Training Multiscale-CNN for Large Microscopy Image Classification in One Hour

Kushal Datta*,1[0000−0003−1608−6040], Imtiaz Hossain*,2[0000−0001−6747−5906], Sun Choi1[0000−0003−4276−7560], Vikram Saletore1[0000−0001−8642−539X], Kyle Ambert1[0000−0002−1688−4408], William J. Godinez3[0000−0003−4753−4942], and Xian Zhang2[0000−0002−7337−747X]

[1] Artificial Intelligence Products Group, Intel Corporation, USA
[2] Novartis Institutes for Biomedical Research, Basel, Switzerland
[3] Novartis Institutes for Biomedical Research, Emeryville, CA, USA
{kushal.datta,sun.choi,vikram.a.saletore,kyle.h.ambert}@intel.com
{imtiaz.hossain,william_jose.godinez_navarro,xian-1.zhang}@novartis.com

Abstract. Existing approaches to train neural networks that use large images require to either crop or down-sample data during pre-processing, use small batch sizes, or split the model across devices mainly due to the prohibitively limited memory capacity available on GPUs and emerging accelerators. These techniques often lead to longer time to convergence or time to train (TTT), and in some cases, lower model accuracy. CPUs, on the other hand, can leverage significant amounts of memory. While much work has been done on parallelizing neural network training on multiple CPUs, little attention has been given to tune neural network training with large images on CPUs. In this work, we train a multi-scale convolutional neural network (M-CNN) to classify large biomedical images for high content screening in one hour. The ability to leverage large memory capacity on CPUs enables us to scale to larger batch sizes without having to crop or down-sample the input images. In conjunction with large batch sizes, we find a generalized methodology of linearly scaling of learning rate and train M-CNN to state-of-the-art (SOTA) accuracy of 99% within one hour. We achieve fast time to convergence using 128 two socket Intel Xeon 6148 processor nodes with 192GB DDR4 memory connected with 100Gbps Intel Omnipath architecture.

1 Introduction

Biomedical image analysis has been a natural area of application for deep convolutional neural networks (CNNs). Several uses of CNN-related topologies have been proposed in radiology[1,2], histopathology [3–5] and microscopy [6–8] (for a review, see [9]). High-content screening (HCS) [10–15], the use of microscopy at

* *To whom correspondence should be addressed.*
* *These authors have made equal contributions to the paper.*

scale in cellular experiments, in particular, has seen progress in applying CNN-based analysis [6,7,16–18]. Instead of the conventional analysis approaches where cellular objects are first segmented and then pre-defined features representing their phenotypes (characteristic image content corresponding to the underlying experimental conditions) are measured, deep learning approaches offer the promise to capture relevant features and phenotypes without *a priori* knowledge or significant manual parameter tuning. In deep CNNs, the deeper layers pick up high-levels of organization based on the input of many features captured in previous layers. Typically, a pooling operation (or a higher stride length in the convolution filter) is used to subsample interesting activations from one layer to the next, resulting in ever-coarser "higher-level" representations of the image content.

Despite the potential of deep learning in analyzing biomedical images, two outstanding challenges, namely the complexity of the biological imaging phenotypes and the difficulty in acquiring large biological sample sizes, have hindered broader adoption in this domain. To circumvent these challenges, architectural changes have been introduced into some models to make training easier without trading off model accuracy. One novel approach is to use wide networks, which explicitly model various levels of coarseness. In these topologies, several copies of the input image are downsampled and used to train separate, parallel convolutional layers, which are eventually concatenated together to form a single feature vector that is passed on to fully-connected layers (e.g., see Buyssens et al.[19]). A recent application of this idea to HCS is the Multiscale Convolutional Neural Network (M-CNN) architecture [16], which has been shown to be generally applicable to multiple microscopy datasets, in particular for identifying the effect of compound treatment.

The computational footprint of M-CNN, although relatively small as compared with other deep CNNs (e.g., Residual Neural Network 152), is still large when applied to high-content cellular imaging. Thus, it is important that model-related aspects of memory utilization and training performance are thoroughly understood, and that an end user knows *a priori* how to get maximum performance on their hardware. Commercial cloud service providers (CSPs) like Microsoft, Google, or Amazon–as well as on-premise HPC centers in academia and industry–are exploring custom hardware accelerator architectures, such as application-specific integrated circuits (ASICs) [20] or GPUs, to expedite training neural network models. In spite of the popularity of these technologies, several factors such as higher financial cost of ownership, lack of virtualization and lack of support for multi-tenancy, leading to poor hardware utilization, may be cited as reasons to consider CPU-centric performance optimizations in reducing the time-to-train for such models. Importantly, since almost all data centers, are already equipped with thousands of general-purpose CPUs, it makes a strong case for such an approach.

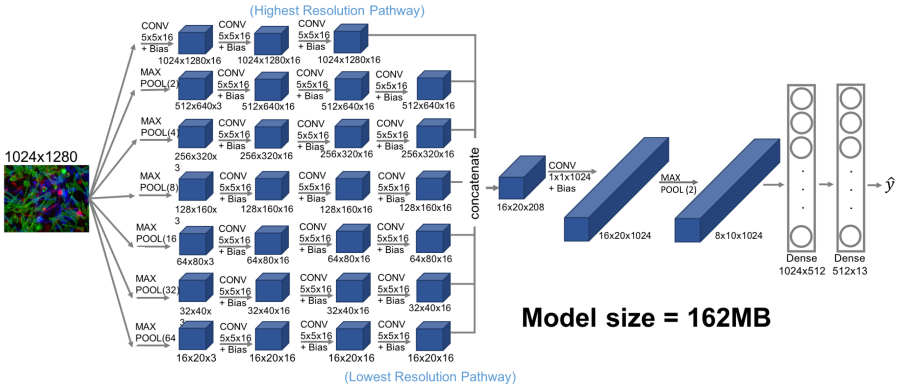


Fig. 1: Operations and kernels of the M-CNN model. Convolution is abbreviated *CONV*, and Max Pooling operations are abbreviated as *MAX POOL*

Existing approaches to improve the time to train convolutional image classification neural network model such as M-CNN designed to work with large high-content cellular images have needed to either crop or down-sample the images during pre-processing. Other ideas are to restrict to small batch sizes or split the model across multiple devices due to the limited memory capacity available on GPUs or accelerator cards. However, these techniques can lead to longer time to convergence or time to train (TTT), and in some cases, lower model accuracy. CPUs, on the other hand, can leverage large memory. Our primary contributions include,

1. Train M-CNN to achieve SOTA accuracy of 99% on multiple CPU servers without tiling or cropping of input images or splitting the model
2. Use large batch sizes per CPU exploiting large memory
3. Use multiple training instances/workers per CPU node to improve utilization
4. Use large batches and learning rate scaling to achieve fast convergence

The ability to leverage large memory capacity on CPUs enables us to scale to larger batch sizes without having to crop or down-sample the input images. In conjunction with large batch sizes, we linearly scale learning rate with global batch size and train M-CNN to SOTA accuracy within one hour. We achieve this fast time to convergence using 128 two socket Intel Xeon 6148 processor nodes with 192GB DDR4 memory connected with 100Gbps Intel Omnipath architecture.

2 Multi-scale convolutional neural network

M-CNNs capture both fine-grained cell-level features and coarse-grained features observable at the population level by using seven parallel convolution pathways (Figure 1). As in [16], image height and width are down-sampled by 64, 32, 16,

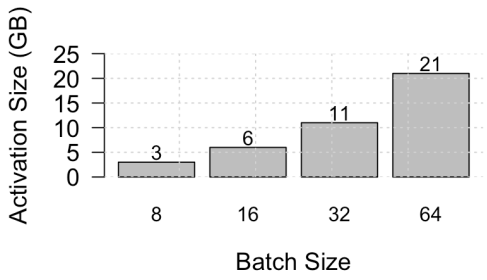


Fig. 2: Activation sizes in M-CNN as a function of batch size.

8, 4, and 2 times in the lower six pathways in ascending order, respectively, while images processed by the top-most path are operated on at the full resolution. The output of the last layers of convolution are down sampled to the lowest resolution and concatenated into a $16 \times 20 \times 208$ tensor. The concatenated signals are passed through a convolution with rectified linear activation (ReLU) and two fully connected layers. A final softmax layer transforms probabilistic per-class predictions associated with each image into a hard class prediction. In Figure 1, the size of convolution kernels are specified below the solid colored cubes, which represent the activations. The sum of the sizes of the convolution kernels and two dense layer, which are 1024×512 and 512×13, respectively, is 162.2 megabytes. Weights are represented as 32-bit floating point numbers.

The network's gradient and activation size determine the lower bound of its memory footprint. We plot the calculated activation size of the feed forward network as the global batch size is scaled from 8 to 64 by factors of two in Figure 2. Note that the size of variables required for back propagation is identical to the size of the gradients and hence is determined by model size, not activation size.

3 Large batch training

Synchronous gradient descent and data-level parallelism are fundamental concepts to training a deep neural network. In this domain, the most common algorithm used for training is stochastic gradient descent (SGD), which exploits the fact that activation functions in a neural network are differentiable with respect to their weights. During training, batches of data are run through the network. This process is referred to as *forward propagation*. A loss function E is computed at each training iteration, which quantifies how accurately the network was able to classify the input. The SGD algorithm then computes the gradient $\nabla_W(E)$ of the loss function with respect to the current weights W. On the basis of the gradients, weights are updated according equation 1, where W_{t+1} are the updated

weights, W_t are the weights prior to the adjustment (or previous iteration), and λ is a tunable parameter called the learning rate (LR).

$$W_{t+1} = W_t - \lambda \nabla_W E \tag{1}$$

Since each neural network layer is a differentiable function of the layer preceding it, gradients are computed layer-by-layer, moving from output to input in a process called backpropagation. Finally, the weights in the network are updated according to the computed gradient, and both forward and backpropagation are repeated with a new batch of data. We continue repeating these procedures until the network has reached a satisfactory degree of accuracy on a hold-out validation data set. Training can require running millions of iterations of this process on a given dataset. The most popular approach to speeding up network training makes use of a data-parallel algorithm called synchronous SGD [21]. Synchronous SGD works by replicating SGD across compute nodes, each working on different batches of training data simultaneously. We refer to these replicas as *workers*. A key requirement for synchronous SGD is for information to be synchronized and aggregated across all computing instances at each iteration. The update equation is show in equation 2, where B denotes the batch sampled from the training data, n is the size of the batch.

$$W_{t+1} = W_t - \lambda \frac{1}{n} \sum_{x \in B} \nabla_W E(x) \tag{2}$$

With k workers each training with B batches and learning rate λ', we updates the weights according to

$$W_{t+1} = W_t - \lambda' \frac{1}{kn} \sum_{j<k} \sum_{x \in B_j} \nabla_W E(x) \tag{3}$$

Thus, if we adjust the learning rate by k, the weight update equation stays consistent with the synchronous SGD update rule, helping the model to converge without changing the hyper-parameters. We refer to n or $|B|$ as the *local batch size*, and kn as the *global batch size*.

3.1 Learning rate schedule

In addition to scaling the model's learning rate parameter (LR) with respect to the batch size, others [22] have observed that gradually increasing it during initial epochs, and subsequently decaying it helps to the model to converge faster. This implies that LR is changed between training iterations, depending on the number of workers, the model, and dataset. We follow the same methodology. We start to train with LR initialized to a low value of $\lambda = 0.001$. In the first few epochs, it is gradually increased to the scaled value of $k\lambda$ and then adjusted following a polynomial decay, with momentum SGD (momentum=0.9).

Reaching network convergence during training is not guaranteed–the process is

sensitive to LR values and features in the data. Scaling this process out to large batch sizes on multiple workers concurrently has the same considerations. If the per-iteration batch size is too large, fewer updates per epoch are required (since an epoch is, by definition, a complete pass through the training data set), which can either result in the model diverging, or it requiring additional epochs to converge (relative to the non-distributed case), defeating the purpose of scaling to large batch sizes. Thus, demonstrating scaled-out performance with large batches without first demonstrating convergence is meaningless. Instead, we measure the time needed to reach state of the art accuracy or TTT. The ingestion method for each worker ensures that each minibatch contains randomly-shuffled data from the different classes.

4 Dataset

The Broad Bioimage Benchmark Collection BBBC021 image set [23] is a collection of 13,200 images from compound treatment on MCF-7 breast cancer cells. Each image consists of three channels: the cells are labeled for DNA, F-actin, and B-tubulin and imaged with fluorescence microscopy. Metadata on compound treatment and concentration is also available[24]. In all, 113 compounds have been used, each with varying concentrations and tested between 2 and 3 times each. Mechanism of action (MoA) labels are available for 103 compound-concentrations (38 compounds tested at between one and seven different concentrations each). In all, 13 MoAs (including the neutral control, DMSO) were available: 6 of the 12 MoAs were assigned visually. DMSO treatments were treated as neutral control and assigned a separate label. The others were defined based on information on the respective compounds in the available literature. We choose 1684 images from the BBBC021 dataset that are representative of all of the MoAs present. The distribution of the images according to MoA classes is shown in Figure 4. The images are preprocessed and normalized as described in [16]. From the 1684 images, we create two datasets with different augmentation strategies:

- *Dataset A*: Images in this dataset are 1024x1280 pixesl wide with 3 channels. They are augmented to produce five copies as 1. 90° rotation, 2. a horizontal mirror, 3. vertical mirror, 4. 90° rotation of horizontal mirror and 5. 90° rotation of vertical mirror. Total number of images in the dataset is $1684 * 6$ (five rotations + original) = 10104. We take a 90-10 split and create a training set of 9093 images and validation set of 1011 images. The total size of the images on disk are 38GB.
- *Dataset B*: This is a larger dataset. The dimensions of the images in this dataset are 724x724 pixesl with 3 channels. Similar to Dataset A, all images have 5 additional augmentations. Additionally, each image is rotated by 15° to create 23 more augmentations. The total size of the images on disk are 512GB. Among them, 313282 images are used for training and 35306 are used for validation.

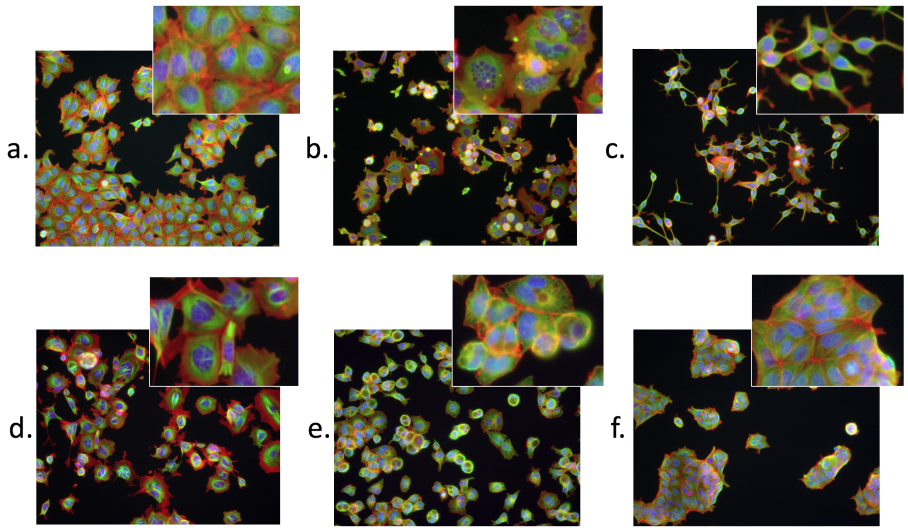


Fig. 3: Example images from the BBBC021[23] dataset showing phenotypes from treatment with compound-concentration pairs with different mechanisms of action: a) DSMO (neutral control), b) Microtubule destabilizer, c) Cholesterol lowering, d) Microtubule stabilizer, e) Actin disrupter, f) Epithelial. DNA staining is shown in blue, the F-actin staining in red and the -tubulin staining in green. The insets show a magnified view of the same phenotypes.

Ideally, we would have allocated a representative out-of-sample set of images as a validation set. However due to the paucity of MOA annotations in this dataset, and the fact that the main objective of this exercise is to reduce time to convergence, we allow for the fact that the validation dataset may contain an augmented version of an image in the training data, although never a copy of the same image.

5 Performance results

5.1 Experimental setup

All experiments are run on two socket (2S) 2.40GHz Intel® Xeon® Gold 6148 processors. There are 20 cores per socket with 2-way hardware multi-threading. On-chip L1 data cache is 32KB. L2 and L3 caches are 1MB and 28MB respectively. For multi-node experiments, we used up to 64 Intel® Xeon® Gold connected via 100gigabits/second Intel® OP Fabric. Each server has 192GB physical memory and a 1.6TB Intel SSD storage drive. The M-CNN topology was added to the standard benchmarking scripts [25] to leverage instantiation mechanisms of distributed workers. Gradient synchronization between the workers

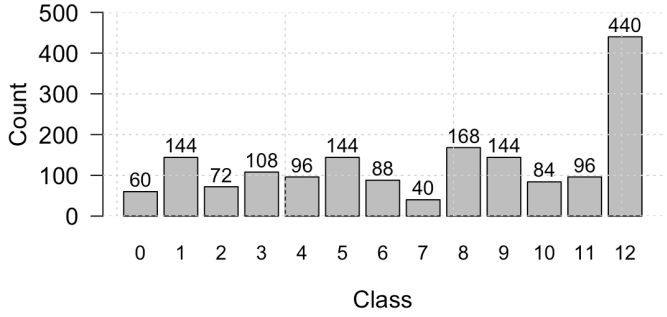


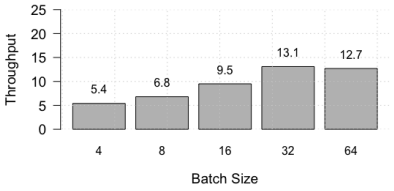
Fig. 4: Class distribution for the 1684 training images used in our experiment.

was done using Horovod, an MPI-based communication library for deep learning training [26]. In our experiments, we used TensorFlow 1.9.0, Horovod 0.13.4, Python 2.7.5 and OpenMPI 3.0.0.

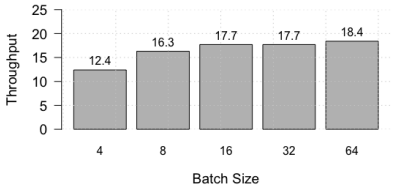
5.2 Scaling up TTT in One Node with Dataset A

We first performed a sweep of batch sizes from 4, 8, 16, 32 and 64 to check how fast we can converge on one CPU server. We acheived convergence in 5hrs 31mins with batch size = 32. The resulting throughput and memory consumed are shown in Figure 5 (a) and Figure 5 (c), respectively. As shown in the latter figure, the memory footprint of M-CNN far exceeds the activation size of the model. For example, in case of batch size of 32, total memory used is 47.5GB which is 4x larger than activation size of 11GB as calculated in Figure 2. The additional memory is allocated by TensorFlow to instantiate temporary variables used in both forward and backward propagation, buffers to read data and others operations. Due to these overheads, memory utilization of M-CNN is prohibitively high and it is difficult to scale to large batch sizes when memory in the system is limited.

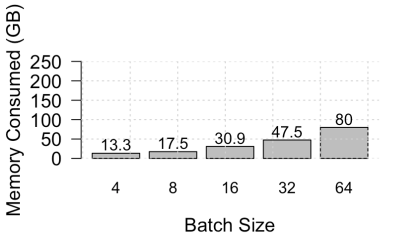
Second, for all batch size configurations, CPU utilization was low meaning the cores were under-utilized. Upon further investigation with system profile, we found 1) there were lots of context switches and 2) processes or threads assigned to one CPU socket are accessing data from the other CPU socket including a long latency hop over the socket-to-socket interconnect. This led to the discovery that using multiple workers or instanes per socket can yield faster TTT. The essence of using multiple workers in a single CPU is to affinitize tasks to cores and bind their memory allocation to local non-uniform memory access (NUMA) banks as shown by the shaded rectangles in Figure 6. Memory binding is key here as it avoids redundant fetches over the interconnect to the memory channels of the adjacent CPU socket. More detailed analysis of multiple workers or instances in training and inference are described in detail by Saletore and colleagues, in [27].



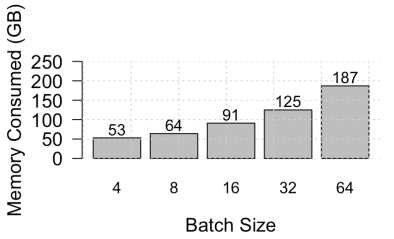
(a) Throughput (in images/sec) – 1 worker



(b) Throughput (in images/sec) – 4 workers



(c) Memory (in GB) – 1 worker



(d) Memory (in GB) – 4 workers

Fig. 5: Throughput (in images/second) and memory utilized (in GB) with batch sizes 4 to 64 for 1 and 4 training workers respectively (a and b) on a single 2S Intel® Xeon® Gold 6148 processor with Dataset A.

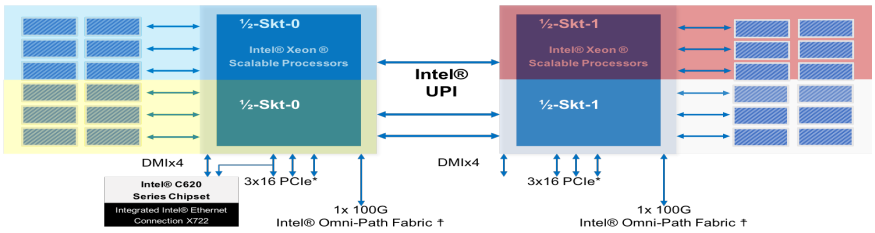


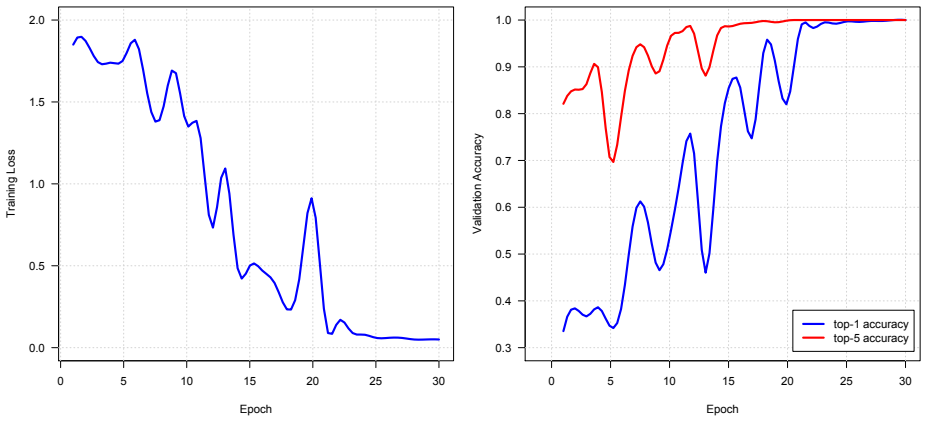
Fig. 6: Two socket Intel® Xeon® Gold 6148 processor NUMA configuration

While the authors mention that instantiating multiple workers boosts performance, they do not specify the optimal number of workers, which can depend on a variety of factors, including the neural network topology being trained, CPU micro-architecture, and characteristics of the input data. To find the best combination of workers and local batch size per worker, we experimented with 1, 2, 4 and 8 workers per CPU. In this case, 4 workers with 8 local mini-batch size resulted in the highest throughput per node. A detailed analysis of throughput and memory utilization for 4 workers is shown in Figure 5 (b) and Figure 5 (d), respectively. Note that throughput with batch sizes of 64, 128, or 256 was

higher than with a batch size of 32, but these configurations did not converge any faster.

5.3 Scaling out TTT on 8 Servers with Dataset A

After determining the number of workers per node, we deployed the training on 8 nodes with 4 workers per node. We used the MPI Allreduce mechanism in Uber's Horovod library to synchronize the gradients. As indicated in Figure 1, the model size is 162MB which was the size of the gradients exchanged between the workers per iteration. Due to this high bandwidth requirement, we used a 100Gbps Intel® Omni-Path Fabric (Intel® OP Fabric). Note here that each layer of M-CNN calls *Horovod_Allreduce*, resulting in a large variation in the MPI negotaition calls. The MPI negotiation times range between 450ms and 858ms. The final time to convergence on 8 nodes is shown in Figure 7. Figure 7(a) shows the training loss over epochs and Figure 7(b) shows the time to achieve state of the art top-1 and top-5 accuracy on Dataset A. From the results, we see that using 8x more hardware resources we were able to scale TTT by 6.6X. With Dataset A, this means a TTT of 31 minutes which is well within our target of one hour. This also encouraged us to explore a larger dataset we would need more hardware resources. Hence, we chose Dataset B with 313,282 images. The experiment results follow in the next section.



(a) Training loss (b) Validation accuracy

Fig. 7: Training loss, top-1 and top-5 accuracy of M-CNN model with Dataset A in 30 epochs on 8x 2S Intel® Xeon® Gold 6148 processors connected with Intel® OP Fabric

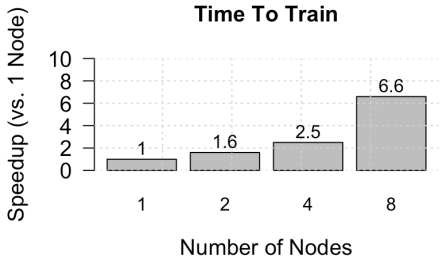


Fig. 8: Scaling M-CNN training with Dataset A from 1X to 8X 2S Intel® Xeon® Gold 6148 processors connected with 100Gbps Intel® OP Fabric

5.4 Scaling out TTT on 128 Servers with Dataset B

Table 1 summarizes the 19.2X performance improvement acheived by scaling from 1 to 128 Intel® Xeon® Gold 6148 processors with Dataset B bringing TTT to 50 minutes. The second column in the table shows number of epochs when training reach 99% top-1 accuracy and 100% top-5 accuracy. Subsequent columns show the global mini-batch size, time to train (in minutes) and effective throughput in images/second for each node configuration. 8 training workers per node were used in these experiments as the image dimensions in Dataset B are smaller than Dataset A.

The key takeaway here is that updates per epoch is critical to acheive convergence. Global mini batch size determines the number of updates per epoch and

Table 1: M-CNN Training Performance on 128 2S Intel® Xeon® Gold processors with Dataset B

# of Nodes	# of Epochs	Batch Size	TTT (mins)	Images/sec
1	6.6	128	960	30
2	8	256	642	72
4	8.7	512	320	141
8	12	1024	240	262
16	15.9	2048	150	553
32	14.9	2048	85	893
64	15	2048	61	1284
128	15.2	2048	50	1587

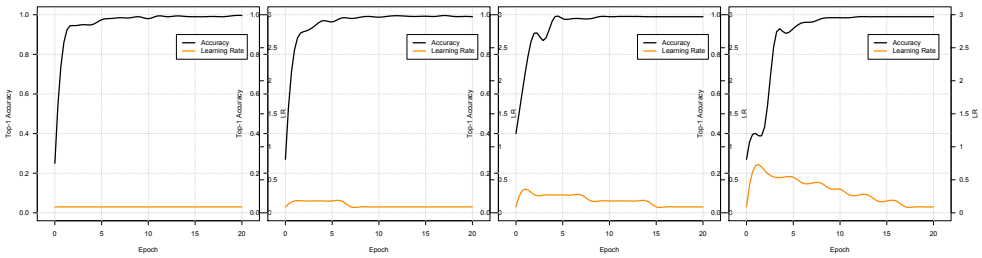
M-CNN did not converge beyond global batch sizes of 2048. Hence, we maintained the global batch size to 2048 while scaling from 16 to 128 nodes – the idea of strong scaling taken from HPC applications. As the same amount of work is increasingly divided across more CPU cores we observe diminishing returns in speedup albeit overall TTT improves. Note that our objective is to not show linear scaling here, but to see what resources will help us acheive a TTT less than one hour.

Anothe key takeaway is that large number of workers required larger number of epochs to converge. This also affects scaling. This is again an artifact of the dataset. Finally, in Figure 9, we show the behavior of top-1 accuracy and learning rate per epoch for each of the configurations. Note here that use the linear learning rate scaling rule discussed in subsection 3.1. The learning rate is scaled according to the ratio of increase in global mini batch size. However, as shown in the Figure 9 similar to global batch size, learning rate scaling has to capped to 2048 beyond 16 nodes for the model to converge.

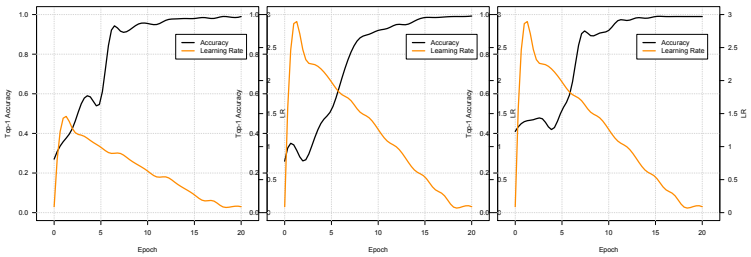
Additionally, we show the scaling efficiency of M-CNN training from 1 to 64 nodes all running for 20 epochs. As shown in Figure 10 time to train efficiently scales up to 16 nodes after which capping the global mini batch size shows diminishing returns.

6 Discussion

In this work, we explored training a multi-scale convolutional neural network to classify large high content screening images within one hour by exploiting large memory in CPU systems. The cellular images used are over million pixels in resolution and are 26 times larger than those in the ImageNet dataset. We used two sets of cellular images with different resolutions to analyze the performance on multiple nodes of M-CNN training. The first set contains 10K full resolution cellular images (1024×1280×3) and the second dataset contains 313K images of smaller dimensions (724×724×3). With the first dataset, we were able to scale time to train linearly using 8X 2S Intel® Xeon® Gold processors. Large mini-batch sizes enabled by the large memory footprint in CPUs helped us achieve the speedup in training time. With the second data set, we were able to achieve TTT of 50 minutes, a 19.2X improvement in time to train using 128 Intel® Xeon® Gold processors. We learned that the updates per epoch is critical to achieve convergence and if the characteristics of the images in the dataset cannot tolerate scaling of updates per epoch beyond a certain threshold (2048 in our case), then adding more computational resources results in diminishing returns. In future work, we intend to explore larger datasets with more variation where images are chosen from different cohorts.



(a) 1 node (b) 2 nodes (c) 4 nodes (d) 8 nodes



(e) 16 nodes (f) 32 nodes (g) 64 nodes

Fig. 9: Top-1 Accuracy achieved in 20 epochs of M-CNN training and Learning Rate used on 1–64 2S Intel® Xeon® Gold processors. Dataset B is used for these experiments. Global minibatch size is capped at 2K from 16 to 64 nodes. The learning rate as shown in (f) – (h) is also scaled only to 0.032 to achieve convergence

Conflicts of interest

Intel® Xeon® Gold 6148 processor, Intel® OPA and Intel® SSD storage drive are registered products of Intel Corporation. The authors declare no other conflicts of interest.

References

1. M. R. Arbabshirani, B. K. Fornwalt, G. J. Mongelluzzo, J. D. Suever, B. D. Geise, A. A. Patel, and G. J. Moore, "Advanced machine learning in action: identification of intracranial hemorrhage on computed tomography scans of the head with clinical workflow integration," *npj Digital Medicine*, vol. 1, 2018.
2. Z. Akkus, A. Galimzianova, A. Hoogi, D. L. Rubin, and B. J. Erickson, "Deep learning for brain mri segmentation: State of the art and future directions," *Journal of digital imaging*, vol. 30, pp. 449–459, 2017.
3. D. C. Cireşan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "Mitosis detection in breast cancer histology images with deep neural networks," in *Medical*

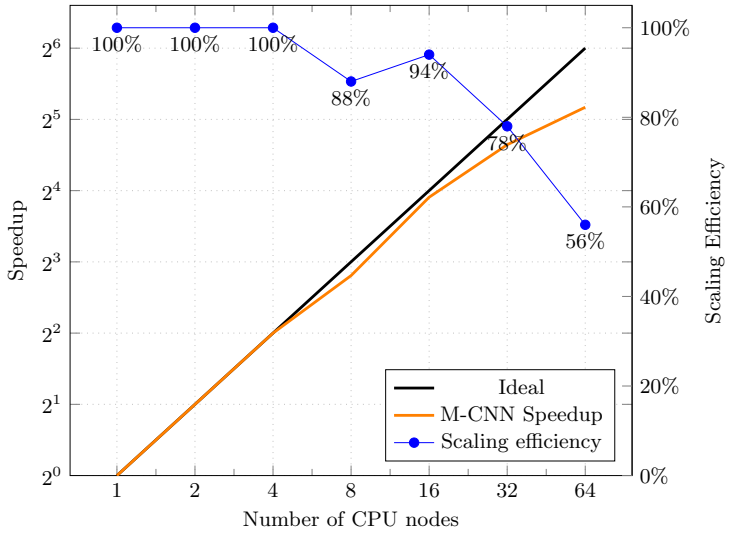


Fig. 10: Scalability of M-CNN training performance for 20 epochs on 64 2S Intel® Xeon® Gold 6148 processors. Note that global batch size is capped at 2K from 16 – 64 nodes. Intel® OP Fabric, TensorFlow-1.9.0+Horovod, OpenMPI v3.0.0, 8 workers/node

Image Computing and Computer-Assisted Intervention – MICCAI 2013, K. Mori, I. Sakuma, Y. Sato, C. Barillot, and N. Navab, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 411–418.

4. G. Litjens, C. I. Sánchez, N. Timofeeva, M. Hermsen, I. Nagtegaal, I. Kovacs, C. Hulsbergen van de Kaa, P. Bult, B. van Ginneken, and J. van der Laak, "Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis," *Scientific Reports*, vol. 6, 2016.
5. A. Janowczyk and A. Madabhushi, "Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases," *Journal of pathology informatics*, vol. 7, 2016.
6. O. Z. Kraus, B. T. Grys, J. Ba, Y. Chong, B. J. Frey, C. Boone, and B. J. Andrews, "Automated analysis of high-content microscopy data with deep learning," *Molecular Systems Biology*, vol. 13, no. 4, 2017.
7. C. Sommer, R. Hoefler, M. Samwer, D. W. Gerlich, and C. Boone, "A deep learning and novelty detection framework for rapid phenotyping in high-content screening," *Molecular Biology of the Cell*, vol. 28, no. 23, pp. 3428–3436, 2017.
8. D. C. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images." in *NIPS*, P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 2852–2860.
9. G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken, and C. I. Snchez, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, 2017.

10. M. M. Usaj, E. B. Styles, A. J. Verster, H. Friesen, C. Boone, and B. J. Andrews, "High-content screening for quantitative cell biology," *Trends in Cell Biology*, vol. 26, no. 8, pp. 598 – 611, 2016.
11. M. Boutros, F. Heigwer, and C. Laufer, "Microscopy-based high-content screening," *Cell*, vol. 163, no. 6, pp. 1314 – 1325, 2015.
12. S. Singh, A. E. Carpenter, and A. Genovesio, "Increasing the content of high-content screening: An overview," *Journal of Biomolecular Screening*, vol. 19, pp. 640–650, 2014.
13. C. Scheeder, F. Heigwer, and M. Boutros, "Machine learning and image-based profiling in drug discovery," *Current Opinion in Systems Biology*, vol. 10, pp. 43 – 52, 2018, pharmacology and drug discovery.
14. J. M. Zock, "Applications of high content screening in life science research," *Combinatorial chemistry & high throughput screening*, vol. 12, no. 9, pp. 870–876, 2009.
15. W. Buchser, M. Collins, T. Garyantes, R. Guha, S. Haney, V. Lemmon, Z. Li, and O. J. Trask, *Assay development guidelines for image-based high content screening, high content analysis and high content imaging.* Eli Lilly & Company and the National Center for Advancing Translational Sciences, 2014.
16. W. J. Godinez, I. Hossain, S. E. Lazic, J. W. Davies, and X. Zhang, "A multi-scale convolutional neural network for phenotyping high-content cellular images," *Bioinformatics*, vol. 33, no. 13, pp. 2010–2019, 2017.
17. W. J. Godinez, I. Hossain, and X. Zhang, "Unsupervised phenotypic analysis of cellular images with multi-scale convolutional neural networks," *bioRxiv*, 2018. [Online]. Available: https://www.biorxiv.org/content/early/2018/07/03/361410
18. D. M. Ando, C. McLean, and M. Berndl, "Improving phenotypic measurements in high-content imaging screens," *bioRxiv*, 2017. [Online]. Available: https://www.biorxiv.org/content/early/2017/07/10/161422
19. P. Buyssens, A. Elmoataz, and O. Lézoray, "Multiscale convolutional neural networks for vision–based classification of cells," in *Asian Conference on Computer Vision*. Springer, 2012, pp. 342–352.
20. N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 2017, pp. 1–12.
21. H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.
22. Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "Imagenet training in minutes," in *Proceedings of the 47th International Conference on Parallel Processing*. ACM, 2018, p. 1.
23. V. Ljosa, K. L. Sokolnicki, and A. E. Carpenter, "Annotated high-throughput microscopy image sets for validation," *Nat Methods*, vol. 9, no. 7, p. 637, 2012.
24. P. D. Caie, R. E. Walls, A. Ingleston-Orme, S. Daya, T. Houslay, R. Eagle, M. E. Roberts, and N. O. Carragher, "High-content phenotypic profiling of drug response signatures across distinct cancer cells," *Molecular cancer therapeutics*, pp. 1535–7163, 2010.
25. Google. Tpu benchmarks. [Online]. Available: https://github.com/tensorflow/tpu.git.
26. A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.
27. V. Saletore, D. Karkada, V. Sripathi, A. Sankaranarayanan, and K. Datta. Boosting deep learning training and inference performance on intel xeon and intel

xeon phi processors. [Online]. Available: https://software.intel.com/en-us/articles/boosting-deep-learning-training-inference-performance-on-xeon-and-xeon-phi