Optimize Virtualized Deep Learning Performance with New Intel Architectures

Performance Study - March 31, 2020

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com

Table of Contents

Executive Summary

Four different image classification tests were used to demonstrate the performance benefits of running deep learning inference on the 2nd Generation Intel® Xeon® Scalable processor compared to previous Intel processors, and to show the performance benefits of running on the VMware vSphere® hypervisor compared to bare metal.

The 2nd Generation Intel® Xeon® Scalable processor's Deep Learning Boost technology includes new Vector Neural Network Instructions (VNNI), which are especially performant with input data expressed as an 8-bit integer (int8) rather than a 32-bit floating point number (fp32). Together with the large VNNI registers, these instructions provide a marked performance improvement in image classification over the previous generation of Intel® Xeon® Scalable processors.

The latest version of vSphere, 7.0, supports VNNI instructions. The work reported in this paper demonstrates a very small virtualization overhead for single image inferencing but major performance advantages for properly configured virtualized servers compared to the same servers running as bare metal.

Introduction

Deep learning typically refers to machine learning on non-tabular data (images, voice, etc.) using neural networks. Some of the applications include image classification (used in license plate detectors or facial recognition systems), object detection (which identifies the objects in an image and is used in autonomous vehicles, for example), or natural language processing (used by voice applications such as Alexa and Siri, and text applications like chatbots).

To train a neural network (Figure 1), one starts with a set of weights w_{ij} that represent the multiplicative factor of joining two adjacent neurons in the network. A set of pre-labeled training data (for example, the pixels of an image together with the classification) x_i is applied to the input layer and the network is run in the forward propagation direction, resulting in a set of predicted classifications y_i. The predicted classifications are then compared to the actual labels, generating a loss function that is used by the back-propagation step to modify the weights in such a manner as to reduce the loss. The cycle of forward propagation followed by back propagation is repeated until a desired accuracy is achieved. The final set of weights along with the model structure are referred to as the trained model.

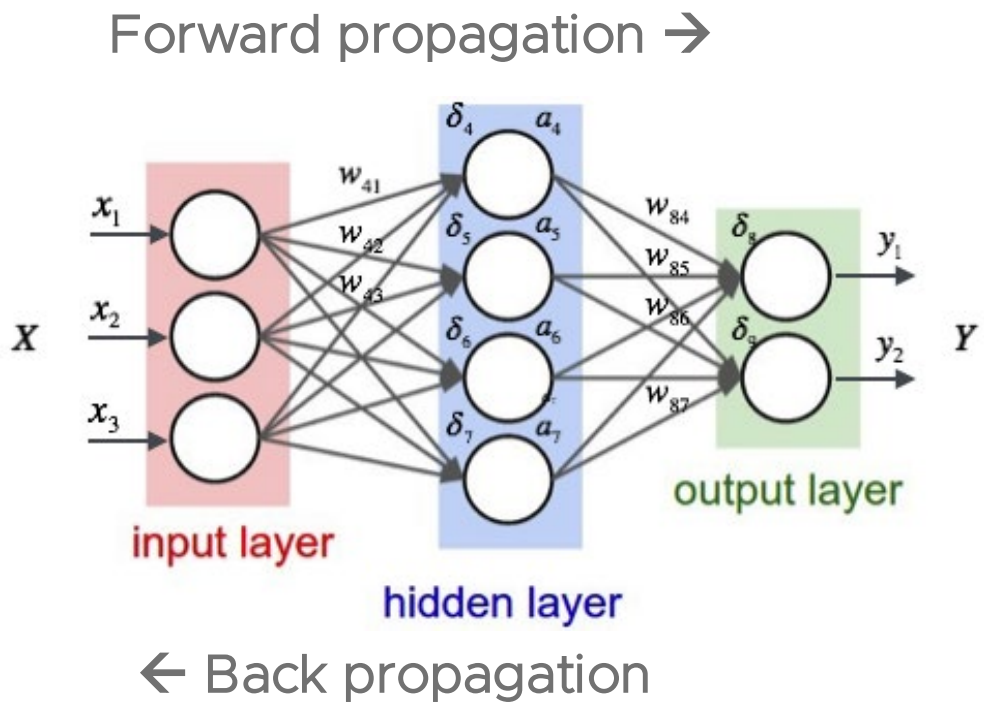

Figure 1: A Neural Network

In inference, a new, unlabeled input is run through the trained model in just one forward propagation step to infer the category of an object or the meaning of a voice command. The two key metrics for inference are *latency*, the response time to a single image or command, and *throughput*, the overall amount of inferencing a system can provide.

Popular current models such as the ResNet50 used in this work can have millions of weights and hundreds of layers. To do all the calculations, even for one forward propagation step, requires processors that can quickly handle large numbers of matrix multiply/add operations.

To that end, Intel has introduced Intel® Deep Learning Boost (Intel® DL Boost) technology, a new set of features in their 2nd Generation Intel® Xeon® Scalable processor ("Cascade Lake"). This feature set includes new Vector Neural Network Instructions (VNNI), which are especially performant with input data expressed as an 8-bit integer (int8) rather than a 32-bit floating point number (fp32). Together with the large VNNI registers, these instructions provide a marked performance improvement over the previous generation of Intel® Xeon® Scalable processors ("Skylake").

vSphere 7 supports the VNNI instructions, which allows VNNI to run on the vSphere hypervisor and to let users take advantage of all that VMware® virtualization offers: enhanced server utilization, scriptable deployment, and ease of management. The work reported in this paper demonstrates a very small virtualization overhead for single image inferencing but major performance advantages for properly configured virtualized servers compared to the same servers running as bare metal.

In the next section, we describe in detail the Intel AI strategy and VNNI architecture. Next, we show the hardware and software configurations used in the tests, followed by descriptions of the individual tests run, and the results of those tests.

Intel's AI Strategy

Over the last few years, we have seen an exponential growth in AI techniques to solve problems in various domains, such as health care, manufacturing, finance, and business operations. This growth has been fueled by the explosion of available data and the need for some sophisticated means for gleaning meaningful insights from the data.

It is expected that the data generated from various "smart" devices will grow from 40 zettabytes to about 175 zettabytes by 2025 [1]. This data will be coming from the greater than 150 billion devices owned by over 6 billion people. This data deluge will need smart analytics to be used in a meaningful way, giving rise to newer innovative and efficient AI techniques. Focusing on developing AI algorithms, or just designing efficient AI chips, will not solve the complexity in data-insight analysis. We need to address this with a holistic system-level view that encompasses hardware (AI chips, storage, memory, and fabric), mature and open software (algorithms, libraries, and tools) and the tight co-design of hardware and software.

Intel's AI strategy offers the most diverse portfolio of highly performant and efficient compute and revolutionizes decades-old memory and storage hierarchies. It fills new gaps identified by AI workloads and enables the development of full-stack software based on open components. This simplifies AI deployments across increasingly heterogeneous hardware environments while integrating with existing frameworks and toolchains. The AI portfolio delivers a robust ecosystem of ready-made solutions for every industry.

From a compute perspective, Intel starts by extending its widely proliferated CPU portfolio to include specific technologies (such as Intel DL Boost) that allow enterprises to accelerate AI applications on existing, familiar infrastructure. From there, Intel offers multiple, discrete hardware accelerators for a wide range of programmability, performance, energy, and latency requirements from cloud to edge, including Intel FPGAs and Intel Movidius Vision Processing Units (VPUs). For the second generation of its Intel Xeon Scalable processors, Intel added specific instructions to accelerate neural network inference, especially when using numeric int8 instructions. These instructions, known as VNNI, give a theoretical speedup of up to 30x over fp32 instructions. The speedup highly depends on the frameworks, models, and the percentage of instructions that can actually use VNNI technology. Intel is also investing in other purpose-built accelerators to speed up deep learning training and inference.

Addressing software is a little more complex due to the various layers of the software stack; the fast growth of tools, libraries, and frameworks in the open source ecosystem; and the optimization of the software for specific hardware to run various types of AI applications and accommodate a variety of developer types.

Kernels for Library Developers

For library developers who write highly optimized code tailored to the specific hardware, Intel has offerings such as the Intel distribution of Python, the Intel Data Analytics Acceleration Library (DAAL), the nGraph Library, and the Intel Deep Neural Network Library (DNNL).

Intel DAAL is a high-performance machine learning and data analytics library. This library helps reduce the time it takes to develop high-performance data science applications, enabling applications to make better predictions faster and analyze larger data sets with available compute resources. It supports highly optimized analytic functions such as logistic regression, which is the most widely used classification algorithm. It extended gradient boosting functionality for inexact split calculations and user-defined procedures to enable with feature extraction of datasets. Intel DAAL supports Python via the Intel distribution for Python.

Intel DNNL, previously known as Intel MKL-DNN, is an open source performance library for deep learning (DL) applications intended for the acceleration of DL frameworks on Intel® architecture. The library includes basic building blocks for neural networks optimized for Intel CPUs and GPUs.

The nGraph Library is an open-source C++ library and runtime/compiler suite for deep learning ecosystems. With nGraph Library, data scientists can use their preferred deep learning framework on any number of hardware architectures, for both training and inference. nGraph Library supports multiple devices (CPU, GPU, ASIC) from multiple frameworks (TensorFlow, MXnet, ONNX, PyTorch).

Libraries for Data Scientists

In addition to the above open source libraries, Intel has optimizations for all popular deep learning frameworks such as TensorFlow, PyTorch, MXNet, PaddlePaddle, and ONNX runtime that can take advantage of Intel hardware features to make machine learning models run faster.

Toolkits for App Developers

Intel provides special toolkits that are designed to help developers to deploy their applications using any framework on the hardware. These toolkits include:

- **Analytics Zoo** – an Open source platform for building analytics and AI applications on Apache Spark with distributed TensorFlow, Keras, and BigDL.
- **OpenVINO** – a highly optimized and popular open source toolkit that deploys applications and solutions that emulate human vision. Based on convolutional neural networks (CNN), the toolkit extends computer vision (CV) and deep learning workloads across Intel® hardware on CPU/GPU/FPGA/VPU for Caffe, TensorFlow, MXNet, ONNX, and Kaldi.
- **Nauta** – an open source, scalable, and extensible distributed deep learning training platform built on Kubernetes and includes open source components and Intel-developed custom applications, tools, and scripts for ease-of-use and deployment.

Experimental Environment

We conducted experiments to show the benefit of using Intel DL Boost technology to accelerate inferencing on 2nd Generation Intel Xeon Scalable processors. Before we delve into the details of the experiments, we discuss the different parameters used for tuning to get the best results.

We ran the experiments with the Intel-optimized version of TensorFlow using pretrained models in fp32 and int8 precisions. These pretrained models are part of the TensorFlow model zoo and are available for everyone to use. We focused on the image classification model, ResNet50; we used the ImageNet validation dataset for inferencing predictions.

The benchmark is launched using a Python script benchmark_launcher.py that can take in the input parameters, validate the arguments, and initialize the platform. It also will initialize the model by locating it in the specified data path.

For the model itself, the framework used, the pretrained model precision and location, and the batch size need to be specified. In addition to these, the script allows the benchmark to be run in a pure “benchmark only” mode, where only the performance is measured, or in "accuracy mode” where the inferencing accuracy

is reported in TOP1 and TOP5. It is critical to run the accuracy mode at least once when changing hyper parameters to ensure that the inference is maintaining the accuracy specified by the pretrained model.

For initializing the platform parameters: socket_id, numbers of cores per socket, number of threads per core, and the number of inter threads and intra threads must be specified. The socket parameter indicates to the script if the inferencing must be run on only one socket of the underlying CPU. In some cases, affinitizing to a single socket gives better performance where the NUMA memory accesses could cause extra latency. However, if the inferencing needs more threads than available on a single socket, then using all the cores from both sockets in the system is useful and will give better performance in terms of throughput. This is especially useful when running large batch sizes where the images are distributed across many cores. The intra threads parameter refers to the number of cores to use, while the inter threads are usually set to indicate how many threads to use per core.

Other environment variables such as KMP_affinity can be specified either in the environment or in the launcher command line. This is used to indicate to the script how to distribute the threads to the physical cores. You can also use this to affinitize the threads to a particular set of cores such as {proclist= 1, 4, 8} that will run all the threads specified to run on core numbers 1, 4 and 8. The numbering scheme for the cores can be obtained by running the lscpu command in the shell.

In addition, you can specify the output file where the results will be collected. This is usually a text file that can be parsed to get the parameters used when the script actually ran and also the latency and throughput of the inferencing. In the output file, you must check the OMP_NUM_THREADS, which will be obtained from the num_intra_threads parameters specified while launching the script. A detailed list of thread assignment to the cores is also stored in the output file so you can determine on which cores the inferencing was executed. Analyzing the output log file is necessary to understand the behavior of the parameters to the launcher script.

Example

```
python3.6 /home/user1/AI/Launcher/models/benchmarks/launch_benchmark.py  --data-location
/home/user1/imagenet/ImageV --benchmark-only --in-graph
/home/user1/AI/pre_trained_models/resnet50_fp32_pretrained_model.pb --model-name resnet50 --
framework tensorflow --precision fp32 --mode inference --batch-size=$u  -num_intra_threads 20 –
num_inter_threads 1  -- warmup_steps=250 steps=500 > output_file

Set KMP_* environment variables to the specified value
Validate the args and initializes platform_util
Create model initializer for the specified model
```

Tests

Our tests used four different workloads, as shown in Table 1, below.

In **Single Image Classification Latency**, the benchmark program sent one image at a time through a pre-trained ResNet50 neural network model (setting the program's batch_size parameter to 1). The metric recorded was the latency, or average time to infer a single image, in milliseconds (msec). The benchmark program has a parameter that sets the number of threads for the calculations of the forward propagation of the neural network needed to classify the image. For this test, this parameter was optimized separately for the bare metal and virtualized tests. The Single Image Classification Latency test was used to compare both the performance of 2nd Generation Intel Xeon Scalable processors using the int8 quantization to that on 1st Generation Xeon Scalable processors using fp32 as well as to compare bare metal vs. virtualized performance.

In **Large Batch Image Classification Throughput**, the performance of large batch classification on Cascade Lake using the int8 quantization was compared to that on Skylake using fp32, both running on a single VM. The batch size was set to 1024 and the metric recorded was throughput in images per second. There was no constraint on the average image classification latency.

Single Image Classification Throughput Scaling measures the throughput of running single image classification in multiple instances as the number of instances are increased. For the bare metal case, the benchmark program was run in a single instance, and then run simultaneously in multiple instances, with the number of instances increasing from 2 to 8. For the virtualized case, the benchmark program was run simultaneously in separate VMs, with the number of VMs increasing from 1 to 8. The metric recorded was throughput, in images per second.

In **Multistream Image Classification Throughput**, the program sent multiple images at once (batch_ size > 1) through the ResNet50 inference engine. The maximum throughput achievable by the bare metal server and the virtualized server meeting a specified latency constraint was measured. The latency constraint used was 33.3 msec, equivalent to 30 video frames per second. The average latency reported by the program was used to compare to the latency constraint. The metric recorded was throughput in images per second.

Workload	Batch Size	Metric
Single Image Classification Latency	1	Latency (average time to infer single image) (milliseconds)
Large Batch Image Classification Throughput	1024	Throughput (images/second), no latency constraint
Single Image Classification Throughput Scaling	1	Throughput (images/second)
Multistream Image Classification Throughput	> 1	Throughput (images/second) with latency constraint

Table 1: Workloads Used in Image Classification Tests

Hardware and Software Configuration

The 1st Generation Intel® Xeon® Scalable processor ("Skylake") server and 2nd Generation Intel® Xeon® Scalable processor ("Cascade Lake") server configurations are shown in Table 2 below.

Component	Skylake Server	Cascade Lake Server
Processor	2x Intel® Xeon® Platinum 8160 @ 2.10GHz, 24C/48T, 33.00M Cache	2x Intel® Xeon ® Platinum 8260 @ 2.4GHz, 24C/48T, 35.75M Cache
Logical Processors (including hyperthreads)	96	96
Memory Type/Speed	DDR4 / 2666 MT/s	DDR4 / 2666 MT/s
Memory Size	768 GiB (24x 32 GiB DIMMs)	768 GiB (24x 32 GiB DIMMs)
NICs	2x 1 GbE ports + 2 x 10GbE ports	2x 1 GbE ports + 2 x 10 GbE ports
Non-Volatile Memory Express storage	2x Intel® Optane™ P4800 375 GiB 1x P4510 8TB	2x 1.6TB NVMe PCIe
Solid State Disks		8x 1.92 TB SSD SATA

Table 2: Server Configurations

The virtualized configurations used in the tests are shown in Table 3.

Configuration	Number of VMs	vCPUs per VM	Memory per VM (GB)	Total vCPUs Allocated	Total Memory Allocated (GB)
1	1	48	360	48	360
2	2	48	360	96	720
3	1	12	90	12	90
4	2	12	90	24	180
5	3	12	90	36	270
6	4	12	90	48	360
7	5	12	90	60	450
8	6	12	90	72	540
9	7	12	90	84	630
10	8	12	90	96	720

Table 3: Virtualized Configurations

Both the virtual machines and the bare metal server ran the Ubuntu 18.04 operating system.

Results

Single Image Classification Latency – Intel Xeon Scalable CPU + fp32 vs 2nd Gen Intel Xeon Scalable CPU + int8

In Single Image Classification Latency, the benchmark program's batch_size parameter was set to 1, sending one image at a time through a pre-trained ResNet50 neural network model. The performance metric was the latency, or average time to infer a single image, in milliseconds (msec).

The first Single Image Classification Latency test (Figure 2) demonstrated the performance improvement from the fp32 quantization running on the Intel® Xeon® Scalable processor ("Skylake") to int8 running on 2nd Gen Intel® Xeon® Scalable processor ("Cascade Lake") using the VNNI instruction. The classification performance improved by 2.18x due to the smaller quantization, the enhanced VNNI instruction set, and the architectural improvements of Cascade Lake over Skylake. In addition, the image classification accuracy of the int8 quantization was measured to be less than 1% less than the fp32 accuracy. The tests used a VM size of 48 vCPU and 360 GB of memory (Configuration 1 in Table 3).

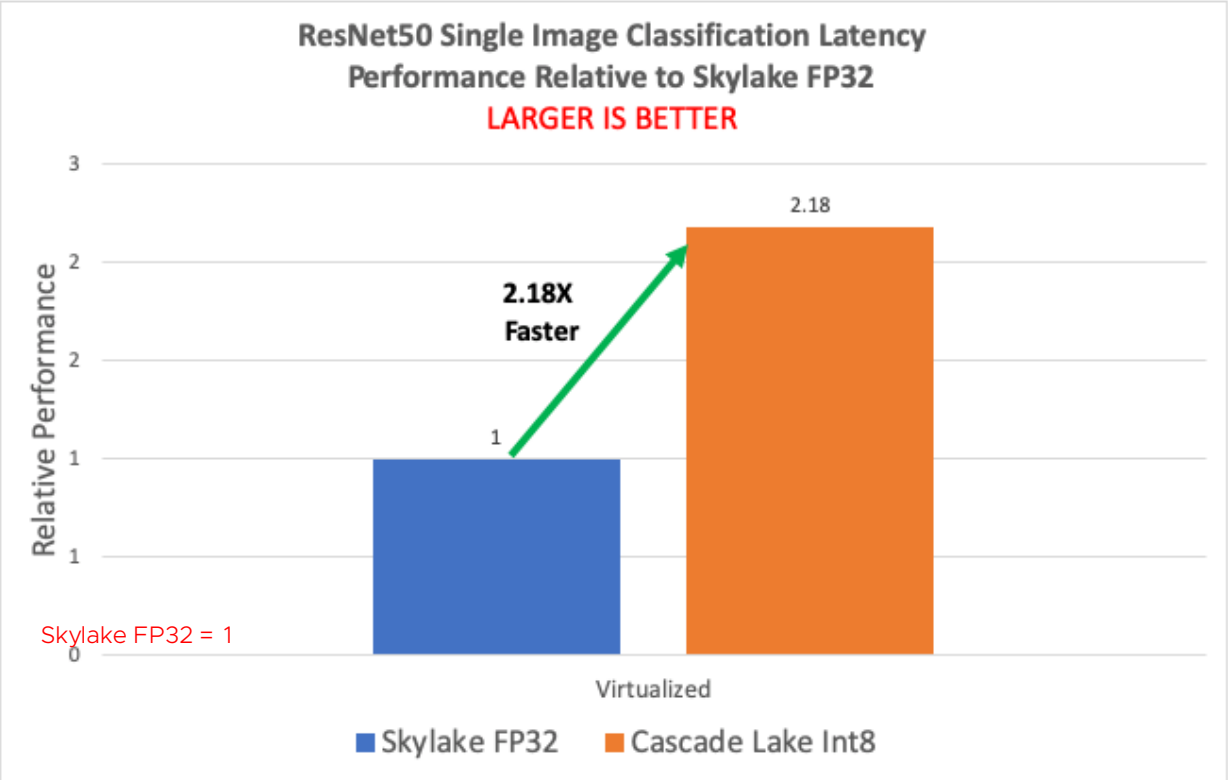

Figure 2: Single Image Classification Latency – Skylake fp32 vs Cascade Lake int8

Single Image Classification Latency – Virtualized vs Bare Metal – int8

The virtualization overhead was shown to be small (1.5%) running the Single Image Classification Latency int8 test on a 48-vCPU, 360 GB memory VM on a Cascade Lake server compared to running the same test on an identically configured Cascade Lake server (Figure 3).

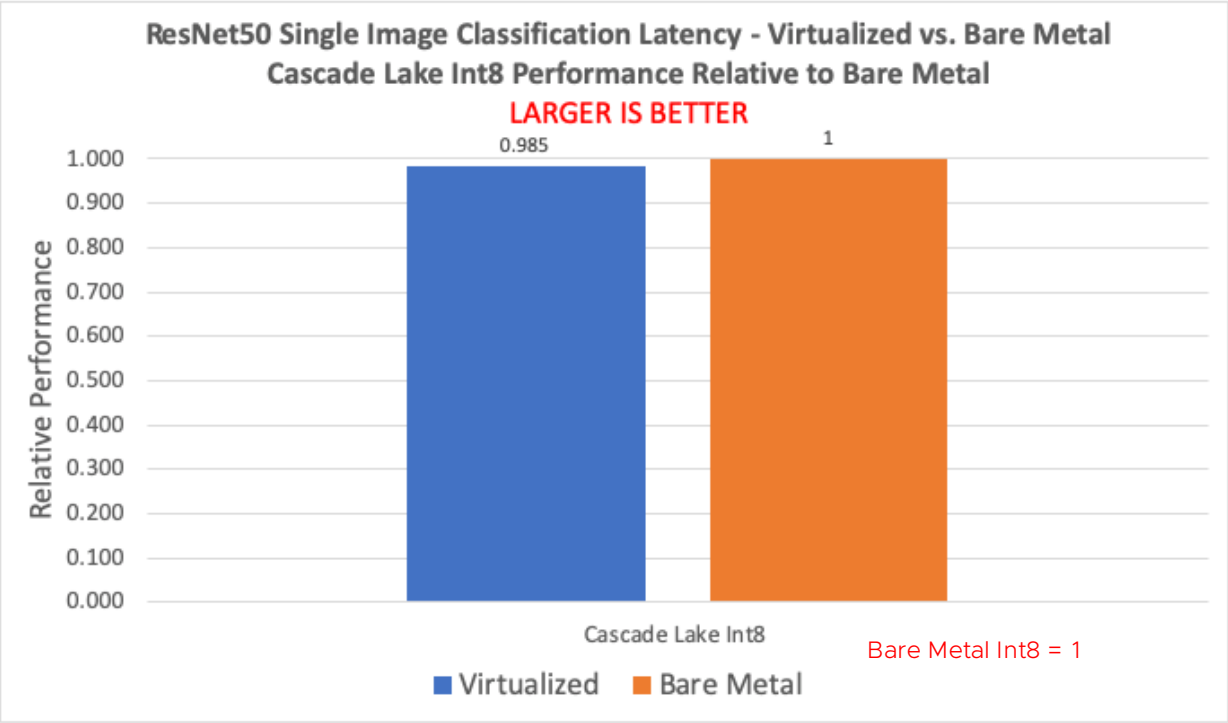

Figure 3: Single Image Classification Latency – Virtualized vs Bare Metal – int8

Single Image Classification Latency – Virtualized vs Bare Metal – fp32

The virtualization overhead using the fp32 quantization was shown as even smaller (0.6%) running the Single Image Classification Latency than the int8 test. Again, the test compared image classification latency on a 48-vCPU, 360 GB memory VM on a 2nd Gen Intel Xeon Scalable server compared to running the same test on an identically configured server (Figure 4).

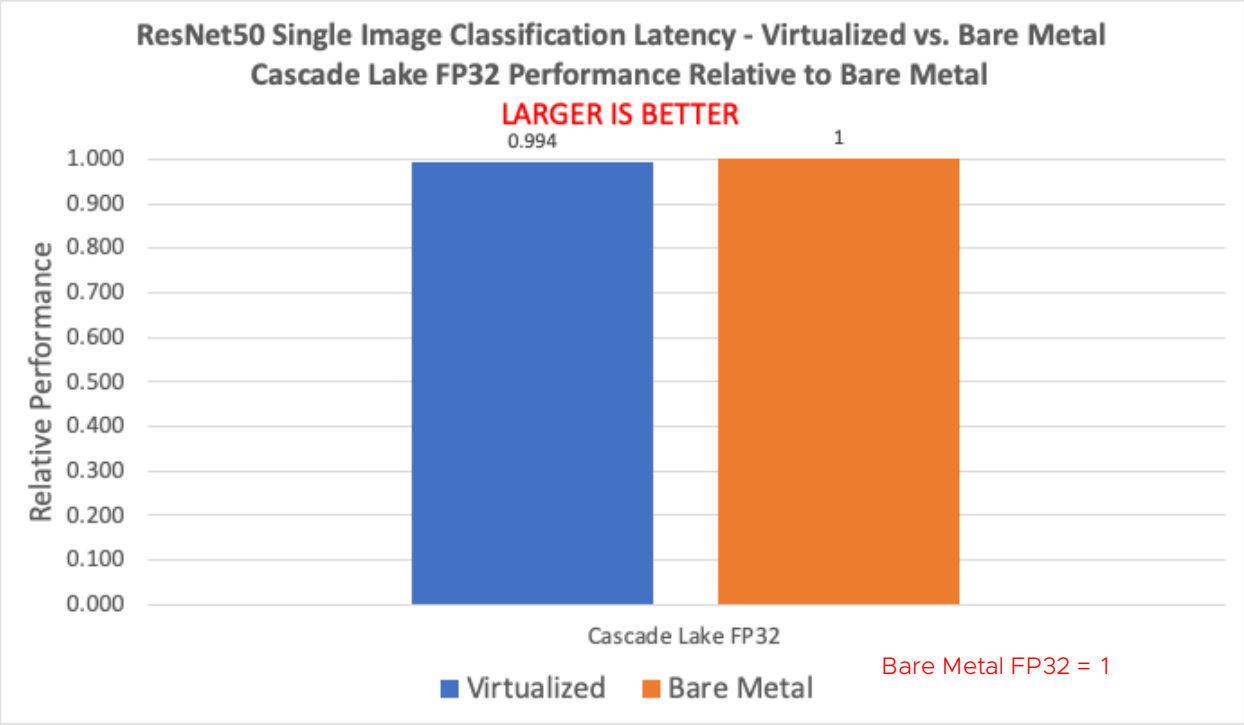

Figure 4: Single Image Classification Latency – Virtualized vs Bare Metal – fp32

Large Batch Image Classification Throughput – Intel Xeon Scalable CPU + fp32 vs 2nd Gen Intel Xeon Scalable CPU + int8

A second method for comparing the performance improvement from Intel Xeon Scalable CPUs using fp32 to 2nd Generation Intel Xeon Scalable CPUs using int8 was Large Batch Image Classification, in which VMs on both platforms were driven with the batch size set to 1024, and the resulting throughput was measured in images per second. A VM size of 96 vCPU and 720 GB was used (Configuration 2 in Table 3). As seen in Figure 5, the performance improvement, 3.49x, was even larger than the 2.18x shown in Figure 2, due to the large VNNI register as well as the other improvements already mentioned (VNNI instruction set, int8 quantization, enhanced Cascade Lake architecture).

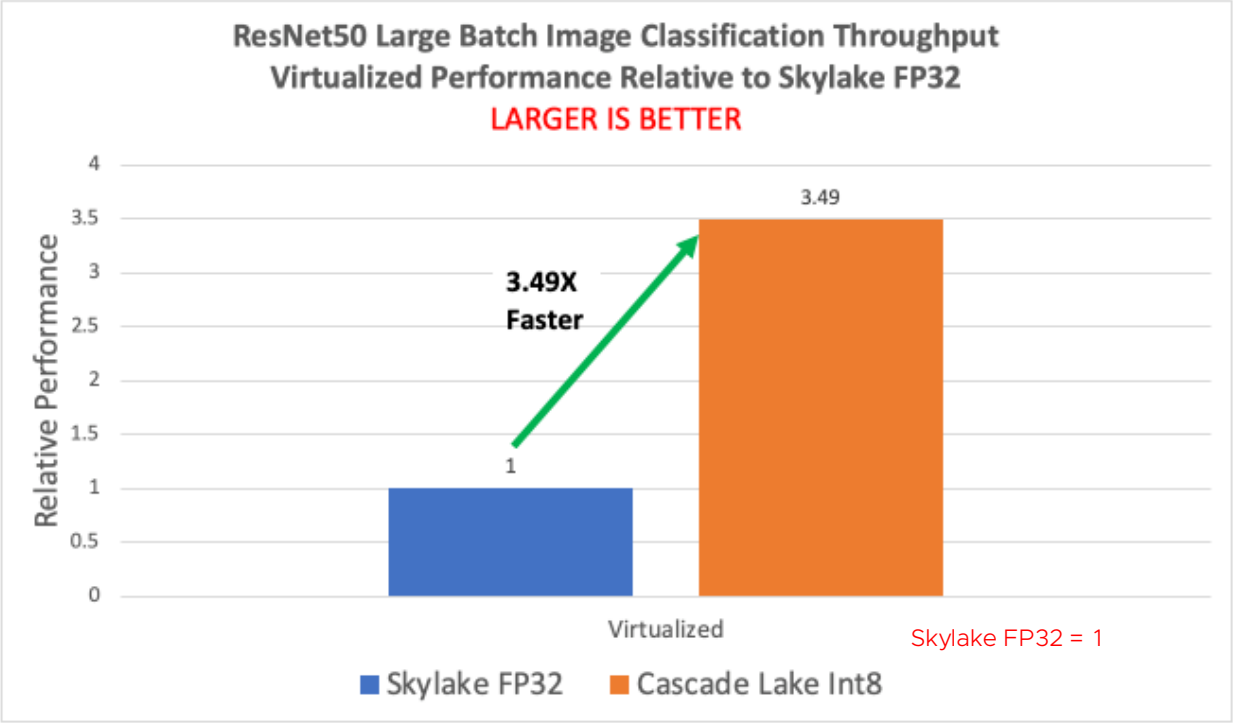

Figure 5: Large Batch Image Classification Throughput - Skylake fp32 vs Cascade Lake int8

Single Image Classification Throughput Scaling – VMs vs Bare Metal

In Single Image Classification Throughput Scaling, the Single Image Classification Latency tests (with either int8 or fp32 quantization) was run in multiple instances. For the bare metal case, the benchmark program was run in a single instance, and then run simultaneously in multiple instances, with the number of instances increasing from 2 to 8. For the virtualized case, the benchmark program was run simultaneously in separate VMs, with the number of VMs increasing from 1 to 8, using Configurations 3-10 from Table 3.

As shown in Figure 6, the single-instance bare metal result (either int8 or fp32), which uses the entire server, is faster than the virtualized result from one small (12 vCPU, 90 GB) VM, but as the number of bare metal instances and VMs increase, the total throughput in the virtualized case overtakes that of the bare metal case and ends up being 2.04x faster (int8) and 1.44x faster (fp32) due to the better resource utilization afforded by virtualization. The int8 throughputs are faster than those of fp32 due to the smaller quantization.

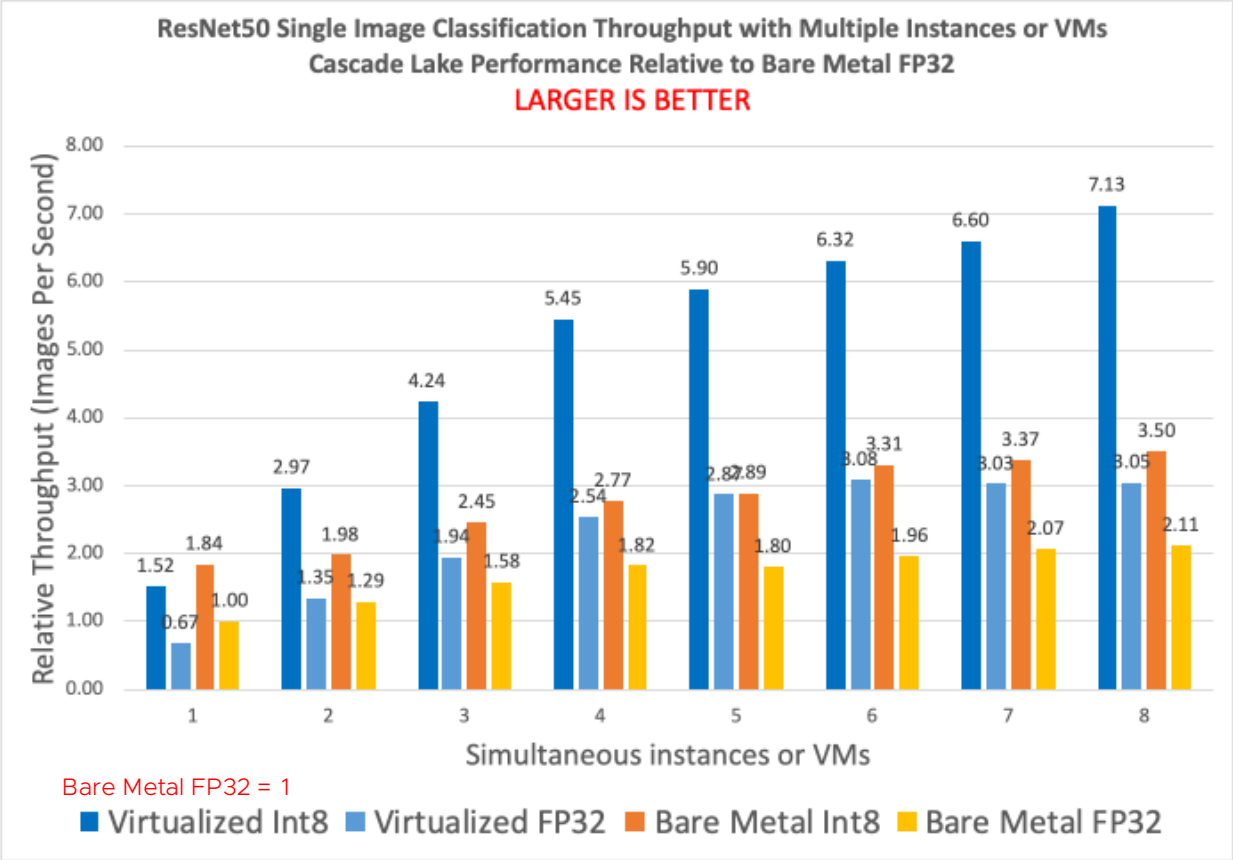

Figure 6: Single Image Classification Throughput Scaling – VMs vs Bare Metal

Multistream Image Classification Throughput – VMs vs Bare Metal

In Multistream Image Classification Throughput, the program sent multiple images at once through the ResNet50 inference engine. For both the bare metal and the virtualized servers, the batch_ size was increased while the average latency remained below 33.3 msec, equivalent to 30 video frames per second. The virtualized configuration consisted of two VMs, each with 48 vCPUs and 360 GB memory, to provide an optimal resource configuration for the workload. The resulting attainable throughput is shown in Figure 7. The virtualized int8 and fp32 were faster than their bare metal equivalents, but not by the same factor seen in the Single Stream Throughput Scaling. The multistream workload makes better use of the bare metal resources than does single stream, but running the same workload on two VMs is still faster due to better resource utilization

The batch_ size used was 15 (int8) and 5 (fp32) for the VMs, and 28 (int8) and 10 (fp32) for bare metal.

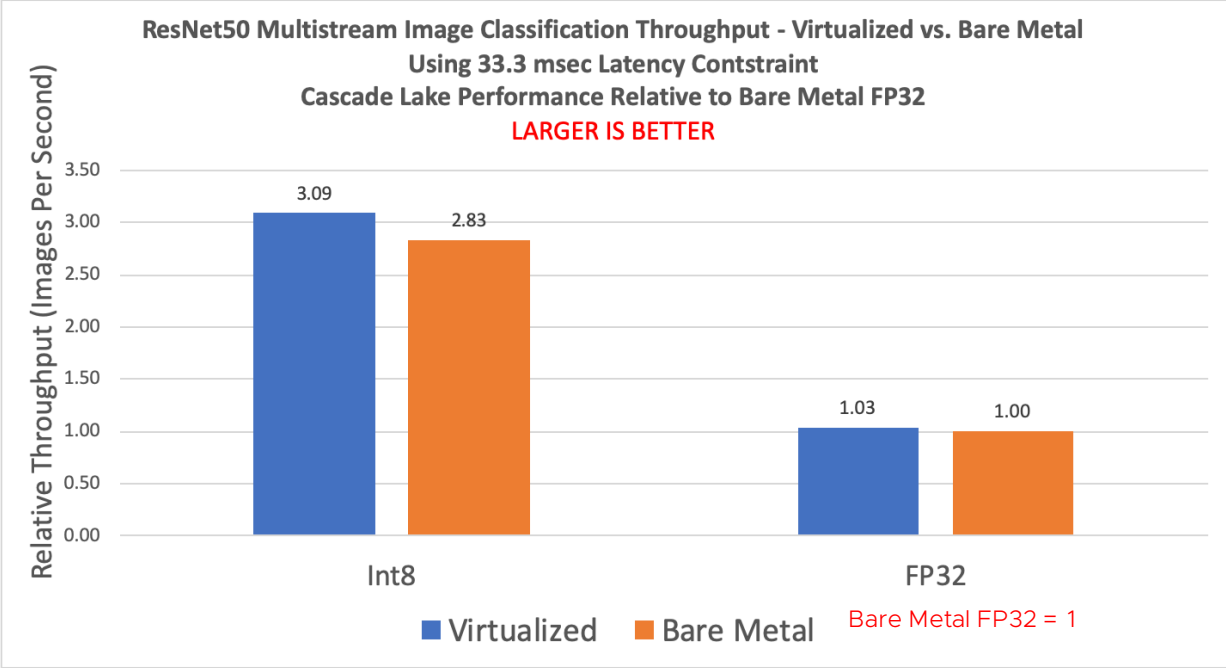

Figure 7: Multistream Image Classification Throughput – VMs vs Bare Metal

Conclusion

This study's results demonstrate the performance benefits of running deep learning inference on the 2nd Generation Intel® Xeon® Scalable processor compared to previous Intel processors, and the performance benefits of running on VMware's vSphere hypervisor compared to bare metal.

Two tests were run to compare the Intel processor generations, Single Image Classification Latency and Large Batch Image Classification. For the Single Image Classification, the classification performance improved by 2.18x from the fp32 quantization running on the Intel® Xeon® Scalable processor ("Skylake") to int8 running on 2nd Generation Intel® Xeon® Scalable processor ("Cascade Lake") due to the smaller quantization, the enhanced VNNI instruction set, and the architectural improvements of Cascade Lake over Skylake.

For Large Batch Image Classification, the performance improvement was 3.49x, even larger than the 2.18x in Single Image Classification, due to the large VNNI register, as well as the other improvements already mentioned (VNNI instruction set, int8 quantization, enhanced Cascade Lake architecture).

The rest of the tests compared Deep Learning inference performance on vSphere vs bare metal. The virtualization overhead was shown to be small (1.5% for the int8 quantization and 0.6% for fp32) running the Single Image Classification Latency int8 test on a VM on a Cascade Lake server compared to running the same test on an identically configured bare metal Cascade Lake server.

Single Image Classification Throughput Scaling shows slightly worse performance for one image classification instance running on a single, small VM versus running on a full bare metal server, but, as the number of bare metal instances and VMs increase, the total throughput in the virtualized case overtakes that of the bare metal case and ends up being 2.04x faster (int8) and 1.44x faster (fp32) due to the better resource utilization afforded by virtualization. The int8 throughputs are faster than those of fp32 due to the smaller quantization.

Finally, virtualized Multistream Image Classification Throughput was 9% (int8) and 3% (fp32) faster than when run on bare metal. The multistream workload makes better use of the bare metal resources than does single stream, but, running the same workload on two VMs is still faster due to better resource utilization.

References

[1] MarketWatch. (2019, February) 175 Zettabytes by 2025! A data deluge is round the corner. https://www.marketwatch.com/press-release/175-zettabytes-by-2025-a-data-deluge-is-round-the-corner-2019-02-20

About the authors

Dave Jaffe is an engineer on the VMware Performance Engineering team, focusing on big data and machine learning.

Padma Apparao is a principal engineer and the chief performance architect in the VMware Center of Excellence team at Intel, optimizing Intel Data Center technologies (artificial intelligence, persistent memory, and accelerators) on the VMware Cloud stack. She has over 20 patents and many published papers.