Abstract—Neural networks are an emerging technology of great interest. Many neural networks are implemented using the TensorFlow library. TensorFlow abstracts the process of constructing and training neural networks, and has supports for several parallel computing interfaces including Compute Unified Device Architecture (CUDA) and Direct Machine Learning (DirectML). However, the structure and training process may not be able to take full advantage of the underlying hardware due to how TensorFlow parallelizes computations. In this paper, we examine how changing a training variable, the batch size, affects the performance of a sample Convolutional Neural Network (CNN). We use the CNN to classify images from the Modified National Institute of Standards and Technology (MNIST) database of handwritten digits. We run the CNN on a desktop with eight central processing unit (CPU) cores plus 4608 graphics processing unit (GPU) cores and on a high-performance computing (HPC) system with 2x18 CPU cores plus 5120 GPU cores. According to the experimental results, larger batch size may reduce training time with negligible or recoverable losses in accuracy for certain CNN parameters. Results also suggest that optimization must be taken on a case-by-case basis to determine the best parameters.

Keywords—Batch size, convolutional neural networks, parallel computing, TensorFlow, training time

I. INTRODUCTION

I1.7 5Tw 4.333 0 Td0.004 Tw32.723 -2.253 m tsie0.0478.6ld0.0071—

algorithm to update the weights. Figure 1 shows two variations of how TensorFlow may accomplish data parallelism (namely, synchronous and asynchronous). TensorFlow dispatches training data to each device by accumulating the error and synchronously updating the parameters of the model or by using multiple training clients asynchronously generating updates.

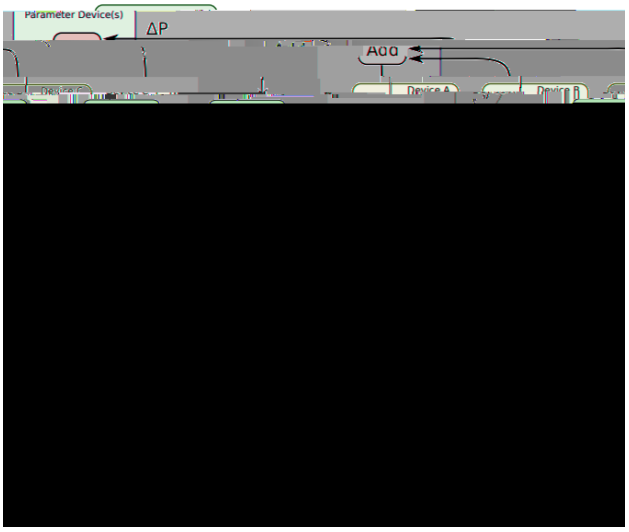

Figure 1. TensorFlow's use of data parallelism [1]

Backpropagation, however, cannot be parallelized to the same degree as forward propagation within the synchronous training process because it cannot take full advantage of data parallelism; it is only run once after each batch and must run after each batch is entirely processed. The processing of the batch, however, can theoretically be parallelized up to the size of the batch; each piece of data can be processed independently of each other and summation of the error is a common reduction operation. Therefore, for neural networks in which the computational cost of forward propagation for an individual piece of data is low, the limiting factor in training speed will be the backpropagation steps.

The batch size determines the maximum limit of how many items can be processed in parallel concurrently, as TensorFlow will dispatch work for the entire batch and wait before performing backpropagation. If the total number of items in a training dataset is much greater than the number of items in a batch and the computational cost of each item is much smaller than the capabilities of the underlying hardware, then it can be assumed that the size of a batch may limit the speed at which the entire dataset is processed; for small batch sizes, the hardware may be able to process the entire batch in parallel but have unutilized computing elements that a larger batch would be able to use. Studies observe this phenomenon, noting that increasing the batch size can yield greater time efficiency on parallel systems [4][5]. Liu et al. focus on dynamically changing the batch size while training to improve the efficiency of the training process (loss improvement over time) [4]. Ramirez-Gargallo et al. also observe the same increase in performance as both batch size and thread counts increase [5]. Increasing the batch size may decrease accuracy though, as the weights of the network are not adjusted as often over the course of training as observed by Liu et al. In this work, a sample CNN is created and run on two different computer systems with varying parameters to evaluate the impact of changing the batch size during training.

III. Experimental Setup

The neural network used for testing the impact of batch size is a CNN designed to classify images from the MNIST database of handwritten digits. The structure of the network can be seen in Figure 2, with a 28x28 input layer for each pixel of the input images, followed by two alternating layers of convolutions and pooling with a kernel size of 4x4 and a pooling size of 2x2. After flattening the network incorporates two alternating layers of dropout and densely connected neurons, the first dense layer having 500 neurons and the final output layer having 10 neurons corresponding to each digit.

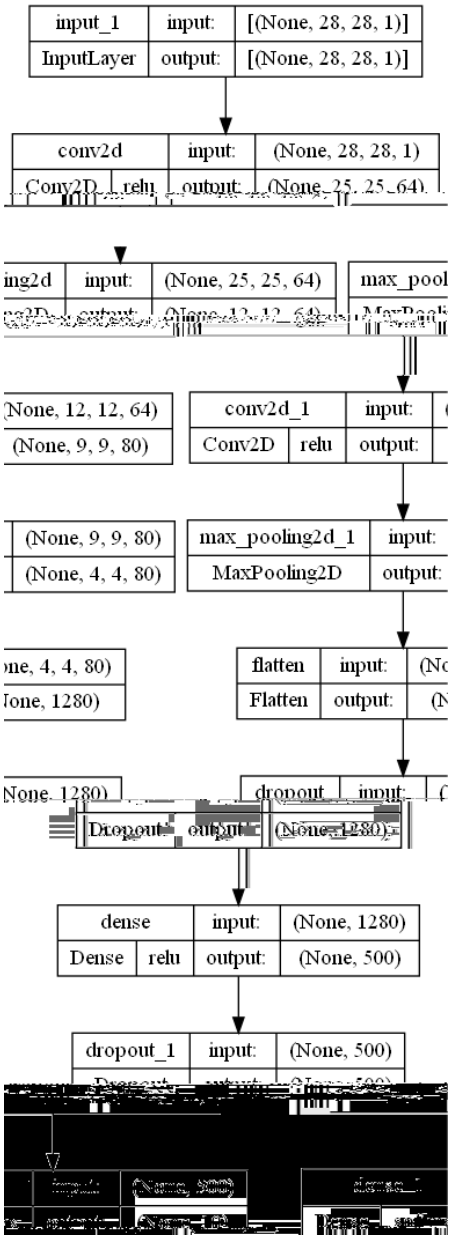

Figure 2. Structure of the CNN used

Each convolution and dense layer is

The accuracy plot in Figure 6 again shows similar behavior to previous tests, with accuracy rising with the number of epochs. This behavior seems to be consistent across all

Accuracy for the BeoShock trials showed the same trends as with the desktop, decreasing with higher batch sizes but recovering with more epochs.

Figure 11. Training time for the BeoShock system using GPU

Unlike the desktop system, for BeoShock, both the CPU and GPU tests show no significant improvement with larger batch sizes. The GPU run took significantly longer than its equivalent on the desktop system (see Figures 7 and 11). A possible explanation for this large difference is that BeoShock is a shared system with different users running concurrent HPC tasks, while the desktop was a single-user system only running background tasks, meaning the BeoShock system would have a much lower limit on the maximum effective computational throughput and would be more easily saturated with work by a smaller batch size. This is a good reminder that optimal performance depends on the conditions of the environment the program is running in.

V.