Generative Adversarial Networks for Fast Simulation: distributed training and generalisation

Federico Carminati[*]
CERN
E-mail: Federico.Carminati@cern.ch

Gulrukh Khattak
CERN
E-mail: Gul.Rukh.Khattak@cern.ch

Sofia Vallecorsa
CERN
E-mail: Sofia.Vallecorsa@cern.ch

Valeriu Codreanu, Damian Podareanu, Maxwell Cai

SURFsara

Vikram Saletore, Hans Pabst

Intel

In High Energy Physics, the Monte Carlo simulation of the detector response to the particles traversing it is one of the most computing intensive tasks. For some applications, a certain level of approximation is acceptable and it is therefore possible to implement fast simulation models that have the advantage of being less computationally intensive, in comparison to standard detailed Monte Carlo techniques. As a consequence, research on fast simulation solutions, including deep Generative Models, is very active. 3DGAN is a Generative Adversarial Network prototype, based on 3 dimensional convolutions, developed to simulate the response of high-granularity calorimeters. We have previously reported on its physics performance and the results we obtained by introducing a data parallel approach to accelerate training, using the Horovod library in conjunction with the Keras and Tensorflow packages. One of the major bottleneck in the 3DGAN application is the training time. In this contribution we present improved results on 3DGAN distributed training: we report on the reduction of training time and high scaling efficiency up to 256 Intel Xeon$^{\text{TM}}$8260 (codenamed "Cascade Lake") nodes. We demonstrate how HPC centers could be utilized to globally optimize deep learning models, thanks to their large computation power and excellent connectivity. In addition we document our preliminary studies on the use of genetic algorithms as an alternative method to train and optimise neural networks.

Artificial Intelligence for Science, Industry and Society, AISIS2019
October 21-25, 2019
Universidad Nacional Autónoma de México, Mexico City, México

[*]Speaker.

1. Introduction

High Energy Physics (HEP) experiments rely on detailed Monte Carlo techniques to simulate particle transport through detectors. Experiments at the Large Hadron Collider (LHC) devote today a large fraction of their worldwide distributed computing power [1], to the simulation task and the next High Luminosity LHC phase [2] is expected to increase the need for simulated data by a factor 100 [3].

Existing fast simulation techniques are mostly based on parametrization [4, 5] or look-up-tables [6] providing different levels of accuracy; deep Neural Networks are also being investigated as alternative solutions [7, 8, 9, 10]. Calorimetes are among the most time consuming detectors as far as simulation is concerned. In particular next-generation calorimeters are being designed as highly segm nted array of cells, capable of reaching very high spatial resolution [11]: their output is a three-dimensional pattern of energy depositions that can be interpreted as pixel intensities in an 3D image. The work presented in [10] demonstrates the performance of a three-dimensional convolutional GAN model (3DGAN) to simulate a highly granular electromagnetic calorimeter designed within the context of the Linear Collider Detector studies [11]. 3DGAN represents the initial step of a wider plan aimed at providing a generic detector simulation tool based on Deep Learning, fully configurable for different detector use cases. Our final goal is to prove that, by using meta-optimization and hyper-parameters scans, it is possible to tune the network architectures to simulate different detectors.

In this perspective, an efficient training process becomes essential and, therefore, we focus on optimizing the computing resources needed to train 3DGAN, studying parallelization on HPC centers and commercial cloud resources [12, 13, 14]. This work presents two main contributions: the results of the optimisation of 3DGAN inference and training on Intel Xeon^{TM}Scalable Processors (Cascade Lake) and the deployment of a distributed training approach over 256 nodes. We also present initial studies aimed at training and optimising 3DGAN using an evolutionary approach based on genetic algorithms (GA).

2. Previous Work

Generative Adversarial Networks (GAN) [15] implement the idea of adversarial training using two neural networks: a generator that reproduces the true data distribution and a discriminator, typically a classifier, discriminating generated from true examples. Training develops as a minimax optimisation process reaching, ideally, a saddle point that corresponds to a minimum for the generator and a maximum for the discriminator (Nash equilibrium).

The Auxiliary Classifier GAN (ACGAN) follows a semi-supervised approach via the introduction of a label resulting in faster convergence and more stable performance [16]. There have been many recent variations of GAN: our work combines an ACGAN-like approach with physics-derived constraints. The LAGAN [17] and CaloGAN [9] models represent the first applications of GAN to HEP simulation: particle showers in simplified calorimeters are simulated as a set of two-dimensional images. Further examples are described in [7, 8]. Simulation of highly granular calorimeters, using true 3D convolutions, to fully exploit the correlations in the volumetric space, achieves promising results [10].

As Deep Learning models increase in complexity, model size and training time, the role played by distributed training becomes of primary importance. Several different algorithms have been developed in recent years. For instance, in the data parallel approach each process trains a copy of the whole model on different parts of the data and the models need to stay consistent across the multiple processes. This results in large communication costs since gradients and/or model updates are exchanged among the nodes or between the nodes and a parameter server. Most existing Deep Learning tools and frameworks implement some form of parallel training, typically using MPI [18] as communication layer. Horovod is one example: it introduces a ring all-reduce pattern in which each of N nodes communicates with two of its peers 2*(N-1) times, strongly reducing communication costs [19].

3. The 3DGAN model training and inference processes

The 3DGAN model is extensively described in several publications [10, 20, 21]. It loosely follows an ACGAN approach, by introducing an auxiliary regression task on the particle energy to improve convergence and provide insights on the quality of the discriminator training. The generator and discriminator networks are build as four layers of three-dimensional convolutions. Batch-normalization and dropout layers are also introduced to improve sparsity and insure regularization. Two dense layers perform the real-fake classification and the auxiliary energy regression task. Overall, 3DGAN includes slightly more than 1 million parameters. However, it should be noted that the 3DGAN training process presents a high compute-to-communication ratio given that three-dimensional convolutions have a cubic computational complexity. The model is implemented using Keras [22], for its simplicity of use. As for Keras's backend, we have used Tensorflow [23], configured and compiled with MKL-DNN [24] support, so that its operations can be well mapped on CPU-based hardware. The binaries are built with architecture specific flags (AVX512 FMA) and XLA support.

Once trained, the GANs simulation tool is a relatively lightweight application: a few MB are enough to describe the layer configuration for the two networks and the inference step is orders of magnitude faster than a standard Monte Carlo. We have run a test on an Intel Xeon$^{\mathrm{TM}}$8180 processor (codenamed "Skylake") measuring the time it takes to generate one electromagnetic shower for our benchmark detector. We obtained 17 s/shower using Geant4 [25] and 1 ms/shower using the 3DGAN model, yielding a speedup factor larger than 20000. Using dedicated hardware, such as GPGPUs, the generation time could further reduce but we choose not to quote comparison results since the Geant4 application cannot run on GPGPUs. As explained in [12], training 3DGAN for on a single Intel Xeon$^{\mathrm{TM}}$8160 node (2 sockets, 24 cores each) required slightly less than 5 hours per epoch using an Intel optimised version of Tensorflow 1.9, linked to Intel MKL-DNN, Horovod 0.13.4 (with IntelMPI) and running 4 worker process per node. These results were obtained on the Stampede 2 cluster at TACC in 2018. Recent upgrades in the Intel MKL-DNN framework and Intel optimised Tensorflow 1.14 have improved this results by a factor 2.8 bringing the training time down to about 1.8 hours/epoch on a Intel Xeon$^{\mathrm{TM}}$Scalable Processor Platinum 8260, 2 sockets, 24 cores per socket.

4. Distributed training

Training large neural networks is a computationally intensive task, taking a substantial amount of time, from several hours to days or even weeks. For instance, training 3DGAN for 30 epochs on an Intel Xeon^{TM}Scalable Processor Platinum 8260, requires slightly more than two days. Parallelizing the training workload is thus critical to obtaining results in a timely manner. To speed up this part of the workflow, we leverage the power of modern supercomputers and decrease the training time.

In this work, we distribute the 3DGAN training via a data parallel strategy, given the limited communication requirements of the 3DGAN architecture. In addition, gradients are updated using a synchronous approach, due to the more controllable nature of synchronous stochastic gradient descent and the relatively limited straggling effects. Finally we use more than one worker per node to maximize the hardware efficiency as described in the sections below. Horovod in the distribution framework, is configured to use Intel MPI, optimized for shared memory communication within a single node and interconnection fabric across multiple nodes. The tests are run on the Intel Endeavour cluster, in particular on the CLX-SP partition composed of dual-socket Intel Xeon^{TM}Scalable Processors Platinum 8260 CPUs, 24 cores each, 192GB RAM. The interconnection fabric is Intel OmniPath. Best result are obtained with 2 MPI processes per node (1 per socket), with tuned Tensorflow interop and intraop parallelism threads and processes pinned to separate NUMA domains in order to minimize NUMA effects.

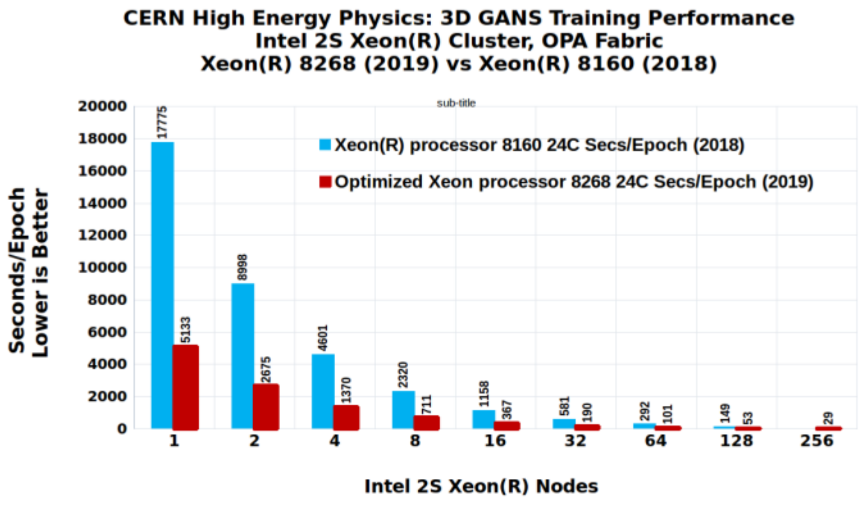

Figure 1: (blue) Training time per epoch on the TACC's Stampede 2 system [12]. (red) Training time per epoch on the Intel Endeavour cluster (Intel Xeon^{TM}Scalable Processors Platinum).

Figure 1 compares the results we obtained on the Endeavour cluster to the 2018 benchmarks described in [12]. Tensorflow optimisation improves the single node results by a factor 2.8. Distributing the training across multiple nodes yields linear scaling and less than 53 seconds/epoch using 128 nodes (compared to 149 seconds/epoch in [12]). Training time on 256 nodes is reduced to 29 seconds/epoch and convergence is reached in less than 15 minutes. The scaling efficiency, up to 256 nodes, is around 98%. No major communication bottleneck is observed.

In general, using a large number of nodes, the effective batch size has a negative impact on physics performance: we observe some degradation on the 256 nodes run in the low energy region; as an example, Figure 2 shows the ratio between the total energy deposited in the calorimeter estimated by the generator and the primary particle energy as a function of the primary particle energy. The 256 training result (green) overestimates the total energy deposited at the low end of the spectrum. Further studies are on-going to better characterize this loss in physics performance.

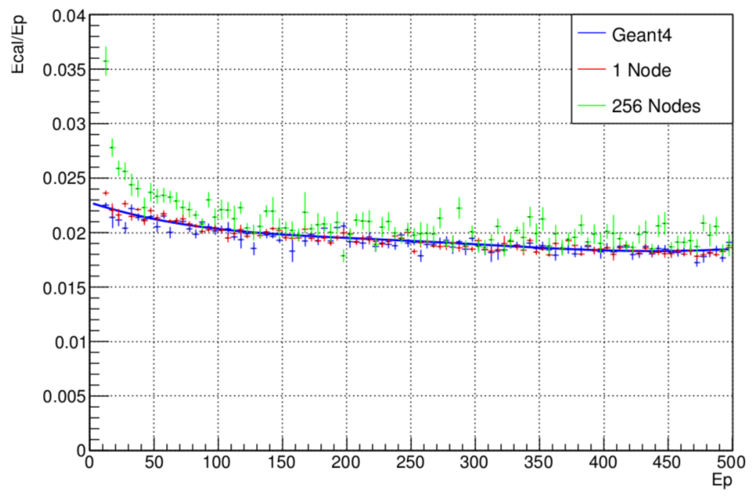

Figure 2: Ratio between the total energy deposited in the calorimeter and the primary particle energy as a function of the primary particle energy. (blue) The Monte Carlo prediction obtained using Geant4 [26]. (red) single node 3DGAN prediction. (green) 256 nodes 3DGAN prediction: the total deposited energy is overestimated, the effect is larger at lower primary particle energies.

5. Genetic Algorithms for Deep Neural Networks optimisation

Hyper-parameters scans are among the simplest way to optimise a neural network architecture for the task at hand. However, the deeper the neural network, the longer is the time required for an exhaustive search, since training and evaluation of every single instance increases. In this context, the training time can be reduced using a distributed approach as explained in the previous sections and in addition, efficient search strategies can be adopted: Bayesian optimisation, reinforced learning and genetic algorithms (GA). Here we explore the GA approach as an alternative to stochastic gradient descent (SGD) for training and hyper-parameter optimisation and we present some initial results.

A GA is a meta-heuristic often used as an iterative search algorithm in many domains where problems are complex or poorly defined. The solution parameters are encoded in a "chromosome" that is evolved according to the principle of best fitness. In the context of deep learning, the parameters encoded in a chromosome can be the network weights [27, 28, 29, 30] (training process) or the network hyper-parameters [31, 32, 33, 34] (architecture optimisation). As an example of GA-based network training, Deep Neuroevolution [29] trains a four millions weights deep neural network using a supervised learning approach. In order to reduce computing resources, distributed training is implemented as well as a compression technique for encoding of chromosomes. In [35] cartesian genetic programming is used to optimise a convolutional neural networks (CNN) architecture: CNN kernels are encoded as nodes in a graph with connections defined by the chromosome. The fitness evaluation is carried out using stochastic gradient descent.

The genetic algorithm can also be used to optimise the network topology as well as its weights, combining the training ad optimisation process and resulting in a faster and more efficient generalization approach, especially when combined with distributed training. However, as of today, a successful implementation of the combined architecture and weight training has yet to be demonstrated for CNNs. Here, we describe a preliminary test using a GA as an optimizer for training a CNN and we compare performance to standard SGD approach. We plan to extend this work incorporating architecture search and distributed training.

5.1 Implementation and evolutionary strategy

For our initial test we setup a regression problem on the two dimensional projection of the 3DGAN data along the transverse plane. To reduce the dimensionality of the problem from 3D to 2D we project the calorimeter three-dimensional images along one of the transverse dimension thus retaining the longitudinal development of the particle showers. Loosely following the structure of the 3DGAN discriminator network, effectively a CNN, we build a regressor CNN with a single convolutional layer and a single dense layer estimating the projected energy. The network is initialized using M random parents drawn from uniform distributions. The $1+\lambda$ evolutionary strategy is employed with *elitist selection* selecting a single parent with lowest loss and generating λ children. Each individual is defined by a chromosome consisting of a set of weights.

The evolutionary operator is a point mutation. The initial implementation is similar to [29] employing the point mutation controlled by two parameters: the mutation rate (μ) and the mutation power (mp). The mutation rate denotes the fraction of the weights to be mutated. The mutation power set a threshold for the weight updates. The actual update to the weight is the product of sampling from a uniform distribution [-1, 1] and the mutation power. The GA objective is a minimization task defined by the relative error on the regression task. The training is performed in batches given the large data size. Some initial tuning of the GA hyper-parameters resulted in the optimal set summarized in Table 1.

No.	Param	Description	Value
1	Weight mutation	Sampling random weights	$\mathcal{N}(\mu, \sigma^2)$
2	M	Number of random parents at initialization	8
3	λ	Number of offsprings	8
4	Bias	Constant term added to weighted sum for a neuron	0
5	Data size	Number of events used for training	5,000
6	Batch size	Data samples used in a single evaluation/iteration of the algorithm	128

Table 1: Parameters selected for GA

5.2 Results and Discussion

We compare the regression performance of the GA-trained network to that of a network trained using RMSprop [36] on a small sample of 5000 images. Figure 3 shows (left) how the training loss, defined as a relative error, changes through the epochs: it can be seen that GA achieves low loss values very quickly but RMSprop eventually reaches a lower loss. The estimated energy is also shown in Figure 3: the network trained using RMSprop has a lower error and steady performance along the whole energy range; the GA trained network performance is not as accurate but the resulting error stays well below 10% over the whole energy range.

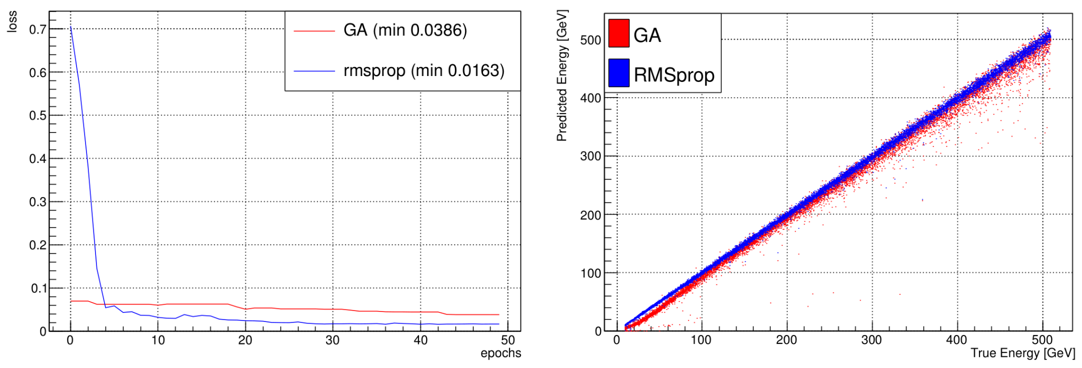

Figure 3: Regression results for GA (red) vs. RMSprop (blue). (left) Test losses. (right) Estimated energy.

This initial exercise suggests that a GA can be used to successfully train a CNN: the result is currently not as accurate as RMSprop but additional tuning might lead to comparable performance. Our plan is to continue this investigation and develop a methodology for combining network training and architecture otpimisation.

References

[1] WLCG, “Worldwide lhc computing grid.” `http://wlcg.web.cern.ch`.

[2] G. Apollinari et al., *High-Luminosity Large Hadron Collider (HL-LHC)*, *CERN Yellow Rep. Monogr.* **4** (2017) 1.

[3] The HEP Software Foundation, J. Albrecht et al., *A Roadmap for HEP Software and Computing R&D for the 2020s*, *Computing and Software for Big Science* **3** (2019) 7.

[4] W. Lucas, *Fast simulation for atlas: Atlfast-ii and isf*, in *International Conference on Computing in High Energy and Nuclear Physics*, vol. 396, 2012.

[5] D. Orbaker, *Fast simulation of the cms detector*, in *International Conference on Computing in High Energy and Nuclear Physics*, vol. 219, 2010.

[6] E. Barberio et al., *Fast simulation of electromagnetic showers in the ATLAS calorimeter: Frozen showers*, *J. Phys. Conf. Ser.* **160** (2009) 012082.

[7] M. Erdmann et al., *Precise simulation of electromagnetic calorimeter showers using a wasserstein generative adversarial network*, *ArXiv High Energy Physics - Experiment e-prints* (2020) [`https://arxiv.org/abs/1807.01954`].

[8] V. Chekalina et al., *Generative models for fast calorimeter simulation.lhcb case*, *ArXiv High Energy Physics - Experiment e-prints* (2019) [https://arxiv.org/abs/1812.01319].

[9] M. Paganini, L. de Oliveira and B. Nachman, *Calogan: Simulating 3d high energy particle showers in multi-layer electromagnetic calorimeters with generative adversarial networks*, *arXiv preprint arXiv:1705.02355* (2017) .

[10] G. R. Khattak, S. Vallecorsa and F. Carminati, *Three dimensional energy parametrized generative adversarial networks for electromagnetic shower simulation*, in *2018 25th IEEE International Conference on Image Processing (ICIP)*, pp. 3913–3917, Oct, 2018, DOI.

[11] M. Pierini, "The Linear Collider Detector Dataset." https://indico.hep.caltech.edu/indico/getFile.py/access?\\url{contribId=16\&sessionId=9\&resId=0\&materialId=slides\&confId=102}, 2016.

[12] S. Vallecorsa, F. Carminati, G. Khattak, D. Podareanu, V. Codreanu, V. Saletore et al., *Distributed training of generative adversarial networks for fast detector simulation*, in *High Performance Computing*, R. Yokota, M. Weiland, J. Shalf and S. Alam, eds., (Cham), pp. 487–503, Springer International Publishing, 2018.

[13] S. Vallecorsa, D. Moise, F. Carminati and G. R. Khattak, *Data-parallel training of generative adversarial networks on hpc systems for hep simulations*, in *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, pp. 162–171, Dec, 2018, DOI.

[14] D. Brayford, S. Vallecorsa, A. Atanasov, F. Baruffa and W. Riviera, *Deploying ai frameworks on secure hpc systems with containers.*, in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6, Sep., 2019, DOI.

[15] I. J. Goodfellow et al., *Generative Adversarial Networks*, *ArXiv e-prints* (2014) [1406.2661].

[16] A. Odena, C. Olah and J. Shlens, *Conditional Image Synthesis With Auxiliary Classifier GANs*, *ArXiv e-prints* (2016) [1610.09585].

[17] L. de Oliveira, M. Paganini and B. Nachman, *Learning particle physics by example: Location-aware generative adversarial networks for physics synthesis*, *arXiv preprint arXiv:1701.05927* (2017) .

[18] M. P. I. Forum, "Mpi: A message-passing interface standard." https://www.mpi-forum.org/docs-draft-report.pdf, 2019.

[19] A. Sergeev and M. D. Balso, *Horovod: fast and easy distributed deep learning in tensorflow*, *CoRR* **abs/1802.05799** (2018) [1802.05799].

[20] G. Khattak et al., *Particle detector simulation using generative adversarial networks with domain related constraints*, in *IEEE International Conference on Machine Learning and Applications, ICML2019*, 2019.

[21] F. Carminati, A. Gheata, G. Khattak, M. L., P. and S. Vallecorsa, *A machine learning tool for fast simulation in geantv*, in *ACAT*, (Seattle), 2017, https://indico.cern.ch/event/567550/contributions/2627179/.

[22] F. Chollet et al., "Keras." https://github.com/fchollet/keras, 2015.

[23] M. Abadi et al., *An end-to-end open source machine learning platform*, 2015.

[24] I. Corporation. https://software.intel.com/en-us/articles/intel-mkl-dnn-part-1-library-overview-and-installation, 2020.

[25] J. Allison, K. Amako, J. Apostolakis, P. Arce, M. Asai, T. Aso et al., *Recent developments in geant4*, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **835** (2016) 186 .

[26] CERN, "Geant." `http://geant.cern.ch/`, accessed July 31, 2017.

[27] P. Verbancsics and J. Harguess, *Image classification using generative neuro evolution for deep learning*, in *2015 IEEE Winter Conference on Applications of Computer Vision*, pp. 488–493, Jan, 2015, DOI.

[28] G. Morse and K. O. Stanley, *Simple evolutionary optimization can rival stochastic gradient descent in neural networks*, in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, (New York, NY, USA), pp. 477–484, ACM, 2016, DOI.

[29] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley and J. Clune, *Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning*, *CoRR* **abs/1712.06567** (2017) [`1712.06567`].

[30] J. Schrum, *Evolving indirectly encoded convolutional neural networks to play tetris with low-level features*, in *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, (New York, NY, USA), pp. 205–212, ACM, 2018, DOI.

[31] L. Xie and A. Yuille, *Genetic cnn*, in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1388–1397, Oct, 2017, DOI.

[32] T. Desell, *Large scale evolution of convolutional neural networks using volunteer computing*, in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, (New York, NY, USA), pp. 127–128, ACM, 2017, DOI.

[33] Y. Sun, B. Xue, M. Zhang and G. G. Yen, *Automatically designing cnn architectures using genetic algorithm for image classification*, *ArXiv* **abs/1808.03818** (2018) .

[34] A. Baldominos, Y. Sáez and P. Isasi, *Hybridizing evolutionary computation and deep neural networks: An approach to handwriting recognition using committees and transfer learning*, *Complexity* **2019** (2019) 2952304.

[35] M. Suganuma, S. Shirakawa and T. Nagao, *A genetic programming approach to designing convolutional neural network architectures*, in *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, (New York, NY, USA), pp. 497–504, ACM, 2017, DOI.

[36] G. Hinton, N. Srivastava and K. Swersky, "Lecture 6a overview of mini–batch gradient descent." 2012.