2018 IEEE/ACM Machine Learning in HPC Environments (MLHPC)

Ramifications of Evolving Misbehaving Convolutional Neural Network Kernel and Batch Sizes

Mark Coletti*, Dalton Lunga*, Anne Berres*, Jibonananda Sanyal*, Amy Rose*
*Oak Ridge National Laboratory
Geographic Information Science and Technology Group
Oak Ridge, TN USA
Email: {colettima,lungadd,berresas,sanyalj,rosean}@ornl.gov

***Abstract*—Deep-learners have many hyper-parameters including learning rate, batch size, kernel size — all playing a significant role toward estimating high quality models. Discovering useful hyper-parameter guidelines is an active area of research, though the state of the art generally uses a brute force, uniform grid approach or random search for finding ideal settings. We share the preliminary results of using an alternative approach to deep learner hyper-parameter tuning that uses an evolutionary algorithm to improve the accuracy of a deep-learner models used in satellite imagery building footprint detection. We found that the kernel and batch size hyper-parameters surprisingly differed from sizes arrived at via a brute force uniform grid approach. These differences suggest a novel role for evolutionary algorithms in determining the number of convolution layers, as well as smaller batch sizes in improving deep-learner models.**

***Index Terms*—deep learning, convolutional neural networks, evolutionary algorithms, hyper-parameters, optimization, settlement detection, satellite imagery**

I. Introduction

Fig. 1. **Building footprint detection.** This depicts detected building footprints, highlighted in purple, by applying a CNN model to satellite imagery.

Deep learners (DLs) have been applied on problems in computer vision, speech recognition, and text analysis [13]. At Oak Ridge National Laboratory (ORNL)'s Geographic Information Science and Technology (GIST) group, scientists have applied DLs toward large scale building detection and mapping, as depicted in Fig. 1 [14, 27].

All DLs have hyper-parameters whose settings will influence model quality. These include specifying kernel sizes for convolution layers, the number of training epochs, batch sizes, and dropout rate. Strategies for finding ideal hyper-parameters has become an active research topic, but generally hyper-parameters are selected using some form of manual trial and error system, a brute-force uniform grid search, or random search [3]. However, one alternative to these hyper-parameter tuning strategies is to use an evolutionary algorithm (EA) to evolve optimal DL hyper-parameters [28].

We share here the results of a prototype system that uses an evolutionary algorithm to tune two hyper-parameters for a satellite image building footprint DL. The same DL was previously tuned using a brute-force, uniform grid search of several hyper-parameters, which took two weeks on a dedicated cluster node with 8-NVIDIA K80 GPUs [14], and which served as a baseline for comparison with the current work. By contrast, the EA approach took 12 hours on ORNL's Titan supercomputer. We found that the evolved kernels sizes suggest that EAs could be used as a diagnostic for determining the ideal number of convolution layers in a CNN. We also found that batch sizes were inversely correlated to model validation accuracy, which perhaps indicates that smaller batch sizes may yield improved validation accuracy possibly as a novel and unique source of noise [30].

II. Related Work

DL has provided significant advances in the areas of computer vision, text processing, and language translation, among others [13]. Within a remote sensing context, DLs have been applied to classify satellite image regions [18]. DLs have also been successfully employed to identify buildings from satellite imagery [29, 14].

Generally, the two most common approaches for tuning DL hyper-parameters are using a brute-force uniform-grid

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

978-1-7281-0180-4/18/$31.00 ©2018 IEEE
DOI 10.1109/MLHPC.2018.00014

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 17:07:03 UTC from IEEE Xplore. Restrictions apply.

approach to exhaustively explore combinations of different hyper-parameter values, or randomly sample different hyper-parameter values [3]. The uniform grid approach is time consuming beyond a certain number of tuned hyper-parameters, the sample intervals for real values may skip over interesting regions of the search space, gives equal weight to hyper-parameters that make little contribution to model fidelity, and adding more hyper-parameters can exponentially increase search times. By contrast, random search does not give equal weight to non-contributing hyper-parameters, and are free to encounter useful solutions that would have been skipped over by a comparative uniform search due to regular sample value spacing. Regardless, the random approach is not satisfactory in that practitioners have to rely on luck to find viable hyper-parameters; this kind of search also does not leverage learned facets of the search space to focus on interesting areas.

An alternative to uniform grid and random search is to use Bayesian optimization to tune DL hyper-parameters [23, 21]. There is also some general guidance for tuning DL hyper-parameters in [2].

EAs solve problems via a biologically inspired approach whereby posed solutions are represented as individuals in artificial ecosystems, and where fitness is represented by solution quality [7]. EAs have been used as an alternative to gradient descent approaches for learning network weights [16], and have also been used to learn weights and simple network topologies [24]. In addition to tuning DL hyper-parameters, MENNDL used an EA to optimize the architecture of CNN layers, including determining the number, kind, and sequence of layers up to an arbitrary fixed upper-bound [28] using a VGG-style DL [22]. Another approach that also used a fixed-length representation allows for arbitrary connections between layers that support not only VGG-type DLs, but also ResNet [10] and DenseNet [11] DLs.

III. Methodology

The research objective was to determine the efficacy of tuning a DL's hyper-parameters using an EA. However, this was a prototype system in that only two hyper-parameters were selected for tuning, the kernel and batch sizes, with the intent that if this system proved viable that the next implementation would consider additional hyper-parameters that includes *the learning rate, number of layers, number of neurons per layer, gradient solver, and activation functions*. A prior DL architecture, including hyper-parameters tuned via a brute force uniform grid approach, provided a baseline for comparison [14]. The expectation was that our approach should provide similar hyper-parameters as found with the brute force method, but in a more timely manner.

To that end, here we detail the software used to implement the experiment, the DL architecture, the EA implementation, as well as the data used for training and validation.

A. Base Deep-learner Architecture

We use a baseline CNN architecture implemented in our previous deep learning settlement mapping task, and which is depicted in Fig. 2 [14]. The four 2D convolution layers are highlighted to emphasize that one of the tuned hyper-parameters, the convolution kernel size, is associated with those layers. We use stochastic gradient descent (SGD) to train the CNN model with the optimizer's parameters set according to Table I.

TABLE I
DL optimizer parameters

optimizer	stochastic gradient descent
momentum	0.9
Nesterov momentum	true
learning rate	0.00273
decay rate	0

The MaxPooling2D layers all used a 2×2 pool size. The 2D convolution and dense layers all used Rectified Linear Unit (RLU) activation functions and weights are initialized from a normal distribution. The dropout layers used rates of 0.5. The last three dense layers had sizes of 512, 256, and 2, respectively, with the last layer using a "softmax" activation function for the classification output.

B. EA implementation

Algorithm 1 EA for optimizing DL hyper-parameters.

1: $P_0 \leftarrow$ GENERATE(30) ▷ random initial population
2: EVAL(P_0) ▷ calculate fitnesses
3: $i \leftarrow 0$
4: $maxgen \leftarrow 4$
5: **while** $i < maxgen$ **do**
6: $C_i \leftarrow$ BIRTH(P_i) ▷ create children
7: EVAL(C_i) ▷ evaluate children
8: $P_{i+1} \leftarrow$ SURVIVE($P_i \cup C_i, 30$)
9: $i \leftarrow i + 1$
10: **end while**

Algorithm 1 shows at a high-level the EA used to optimize the DL kernel and batch size hyper-parameters. First, an initial population of 30 individuals was generated by creating random bit vectors for each of them. Then the individuals were concurrently evaluated on separate Titan nodes, which entailed returning the validation accuracy for the corresponding DL built and trained with the kernel and batch sizes described in the individual. Then for each generation created a set of children from the current set of parents; then evaluated them concurrently on separate Titan nodes as was done with the initial parents; and then determined the next set of parents from the 30 best of the current parents and children.

Algorithm 2 details the BIRTH operator invoked in line 6 of Algorithm 1. This operator gets the current set of parents, P_i, and returns a set of children created from those parents, C, with the number of children equal to the number of dedicated Titan worker nodes, or 382. We chose 382 nodes because the minimum number of nodes needed for a single job to get a maximum of 12 hours per run as dictated by ORNL Leadership Computing Facility (OLCF) policy [1] was 383.

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 17:07:03 UTC from IEEE Xplore. Restrictions apply.

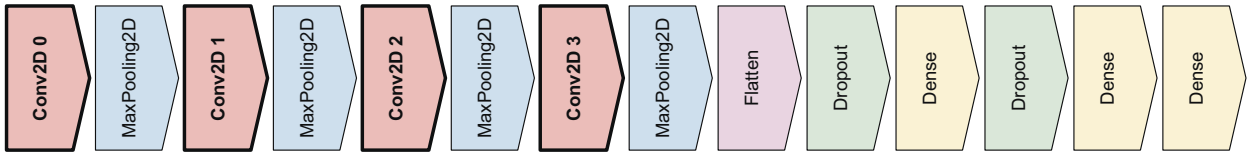

Fig. 2. **CNN architecture used.** This depicts the CNN architecture that was used for building detection from satellite imagery. The four 2D convolution layers for which the corresponding kernel sizes were evolved are highlighted.

Algorithm 2 The BIRTH operator.

1: **function** BIRTH(P_i)
2: $C \leftarrow \{\}$
3: $n \leftarrow 0$
4: $maxchildren \leftarrow 382$ ▷ 382 Titan worker nodes
5: **while** $n < maxchildren$ **do**
6: $P_a, P_b \leftarrow$ SELECT(P_i)
7: $C_a \leftarrow$ CLONE(P_a), $C_b \leftarrow$ CLONE(P_b)
8: MUTATE(C_a), MUTATE(C_b)
9: $C_t, C_u \leftarrow$ CROSSOVER(C_a, C_b)
10: $C \leftarrow C \cup C_t \cup C_u$ ▷ append children
11: $n \leftarrow n + 2$ ▷ created 2 children
12: **end while**
13: **return** C
14: **end function**

(I.e., 382 worker nodes plus one controller node.) 12 hours was deemed enough time for five generations, which was enough to get a sense of the efficacy of this approach.

Lines 5 – 12 show the details of creating offspring. First, binary tournament selection is used to pick two parents from P_i; that is, binary tournament selection works by selecting two individuals from P_i with replacement, and the best of the two is returned [7]. This process is done twice to select two parents, P_a and P_b. These two individuals are cloned into what will become the new children, C_a and C_b. Then bit-flip mutation is applied with a chance of 0.1 per bit. Then uniform crossover [8] is applied to create the final children, C_u and C_t, that are then appended to C. After 382 children are created, C is returned.

Fig. 3 shows how the individuals in the population represent the DL kernel and batch sizes to be optimized as a sequence of 19 bits. The first set of bits correspond to the kernel sizes, with each kernel size taking three bits, thus supporting eight possible values mapped to $\{3, 5, 7, 9, 11, 13, 15, 17\}$. The remaining seven bits represented the batch size. Since sensitivity tests indicated that the Nvidia K20 graphics processing units (GPUs) on Titan could only support a maximum batch size of 150, and the seven bits had 127 unique values, the valid range of batch sizes to be explored was in $[22, 150]$.

Mutation for such a binary representation is implemented as simple bit-flipping. However, it has been long known in the EA community that bit flipping the most significant binary digit of

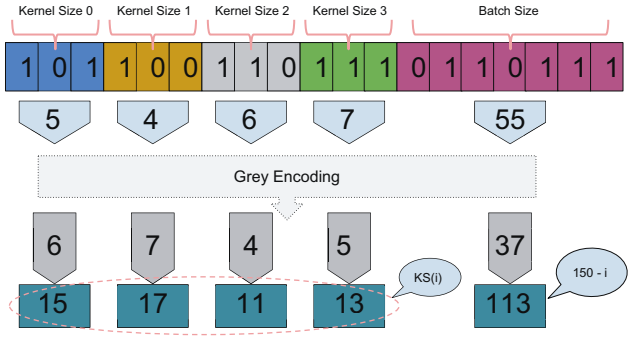

Fig. 3. **Problem representation.** Each individual in the population represents a unique DL configuration using 19 bits. The kernel sizes for the four 2D convolution layers are represented by four sets of three bits, respectively; the last seven bits correspond to the batch size. The integer values for the kernel and batch sizes are remapped via Grey Encoding to mitigate the disruptive effects of bit-flip mutation. The kernel sizes after Grey encoding are indices to look up the actual kernel size $\in \{3, 5, 7, 9, 11, 13, 15, 17\}$ – e.g., a kernel size index value of four corresponds to a kernel size of eleven. Lastly, since the batch sizes must map to [22,150], the Grey encoded value is subtracted from 150. These kernel and batch sizes are then passed to Keras to build a corresponding DL that is then trained to be evaluated for its validation accuracy.

a numeric value could be very disruptive – the notion is that evolutionary processes should generally be gently incremental, which is at odds to such drastic mutation effects. The typical solution, which we implemented, is to use Gray codes to remap numeric binary sequences such that a single bit flip only nudges the value by a small amount regardless of where the mutation occurs [26, 4]. The resulting integer values after Gray encoding then yields a final kernel size from $(n + 1) \times 2 + 1$, where n is the Gray encoded integer. (I.e., again, yielding kernel sizes in $\{3, 5, 7, 9, 11, 13, 15, 17\}$.) The Gray encoded value for the batch size is subtracted from 150 to yield values in $[22, 150]$.

For every generation each Titan worker node receives these kernel and batch sizes that are then used to configure our DL model.The resulting DL is trained and the validation accuracy is returned as the corresponding individual's fitness. However, we do observe pathological configuration of kernel sizes that do not fit the commonly human picked sizes, such as the kernel sizes $\{17, 17, 17, 17\}$, for which our model estimation process will throw an exception. We flag these exceptions with a -1 to account for the corresponding fitness; such individuals will

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 17:07:03 UTC from IEEE Xplore. Restrictions apply.

quickly fall out of the population due to selection given that this is a maximization problem

C. Dependent Software

TABLE II
DEPENDENT SOFTWARE

software	version
python	3.5.6
Keras	2.0.8
TensorFlow	1.3.1
CUDA	7.5
singularity	2.5.2
inspyred	1.0.1
schwimmbad	0.3.0

Table II shows the software used to implement our experiments, but does not include an exhaustive list of the dependencies of dependencies for the sake of brevity. We used Keras [5] with the TensorFlow [15] backend for our DL implementation, which, in turn, used CUDA [17] to support GPU operations. We also relied on Singularity containers [12] supplied by the OLCF to provide support for TensorFlow on Titan [25]. inspyred provided a python-based evolutionary computation (EC) framework [9] in which we incorporated schwimmbad [19] to support a controller/worker configuration for submitting DLs to Titan nodes for evaluation with each generation.

D. Experiments

Four panchromatic (single band) image tiles, of 0.5-meter resolution, that cover extremely varying landscapes and settlement structures from Daykundi, Afghanistan are selected. A *ground-truth* collection of $20,000$ image patches, each of size 144×144 pixels, is manually cropped from the tiles. These are randomly sampled however are representative enough for training a convolutional network based settlement detection model. Model evaluation is conducted using a validation set of $2,300$ image patches.

IV. RESULTS

That 19 bits were used to represent these DLs meant that there were 2^{19} possible combinations, or 524,288, possible DL configurations. Presuming an average training time of one hour for a single GPU, and an Nvidia DGX-1 with eight such GPUs, would take nearly eight years to explore every such combination. However, for the five runs performing on the ORNL Titan supercomputer evaluated just 7,830 DLs. Again, presuming an average training time of one hour per DL, that number of models would have taken approximately 40 days on a dedicated DGX-1 with eight GPUs. However, the original brute-force uniform grid hyper-parameter sweep that we used as a baseline comparison took two weeks on a cluster node with 8-NVIDIA K80 GPUs — by comparison, all five runs done for this work occurred within a 12 hour period.

We will next share the validation accuracy, kernel, and batch size results for these five 12-hour runs.

A. Validation Accuracies

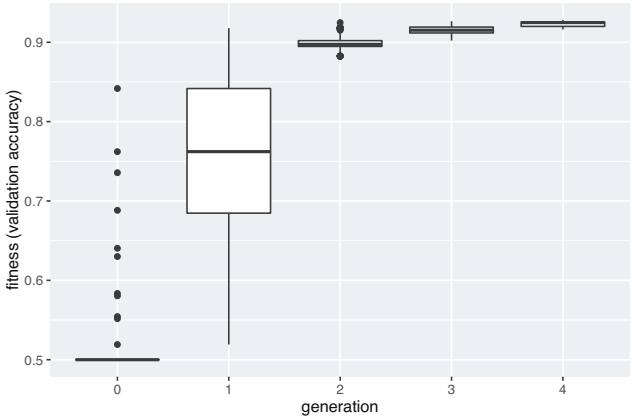

Fig. 4. **DL fitnesses.** This shows the aggregate fitnesses for all five runs. The intial population is random, and most of the fitnesses are around 50%. However, very quickly the validation accuracies of the five runs asymptotically approach 95%.

Fig. 4 shows the collective fitnesses for all five runs, which is the final validation accuracies for 30 DL by five runs, or 150 DLs, per generation. The first generation is comprised of entirely random kernel and batch sizes with most of the corresponding validation accuracies around 50%. The next generation is comprised of the best offspring and parents, or DLs, from the previous generation, and shows a jump in validation accuracies, with a median validaiton accuracy of 76.22%. Over the next three generations, the five runs converged on DL validation accuracies to a median value of 92.48%.

These fitness trajectories were a diagnostic that the prototype system was behaving as expected from the perspective of creating viable DL models. That is, the "best-so-far" curve of the evolved DL model validation accuracy quickly converged on the validation accuracy of 95.83% that was achieved with similar DL models derived from the prior uniform grid brute force approach [6]; if the system was not performing correctly, then the runs would not have improved from generation to generation, or achieved a validation accuracy close to that of the brute force approach. Moreover, we are confident that the runs would have converged on similar validation accuracy if allowed to run for a few more generations based on the observed best-so-far trajectories of the five runs.

B. Kernel Sizes

The brute force uniform grid approach determined that a kernel size of three for all four 2D convolutional layers was optimal [6], and the expectation was that the EA would converge on similar results. Fig. 5 shows the kernel sizes for all five runs by generation for the EA. For the first three convolutional 2D layers, the EA readily converged on a kernel size of three by the third generation. However, the EA significantly differed from the brute force approach for the fourth and final 2D convolutional layer by not converging on

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 17:07:03 UTC from IEEE Xplore. Restrictions apply.

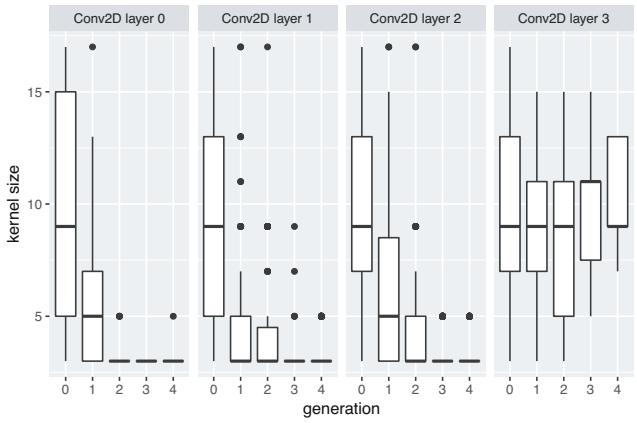

Fig. 5. **Conv2D layer kernel sizes.** This shows the aggregate kernel sizes for each of the four 2D convolutional layers for 30 DLs for all five runs, or 150 DLs per generation. For the first three 2D convolution layers all five runs converged to a kernel size of three by the third generation, which matches what the uniform grid brute force approach discovered. However, for the fourth 2d convolutional layer, none of the five runs converged on a single kernel size; by contrast, the uniform grid brute force approach determined that a kernel size of three was ideal.

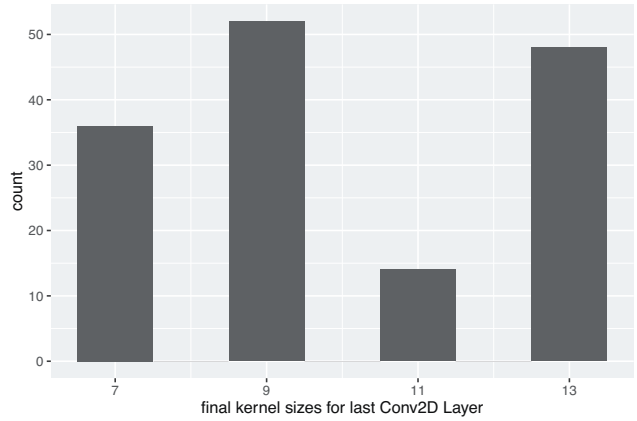

Fig. 6. **Fourth Conv2D layer final kernel sizes.** This is a histogram of the final, aggregate kernel sizes for all five runs, and shows that the kernel sizes had converged to one of four values — 7, 9, 11, or 13; which, again, significantly differed from the kernel size of three determined to be ideal by the uniform grid brute force approach.

a kernel size of three; in fact, the EA did not converge on any single value.

Fig. 6 shows a histogram of the final fourth 2D convolutional layer kernel sizes for all five runs, or the final, and best 150 DLs. There we observe that the final kernel sizes fall within one of four bins in $\{7, 9, 11, 13\}$. In descending order of count, there were 52 kernel sizes of size 9, 48 of 13, 36 of 7, and 14 of 11 — none of which were anywhere near the kernel size of three arrived at by the baseline brute force approach.

We pose one possible explanation for the differences in kernel sizes between the EA and the brute force approach. That is, these results serve as an architectural diagnostic signaling that the fourth 2D convolutional layer is superfluous, at least within the context of the training and validation data used. There is also the distant possibility that the EA is leveraging some larger scale structures relevant for that layer, and that after several more generations it may converge on one of the four kernel sizes found in the final generation. However, given the rarity of having such large kernel sizes on convolutional layers, and the significant distance of these kernel size values from the brute force size of three, it is likelier that these different sizes are the result of genetic drift, or arbitrary convergence via sampling. That is, if the kernel size values for the fourth 2D convolutional layer have no or little bearing on the corresponding fitness (validation accuracy), then from generation to generation the size of the set of kernel sizes will monotonically decrease [20].

C. Batch Sizes

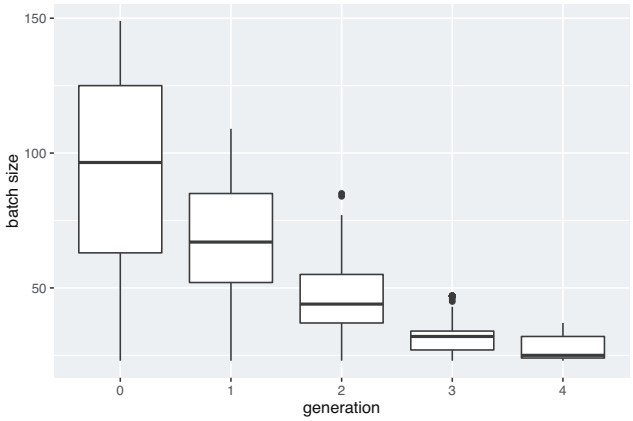

Fig. 7. **Batch sizes.** This batch sizes for all five runs gradually converged to a median value of 25, the convergence trajectory of which has an inverse correlation to the validation accuracies, as shown in Fig. 4.

Fig. 7 shows the batch sizes for all five runs. The first generation, which is comprised of random batch sizes, shows the expected uniform spread of batch sizes in [22,150]. However, the batch sizes for all the runs gradually converged on a median value of 25. This was surprising because the expectation is that the batch size should have no impact on the corresponding validation accuracy. That is, the DL is going to "see" all the training and validation data with each training epoch, so the amount of training and validation it sees at once should not color its perspective of the problem space. Yet the batch size trajectories are inversely correlated with the validation accuracy trajectories as noted in Fig. 4.

One possible explanation for this phenomenon is that smaller batch sizes introduce a novel form of noise to the training data that gives a small, but notable, boost in validation accuracy. That is, adding noise to different aspects of DL training can improve model fidelity [30]. The smaller batch sizes may provide a rougher perspective of the training and validation

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 17:07:03 UTC from IEEE Xplore. Restrictions apply.

data almost as if noise had been added — think of it as a kind of drop out layer that is added in such that each batch is not so much giving data from which to be training, but giving data with the rest of the data occluded.

V. Conclusions and Future Work

Here we share our experimental results, which indicate that the evolved kernel sizes are a potential diagnostic for DL architectural configuraitons, and that smaller batch sizes may be a novel source of helpful noise. We also share follow-on plans for future work.

A. Kernel Sizes as Possible Diagnostic

The EA converged on kernel sizes for the fourth 2D convolutional layer that significantly differed from that which the corresponding brute force approach discovered. That is, the latter determined that a kernel size of three was ideal, whereas the EA converged on four possible kernel sizes — 7, 9, 11, or 13. This is possibly a diagnostic that the fourth 2D convolutional layer is superfluous within the context of the training and validation data that was used in the experiments. That is, if the fourth layer is not needed, then there will be no or little contribution to the individual's fitness for the fourth layer's kernel size, and so any observed convergence will be due to genetic drift.

There are two ways to test this hypothesis. First, the experiments can be re-run after removing the fourth 2D convolutional layer; if there are no significant differences between the quality of these newly evolved models and the original models, then this would support the notion that that layer was not needed. Second, the training and validation data for this work was for a single Afghanistan province. The experiments can be re-run keeping the fourth layer, but for the entire country instead of just a single province, with the intuition that a larger training and validation dataset with more variance may need the leverage of that fourth layer. If this is true, then we should observe the kernel sizes converge to three, which, again, was the same as what was noted in the baseline brute force approach.

B. Smaller Batch Sizes as Possible Novel Source of Jitter

First, see if this phenomenon can be reproduced with other training and validation data. If this is a general observation, then run another experiment with one set of runs with the maximum batch size supported by the GPU and another with a batch size of 25, as converged to by the EA. If there is a significant difference between the two runs, then we know this phenomenon is real.

Then, to determine if this behaves as a unique source of noise, intentionally add noise or manipulate the existing dropout layers to see if that has an impact on the evolved batch sizes. If the batch sizes no longer converge, then we can be satisfied that this is a viable explanation.

C. Improvements and Expanding Scope

The results shown here were from a prototype system intended to show the viability of this approach. Given that the system appears to evolve ideal hyper-parameter values, we plan to include the other hyper-parameter values used with the original brute force search to do a comparison between the two.

The by-generation implementation of the EA did not make the best use of the available GPU resources on Titan. That is, parallel evaluation of DLs were done in lock-step by generation; this meant that GPU's that finished training their respective DL would be idle until the last DL of that generation was done. Considering that pathological configurations, such as sequences of kernel sizes that made no sense, would terminate evaluation immediately, that meant that GPU could sit idle for hours until the last DL completed training. Moreover, not all DL take the same amount of time to train, so once again there would be idle GPUs for those DLs that finished their training early.

A better approach that we intend on implementing, and one used by MENNDL [28], is to use an asynchronous steady-state EA. That is, a pool of individuals is continually updated with new DLs as they complete training. As soon as a node has completed training a DL, the newly evaluated DL replaces an inferior individual in the population, and a DL awaiting evaluation is then assigned to that GPU for evaluation. This configuration ensures that no GPUs remain idle for long during the entire run.

VI. Acknowledgements

Thanks to Jeanette Weaver of ORNL for supplying the building detection image used for Fig. 1. Thanks to Dalton Lunga, also of ORNL, for supplying the training and validation data as well as the original architecture with attendant source code. We would also like to thank the ASCR Leadership Computing Challenge (ALCC) for the 25 million core hours on Titan that made this research possible. This research was funded by the Federal Emergency Management Association (FEMA), the Bill and Melinda Gates Foundation, and via undisclosed federal funding sources.

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 17:07:03 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] Aug. 2018. URL: https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs/#job-priority-by-processor-count.

[2] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[3] James Bergstra and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research* 13 (Feb Feb. 2012), pp. 281–305. ISSN: ISSN 1533-7928. URL: http://www.jmlr.org/papers/v13/bergstra12a.html (visited on 03/13/2018).

[4] Richard A. Caruana and J. David Schaffer. "Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms". In: *Machine Learning Proceedings 1988*. Ed. by John Laird. San Francisco (CA): Morgan Kaufmann, 1988, pp. 153–161. ISBN: 978-0-934613-64-4. DOI: 10.1016/B978-0-934613-64-4.50021-9. URL: https://www.sciencedirect.com/science/article/pii/B9780934613644500219.

[5] François Chollet et al. *Keras*. https://keras.io. 2015.

[6] D. Lunga and L. Yang. "On Training 14,000 ConvNets for Settlement Mapping". Urban Dynamics Institute Brown Bag Lecture Series. Oak Ridge National Laboratory, Dec. 21, 2016.

[7] Kenneth De Jong. *Evolutionary Computation: A Unified Approach*. The MIT Press, 55 Hayward St., Cambridge, MA 02142: The MIT Press, 2006.

[8] Kenneth A. De Jong and William M. Spears. "A formal analysis of the role of multi-point crossover in genetic algorithms". In: *Annals of Mathematics and Artificial Intelligence* 5.1 (1992), pp. 1–26. ISSN: 1573-7470. DOI: 10.1007/BF01530777. URL: http://dx.doi.org/10.1007/BF01530777.

[9] A. Garrett. *inspyred: Bio-inspired Algorithms in Python*. June 2017. URL: http://aarongarrett.github.io/inspyred/.

[10] K. He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[11] Gao Huang et al. "Densely Connected Convolutional Networks." In: *CVPR*. Vol. 1. 2. 2017, p. 3.

[12] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. "Singularity: Scientific Containers for Mobility of Compute". In: *PLOS ONE* 12.5 (May 11, 2017), e0177459. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0177459. URL: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0177459 (visited on 08/24/2018).

[13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521.7553 (May 27, 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: https://www.nature.com/articles/nature14539 (visited on 02/22/2018).

[14] D. Lunga et al. "Domain-Adapted Convolutional Networks for Satellite Image Classification: A Large-Scale Interactive Learning Workflow". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 11.3 (Mar. 2018), pp. 962–977. ISSN: 1939-1404. DOI: 10.1109/JSTARS.2018.2795753.

[15] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[16] David J Montana and Lawrence Davis. "Training Feedforward Neural Networks Using Genetic Algorithms." In: *IJCAI*. Vol. 89. 1989, pp. 762–767.

[17] John Nickolls et al. "Scalable Parallel Programming with CUDA". In: *Queue* 6.2 (Mar. 2008), pp. 40–53. ISSN: 1542-7730. DOI: 10.1145/1365490.1365500. URL: http://doi.acm.org/10.1145/1365490.1365500.

[18] S. Paisitkriangkrai et al. "Semantic Labeling of Aerial and Satellite Imagery". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 9.7 (July 2016), pp. 2868–2881. ISSN: 1939-1404. DOI: 10.1109/JSTARS.2016.2582921.

[19] Adrian M. Price-Whelan and Daniel Foreman-Mackey. "schwimmbad: A uniform interface to parallel processing pools in Python". In: *The Journal of Open Source Software* 2.17 (Sept. 2017). DOI: 10.21105/joss.00357. URL: https://doi.org/10.21105/joss.00357.

[20] J. Sarma and K. De Jong. "Handbook of evolutionary computation". In: C2.7. CRC Press, 1997. Chap. Generation gap methods.

[21] B. Shahriari et al. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104.1 (Jan. 2016), pp. 148–175. ISSN: 0018-9219. DOI: 10.1109/JPROC.2015.2494218.

[22] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[23] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[24] Kenneth O. Stanley and Risto Miikkulainen. "Evolving Neural Networks through Augmenting Topologies". In: *Evolutionary Computation* 10.2 (June 1, 2002), pp. 99–127. ISSN: 1063-6560. DOI: 10.1162/106365602320169811. URL: http://dx.doi.org/10.1162/106365602320169811 (visited on 01/31/2017).

[25] *Titan*. URL: https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/ (visited on 08/24/2018).

[26] Wikipedia. *Gray code*. In: *Wikipedia*. Page Version ID: 826967133. Feb. 22, 2018. URL: https://en.wikipedia.org/w/index.php?title=Gray_code&oldid=826967133.

[27] Hsiuhan Lexie Yang et al. "Building Extraction at Scale using Convolutional Neural Network: Mapping of the

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 17:07:03 UTC from IEEE Xplore. Restrictions apply.

United States". In: *CoRR* abs/1805.08946 (2018). arXiv: 1805.08946. URL: http://arxiv.org/abs/1805.08946.

[28] Steven R Young et al. "Evolving Deep Networks Using HPC". In: *Proceedings of the Machine Learning on HPC Environments*. ACM. 2017, p. 7.

[29] Jiangye Yuan. "Automatic Building Extraction in Aerial Scenes Using Convolutional Networks". In: (Feb. 21, 2016). arXiv: 1602.06564 [cs]. URL: http://arxiv.org/abs/1602.06564.

[30] Le Zhang and P. N. Suganthan. "A Survey of Randomized Algorithms for Training Neural Networks". In: *Information Sciences* 364-365 (Oct. 10, 2016), pp. 146–155. ISSN: 0020-0255. DOI: 10.1016/j.ins.2016.01.039. URL: http://www.sciencedirect.com/science/article/pii/S002002551600058X (visited on 05/29/2018).

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 17:07:03 UTC from IEEE Xplore. Restrictions apply.