BAIPAS: Distributed Deep Learning Platform with Data Locality and Shuffling

Mikyoung Lee, Sungho Shin, Seungkyun Hong, and Sa-kwang Song

***Abstract*—**In this paper, we describe a distributed deep learning platform named BAIPAS, a Big Data and AI based Predication and Analysis System. Recently, research on deep learning using big data has been actively carried out. For deep learning to use big data, it takes a lot of time to learn with the training data. To reduce training time, there is a method that uses distributed deep learning framework. However, due to the size of storage, there is a problem that the full training data can't be used for deep learning. When the big data is in external storage, training takes a long time because it requires additional training time due to network I/O and bottlenecks for data to be loaded during deep learning operations. To solve this problem, we propose BAIPAS with data locality module as a way to reduce training time with big data. In a cluster environment consisting of a master server and worker servers, the master server distributes training data to worker servers using optimal scheduling method. The goal of BAIPAS is support to enable to make effectively deep learning model, to easy install and monitoring of the platform. In order to provide fast training speed, data is distributed and stored in worker-server storage using data locality module, and then training is performed. The data locality module analyzes the training data and the state information of the worker servers. This distributes the data scheduling according to the available storage space of the worker server and the learning performance of the worker server. The worker server quickly performs training using training data (subset data) stored in the local storage. However, if each worker server conducts deep learning using the distributed training data, model overfitting may occur, which does not occur when the method of learning uses full training data set. To solve this problem, we applied a data shuffling method that moves already learned data to another worker server when training is performed. In this way, each worker server can contain the full training data set. BAIPAS uses Kubernetes and Docker to provide easy installation and monitoring of the platform. The master server used remote commands to install the distributed deep learning platform and the libraries and source code associated with the platform on the slave servers. It also monitored the resources of the slave servers to support efficient deep learning operations. It also provides pre-processing modules, management tools, automation of cluster creation, resource monitoring, and other resources; so developers can easily develop deep learning models.

This research was supported by the Korea Institute of Science and Technology Information (KISTI).

Mikyoung Lee is with the Korea Institute of Science and Technology Information, Daejeon, 34141 Korea (e-mail: jerryis@kisti.re.kr).

Sungho Shin is with the Korea Institute of Science and Technology Information, Daejeon, 34141 Korea (e-mail: maximus74@kisti.re.kr).

Sungkyun Hong is with the University of Science and Technology, Daejeon, 34113 Korea (e-mail: xo@kisti.re.kr).

Sa-kwang Song is with the Korea Institute of Science and Technology Information, Daejeon, 34141 Korea. He is also with the professor of University of Science and Technology, Daejeon, Korea (corresponding author to provide phone: +82-42-869-0757; fax: +82-42-869-1133; e-mail: esmallj@ kisti.re.kr).

***Keywords*—**Distributed Deep Learning Platform, Data Locality, Data Shuffling, Deep Learning Platform, Big Data

I. INTRODUCTION

A few years ago, since the advent of Google's AlphaGo[1, 2], many people have become interested in deep learning technology. Recently, deep learning technology has been actively researched in many fields, such as image processing, video recognition, speech recognition, natural language translation, autonomous vehicles, and artificial intelligence robot. Also, it is applied to various research fields such finance, medicine, art, natural science and so on. Deep learning deals with big data and is now being used as an alternative option to support existing research methods in many areas.

The performance of existing machine learning is well below the level of people, and training from data takes a long time because of hardware limitations. However, machine learning is being used in more fields as the range of problems that can be processed by machine learning has widened. This is due to the recent development of deep learning, the activation of big data, and the development of hardware. The biggest advantage of deep learning is that you can make the best decisions based on massive amounts of data. Deep learning trains data on its own like a human being and finds optimized values when there is more data available and training is repeated. Much computing power is needed to improve the accuracy of data analysis. This is because, as the size of the learning model increases, the amount of data that needs to be computed increases.

Currently an issue has arisen about storage of training data in deep learning using big data. When the capacity of the entire amount of training data exceeds the storage capacity of the node, it is necessary to learn using a subset of the entire training data. To solve this problem, large capacity external storage (NAS, Luster, HDFS, etc.) is used, but it slows down training speed due to network I/O and bottlenecks. We are developing a deep learning model using image data with a size of over several hundreds of TB, and we aim to develop a platform that shortens training time and improves learning effectiveness. Therefore, it was very important find a way to reduce the time needed to train with large amounts of data.

This paper introduces the BAIPAS platform using data distribution and shuffling to train a large amount data in a distributed deep learning environment.

II. Related Work

A. TensorFlowOnSpark

TensorFlowOnSpark (TFoS) [3, 4], developed by Yahoo, is a framework that enables distributed TensorFlow training and inference on Apache Spark clusters. TensorFlowOnSpark enables distributed deep learning on a cluster of GPU and CPU servers. This can be used on the platform, with minor code changes, to run existing TensorFlow programs. TFoS supports all TensorFlow functions, including synchronous /asynchronous learning, model/data parallelism, and TensorBoard. It provides architectural flexibility for data ingestion to TensorFlow (pushing vs. pulling) and network protocols (gRPC and RDMA) for server-to-server communication. Its Python API makes the integration with existing Spark libraries like MLlib easy. The speakers will walk through multiple examples to outline these key capabilities, and share benchmark results about scalability. This provides fast training through server-to-server communication. It has the advantage of using HDFS and Spark. However, TFoS is undergoing frequent system upgrades, making it unsuitable for stable use.

B. BigDL

BigDL [5, 6] is distributed as a deep learning library for Apache Spark and is highly scalable. It is an open source distributed deep learning framework for big data platform developed by Intel. With the help of BigDL, we can write directly into a spark program on a Spark or Hadoop cluster to run the deep learning application directly. It provides rich training support and uses Intel's Math Kernel Library (MKL) to ensure high performance. Using BigDL, we can load a pre-trained Torch or Caffe model into Spark. It is a very useful library that we can use if we want to add deep learning to large data sets stored in a cluster. BigDL also provides 100+ basic neural networks building blocks allowing users to create novel topologies to suit their unique applications. Thus, with Intel’s BigDL, the users are able to leverage their existing Spark infrastructure to enable Deep Learning applications without having to invest into bringing up separate frameworks to take advantage of neural networks capabilities. However, it does not support various advanced models.

C. TensorFrame

TensorFrames [7, 8] was developed by Databricks, and provides a bridge between Spark and the TensorFlow framework. This involves native embedding of TensorFlow in Spark Dataframes. If TensorFlow is used directly in Spark, it is inefficient because of the need to go through the process of object conversion. However, with TensorFrames much better performance is possible when working with TensorFlow on Spark. TensorFrames uses Spark's DataSet/DataFrame API and has in-depth knowledge of the memory-efficient data representation in Spark, minimizing memory redundancy between the two frameworks. TensorFrames can be run in Python and Scala. However, this has the disadvantage of providing data parallelism only in the distributed training stage.

III. Platform configuration

A. Platform Install

The library installation required for the BAIPAS Deep Learning Platform is shown in Fig. 1. The operating system installed on the platform was Linux CentOS. We installed the Nvidia driver to accelerate the use of GPU in the Linux environment. Docker [9] (a tool for building a virtual environment) and Nvidia-Docker (a docker addition module) were installed to utilize the host's GPU in a virtual environment. Our platform used Kubernetes to execute and monitor commands in a distributed environment using a master and slave servers. Kubernetes [10] is powerful container management software developed by Google. It monitors, replicates, and deploys Docker containers.

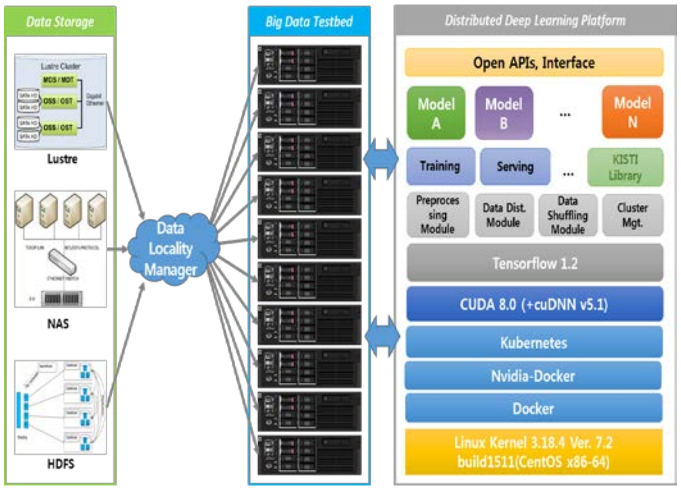

Fig. 1. Platform diagram

CUDA [11] is a GPGPU technology for accelerating the use of GPUs. It installs CuDNN [12] to support the CUDA-based Deep Neural Network library. We installed the TensorFlow deep learning library [13] to support distributed deep learning. After all the basic essential libraries were installed, the modules developed by the BAIPAS Deep Learning Platform were installed. These consisted of a data preprocessing module, a data distribution module, and the cluster monitoring and automatic management modules. In addition, training and serving are provided by TensorFlow, and the library necessary for the platform is developed directly and provided to the user. This is a system that generates various prediction models and makes inferences.

B. Configuration of Clusters

The clusters of BAIPAS built on each server are shown in Fig. 2. One of the 10 nodes was used as a master server, and nine were configured as slave servers. A distributed deep learning platform was installed in the virtual container of each slave node. Each virtual container using Docker included

TensorFlow, related library images, and platform modules. Nodes in a cluster are easily scalable. We install the Kubernetes on the node to be added and register it as a slave server to the master server. Then, we only need to install the platform-specific image on the docker of the node where the master server is added.

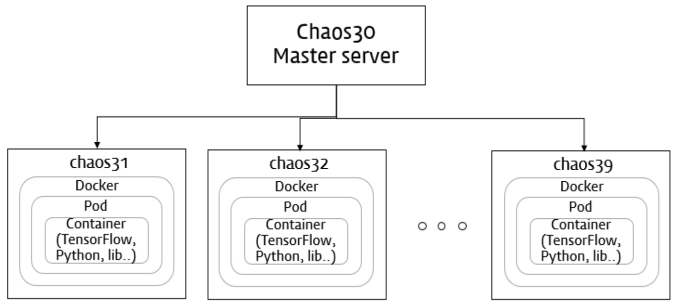

Fig. 2. Configuration of clusters

IV. Distributed deep learning platform

A. Overview

The goal of our platform is to quickly and effectively support making prediction models using big training data. We focused on quick training with big data, easy installation and maintenance of the platform.

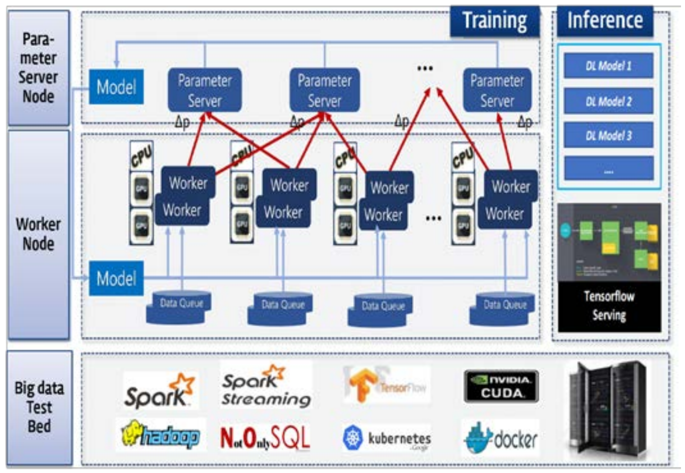

Fig. 3. BAIPAS distributed deep learning platform conceptual diagram

We designed the BAIPAS to ensure fast training speed. BAIPAS is an optimal deep learning platform that improves learning efficiency of big data. The platform performs learning by distributing data to the local repository of each worker server to process large amounts of data. Therefore, the problem of slow learning due to network I /O and bottleneck is solved. It has the function to preprocess the data in a format suitable for Tensorflow. It also maximizes efficiency of CPU/GPU in platform by using high-speed operation support library. This platform improves the accuracy of the deep learning model by data shuffling method. This template is related to the creation of distributed clusters and the configuration of the distributed environment in TensorFlow. Users can easily install, use and operate this platform. The master server used remote commands to install the distributed deep learning platform and the libraries and source code associated with the platform on the slave servers. It also monitored the resources of the slave servers to support efficient deep learning operations. It also provides templates for model developers to easily develop a distributed deep learning model.

B. Data Locality

The distributed deep learning method implemented in BAIPAS is different from the data parallelism method used in a general distributed deep learning framework. Data parallelism [14] is the process of distributing data across different nodes, which operate on the data in parallel computing environments. This is similar to training with distributed data in parallel on multiple worker servers. It is training with the entire data set. However, worker servers on BAIPAS train using different subsets of the same data set (see Fig. 4). When training, each worker server is able to train using the entire data set by use of a shuffling method that moves subsets of the trained data among the servers.

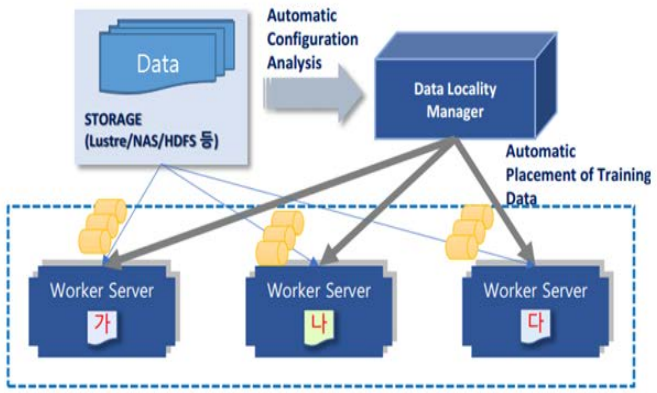
Data
STORAGE (Lustre/NAS/HDFS 등)
Automatic Configuration Analysis
Data Locality Manager
Automatic Placement of Training Data
Worker Server
가
나
다

Fig. 4. Data locality module

Our goal was to solve the slow learning speed due to network I/O and bottleneck through distributed training data in each node. This is a method for efficient storage considering the size of the entire data set, the space available for storing the worker server, and the performance of the worker server. The movement of the subsets was handled by the data locality module, and it operated as follows.

1) Analyze status information of training data and worker servers

The master server stores status information, such as the data name, data size, data format, total number of data sets, and total size of the data set. It stores state information of the worker servers, such as the size of the worker server, the amount of available storage space, CPU/GPU information, total number of worker servers, and available space information. We use the analyzed state information for distribution scheduling.

2) Data distribution scheduling

The idea of basic data distribution scheduling is to equally distribute the training data among all the worker servers. However, there are some things to consider: the total size of the data set, the available space on the server's local storage, and the CPU/GPU performance of the server.

- First, if the performance and available local storage space of each worker server are the same, and the total size of the data set is smaller than the sum of the available space, each data subset is sequentially distributed to each server's local storage and stored.
- Second, when the size of the learning data is too large, the worker server distributes as much of the data as can be stored and proceeds with learning. The worker server deletes the learned data in the shuffling step and shuffles unallocated data from the external storage to itself to proceed with learning.
- Finally, the speed of learning depends on the performance of the CPU/GPU and the number of GPUs. Thus, the amount of data distributed is in proportion to the learning performance of the worker server. The faster the learning performance, the more data is given. Then, the remaining data is distributed according to the ratio of the available storage space on the worker server with the highest performance.

C. Data Shuffling

In our BAIPAS platform, each worker server has a subset of the entire training data set, so overfitting can occur during training. To solve this problem, we performed data shuffling by moving already learned data to another server during training in mini- batch units (see Fig. 5). This method allows all worker servers to learn from the full data set. Shuffling is an ongoing process during training. Therefore, no extra shuffling time was required. When training with the mini-batch unit was completed, the work was terminated, even if all of the data targeted for movement had not been moved. For learning the next mini-batch unit, training data was selected from the newly shuffled data. This method achieved the same model accuracy as learning from all the data.

The data shuffling method is as follows.

- First, the worker server performs learning by a predetermined amount of mini-batch size data stored in each server.
- Second, to perform 1 epoch, n mini-batches are performed. During the training of mini-batch units, the data used in the previous mini-batches are moved. (File movement does not occur during initial learning.)
- Third, the target data is sequentially moved to the next worker server (copy and delete) on a file basis. When one piece of data is copied to another server, the copied data is deleted from the corresponding server.
- Finally, moving work progresses sequentially while mini-batch unit learning occurs. When learning ends, only move to the file that is copied now, and work is started from a new one. When the next mini batch is started, the moving operation is performed based on the new mini batch data set.

However, the shuffling does not sequentially send and receive files as shown in Fig 5. Data shuffling is possible while both servers are training. In addition, if data is shuffled at a maximum of 100%, all the data can be learned, but there is a possibility that the data will not be completely shuffled.

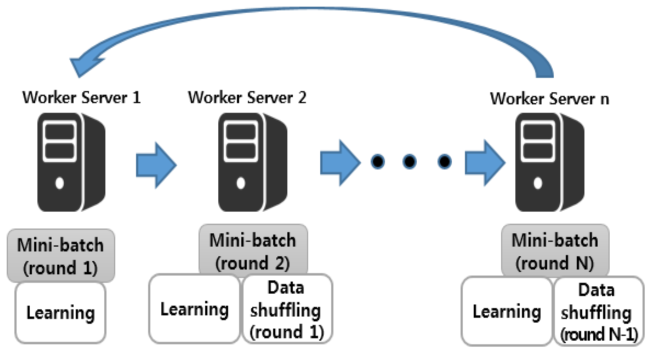

Fig. 5. Data shuffling concept

V. Experiments

A. Training Speed Test

We compared the training speed of the BAIPAS and distributed TensorFlow that load training data from external storage. The test environment is as follows. One parameter server and 3 worker servers were used in the experiment. Each server has 2 GPUs (12 Gbytes) and 128 Gbytes of memory. We selected DCGAN with a slow learning speed as a test algorithm. The training data used in the experiment was 61 Gbytes in size and the number of instances was 10,000. The mini-batch size was 20, the iteration was 500, and the epoch was 24 times.

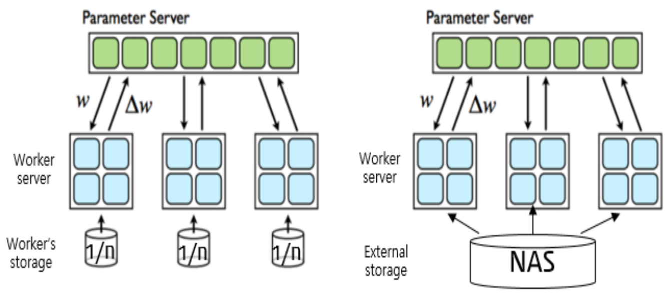

Fig. 6. BAIPAS(L) vs. Distributed TensorFlow(R) test concept

In the test results as shown Table1, the learning time of the mini-batch unit was 36.53 seconds for distributed TF learning using external storage. The training time of BAIPAS was 30.98 seconds. BAIPAS had a faster average training time than distributed TF because of the network I/O and the bottleneck. Several nodes simultaneously access external storage and load data so bottlenecks occur. The remaining nodes wait until the node that has preempted the resource finishes the work, so if the number of nodes is large, the learning time is longer.

Table 1. Distributed TF vs. BAIPAS learning time results

	Distributed TF	BAIPAS
Server 1	34.39852295	30.70884229
Server 2	32.44939505	31.79591312
Server 3	42.7465349	30.46060936
Average speed	36.5314843	30.98845493

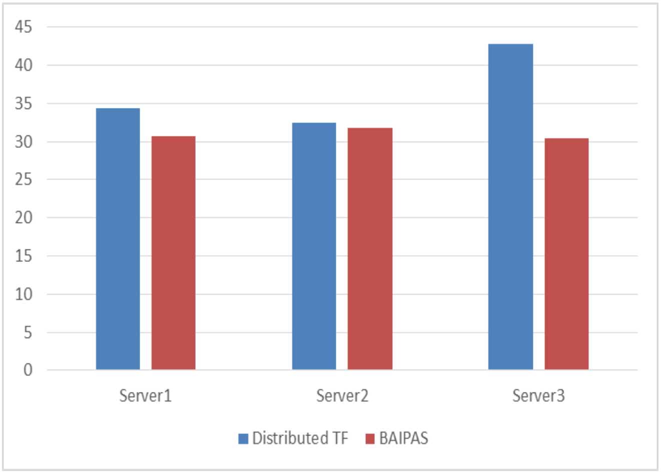

Fig. 7. Learning speed test results

B. *Platform Stability Test*

We tested the distribution and training speed of large amount of data of BAIPAS. This task is to determine platform stability and issues. We tested the problems and weaknesses of BAIPAS by learning massive typhoon satellite images.

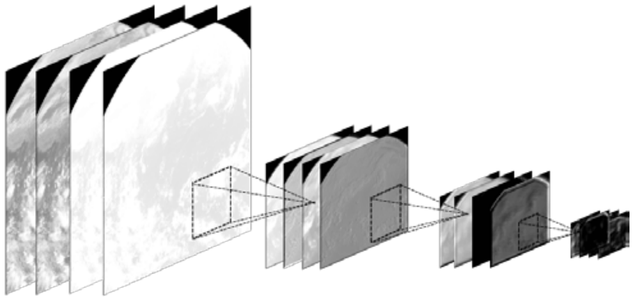
Fig. 7. CNN based algorithm to recognize the eyes of the typhoon

The test environment is as follows. Three parameter servers and 6 worker servers were used in the experiment. Each server had 2 GPUs (12 Gbytes) and 128 Gbytes of memory. We used a CNN-based algorithm to perform the task of recognizing the eyes of the typhoon. The total size of the data was about 2 TB and the number of data items was 90,715. Epoch was run 20 times. The mini-batch size was 40.

The test results were as follows (see Table 2). It took about 25 hours to distribute the data to the six worker servers. The training time required for each node was 50 hours and 5 minutes.

Table 2. Platform stability test result

Node Name	File #	Data size	Training Time
Chaos 32	15097	336.5GB	50:05:22.849968
Chaos 34	14960	333.4GB	31:57:17.088341
Chaos 35	14936	332.9GB	26:19:37.918942
Chaos 36	15239	339.6GB	8:29:04.113279
Chaos 37	15559	346.8GB	31:36:46.99236
Chaos 39	14924	332.6GB	45:54:34.454741

VI. CONCLSUIONS

In this paper, we introduced BAIPAS (Big data and AI based Predication and Analysis System), a distributed deep learning platform. When developing a deep learning model, it takes a lot of training time to learn big data. Also, if the size of the data is larger than the size of the storage, the full data can't be stored and learned. When training data stored in external storage, training speed is slowed down by network I/O and bottleneck.

To solve this problem, BAIPAS is done by distributing data to multiple worker servers in order to reduce the time required for big data learning. Our goal was to reduce the network I/O time required when training with big data stored on external storage devices. Then, each worker server learns using the subset training data stored in the local storage. Each worker server shuffles the trained data to another worker server to learn another subset of the full data set. To install the necessary libraries and environments on the platform, it is possible to install, operate, and utilize the necessary tools through the master server without installing them directly on each worker server. Therefore, the platform has easy extensible. BAIPAS provides management tools, including clustering monitoring, which make using it easy for developers of deep learning models.

In the future, we will improve the performance of the platform by improving the data scheduling method and optimizing the training speed.

REFERENCES

[1] AlphaGo, https://en.wikipedia.org/wiki/AlphaGo
[2] D. Silver, A. Huang, and A. Guez et al., "Mastering the game of Go with deep neural networks and tree search", Nature 529, pp 484-489, doi:10.1038/nature16961, 2016.
[3] TensorFlowOnSpark, https://github.com/yahoo/TensorFlowOnSpark
[4] Lee Yang, Jun Shi, Bobbie Chern, and Andy Feng, "Open Sourcing TensorFlowOnSpark: Distributed Deep Learning on Big-Data Clusters", http://yahoohadoop.tumblr.com/post/157196317141/open-sourcing-tensorflowonspark-distributed-deep, 2017.
[5] Sergey E. "BigDL: Distritubted Deep Learning on Apache Spark",

https://software.intel.com/en-us/articles/bigdl-distributed-deep-learning-on-apache-spark, 2017.

[6] BigDL, https://github.com/intel-analytics/BigDL

[7] Tim Hunter, "TensorFrames: Deep Learning with TensorFlow on Apache Spark", https://databricks.com/session/tensorframes-deep-learning-with-tensorflow-on-apache-spark, Spark summit, 2016.

[8] TensorFrames, https://github.com/databricks/tensorframes

[9] Docker, https://en.wikipedia.org/wiki/Docker_(software)

[10] David Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," IEEE Cloud Computing, vol 1, issue 3, pp81-84, 2014.

[11] CUDA Nvidia. Cublas library. https://developer.nvidia.com/cublas

[12] S. Chetlur, C. Woolley, P. Vandermersch, "cuDNN: Efficient primitives for deep learning," arXiv:1410.0759, 2014.

[13] M. Abadi, A. Agarwal, and P. Barham et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," arXiv:1603.04467v2, Google Research whitepaper, 2016.

[14] C. Chambers, A. Raniwala, F. Perry and et al. "efficient data–parallel pipelines," In ACM Sigplan Notices, volume 45, pp 363-375, ACM, 2010.