Comparing ML Hardware

Deepanshu Gera (dgera@wisc.edu)
Sreejita Dutta (sdutta7@wisc.edu)
Srujana (srujana@wisc.edu)

Problem Statement

Deep learning algorithms like VGG, ResNet are revolutionizing various areas of machine learning. On a high level, working with machine learning is a two-stage process: First, a model is trained, i.e. its parameters are determined using labeled examples of inputs and desired output. Then, the model is deployed to run inference, using its previously trained parameters to classify, recognize, and generally process unknown inputs. Training these algorithms is computationally intense, and often take up to several weeks for a training session if performed sequentially on a CPU. However, recently with the ever-increasing trend of using GPUs in Machine Learning, it is widely believed that they are the state of the art, due to both speed and energy efficiency advantages compared to more traditional CPU-based platforms[3].

There is an unclear understanding of how architecture design affects the performance of the ML jobs as well as the utilization of the hardware. As people believe that the major improvement in cost-performance can be achieved using domain-specific hardware[1,2], we want to test this hypothesis by comparing various Machine Learning workloads on different hardware. In this project, we plan to use Nvidia K40 (Baseline), Nvidia V100, and Nvidia P100 (16GB and 12 GB variation) and compare their performance with respect to the various state of the art algorithms. Specifically, we want to analyze CPU/GPU utilization, performance, inference time and accuracy of these models.

Motivation and Background

In recent years, GPUs have become increasingly popular with the rise of deep learning. Typical CPUs are designed to tackle calculations in sequential order. A CPU with multiple cores may marginally speed up the calculation by offloading the operations to each core. But CPUs with multiple cores are prohibitively expensive, making them less optimal for training neural networks. A GPU is a processor with thousands of cores capable of performing millions of mathematical operations in parallel. There is a similarity between graphic rendering and deep learning. Both these scenarios deal with a huge number of matrix multiplication operations per second. That is one of the reasons why high-end GPUs are preferred for deep learning.
The efficient architecture of GPU performs more floating point operations per watt of power consumed. The superior floating point performance provided by GPUs is due to the large number of cores. Matrix algebra applications can divide their work evenly among a large number of cores, making GPUs an ideal processor for these applications. The result is faster solution times and the ability to solve large problems compared to general purpose CPU processors. GPUs provide an order of magnitude or more in processing power for the same capital cost. In recent years, TPUs and FPGAs are also on the horizon for performing deep learning tasks.
Since the emergence of Deep Neural Networks (DNNs) as a prominent technique in the field of computer vision, the ImageNet classification challenge has played a major role in advancing the state-of-the-art. Various CNNs were discovered during this competition which obtained the highest accuracy in a multi-class image classification problem. In our report, we will focus on some of these algorithms for various comparisons.

1. **VGG(16/19)** is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”[6]. VGGNet consists of 16/19 convolutional layers and is very appealing because of its very uniform architecture[5].
2. **Residual Neural Network** (ResNet) is an artificial neural network introduced by Kaiming He et al[7], a novel architecture with “skip connections” and features heavy batch normalization. Such skip connections are also known as gated units or gated recurrent units and have a strong similarity to recent successful elements applied in RNNs. ResNet's will only skip over a single layer. And they train a NN with 152(51, 101) layers while still having lower complexity than VGGNet[5].

Experiment Setup

For our performance analysis, we focus on four classical neural network architectures: ResNet 50, ResNet 152, VGG16 and VGG19. We train these networks on the Imagenet downsampled dataset of 32X32 images.

To cover a range of possible scenarios, we divide our experiments into training and inference. In both the training and inference step, we also measure output based on various batch sizes. Using multiple batch sizes is important as it impacts both the training time and inference time. Also, this enables us to capture real-world applications that have high latency bound. For the hardware comparison, we chose four different hardwares: 1) Nvidia K40m 2) Nvidia P100 with 12 GB memory 3) Nvidia P100 with 16GB memory 4) Nvidia V100. The specifications of these processors can be found in the table below. Nvidia K40 is based on Nvidia’s Kepler Architecture which supports over 1 TFlop of double precision throughput. It supports dynamic parallelization and has optimizations which show improvements when dealing with image data. Nvidia P100 is based on Nvidia’s Pascal architecture[8] with over 5 TFLOP of double precision performance. Based on the optimizations done specifically for machine learning, Pascal architecture offers reduced training time for neural network training and a significant increase in deep learning inference throughput. Nvidia V100 is based on the latest Volta architecture[9]. It has over 640 Tensor cores with almost five times increase in deep learning performance as compared to the Pascal architecture.

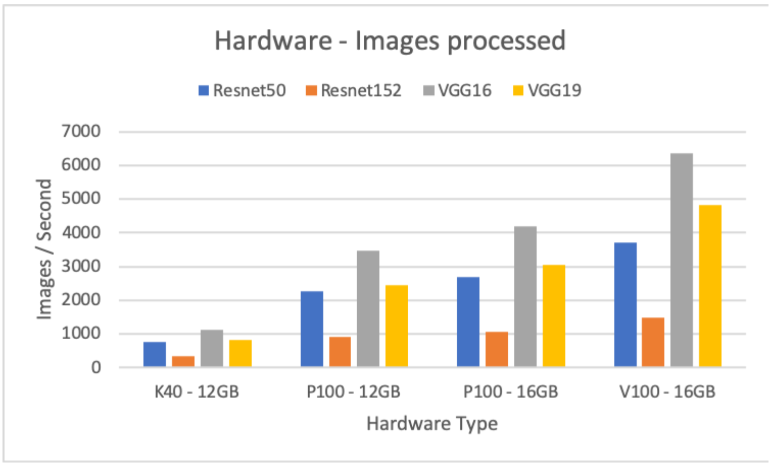
We also perform experiments by dividing workload across four Nvidia V100[4]. To run the neural networks on the GPU, we use Tensorflow compiled for GPU usage using cuDNN. The code to perform training and inference, as well as scripts for evaluation can be found here. The data collected and the reports generated can be found here.

Processor	SMs	CUDA Cores	Tensor Cores	Frequency	TFLOPs (double)[1]	TFLOPs (single)[1]	Cache	Memory	Memory B/W	Wattage
Nvidia P100 (Pascal)	56	3,584	N/A	1,126 MHz	4.7	9.3	4 MB L2	12/16 GB	720 GB/s	250W
Nvidia V100 (Volta)	80	5,120	640	1.53 GHz	7	14	6 MB L2	16 GB	900 GB/s	300W
Tesla K40M (Kepler)	15	2,880	N/A	745 MHz	1.43	4.29	1536 KB L2	12 GB	288 GB/s	235W

Training Results

Performance

We use images processed per second as a performance metric to compare the different hardwares. K40M is an older GPU architecture and does not have any deep learning-specific optimizations (which P100 and V100 have). Hence, even though it has the same memory as one of the P100 machines, K40M performs poorly compared to the P100. As expected, the higher memory P100 performs better than the lower memory P100 as it can store more weights in memory. Due to the advanced architecture of V100 that uses Nvidia Volta (with Tensor cores), it delivers higher number of TFLOPs per second compared to the previous generation Nvidia Pascal (used by P100). This is why V100 exhibits the best performance for every model.



Utilization

In this section, we show the results of various GPUs and their utilization while running ResNet50 model. Even though we only show the results for ResNet50, the trend is the same across all the models. Here, we can clearly see that all the GPUs are not fully utilized, which leads us to believe that GPU cores are not a bottleneck, instead, the performance is mainly bounded by memory efficiency (i.e., bandwidth and latency). Also, every step in the softmax layer involves element-wise matrix computation. The low arithmetic intensity in these matrix-vector operations and the intermediate data communication across different steps also make them memory bound[10].

GPU Model	Nvidia P100 (12GB)	Nvidia P100 (16GB)	Nvidia V100	Nvidia K40M
Memory	11837	15715	15772	11019
Min Utilization	48	65	54	58
Max Utilization	92	94	100	90
Avg Utilization	78.75008831	86.14081547	71.73395321	75.24019258

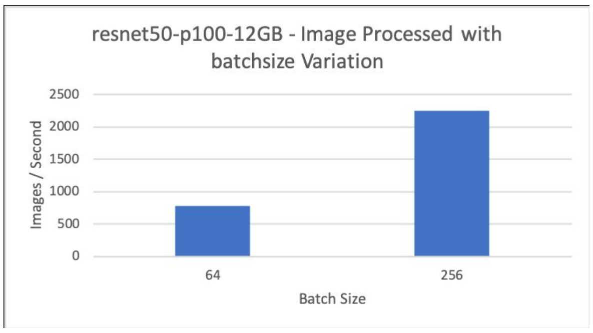
Power Consumption and temperature

We observe the power consumption using nvidia-smi. Here, we report the power consumption of the ResNet50 model on various hardwares as the trend is consistent among models. Also, larger the model is, it consumes more power.

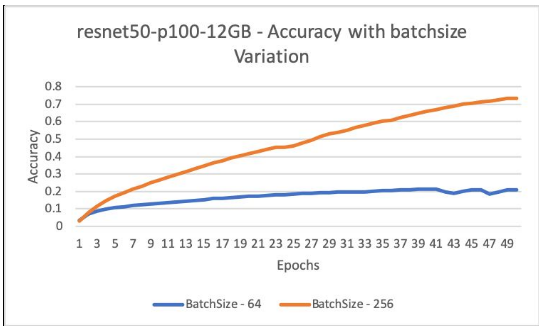
GPU Model	Nvidia P100 (12GB)	Nvidia P100 (16GB)	Nvidia V100	Nvidia K40M
Min Consumption	38	89	110	99
Max Consumption	150	182	247	145
Avg Consumption	103.7762275	140.1445986	170.9325135	120.586543
Idle Power	24	30	55	20

Batch size

We perform training using mini-batch sizes of 64 and 256. Images processed per second is much lower for batch 64 as compared to batch 256. Higher batch size involves lesser gradient updates and this results in higher throughput. Likewise, total training time taken for batch size 256 is lower as compared to that of 64.

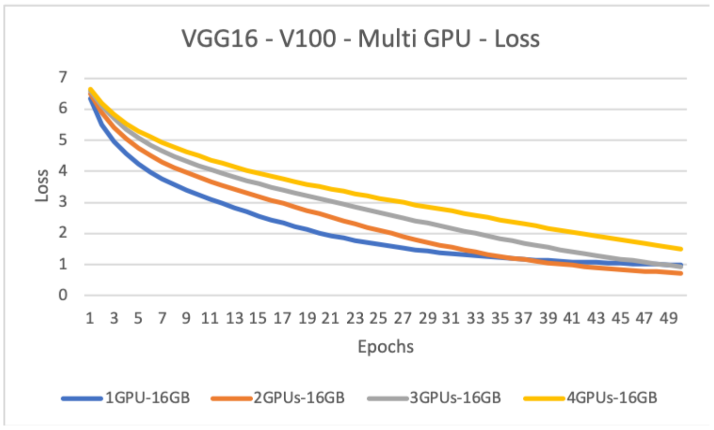


Batch size is a hyperparameter that needs to be tuned to select a model that provides optimal accuracy. In our case, the batch size of 256 provides significantly better accuracy over 50 training epochs as compared to the batch size of 64.



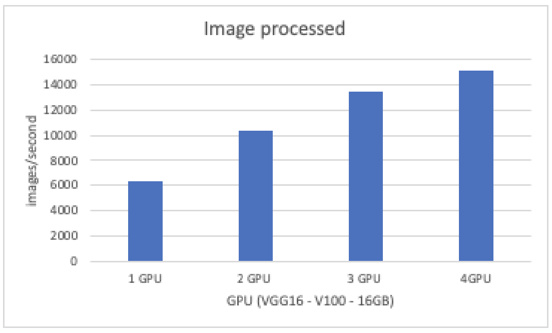
Multiple GPUs: To compare various aspects of hardware, we also analyze our models on multiple GPUs. We examine the impact of using single system multi-GPU configurations as it partially solves problems with limited performance and memory.
Choosing a specific device to train the neural network does not paint the complete picture. It's normal to train several models at once (maybe trying different parameters) and also to train larger models faster. For this, we need to split the work across multiple GPUs in the system by doing distributed training.
There are two main approaches to parallelizing neural network training: model parallelism and data parallelism. In model parallelism, different machines in the distributed system are responsible for the computations in different parts of a single network — for example, each layer in the neural network may be assigned to a different machine. In data parallelism, different machines have a complete copy of the model; each machine simply gets a different portion of the data, and results from each are somehow combined.
For our experiments, we perform data parallel computation.

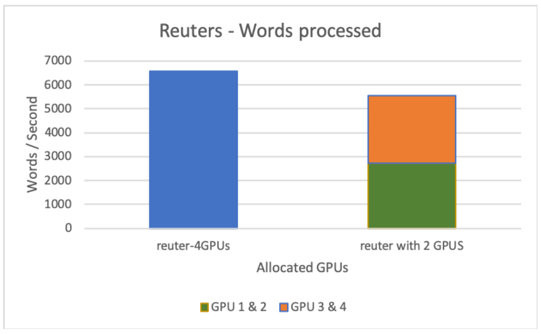
1) **Convergence:** As we can clearly see in the graph below, the model running one GPU converges in lesser number of epochs than using multiple GPUs. This is expected as multiple GPU workloads use stale parameters, which can delay convergence.



2) **Performance:** We trained a VGG16 model on multiple GPUs to compare the performance of distributing the workload. We can clearly see that the number of images per second increases almost linearly when using two GPUs instead of one GPU. However, the performance increase is not consistent and follows the law of diminishing returns. This means that the increase in performance (seen by addition of GPUs) declines as we keep on adding more GPUs to the system.

And, while using four Nvidia V100 GPUs, we noticed that the total time to train two VGG16 models simultaneously is lesser than training two models sequentially (one after the other) on four GPUs. To verify this, we ran another lightweight model (Fully connected neural network with 512 hidden layers on Reuters dataset), and we can clearly see that it takes almost double the time to train it sequentially on 4 GPUs than to train them parallelly on 2 GPUs each. In fact, the time taken to train two models simultaneously on 4 GPUs is almost similar to the time taken to train one model on 4 GPUs. This happens because the GPUs are not fully utilized to their potential (can be seen in the next section). Thus, instead of training a single model with more GPUs, it might be better to train multiple models depending on the hardware and the models used.





3) **Utilization:** While running VGG16 on four GPUs, we notice that none of the GPU is being utilized entirely. In fact, the utilization is almost 50 percent. This leads us to believe that running VGG16 on more than two GPUs is a waste of resources. The performance gain is not high enough to warrant the use of multiple GPUs. Instead, it would have saved time and resources if we had run multiple models in parallel as shown in the previous section.

GPU Model	GPU 1	GPU 2	GPU 3	GPU 4
Memory	15772	15772	15772	15772
Min Utilization	35	18	17	12
Max Utilization	68	53	53	54
Avg Utilization	56.27317073	43.77926829	43.89268293	43.72682927

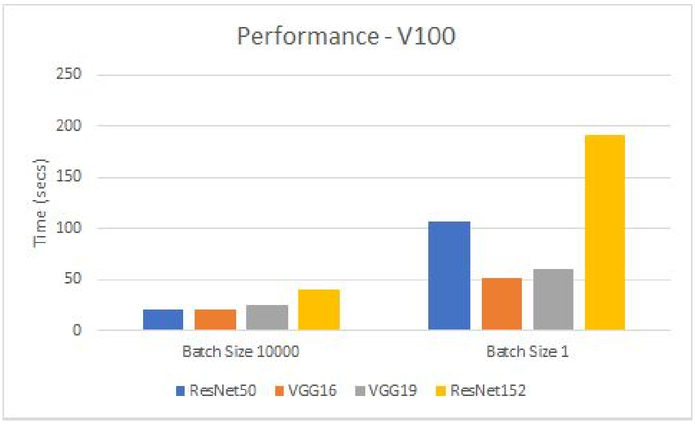
Inference Results

Both training and inference start out with the same forward propagation calculation, but training goes further. After forward propagation, the results from the forward propagation are compared against the (known) correct answer to compute an error value. A backward propagation phase propagates the error back through the network's layers and updates their weights using gradient descent in order to improve the network's performance at the task it is trying to learn. It is common to batch hundreds of training inputs (for example, images in an image classification network or spectrograms for speech recognition) and operate on them simultaneously during training in order to prevent overfitting and, more importantly, amortize loading weights from GPU memory across many inputs, increasing computational efficiency. For inference, the performance goals are different. To minimize the network's end-to-end response time, inference typically batches a smaller number of inputs than training, as services relying on inference to work (for example, a cloud-based image-processing pipeline) are required to be as responsive as possible so users do not have to wait several seconds

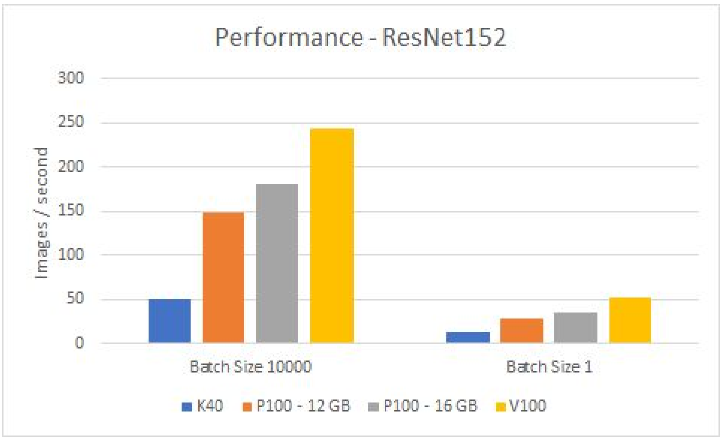
while the system is accumulating images for a large batch. In general, we might say that the per-image workload for training is higher than for inference, and while high throughput is the only thing that counts during training, latency becomes important for inference as well.
Here, we analyze how various different types of hardware affect inference properties like performance, utilization. We performed inference using all four models (ResNet50, ResNet152, VGG16, and VGG19) on 10,000 images using all four GPU architectures mentioned above. We used two batch sizes for inference - 1 and 10,000, to compare the difference in performance and utilization.

Performance

We use time taken and images processed per second as metrics for performance. The graph below shows the time taken to perform inference on V100 hardware with two different batch sizes. For both batch sizes, ResNet152 takes the maximum time as it is a bigger model compared to the other models. It also observed that for batch size of 1, ResNet50 takes more time than VGG16 and VGG19. These results are consistent across all four hardwares.

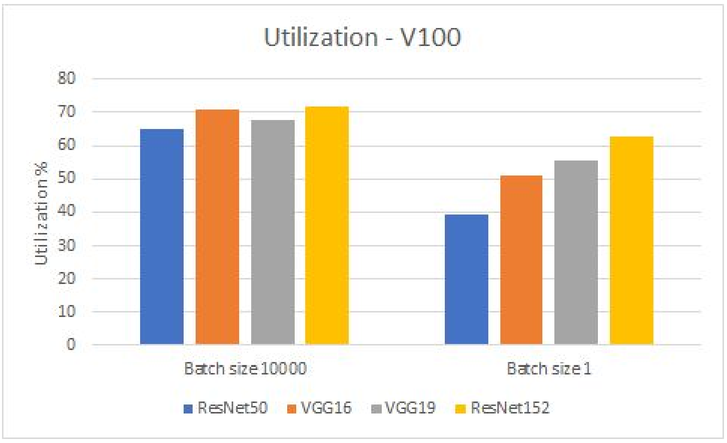


Also, as observed in training, better hardware process more images per second which means K40M shows the poorest performance and V100 outperforms other architectures during inference as well. When we compare the images processed per second between batch sizes, we find that increased batch size processes more images per second. This is consistent across hardwares and across models. When batch size is 1, each time a single image is read into memory and weights are applied for inference. However, in case of a larger batch size, all the images are read at once and kept in memory, which reduces the I/O cost.



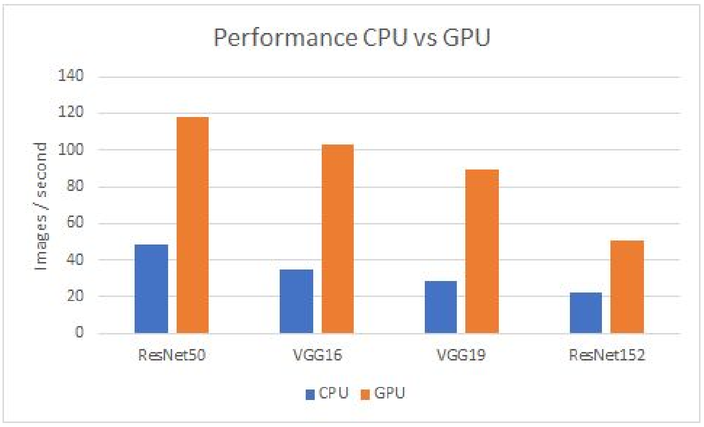
Utilization

We analyze GPU utilization for the two batch sizes. Batch size of 1 has a lower utilization compared to batch size of 10000. This is because the larger batch size is able to process more images per second. Hence, this increases utilization and lowers total time taken. We did not find any other significant difference in utilization between training and inference.



CPU vs GPU

We also compare performance on a CPU to that of GPU architectures. We use two Intel Xeon E5-2667 8-core CPUs(total 16 cores) and the graph below shows its comparison to Nvidia K40M, which is our poorest performing GPU. We report performance in terms of images processed per second and even the slowest GPU outperforms the CPU significantly.



Future Work

In this report, we analyzed multiple state-of-the-art deep neural networks on ImageNet, in terms of loss, batch sizes, performance, distributed training, inference time for various hardware. Our goal is to provide insights into the design choices that can lead to efficient neural networks for practical application and optimization of the often-limited resources in actual deployments.

We believe that another important aspect of analyzing hardware for machine learning is to check how various GPUs perform during quantization and how the inference time is affected by it. Although, we did see a significant improvement in prediction time while working with fully quantized ResNet (trained on 8-bit fixed point for integer weights), but the accuracy drop was too high and hence we decided not to include it in our report. Still, we think that it's an interesting area of research and would help advance how inference is calculated on edge devices.

References

1) Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., ... & Boyle, R. (2017, June). In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)* (pp. 1-12). IEEE.
2) Viebke, A., & Pllana, S. (2015, August). The potential of the intel (r) xeon phi for supervised deep learning. In 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems (pp. 758-765). IEEE.
3) NVIDIA, "Whitepaper: GPU-based deep learning inference: A performance and power analysis."
4) Yadan, O., Adams, K., Taigman, Y., & Ranzato, M. A. (2013). Multi-gpu training of convnets. arXiv preprint arXiv:1312.5853.
5) CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more by Siddarth Das, https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5
6) Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
7) He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
8) Nvidia Corporation. (2017). NVIDIA Tesla P100. Retrieved from https://www.nvidia.com/object/pascal-architecture-whitepaper.html
9) Nvidia Corporation. (2018). NVIDIA Volta. Retrieved from https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/
10) Li, C., Yang, Y., Feng, M., Chakradhar, S., & Zhou, H. (2016, November). Optimizing memory efficiency for deep convolutional neural networks on GPUs. In SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 633-644). IEEE.