, wildlife, music, food, hotels and so on. And we had our target attribute a set of selected 'classes'. Since there were thousands of different topic classes in the original dataset, we implemented the actual classification on selected 10 classes to limit our data size. In order to do so, we preprocessed the dataset and extracted the id, RGB value and label of every video-level feature to determine the certain set of classes.

3.1 Dataset Description

Why Youtube-8M Dataset?
When we searched a video on YouTube for given keywords, almost all of the search results had the

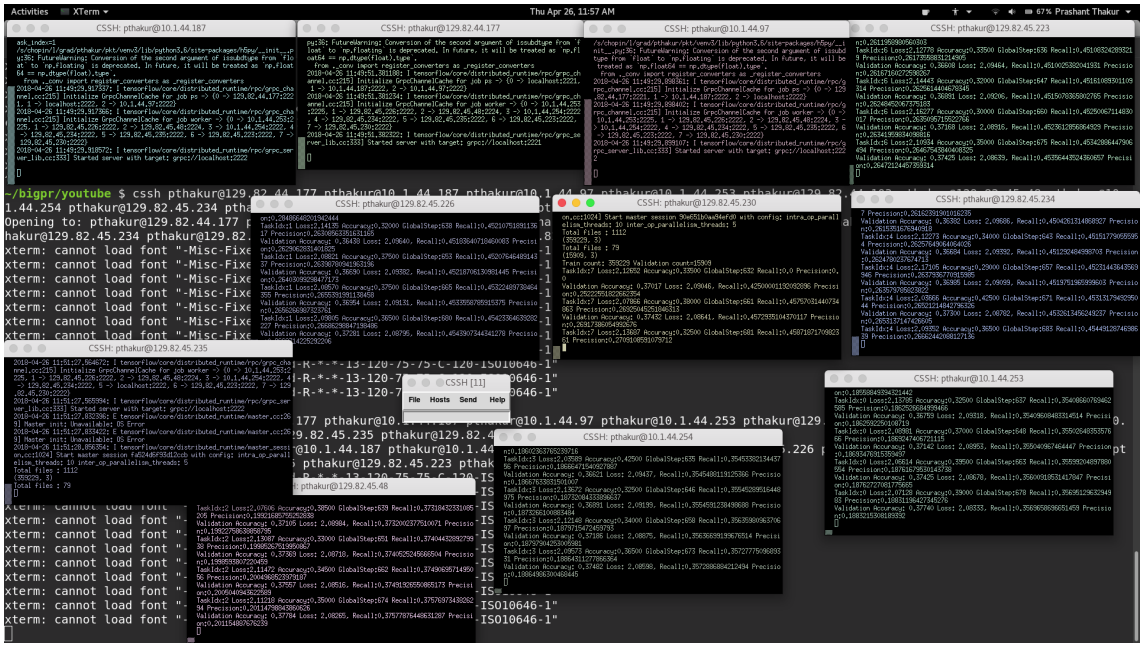

Figure 3: Screenshot of TensorFlow training in cluster

keyword either in a video title or in a description. However, predicting any video class based on its video-level features would provide the following benefits :

- Optimize videos filtering regardless of the misleading titles and descriptions.
- Eliminates the term-spam for the videos as video-level features are used for classifying. For example, try searching simple query on youtube as "Black Panther full movie". The results are highly viewed videos which are either the trailers or long videos containing links highly possibly a spam.
- Easy implementation of a control mechanism for videos recommendation, parental control for a video channel.
- Use video analysis to categorize any video for detecting emergency conditions under real-time CCTV feeds for traffic control, emergency medical assistance and several other.
- With optimization, it can be used to detect Copyright Infringement by correlating different videos.
- Finally, due to the dataset large size, it is well suited to solve this as a big data problem.

Youtube 8M dataset - YouTube-8M is a large-scale labeled video dataset that consists of millions of YouTube video ids, RGB values and associated labels from a diverse vocabulary of 4700+ visual entities.

Link for dataset: `https://research.google.com/youtube8m/download.html`

This dataset is one of the largest multi-labeled video classification dataset, composed of around 8 million videos each about 120 to 500 seconds long (total 500,000 hours of videos) in the past 50 years, also each video has at least one entity from target classes, the average number of labels per video is 3.4. The (multiple) labels per video are Knowledge Graph entities, with the total number 4800, being organized into 24 top-level verticals. Each entity represents a semantic topic that is visually recognizable in video, and the video labels reflect the main topics of each video. The dataset was generated by Video Understanding group within the Machine Perception Research organization at Google using a YouTube video annotation system, which labeled videos with their main topics. The labels are machine-generated but they have high-precision and are derived from a variety of human-based signals including metadata and query click signals. The dataset is created in form of tfrecords. So we can directly read the entire dataset into TensorFlow.

3.2 Video-level Features

As mentioned in previous section, we only analyzed the video-level features. Video-level features are stored as TensorFlow.Example protocol buffers. A TensorFlow.Example proto is shown in Figure 2. Each video-level feature contains attributes video ID, one or multiple label, mean RGB values and mean audio features.

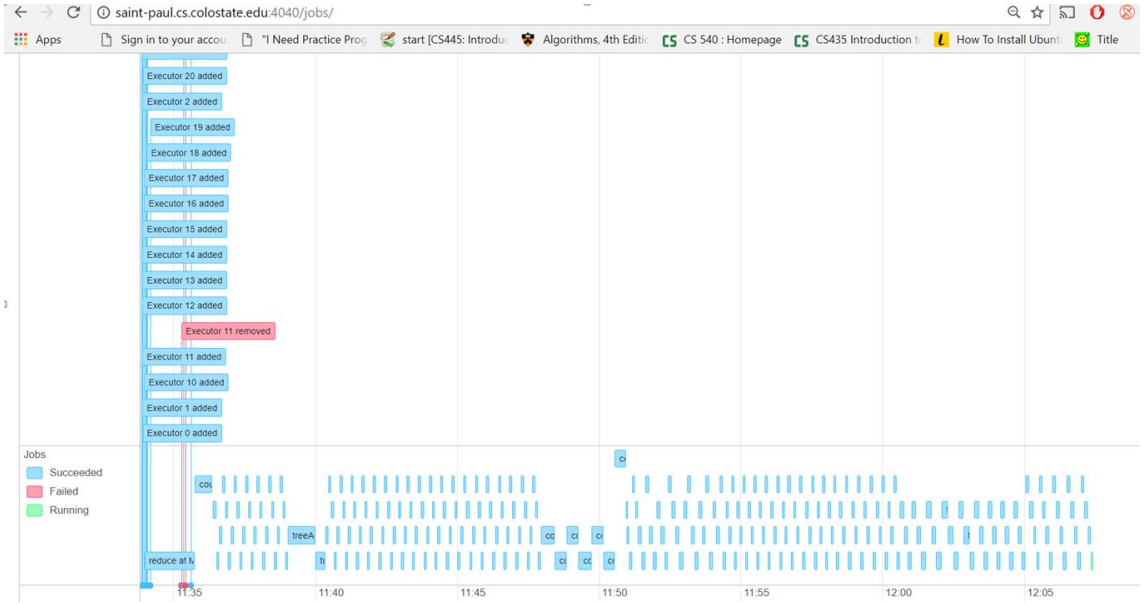

Figure 4: Event timeline in Apache Spark

4 Experiment and Discussion

4.1 Experimental Environment Setup

We conducted data processing and Classification in cluster mode which contains 8 desktop computers running Linux operating system (4.13.5-100 kernel, with 32GB memory and 16 2.60GHz Intel Core E5-2650 processors.) The number of worker nodes been used between 1 to 8, included parameter manager and chef worker. The programming language is Python and Scala for TensorFlow and Spark, respectively. We used maximum 8 nodes using following values-
SPARK_WORKER_INSTANCES=4
SPARK_WORKER_CORES=8
SPARK_WORKER_MEMORY=4g

A run time screenshot when one node in TensorFlow is training in cluster mode is shown in Figure 3. Figures 1 and 4 are snapshots of the web UI provided by Spark. Figure 1 shows the DAG visualization of job multilayer perceptron classifier using Mllib. Figure 4 is the timeline in Spark illustrates how jobs are added and executors are assigned and removed, shows the active and pending stages of jobs, including the number of tasks completed.

4.2 Classification Performance Evaluation

We evaluated two frameworks' performances based on four main aspects:

- Percentage of accuracy versus the number of iterations when training the model.
- Log loss tendency (cross entropy obtained from Logistic Regression model). For assessing the performance of our classifier, we used Logarithmic Loss function. Log loss measures the performance in terms of accuracy by penalizing false classifications. The log loss increases as the predicted probability diverge from the actual label. So log loss as 0 means a perfect model.

  We successfully had log loss from TensorFlow implementation. However, the multilayer perceptron classifier from Mllib didn't provide with training summary for the Neural Net. This was the limitation we found while using the Mllib.
- Time cost versus the number of iterations.
- Precision and recall values which measure the proportion of true positive classification result in percentage. Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. Both precision and recall are therefore based on an understanding and measure of relevance.

Evaluator	Framework	
	TensorFlow	ApacheSpark
Accuracy	0.69	0.69
Precision	0.7	0.65
Recall	0.65	0.67

Figure 5: Accuracy, precision and recall values of TensorFlow versus Apache Spark

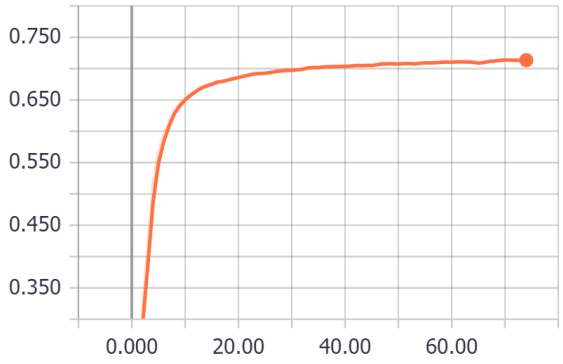

Figure 6: Classification accuracy in TensorFlow

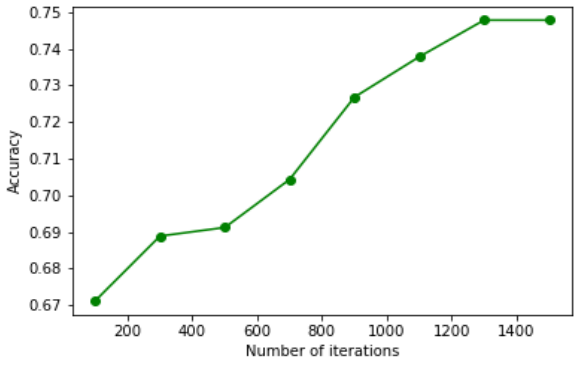

Figure 7: Classification accuracy in Apache Spark

Precision and recall can be defined as:

$$Precision = \frac{tp}{tp + fp}$$

$$Recall = \frac{tp}{tp + fn}$$

Where tp in the above equation represents true positive rate, fp means false positive rate and fn indicates false negative rate.

4.3 Result Discussion

Figure 5 indicates an overall quantitized performance of both frameworks in terms of accuracy, precision and recall values. The accuracy of two frameworks are almost the same, 69%, the precision of TensorFlow is 70%, slightly higher than 65% of Spark, while in terms of recall value Spark is better, 67% compared with 65%.

We deployed total 8 nodes to run the training jobs, with the learning rate 0.001. A one time training result are shown in Figure 6 and 7, the horizontal axis is the number of iterations of training, the vertical axis is the accuracy in percentage measures if the predicted class matches the target class. In both figures the accuracy continuously increasing in the whole process. The tendency of TensorFlow accuracy towards stability after 74 iterations while the tendency in Spark becomes stable after nearly 1300 iterations. At last the accuracy difference between TensorFlow and Spark is not huge, around 71% and 75%, respectively.

We also calculated the log loss in TensorFlow as shown in Figure 8, the algorithm of log loss has been discussed in section 2.2.6. Like figures above, the horizontal axis is the number of iterations of training process, the vertical axis is the logarithmic loss computed by the system. The log loss dropped to 1.75 after 74 iterations.

We also experimented to see how much time does it take to perform small number of iterations on Spark.The result is shown in Figure 12. The figure depicts the time spent in seconds by training in Apache Spark versus the number of iterations. As we can see from the plots 40 iterations on huge datasets just took 7.9 mins(478 seconds). The time taken to train the model is nearly a linear relationship with the number of iterations. Time taken reached up to 30 mins(1300 secs) at 100 iterations.

The framework performed really good in terms of turn around time. However, we wanted to achieve

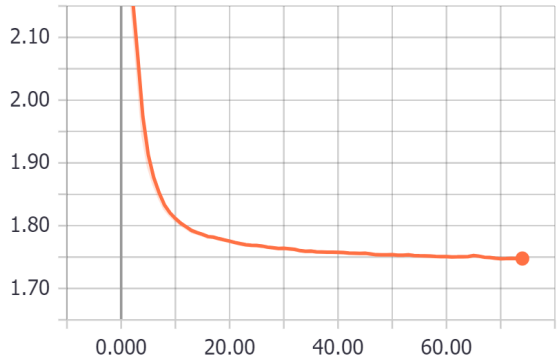

Figure 8: Classification log loss in TensorFlow

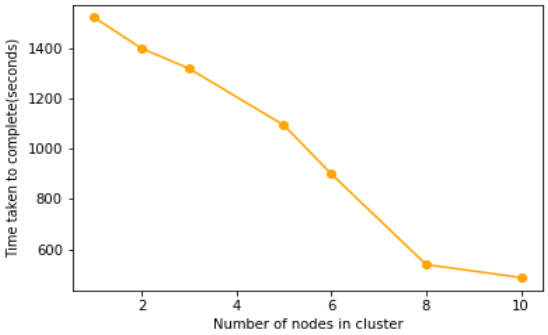

Figure 9: Time cost versus number of nodes in TensorFlow

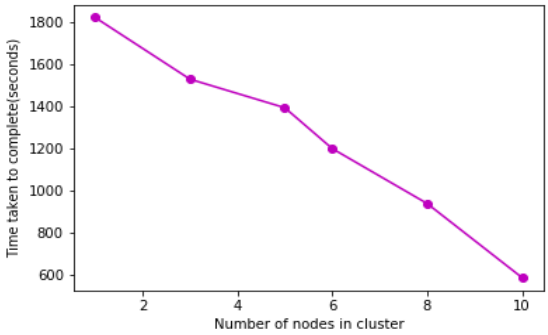

Figure 10: Time cost versus number of nodes in Apache Spark

highest accuracy possible with Youtube-8M dataset. Therefore, we measured accuracy with small number of iterations. Figure 13 shows the accuracy curve increasing as number of iterations is increased. It is still moving upwards which shows scope of improvement. However, at less number of iterations accuracy lies between 63-65% only. Therefore,it is trade-off between time consumption and accuracy.

As Figures 9, 10 shown the use of cluster has increased the performance several folds. The time to train the model with the given dataset for larger number of nodes is very less as the workers read only unique portion of data. Since training the data with the same batch on different workers would not improve the accuracy, the load can be easily distributed on several clusters. So with the increase of cluster the number of times the workers have to go through the entire dataset decreases and the time taken to compute the given iteration on different cluster size differs. Figure 11 shows the benefits of the distributed environment. When the number of nodes increases the time taken to achieve a given loss decreases drastically. Similarly, the accuracy on the dataset was also achieved faster in case of the larger cluster which is complementary to loss function.

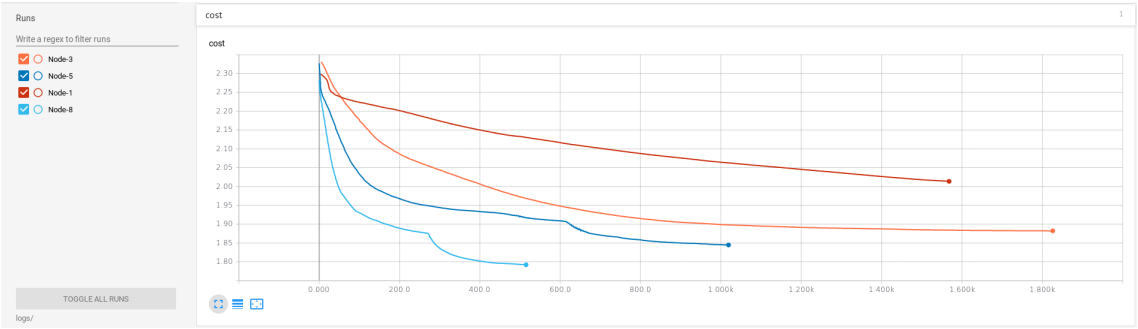

Figure 11: Time(sec) versus Loss/Cost for different number of Nodes in cluster.

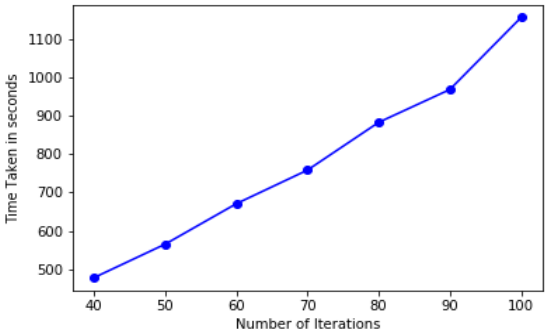

Figure 12: Time Taken versus maximum iterations

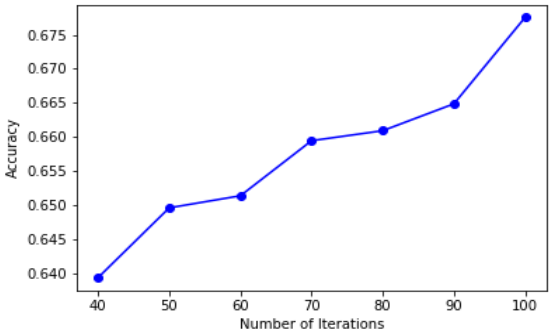

Figure 13: Accuracy versus Number of Iterations from 40 - 100

5 Conclusion & Future Work

While dealing with two popular machine learning libraries – TensorFlow and Spark Mllib, we noticed that they both have their own advantages. The TensorFlow is very well designed when it comes to the computation of different parameters during the training and testing phase. The computation speed of TensorFlow was fast in compared to the Spark Mllib. In the distributed setup the computation of both the framework got better with the increase in the number of cluster size. We found that TensorFlow is better at computing the task better in standalone mode and cluster mode as well. Though spark Mllib is hand in hand. However, while designing the TensorFlow we noticed that the data is loaded in all the workers node because of which the total memory utilization of the entire cluster is higher compared to the Spark where the workers read the corresponding partition of the data allocated to them.

Adding the number of nodes also increases the way the computations are done in both the framework and the time taken to train the model gets lesser with the number of nodes. We, however, did not test the nature with the increase in the cluster size. We did our computation for 1, 3, 5, 8 nodes only. This could be one of the implementations that can be carried out in the future to see if adding the number of nodes has any saturation effects, i.e. the model does not get better with the increase in the number of nodes.

Working with the distributed TensorFlow was also difficult because it needs to be configured manually and maintaining the project once the cluster is large is very tedious. Additionally, the jobs needed to be submitted on every node with corresponding task_index which created much overhead. Therefore, TensorFlow is generally used with Spark, Kubernet, Yarn or similar cluster managers. The future implementation of the benchmarking can also be done by implementing the clustering on popular cluster managers. On the other hand, Spark cluster manager is very clean and it handles all the burden that the users can face.

So the selection of framework boils down to the requirement and the ease and control the users require. If the speed is the required criteria with no limitation on network communication, memory usage per worker nodes then TensorFlow is the best match. However, if the cleaner interface is required with better ways of handling things Spark Mllib could be a better choice.

Other than frameworks, our classifier gave approximately 75% accuracy. In future, we can explore the frame-level features of the video instead of video-level features. We can clearly see, large dataset doesn't always mean better classification. It will be fun to explore the performance by training the model using multi-labels as target classes.

6 Project Contribution

Prashant	Paahuni	Heting
TensorFlow cluster setup and performed data filtering	Setup cluster for Spark, Data modification	Analyzed the dataset and extracted the required information
Formulation of the algorithm using TensorFlow and tuning the parameters	Experimenting parameters to get better performance from Mllib classification on Spark.	Learned about different evaluation methods to evaluating our model
Verified the results and improved the method to reduce errors	Collected evaluation metric with different iterations and different cluster setup	Helped in performing TensorFlow experiments on cluster
Analyzed the results and performed the comparative study between TensorFlow and Spark	Helped in doing the comparative study between two frameworks	Wrote Project report and performed analysis on results from both the frameworks

References

[1] TensorFlow DNNClassifiers,
`https://www.tensorflow.org/api_docs/python/tf/contrib/learn/DNNClassifier`

[2] Mllib - Classification and Regression,
`http://spark.apache.org/docs/1.2.1/mllib-classification-regression.html`

[3] Hadoop MapReduce Tutorial,
`https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html`

[4] YouTube-8M Dataset,
`https://research.google.com/youtube8m/`

[5] Distributed TensorFlow,
`https://www.oreilly.com/ideas/distributed-tensorflow`

[6] TensorFlow Vs. Spark: How Do They Differ And Work In Tandem With Each Other,
`https://analyticsindiamag.com/tensorflow-vs-spark-differ-work-tandem/`

[7] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan and Sudheendra Vijayanarasimhan. (Google Research) [*YouTube-8M: A Large-Scale Video Classification Benchmark.*] abs/1609.08675, 2016.

[8] Janani Gururam. [*Performance Tuning and Evaluation of Iterative Algorithms in Spark.*] 2016.