An analysis of image storage systems for scalable training of deep neural networks

Conference Paper · April 2016

CITATIONS
4

READS
3,204

3 authors:

Seung-Hwan Lim
Oak Ridge National Laboratory
30 PUBLICATIONS **332** CITATIONS
SEE PROFILE

Steven Robert Young
Oak Ridge National Laboratory
29 PUBLICATIONS **278** CITATIONS
SEE PROFILE

Robert Patton
Oak Ridge National Laboratory
72 PUBLICATIONS **363** CITATIONS
SEE PROFILE

An analysis of image storage systems for scalable training of deep neural networks

Seung-Hwan Lim
Oak Ridge National Laboratory
lims1@ornl.gov

Steven R. Young
Oak Ridge National Laboratory
youngsr@ornl.gov

Robert M. Patton
Oak Ridge National Laboratory
pattonrm@ornl.gov

Abstract

This study presents a principled empirical evaluation of image storage systems for training deep neural networks. We employ the Caffe deep learning framework to train neural network models for three different data sets, MNIST, CIFAR-10, and ImageNet. While training the models, we evaluate five different options to retrieve training image data: (1) PNG-formatted image files on local file system; (2) pushing pixel arrays from image files into a single HDF5 file on local file system; (3) in-memory arrays to hold the pixel arrays in Python and C++; (4) loading the training data into LevelDB, a log-structured merge tree based key-value storage; and (5) loading the training data into LMDB, a B+tree based key-value storage. The experimental results quantitatively highlight the disadvantage of using normal image files on local file systems to train deep neural networks and demonstrate reliable performance with key-value storage based storage systems. When training a model on the ImageNet dataset, the image file option was more than 17 times slower than the key-value storage option. Along with measurements on training time, this study provides in-depth analysis on the cause of performance advantages/disadvantages of each back-end to train deep neural networks. We envision the provided measurements and analysis will shed light on the optimal way to architect systems for training neural networks in a scalable manner.

1. Introduction

Deep learning methods are under the spotlight primarily because of their ability to find hierarchical patterns in high-dimensional data [26]. Exemplary problems that favor deep learning are visual object recognition in images [33] and speech recognition in wavelets [17]. Training these neural network models over large datasets achieved a break-through in machine learning and artificial intelligence, which often requires significant effort to architect the system [5, 7, 11].

When using popular machine learning algorithms, including deep learning algorithms, a common practice is to use existing platforms, instead of developing new software for each algorithm [21]. Deep learning is not an exception. There have been several efforts to develop dedicated systems for training deep learning models [5, 11]. Caffe [19], Theano [2], and Torch [8] are just a short list of popular general purpose deep learning platforms. Google has recently revealed that they developed a general purpose machine learning platform, including deep learning [13]. After choosing the option of platforms, another important decision should be done: *where to store training data.*

Considering specifically image data, we have multiple options to store the datasets: (1) store image files individually on a file system; (2) store a few aggregated files in a file system by putting multiple images into a file; (3) store images as values (or records) in a key-value storage (or relational database). When designing systems for training deep neural networks, prior studies [5, 11] have discussed data parallelism, which can be exercised with any of storage options. We hypothesize that the combination of the characteristics of the data sets often used in conjunction with deep learning will pose a unique challenge for choosing data storage options. In order to train a deep neural network model on image data, a large number of small images must be read, and the entire dataset must be iterated over many times.

Potential bottlenecks to store large collections of small files have been pointed out across the file system community [3, 30], but none of these studies specifically considered the problem in the context of deep learning. Hence, the scope of this study is to analyze the bottlenecks for deep neural network models to read training image samples from a storage backend, so as to validate our hypothesized research question. The use of a GPU dramatically reduces the system requirements to train a specific model over a particular data set in the same amount of time [7]. Thus, our study specifically considers the case where a GPU is utilized for training the model.

Our analysis results demonstrate that our hypothesis is true. We observe up to $17x$ slower training time when we store training images as individual image files than when we store training images as values in a key-value storage. This pattern occurs regardless of the scale of the datasets for either small MNIST set, or large ImageNet set. When the data set is small and the training model is simple, slower training still can be neglected; for instance, 20 seconds versus 400 seconds. However, for large data sets like ImageNet, training time can vary more dramatically: an hour or a day, depending on storage opstions. Such a big difference can alter the ability for researchers to explore deep learning models.

Considering the variety of deep neural network algorithms and the explosiveness of hyperparameters of any deep neural network models (the number of training weights, the way of stacking up neural network, etc.), researchers would like to train as many models as possible in order to find a model close to the best fit to given training data. If training a deep neural network model takes a day, researchers may explore only a handful of choices for either algorithms or hyperparameters of the chosen algorithm. In other words,

CONF 'yy, Month d–d, 20yy, City, ST, Country.

http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

training deep neural nets can be a delicate art to manage, or a black magic [20]. In order to overcome this situation, system designers should consider the most efficient way to utilize every bit of computing power, for which this study will provide a valuable insight.

The rest of this paper is organized as follows: § 2 overviews convolutional neural networks as an example of deep learning algorithms and presents prior work relevant to this study. § 3 describes our empirical evaluation results, along with the description of our evaluation environment. § 4 concludes this study, suggesting the future directions.

2. Backgrounds

This section describes relevant related work and backgrounds to understand the rest of this study. Among the various deep learning methods, this study chose convolutional neural networks (CNNs) to construct deep neural network models. Thus, CNNs will be the focus of this section.

2.1 Related work

This section describes two categories of prior work to this study: (1) clean and flexible framework for training large deep neural networks, with their computer architectural issues; and (2) backend image storages that considered in this study. Caffe [19] provides data scientists with a clean and modifiable framework for state-of-the-art deep learning algorithms and a collection of reference models. The framework is a C++ library with Python and Matlab bindings for training and deploying general purpose convolutional neural networks and other deep models on commodity hardware. Caffe can process 40 million images on a day on a single NVIDIA K40 GPU ($\sim$ 2.5 ms per image). Similar software packages are Theano [2], Torch7 [8], and TensorFlow [13].

Deep neural networks have been a hot topic in recent years. Since 2012, models trained by large scale deep neural networks have dominated the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), and a recent Google study reports 3.5% top-5 error, using their open-source machine learning system TensorFlow [33]. Training of large deep neural networks has been made possible by utilizing an unprecedented scale of computational resources such as a large number of CPU cores [5, 11] and a large number of GPUs [7].

Consequently, the computer architecture communities have begun paying attention to deep learning algorithms [15]. However, the aforementioned study focused on the service of trained models, instead of efficient training. As the ability to train large neural networks is at the center of the popular adoption of deep neural networks, this study focuses on the efficient training, with the emphasis of utilizing storage backends to feed image data. In principal, deep learning algorithms repeatedly read image data from storage systems until the weights converge, which results in a system that must ready each image many times. Thus, the choice of storage backends could be critical to efficiently feeding the input image data into the training model.

Let us review back-end storage systems considered in this study. In this study, we utilized two in-memory key-value storages, LevelDB [12] and LMDB [6], along with two file system based approaches, HDF5 [16] and raw image files. In addition, we considered memory buffers to hold image data in two different languages, Python and C++.

Key-value systems are widely used to store and retrieve simple information, where complex relational operations upon the data (e.g. join) are not required, such as graph analysis [10] and instant message system [14]. LevelDB is a key-value storage library written at Google that provides an ordered mapping from string keys to string values. It uses log-structured merge tree and stores each row in a compressed format. Write operations go straight into a log/memtable, which is stored in memory. This operation makes log-structured merge tree, which is also used in HBase [1], optimized for write operations [4, 9, 32]. If the memtable grows large, it flushes into a 2MB string sorted table, sstable. As the level gets deeper, the table size becomes larger and read operations might involve accessing several levels. LMDB [6] is a B+tree-based database management library modeled loosely on the BerkeleyDB API, but much simplified. The entire database is exposed in a memory map, and all data fetches return data directly from the mapped memory, so no malloc's or memcpy's occur during data fetches. As reported in a microbenchmark result [32], LMDB is optimized for read operations.

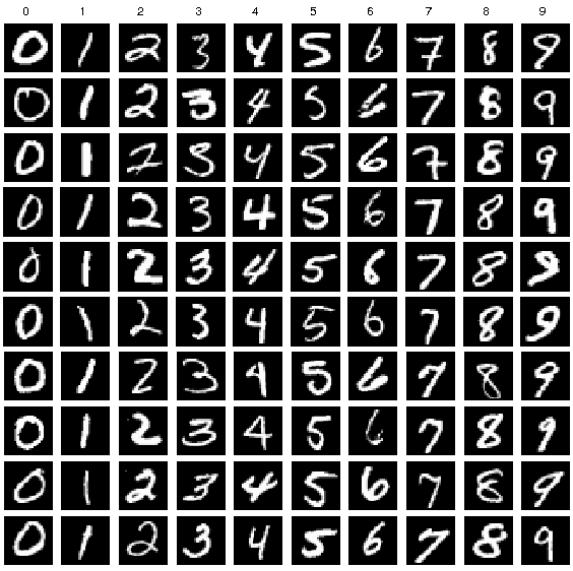


Figure 1. Example images from the MNIST dataset.

Using file systems to store and share image data is a common practice as written in articles on Facebook's photo sharing systems [3, 18]. It eliminates the efforts to extract, transform, and load the uploaded image files into another formats and systems to be shared. Another distinctive characteristic of image data is that they are typically write-once and read-many data. Even when we resize the image, we keep the original copy and store the resized copies as different files. However, the downside is that the size of individual images is not large, which burdens file systems without special treatment [3, 30]. A lesson from Facebook's photo sharing is to archive multiple images into a single large file in order to limit the total number of files in a file system, which will provide efficient file system operations, e.g. caching file system metadata and caching the file contents. In the same line, we consider using HDF5 file format [16], a widely used format by scientific community, where we can put multiple images into a single file, and APIs for HDF5 format allow us to retrieve individual images as if we were accessing them from a database.

2.2 Convolutional neural network

Convolutional neural networks (CNNs) [26] are a class of deep learning models used primarily in supervised settings, which have been successfully scaled to networks containing over 2 billion connections [5]. CNNs consists of one or more convolutional layers (often with a subsequent pooling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the spatial structure of an input image. This is achieved with local

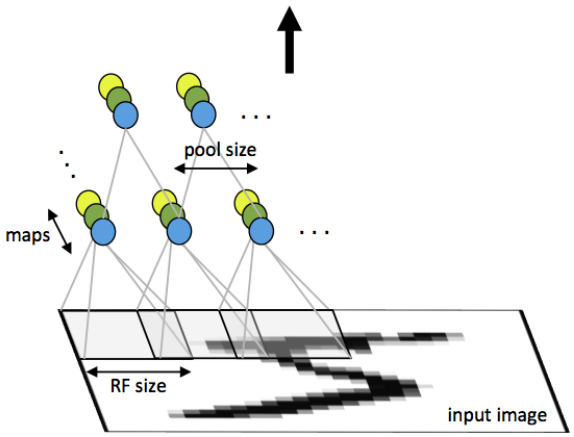


Figure 2. The first layer of a convolutional neural network with pooling [35]. Units of the same color have tied weights and units of different color represent different filter maps.

connections and tied weights followed by some form of pooling which provides translation invariant features. Another benefit of CNNs is that they are easier to train and have fewer parameters than fully connected networks with the same number of hidden units.

Let us introduce a convolutional layer. The input to a convolutional layer is a $m \times m \times r$ image where m is the height and width of the image (images used are typically square) and r is the number of channels, e.g. an RGB image has $r = 3$. The convolutional layer will have k kernels (or filters) of size $n \times n \times q$, where n is smaller than the dimension of the image and q is typically equal to the number of channels r in output of the layer below. The kernels are convolved with the image to produce k feature maps of size $\lfloor (m-n)/s \rfloor + 1$ in each spatial dimension. Each kernel can be convolved with the image using a stride size, s, which is used to reduce the size of the feature map by skipping positions in the input. The stride is often set to 1, which results in feature maps of size $m - n + 1$.

Each map is then subsampled typically with mean or max pooling over $p \times p$ regions using a stride, s, where p ranges between 2 for small images (e.g. MNIST [25]) and is usually not more than 5 for larger inputs. This results in an feature maps of size $\lceil (m-p)/s \rceil + 1$. Either before or after the subsampling layer an additive bias and nonlinearity (typically ReLU or sigmoidal) is applied to each feature map. Figure 2 illustrates a full layer in a CNN consisting of convolutional and subsampling sublayers. Units of the same color have tied weights. After the convolutional layers there may be any number of fully connected layers, otherwise known as Inner Product layers. The densely connected layers are identical to the layers in a standard multilayer neural network.

Let us calculate the number of parameters (weights) in a convolutional neural network from the trained model for MNIST data set in this study, shown in Figure 3. In this feature, the first convolutional layer has 20 feature maps, each of which generates a 2D output with the size of $\times(28-5+1) \times (28-5+1)$. The output of the first convolutional layer is subsampled by a max pool layer, which samples the max value in 2×2 area without overlapping (we have the stride size of 2). Thus, the first max pool produces 20 feature maps, each of which has the size of 12×12. Repeating the same procedure, after the second max pool layer, we will have 50 feature maps, each of which will have the size of 4×4. After that we have a fully connected Inner Product layer, which has 500 feature maps. These maps are the input to the second fully connected layer, ip2, which outputs 10 feature maps. The value in each 10 feature maps can be used to categorize the digits, ranges from 0 to 9, written in each image.

When we address the number of weights (or parameters), the first and the second convolutional layers have $20 \times 5 \times 5 \times 1 + 20$ and $50 \times 5 \times 5 \times 20 + 50$ different weights, respectively, since the connections between neurons in the same feature map share weights and each kernel has a bias weight. The third layer has $500 \times 4 \times 4 \times 50 + 500$ different weights as it is a fully connected layer and the weights are not shared across connections, and the fourth layer has $10 \times 500 + 10$ weights. Hence, we have a total of 431,080 weights (or parameters) in this model. Note that the relation between the number of parameters and the accuracy is not linear. Therefore, a larger trained model does not automatically guarantee the higher accuracy and the trained model in this study may or may not produce the best accuracy for given data sets.

Table 1. System specification

Parameter	Value
CPU cores	8
Memory	16 GB
Local storage	1.1 TB HDD
GPU	NVIDIA Titan X (12GB RAM/3072 CUDA cores)
CUDA	ver. 7.5
cuDNN	yes (ver. 3.0.07)
BLAS library	ATLAS
File system	ext4

Table 2. Training data sets

data set	storage option	size
MNIST	LevelDB	15MB
	LMDB	59MB
	pymem	14MB
	cppmem	180MB
	HDF5	180MB (a single file)
	image files	238MB(60,000 files)
CIFAR10	LevelDB	148MB
	LMDB	197MB
	pymem	530MB
	cppmem	586MB
	HDF5	587MB (a single file)
	image files	200MB (50,000 files)
ImageNet	LevelDB	222GB
	LMDB	240GB
	image files	143GB (1,281,167 files)

3. Evaluation

This section describes the empirical evaluation of the trade-offs between back-end storage systems as image storage systems for training deep neural networks. System specification of the tested system is as shown in Table 1. We employed Caffe [19] in order to obtain deep neural network models. As it is well-known that GPUs are more efficient than CPUs to train deep neural networks [7], we trained the model on GPU in this study. First, we measured training times and system resource utilization (CPU, GPU, memory, and disk) under warm cache condition. Then, we performed the same experiments with cold cache by dropping all the data pages and file system metadata (dentries and inode information) using following commands,

```
sync && echo 3 > /proc/sys/vm/drop_caches ,
```

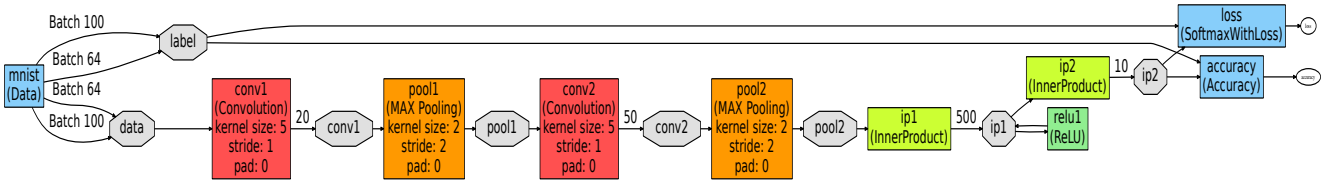
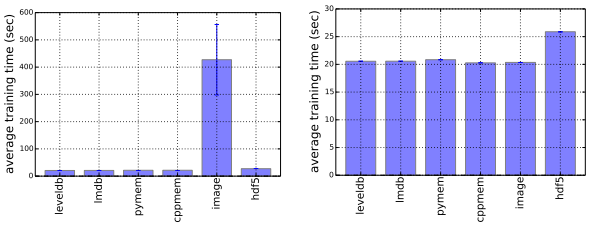
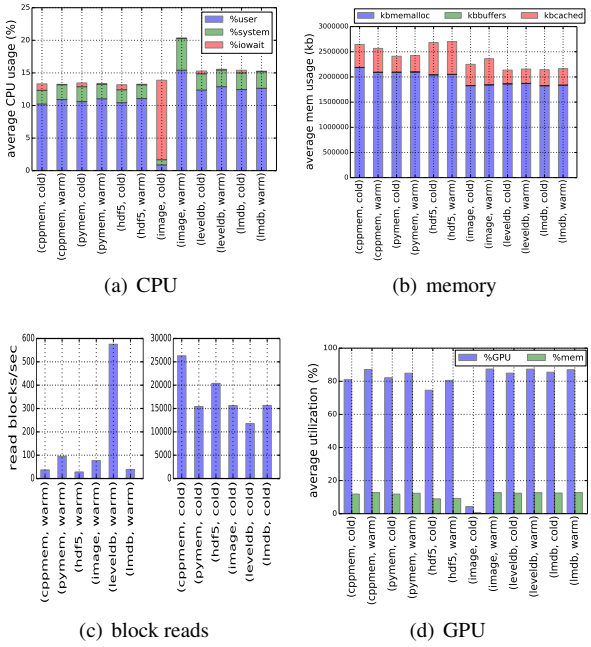


Figure 3. The trained network architecture for MNIST data set, which contains 431,080 weights. The data set consists of 60,000 28×28 pixel grayscale images.



(a) Cold cache (b) Warm cache

Figure 4. Average training time for MNIST. Error bars represent standard deviations.



(a) CPU (b) memory

(c) block reads (d) GPU

Figure 5. Resource utilization for MNIST

For each dataset, we employed different neural network models in order to represent the training of reasonably accurate models for each data set. For instance, with the MNIST data set, we trained the model with samples from 60,000 images for 10,000 iterations, obtaining 99.04% of accuracy against 10,000 images of test data set.

3.1 MNIST

The MNIST dataset [25] is a collection of handwritten digits (0-9). Figure 1 shows examples in this data set. The images are cropped and scaled to fit within a 28×28 pixel image. The images are grayscale and the majority of pixels are approximately black or white as the original images before cropping and scaling were black and white. There are $60,000$ images designated as training images, and $10,000$ images are designated as test images. This dataset was used in the early research in deep learning and is often used to quickly evaluate new ideas. The network used for this dataset is LeNet [26]. This network model is comprised of two convolutional layers, each followed by a max pooling layer, and two inner product layers with a ReLU layer between them. Complete details for the network are shown in Figure 3.

We measured the training times with different back-end storage systems to train the same network configuration. As shown in Figure 4, two key-value storage systems (LevelDB and LMDB) and in-memory options (pymem and cppmem) showed similar training time, but file based options (image and HDF5) showed longer average training time than other options. The option of image files demonstrated serious performance difference, depending on cache condition. When we warm-up the cache for the image files, the average training time was 20.40 sec, with the standard deviation of 0.070, which was similar to other options. However, when the cache was cold, the average training time was 427.16 sec, with the standard deviation of 129.3. The major contributor of this phenomenon is the inefficient support from file systems.

Using the image file option, each training sample is a 28×28 PNG formatted file, each of which is of a size just less than 300 bytes. We used a total of 60,000 such files for training the neural network. File systems provides caching mechanisms, but it caches blocks within a file since file system cannot judge if another file in the same directory can be accessed or not. As a result, as shown in Figure 5, under cold cache condition, training with image files spent the majority of time in waiting for I/O operations, resulting extremely low GPU utilization. The statistics on the memory utilization and block device read support the hypothesis that this I/O wait does not stem from a large data set size.

When the data is cached, as the total footprint of MNIST is small enough to be cached in the system memory, all back-ends perform similar to each other. With HDF5 option, however, the training time was 28% longer than the fastest option, cppmem, independently of cache condition. We hypothesize that the increase in training time when using the HDF5 format is a result of two factors: (1) additional processing to interpret the file format and (2) increased size for the same training data, 180MB, which is up to 12 times larger than other options (refer to Table 2), although not larger than storing as individual image files. When we looked into resource utilization, we confirm that HDF5 used the largest memory footprint during the model training, which resulted in slightly less efficiency in utilizing GPUs, as shown in Figure 5. As was observed with the MNIST dataset, we cannot find clear correlation between the training times and block reads from storage system, which might be attributed to the small data set size and

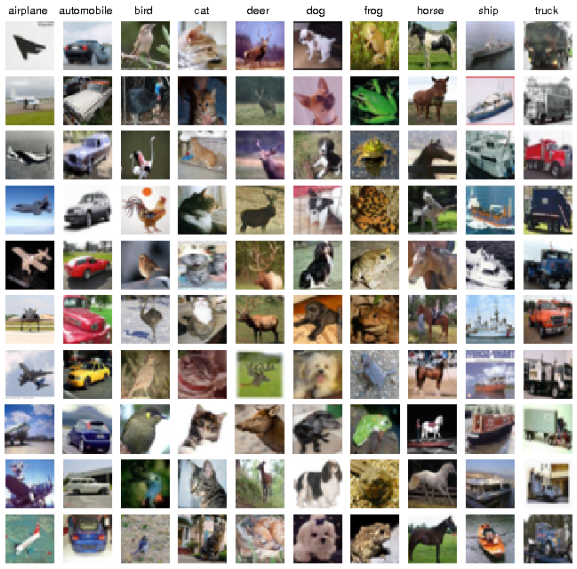
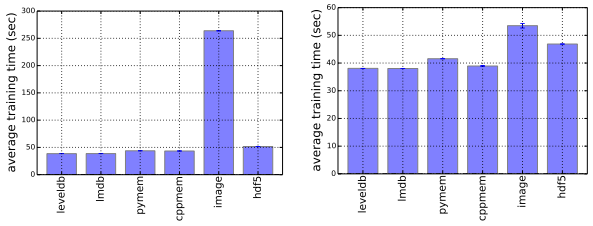


Figure 6. Example images from the CIFAR-10 dataset.



(a) Cold cache (b) Warm cache

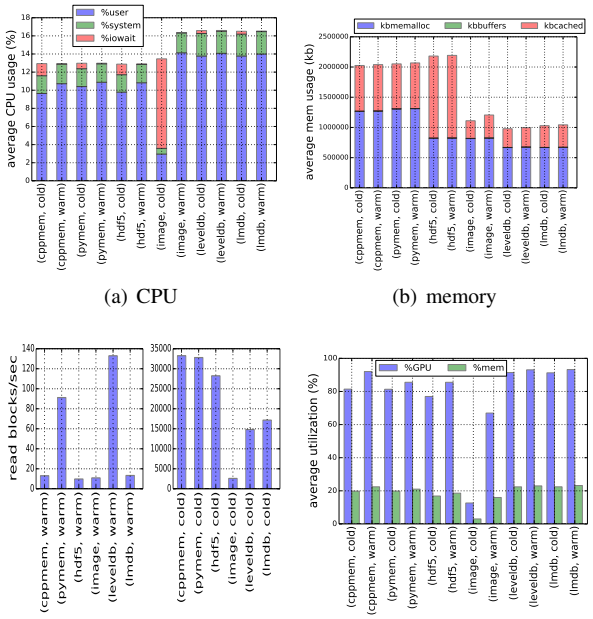
Figure 7. Average training time for CIFAR10. Error bars represent standard deviations.

the bandwidth of the tested system is around 600MB/sec, which supports the block reads well.

3.2 CIFAR10

The CIFAR-10 dataset [23] is a collection of $60,000$ labelled images from the 80 Million Tiny Images dataset [34]. The images 32×32 pixel color images of 10 unique classes. There are $5,000$ images designated as training images for each class, and $1,000$ images from each class are designated as test images. This dataset is used widely in deep learning papers and is often used to quickly evaluate new ideas. The network used for this dataset is the fastest network from Alex Krizhevsky's cuda-convnet work [22]. This network model is comprised of three convolutional layers, each followed by a pooling layer, and two inner product layers. Complete details for the network are shown in Figure 9.

As with MNIST, we measured the training times with different back-end storage systems to train the network shown in Figure 9. The trend of training time to classify image categories in CIFAR-10 data set is shown in Figure 7. Overall, the trend is similar to MNIST cases. Two key-value storage systems (LevelDB and LMDB) and in-memory options (pymem and cppmem) showed similar training time, but file based options (image and HDF5) showed longer average training time than other options. The option of image files demonstrated serious performance deviation, depending on cache condition as with MNIST data set. When we warmed up the cache



(a) CPU (b) memory

(c) block reads (d) GPU

Figure 8. Resource utilization for CIFAR-10

for the image files, the average training time was $53.29 sec$, which was only 40% slower than then fastest option (LMDB). However, when the cache was cold, the average training time was 264.81 sec, which was 6.8 times slower than the fastest option (LevelDB) under the same condition. Again, the major contributor of this phenomenon is the inefficient supports from file systems.

Using the image file option, each training sample is a 32×32 PNG formatted color image file, each of which is of a size less than 4KB. We used a total of 50,000 of such files for training the neural network. As with MNIST data set, we can attribute the serious performance deviation of image file option to the inefficiency of file systems that we have employed. However, since the total footprint of CIFAR-10 is also small enough to be cached in the system memory, all back-ends perform similar to each other when the cache is warmed up. Readers may refer to Table 2 for the storage footprint of CIFAR-10 data set for each storage option. The inefficiency of file-based approaches is highlighted also by HDF5 option in CIFAR-10 data set cases. HDF5 option shows the second longest training time under both warm cache condition and cold cache condition, though it does not show dramatic performance difference according to cache condition, 46.86 seconds for warm cache and 51.67 seconds for cold cache.

Interestingly, key-value storage based options, LevelDB and LMDB, provided the shorter average training time than in-memory options. Key-value storage based option showed around 6 percent higher CPU utilization, which might be required to support database operations. However, Figure 8 (a) shows lower iowait time under cold cache condition, as for both LevelDB and LMDB, which made less deviation of average training time depending on the cache condition. For instance, the average training times of LevelDB were 38.06 seconds under the warm cache and 38.68 seconds under the cold cache. In contrast, cppmem showed 4.5 seconds of differences in average training time, depending on the cache condition.

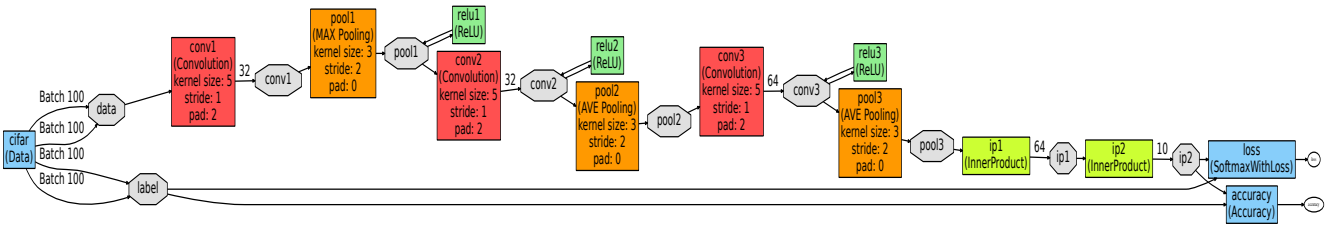


Figure 9. Trained network for CIFAR-10: 145,578 learned parameters

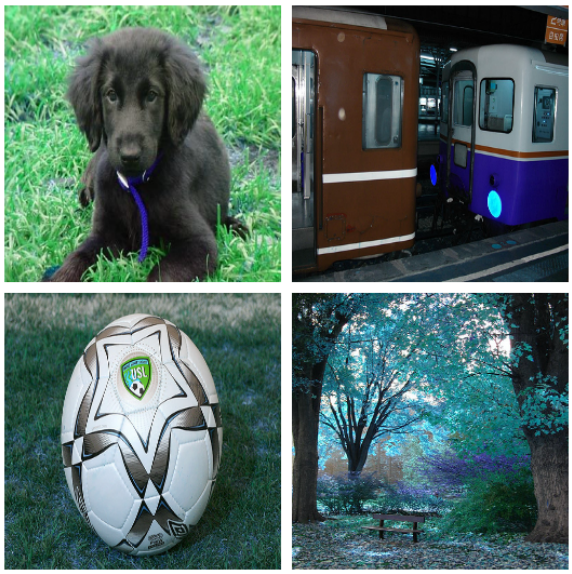
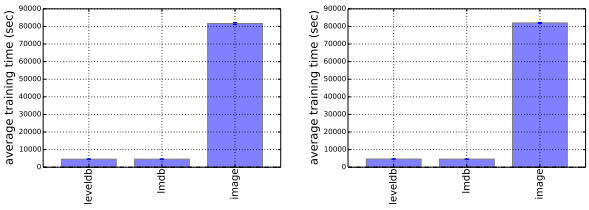
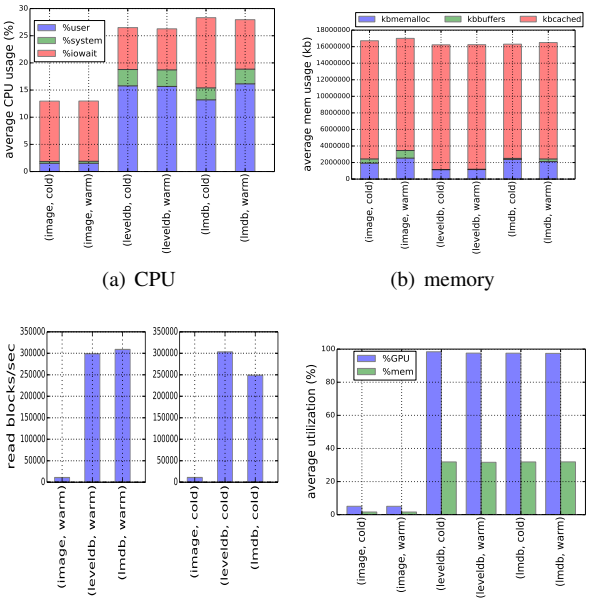


Figure 10. Example images from the ImageNet dataset.



(a) Cold cache (b) Warm cache

Figure 11. Average training time for ImageNet. Error bars represent standard deviations.



(a) CPU (b) memory

(c) block reads (d) GPU

Figure 12. Resource utilization for ImageNet

3.3 ImageNet

The ImageNet competition dataset [31] is a collection of images organized according to the WordNet hierarchy [28]. The images are of various sizes and are typically rescaled to a 256×256 pixel image for use. There are $1.28M$ images designated as training images, and $50,000$ images are designated as test images. This dataset is the most widely used benchmark for deep learning today. The network used for this dataset is a modified AlexNet model [24]. This network model is comprised of five convolutional layers, three max pooling layers, and three inner product layers. Complete details for the network are shown in Figure 13.

We trained the network shown in Figure 13, which is also known as caffe-net included in Caffe. We modified the total number of iterations from 45,000 to 15,000 in order to reduce the training time for benchmark purposes. The storage space footprint to store ImageNet data is shown in Table 2, around 200GB for key-value systems, and 143 GB for image file cases. We stored image files in PNG format. As databases store additional metadata information than pixel-arrays, their storage footprints were higher than image file options. For the ImageNet dataset, we did not experiment with HDF5 since the reference implementation in Caffe framework required to perform cropping operations on image data, which is not supported for the HDF5 option. In addition, we were not able to experiment with cppmem and pymem options as the data set size exceeds several folds of the memory capacity used in this study.

From experiments with ImageNet data set, we clearly confirm that the use of raw image files seriously slows down the model training. For smaller data sets like MNIST and CIFAR-10, we can easily warm up the cache even with the image file option. However, as with ImageNet, when the data set size is multiple times larger than the memory capacity, warming up the cache to achieve performance gain was nearly impossible, shown in Figure 11. When we feed the training samples into the training model, we seek the beginning of the batch and sequentially fetches following image data, also known as mini-batch training for stochastic optimization [27].

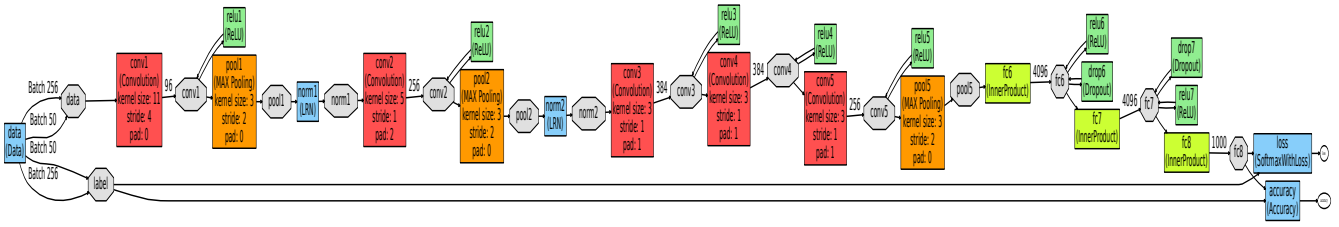


Figure 13. Trained network for ImageNet: 62,378,344 learned parameters

Due to such data feeding operations, caching individual files are not sufficient when the data size is huge. However, key-value systems, where keys represent image identifiers and values represent pixel arrays, provide better caching and indexing strategies to retrieve image data such as B+tree index with LMDB and log-structured merge tree with LevelDB. As a result, key-value systems trained the same network on ImageNet data, with less than 6% time of the image file option, regardless of cache condition.

As reported in many file system studies, when the number of files are large, file system metadata operations to locate the blocks in the file system require very large memory for the operating system kernel [3]. As shown in Figure 12, with image file options, the system simply cannot read the file contents as it cannot locate the files in an efficient manner. Therefore, GPU remained almost idle during the entire training, resulting extremely long training time on average. If we train a larger model than the trained model in this study, it might require multiple GPUs across a cluster of machines [7]. Then, the computation and communication overheads might hide inefficient file system operations to some degree, but prior studies reported the importance of efficient data feeding [5, 11]. Hence, we highly recommend the researchers to consider the reduction of the total number of files to be accessed, along with choosing systems with advanced caching and indexing mechanisms than regular file systems.

4. Conclusion

This study evaluated the efficiency of image storage backend options in training deep neural networks. We trained the deep neural networks using a GPU, as it is well known to significantly reduce the computing resource requirements. We analyzed the average training time according to image storage backends, along with measuring the activities of major computing resources – CPU, memory, I/O, and GPU. On top of our principled evaluation, we observed serious inefficiency of using image files on local file system like ext4, which can result in up to 17 times slower training times for a large data set such as ImageNet. According to our analysis, it is the combination of the characteristics of data set and the characteristics of data access pattern for training deep neural network models. The typical data sets for training deep neural network consist of numerous number of training samples, which are accessed by a series of mini-batches across the entire training samples. Thus, as for training deep neural networks, file systems provide a poor indexing to locate files in the storage system and inappropriate caching mechanism (caching blocks within the same file, instead of caching another files in the same directory.) Therefore, we suggest image storage backends with efficient indexing capabilities of training samples and advanced caching mechanisms, for training deep neural networks.

To complete the evaluation, we might consider other scenarios: (1) utilizing parallel file systems such as GlusterFS, Lustre, and GPFS; (2) the cases with high resolution and spectral images with several thousands of channels, where an individual image could reach at Giga bytes; and (3) concurrently training multiple models for the same data set in order to search the best hyper-parameters to tune the model or to construct an ensemble of models so as to boost the accuracy of prediction. We plan to perform the evaluation that includes these scenarios. Finally, for training deep neural network models, we suggest key-value based approaches to efficiently retrieve the image data. If the image data exceeds the capacity of local file systems, we might consider distributed key-value storage systems such as Redis [29], and HBase [1]. However, we need to use the caution as the size of individual pixel arrays to be trained can be much larger than the desirable range of value sizes for key-value storages. For instance, HBase generally recommends less than 100MB of value size. Another direction might be to improve the metadata operation of file systems as with [3].

Acknowledgment

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

References

[1] Apache HBase. `http://hbase.apache.org`.

[2] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[3] D. Beaver, S. Kumar, H. C. Li, J. Sobel, P. Vajgel, et al. Finding a needle in haystack: Facebook's photo storage. In *OSDI*, volume 10, pages 1–8, 2010.

[4] R. Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.

[5] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 571–582, 2014.

[6] H. Chu. Mdb: A memory-mapped database and backend for openldap. *LDAPCon11*, 2011.

[7] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew. Deep learning with cots hpc systems. In *Proceedings of the 30th international conference on machine learning*, pages 1337–1345, 2013.

[8] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[9] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[10] P. Cudré-Mauroux, I. Enchev, S. Fundatureanu, P. Groth, A. Haque, A. Harth, F. L. Keppmann, D. Miranker, J. F. Sequeda, and M. Wylot. Nosql databases for rdf: an empirical evaluation. In *The Semantic Web–ISWC 2013*, pages 310–325. Springer, 2013.

[11] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[12] S. Ghemawat and J. Dean. `https://github.com/google/leveldb`.

[13] Google. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[14] T. Harter, D. Borthakur, S. Dong, A. S. Aiyer, L. Tang, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Analysis of hdfs under hbase: a facebook messages case study. In *FAST*, volume 14, page 12th, 2014.

[15] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang. Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 27–40. ACM, 2015.

[16] HDF Group. `https://www.hdfgroup.org/HDF5/`.

[17] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[18] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li. An analysis of facebook photo caching. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 167–181. ACM, 2013.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[20] N. JONES. The learning machines. *NATURE*, 505, 2014.

[21] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer. Enterprise data analysis and visualization: An interview study. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2917–2926, 2012.

[22] A. Krizhevsky. cuda-convnet. URL `https://code.google.com/archive/p/cuda-convnet/`.

[23] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images, 2009.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[25] Y. Lecun and C. Cortes. The MNIST database of handwritten digits. URL `http://yann.lecun.com/exdb/mnist/`.

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[27] M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM, 2014.

[28] G. A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, Nov. 1995. ISSN 0001-0782. . URL `http://doi.acm.org/10.1145/219717.219748`.

[29] Redis. `http://redis.io/`.

[30] K. Ren and G. A. Gibson. Tablefs: Enhancing metadata efficiency in the local file system. In *USENIX Annual Technical Conference*, pages 145–156, 2013.

[31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. .

[32] Symas, 2012. `http://symas.com/mdb/microbench/`.

[33] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. *ArXiv e-prints*, Dec. 2015.

[34] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(11): 1958–1970, 2008.

[35] UFLDL Tutorial. `http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/`.