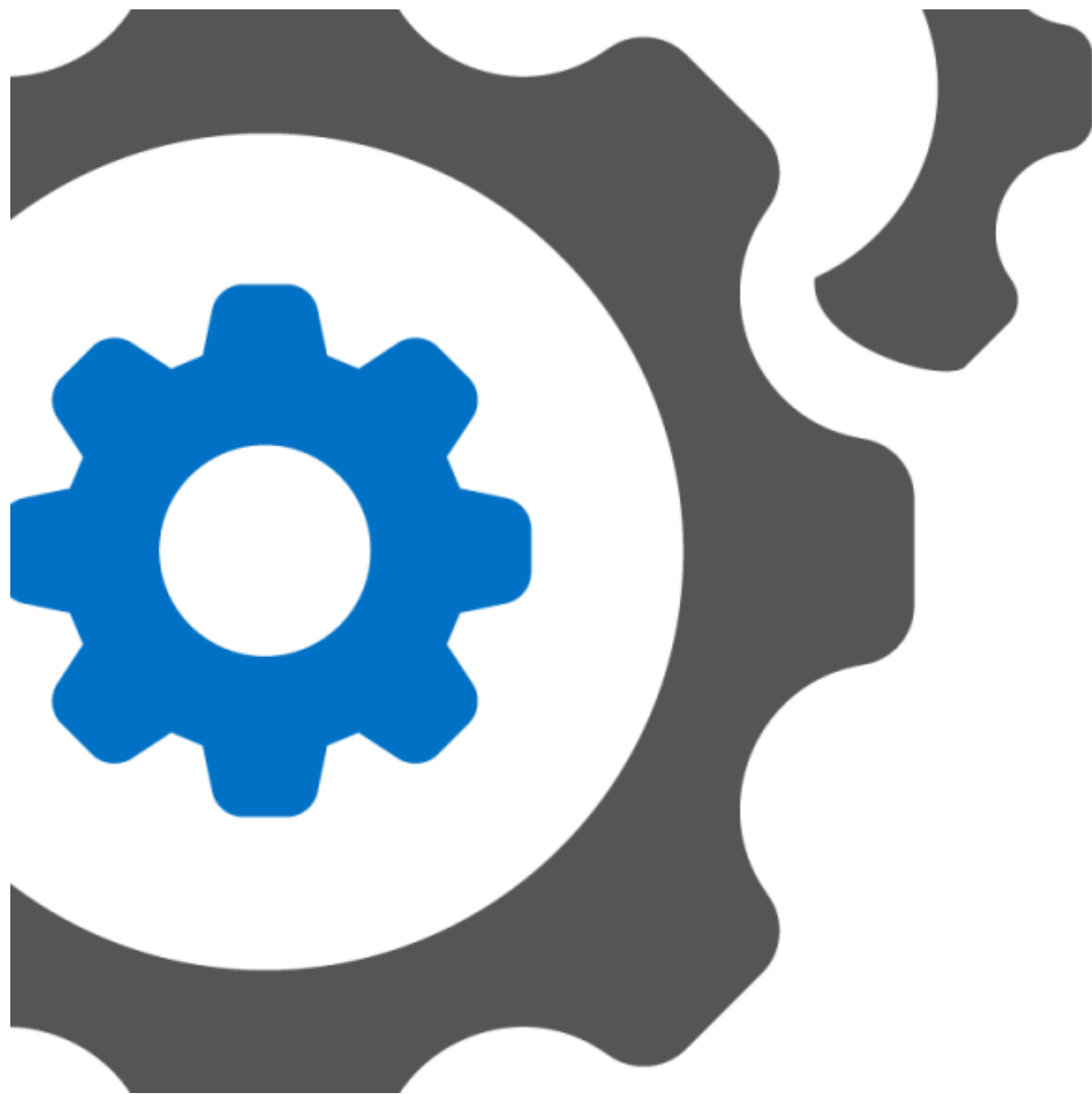




B L O G

04.26.18



INTEL
+
AWS

AMAZON WEB SERVICES WORKS WITH INTEL TO ENABLE OPTIMIZED DEEP LEARNING FRAMEWORKS ON AMAZON* EC2 CPU INSTANCES

With a wide array of compute, memory, and communication configurations, Amazon Web Services (AWS) offers a rich platform for building deep learning (DL) systems ^[1]. On top of this, their introduction of the Amazon* Deep Learning AMI (Amazon Machine Image) – which comes with pre-built deep learning environments – further enhances their DL offering by decreasing the time and effort needed to setup the most common frameworks ^[2].

Previously, these pre-built environments did not come optimized for execution on CPUs, and if a user desired a CPU instance for deep learning, they had to rebuild their framework of choice from source or install via a custom wheel to exploit its entire suite of available CPU optimizations. In response to this, AWS has updated their DL AMI to v6.0 which includes Intel® Optimizations for TensorFlow* – pre-built and ready to use with Intel® Math Kernel Library and Math Kernel Library for Deep Neural Networks (Intel® MKL and Intel® MKL-DNN) primitives – allowing data scientists and deep learning practitioners to get to work knowing their framework is optimized regardless of the hardware it's running on ^[3].

In this article, we will outline some of the optimizations enabling the increased throughput in Intel Optimization for TensorFlow and review additional BKM's for specializing the deep learning environment.

OPTIMIZATIONS FOR INTEL® ARCHITECTURE





1. Vectorization of primitive operations to the latest SIMD instructions (Intel® Advanced Vector Extensions 2 and Intel® Advanced Vector Extensions 512 (AVX2 and AVX512) for Xeon® processors) maximally exploits the ability for the CPU to parallelize computation. Operations like convolution, inner products, pooling, normalization, activation, etc. are built to leverage Intel MKL-DNN kernels wherever possible.
2. Graph optimizations seamlessly replace default TensorFlow operations in the Intel-optimized version, allowing the programmer to realize performance gains without making changes to the network architecture. These graph optimizations also eliminate costly data layout conversions while handling intermediate states for faster backpropagation.
3. Memory optimizations allow for TensorFlow and Intel MKL kernels to share the same memory pools while maintaining a symbiotic relationship on the CPU by not competing for compute resources. In addition to this, balanced use of prefetching, cache-blocking techniques and data formats are used to promote spatial and temporal locality.

In short, these optimizations enable greater performance while being transparent to the programmer. For more details on Intel's contribution to accelerating TensorFlow on CPUs, please see the excellent article by Elmoustapha et.al. ^[4].

For the data scientist, consequences of using Intel® Optimization for TensorFlow manifest as significant speedups across the deep learning design space. Figure 1 compares default TensorFlow (previously available in AWS' DL AMI v5.0) vs. Intel Optimization for TensorFlow (currently available in AWS' DL AMI v6.0) training throughput on four benchmark topologies: InceptionV3, ResNet50, ResNet152 and VGG16.

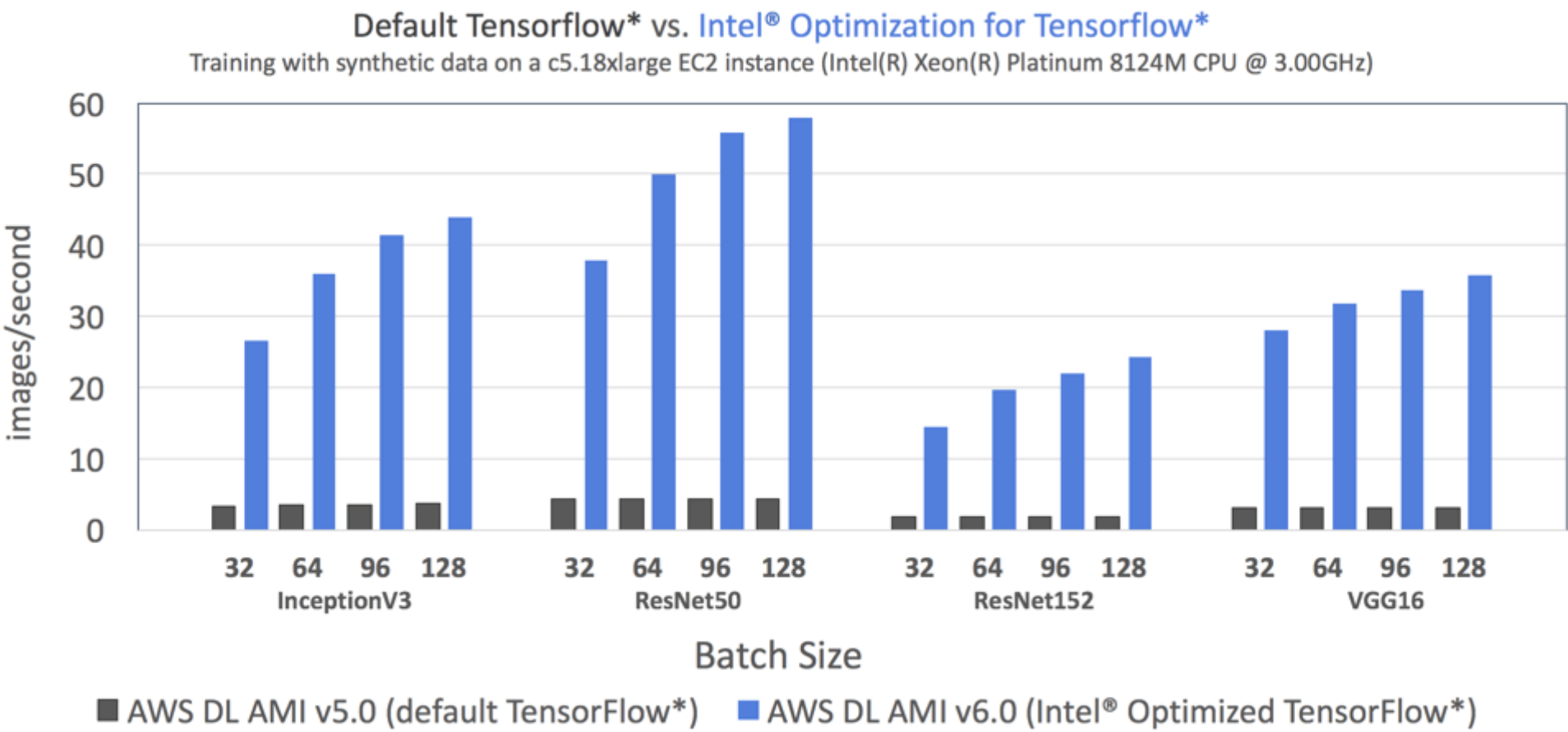


Figure 1: Utilizing Intel® MKL and MKL-DNN optimizations, Intel® Optimization for TensorFlow* in Amazon* Deep Learning AMI v6.0 outperforms default TensorFlow by up to 7.4X (with a batch size of 32) on all benchmark topologies ^[3]. See configuration details below.

Platform	Amazon EC2 c5.18xlarge instance
Computer Type	Cloud Compute Server
# of Sockets	2S

Processor	Intel® Xeon® Platinum 8124M CPU @ 3.00GHz (Skylake)
Enabled Cores	18 cores / socket
SSD	EBS Optimized, 128GB, Provisioned IOPS SSD
OS	Amazon Deep Learning AMI v6.0 (Ubuntu)

ENVIRONMENT BKMS

While using Intel Optimization for TensorFlow will help ensure that the aforementioned optimizations are being exploited during graph execution, there are a few additional environmental variables and TensorFlow parameters that must be set to ensure optimal core/memory utilization. Set the following environment variables, required for Intel MKL to achieve optimal performance, via the following commands within the TensorFlow virtual environment:

```
export OMP_NUM_THREADS=<num_physical_cores>

export KMP_AFFINITY=granularity=fine,verbose,compact,1,0

export KMP_BLOCKTIME=1

export KMP_SETTINGS=1

export OMP_PROC_BIND=true
```

Please see the official documentation on supported environment variables for detailed explanations of each of the above settings [5]. Additionally, there are two variables within TensorFlow that must be set for it to take full advantage of multi-core Intel® CPUs:

1. Intra-op threads: the size of the thread pool to parallelize kernels within some operation.
2. Inter-op threads: the size of the thread pool to run operations in parallel.

The optimal value for intra-op threads will be around the number of physical cores available to the process executing the computation graph, but will vary with topology, dataset and IO demands during training, and should be optimized by the user on a case-by-case basis. Some threads must be left open for extraneous processes of the OS, communication, etc. Setting the intra-op threads value to the full number of physical cores could result in resource exhaustion and crash the process – particularly if performing distributed computation.

Given the structure of most modern topologies, the optimal value for inter-op threads is usually 1. If using a custom topology, set this value to the number of disjoint branches in the computation graph.

These variables are declared within the TensorFlow script as follows:

```
config = tf.ConfigProto(

    inter_op_parallelism_threads=num_inter_op_threads,

    intra_op_parallelism_threads=num_intra_op_threads)
```

Note that Intel MKL is optimized for NCHW (channels_first) data format and there is an ongoing effort to get near performance parity when using NHWC.



BENCHMARKING

To replicate the benchmarking shown in Figure 1, take the following steps:

1. Launch a CPU instance (C4 or C5), choosing the Deep Learning AMI (Ubuntu). See ^[6] for instructions on launching and connecting to a Deep Learning AMI for CPU.
2. Once connected to the instance, run the following command to download the benchmark script:

```
git clone https://gist.github.com/MattsonThieme/60e7eba13d6dc80f7d69c52bebae4d19
```

If desired, the script is also available in plain text below.

1. Modify num_cores in `intel_tf_cnn_benchmarks.sh` to reflect the number of physical cores on your instance. This will be used as value of the `OMP_NUM_THREADS` environment variable and `intra_op_threads` TensorFlow parameter.
2. Initiate the benchmarking by running `bash intel_tf_cnn_benchmarks.sh` in your terminal.

```
#!/bin/bash
```

```
# Usage: bash intel_tf_cnn_benchmarks.sh
```

```
# Activate TensorFlow virtual environment
```

```
source activate tensorflow_p36
```

```
# Assign num_cores to the number of physical cores on your machine
```

```
num_cores=36
```

```
# Set environment variables
```

```
export KMP_AFFINITY=granularity=fine,verbose,compact,1,0
```

```
export KMP_BLOCKTIME=1
```

```
export KMP_SETTINGS=1
```

```
export OMP_NUM_THREADS=$num_cores
```

```
export OMP_PROC_BIND=true
```



```
git clone -b mkl_experiment https://github.com/tensorflow/benchmarks.git

cd benchmarks/scripts/tf_cnn_benchmarks

rm *.log # remove logs from any previous benchmark runs


# Run training benchmark scripts

networks=( alexnet googlenet inception3 resnet50 resnet152 vgg16 )

batch_sizes=( 1 32 64 96 128 )

for network in "${networks[@]}" ; do

for bs in "${batch_sizes[@]}"; do

echo -e "\n\n#### Starting $network with batch size = $bs ####\n\n"

time python tf_cnn_benchmarks.py --device cpu --data_format NCHW --cpu skl --data_name synthetic --model
"$network" --learning_rate 0.001 --num_epochs_per_decay 2 --batch_size "$bs" --optimizer rmsprop --
num_intra_threads $num_cores --num_inter_threads 1 --num_omp_threads $num_cores --num_batches 100 2>&1 | tee
net_"$network"_bs_"$bs".log

done

done


# Print training benchmark throughput

echo -e "\n Network batch_size images/second \n"


for network in "${networks[@]}" ; do

for bs in "${batch_sizes[@]}"; do

fps=$(grep  "total images/sec:"  net_"$network"_bs_"$bs".log | cut -d ":" -f2 | xargs)

echo "$network $bs $fps"

done

echo -e "\n"

done
```

[source](#) [deactivate](#)

GETTING STARTED

AWS' adoption of Intel Optimization for TensorFlow begins the democratization of CPU optimizations for deep learning, and as access to the full suite of Intel MKL and MKL-DNN optimizations becomes the default, reduced cycle times will accelerate the pace of development on CPUs. As part of an ongoing collaboration, Intel and Amazon plan to add additional CPU-optimized deep learning frameworks to complete the CPU DL AMI offering. To launch your own deep learning instance and start building with Intel Optimization for TensorFlow, please visit <https://aws.amazon.com/machine-learning/amis/>.

References

1. <https://aws.amazon.com/deep-learning/>
2. <https://aws.amazon.com/machine-learning/amis/>
3. <https://aws.amazon.com/blogs/machine-learning/faster-training-with-optimized-tensorflow-1-6-on-amazon-ec2-c5-and-p3-instances/>
4. <https://software.intel.com/en-us/articles/tensorflow-optimizations-on-modern-intel-architecture>
5. <https://software.intel.com/en-us/node/522775>
6. Get Started with Deep Learning Using the AWS Deep Learning AMI with Intel CPUs: [PDF](#)

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Intel, the Intel logo, and Xeon are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.


**Other names and brands may be claimed as the property of others.*

© Intel Corporation.


AUTHORS

Mattson Thieme


Ravi Panchumorthy




Machine Learning Engineer, Artificial Intelligence Products Group




Machine Learning Engineer, Artificial Intelligence Products Group



Prashant Shah
Director of Engineering of Health and Life Sciences,
Artificial Intelligence Products Group



Search AI...



T A G S

Amazon Web Services (AWS), Intel Xeon Processors, AWS, Benchmarking, Framework Optimizations, Frameworks, Intel® Xeon® Processors, Open Source, TensorFlow, Xeon®

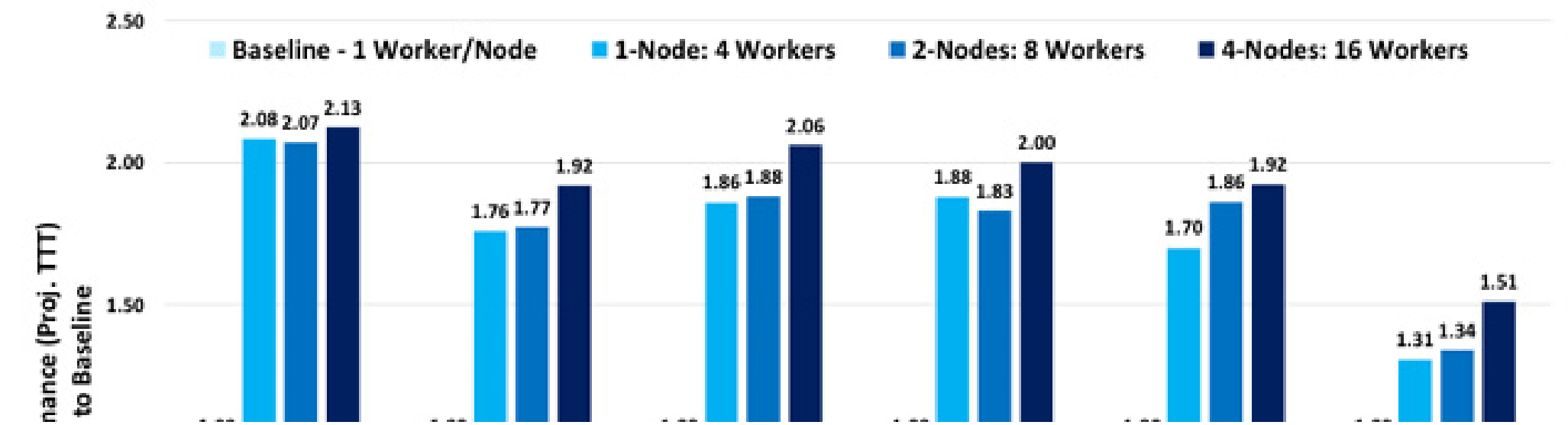
R E L A T E D C O N T E N T

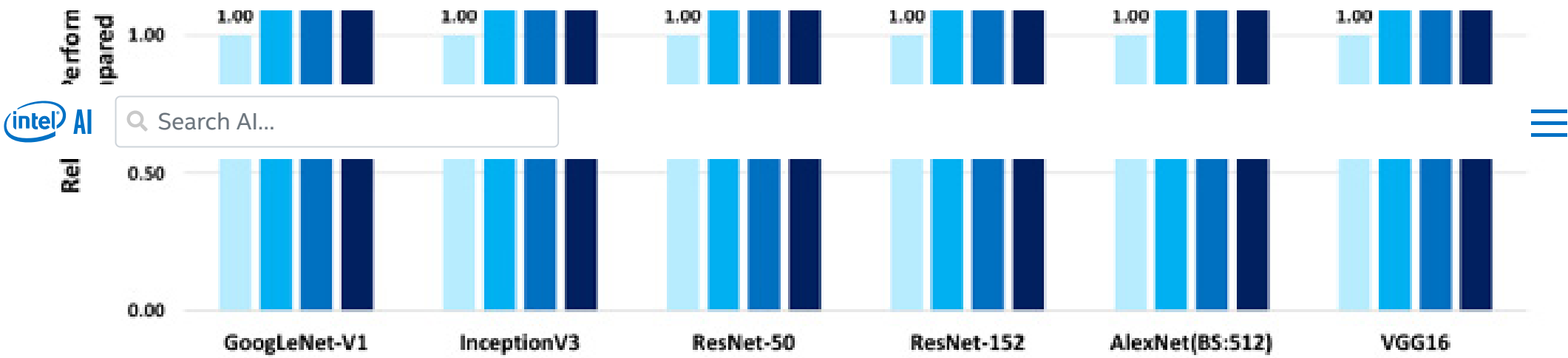


AMAZING INFERENCE PERFORMANCE WITH INTEL® XEON® SCALABLE PROCESSORS

Over the past year, Intel has focused on optimizing popular deep learning frameworks and primitives for Intel® Xeon® processors. Now,...

[Read More](#)





ACCELERATING DEEP LEARNING TRAINING AND INFERENCE WITH SYSTEM...

The neon™ deep learning framework was created by Nervana Systems to deliver industry-leading performance. As of 2018, the neon framework...

[Read More](#)










Stay Connected

Keep tabs on all the latest news with our monthly newsletter.

Subscribe

-  @intelai
-  @intelairesearch
-  @intelai
-  intel-ai
-  intelai
-  nervanasystems