A Deep Neural Networks ensemble workflow from hyperparameter search to inference leveraging GPU clusters

PIERRICK POCHELU, TotalEnergies SE, France
SERGE G. PETITON, Univ. Lille, CNRS, UMR 9189 CRIStAL, France
BRUNO CONCHE, TotalEnergies SE, France

Automated Machine Learning with ensembling (or AutoML with ensembling) seeks to automatically build ensembles of Deep Neural Networks (DNNs) to achieve qualitative predictions. Ensemble of DNNs are well known to avoid over-fitting but they are memory and time consuming approaches. Therefore, an ideal AutoML would produce in one single run time different ensembles regarding accuracy and inference speed. While previous works on AutoML focus to search for the best model to maximize its generalization ability, we rather propose a new AutoML to build a larger library of accurate and diverse individual models to then construct ensembles. First, our extensive benchmarks show asynchronous Hyperband is an efficient and robust way to build a large number of diverse models to combine them. Then, a new ensemble selection method based on a multi-objective greedy algorithm is proposed to generate accurate ensembles by controlling their computing cost. Finally, we propose a novel algorithm to optimize the inference of the DNNs ensemble in a GPU cluster based on allocation optimization. The produced AutoML with ensemble method shows robust results on two datasets using efficiently GPU clusters during both the training phase and the inference phase.

CCS Concepts: • **Computing methodologies** → **Ensemble methods**; **Search methodologies**; **Massively parallel algorithms**.

Additional Key Words and Phrases: neural networks

ACM Reference Format:
Pierrick Pochelu, Serge G. Petiton, and Bruno Conche. 2022. A Deep Neural Networks ensemble workflow from hyperparameter search to inference leveraging GPU clusters. In *International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia2022), January 12–14, 2022, Virtual Event, Japan.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3492805.3492819

Deep Neural networks (DNNs) are notoriously difficult to tune, train, and ensemble to achieve state-of-the-art results. Automatic machine learning with ensembling or "AutoML+ensembling" tools provide a simple interface to train and evaluate many ensembles of DNNs to achieve high accuracy by reducing overfitting.

Nowadays, multiple researchers and practitioners have well understood the benefit of ensembling DNNs. For example, in cyber-attack detection [19], time series classification [43], medical image analysis [7], semi-supervision [50] and unbalanced text classification [49]. Further, several winners and top performers on challenges routinely use ensembles to improve accuracy. However, ensembles of DNNs suffer from three main limitations to be widely deployed in research and industrial applications.

The first limitation is a lack of understanding about the best way to build base DNNs to construct an ensemble. Ensembling is still not fully understood [29] [37] [4] but authors generally agree that a large number of models, high diversity, and high individual performance are the three key components but we ignore in which proportion. That is why automatic procedures assessing multiple possible ensembles have been proposed. AutoML with posthoc ensembles

Manuscript submitted to ACM

[14] works as follow: first, it automatically builds a library of hundreds of models, and then, it explores thousands of ensembles among the huge number of possible ensembles (combinations) [6] [5] [53] based on their validation score. However, it is still unclear how best to construct the best library of models to perform ensemble selection and only experimental evidence seems able to drive the algorithmic choices of the workflow.

The second limitation is the lack of control of the computing cost of the produced ensemble. In the previous works, authors apply ensembling [6] [14] of non-deep machine learning and propose that the number of models to put in the ensemble as a threshold between accuracy and computing cost. In DNNs such as applied on image recognition, the combined models are heterogeneous with orders of magnitude of resource requirement. That is why the number of models as the constraint is not relevant to control the ensemble computational cost. Moreover, when generating automatically multiple models it is known that [26] there is no clear correlation between accuracy and DNNs cost, meaning that sometimes fast DNNs can be prioritized without significantly lowering the accuracy.

Finally, no inference server enables the deployment of heterogenous DNNs and fully leverages modern GPU clusters. Current inference server allows to deploy deep neural networks [56] [40] [42], but the administrator of those servers has to attach manually DNNs to GPUs and set the batch size. Ensemble of DNNs is much more complex because we must deal with multiple DNNs sometimes co-localized into the same device and multiple devices. An ideal inference server of DNNs ensemble would allow computing automatically the best localization and batch size settings at the initialization phase.

It is time to address this missing piece in deep learning pipelines between AutoML, ensemble, and GPU clusters. To summarize our contributions, we run extended benchmarks with seven algorithms to generate the best library of models compared to 7 other algorithms and conclude that asynchronous Hyperband [34] suits this goal. After that the library is generated, we propose a new simple algorithm named SMOBF (scalarized-multi objective with budget greedy forward) to build an efficient ensemble based on their accuracy and a desired maximum computing cost. Third, we propose a novel server design to deploy with high efficiency, high flexibility, and low overhead a heterogeneous ensemble of DNNs.

This paper is as follows. In section II we go into further detail in the related fields: AutoML and AutoML+ensembling. In section (III) the different steps of the workflow AutoML+ensembling are analyzed and introduced. In section (IV) we introduce the inference server design for deployment. In section (V) we analyze our AutoML procedure then (VI) we benchmark our inference server on 2 generated ensembles.

1 ANALYSIS OF THE AUTOML FIELD

1.1 AutoML

The empirical nature of research in Deep Learning leads scientists to try many model architecture settings, optimization settings, and pre-processing settings to find the best-suited one for data. AutoML is made of 3 modules, the DNNs search space (the “hyperparameter space”), the DNNs sampling strategy (or “hyperparameter optimization”), and the evaluation phase consisting in returning the score associated with one hyperparameter value.

Any model previously trained with hyperparameter λ sampled from hyperparameter space Λ is written M_λ. The hyperparameter optimization goal defined in equation 1 consists in finding the best hyperparamer $\lambda* \in \Lambda$ building a model to reduce the error measured by E. The error is measured on the validation data x_{valid} matching labels y_{valid}.

$$\lambda* = \underset{\lambda \in \Lambda}{\arg\min}\, E(M_\lambda(x_{valid}), y_{valid}) \tag{1}$$

In the literature, AutoML algorithms are typically compared based on their results in the evaluation phase. While this may seem intuitive, this field is now facing multiple methodological questions [36] on the relevance of comparing multiple workflows made of different stages and different initial conditions. And more, to fairly compare their robustness, multiple datasets, and multiple random seeds must be reported which is a computing-intensive research area limiting initiatives.

Our paper does not claim the superiority of our AutoML+ensembling method in all cases but it is rather the first step toward AutoML+ensembling of DNNs aiming at a desired trade-off between the produced ensemble accuracy and its cost for practical usage. Our work is applied on the computed vision task but can be generalized on other tasks where qualitative prediction is required leveraging one or more clusters of GPUs.

No Free Lunch theorem [54] proves that no black-box hyperparameter optimization (HPO) can show superior performance to random search in all cases. Nevertheless, methods searching between global exploration and local exploitation have shown a stable performance on diverse applications. Early stopping [45] [33] has been also proposed to stop the less promising trials to focus hardware on the most promising trials accelerating the overall optimization process.

1.2 AutoML frameworks

Recently, AutoML methods sequentially updating the current DNN (neural network morphism) have been proposed based on Bayesian optimization (Auto-Keras [25]), reinforcement learning (ENAS [44]) or differentiable architectures (DARTS [39]). These methods require several hours on a single modern GPU and converge to a sub-optimal solution.

A more general approach consists in searching not only neural network architectures but optimization settings and data processing too with fixed-length vector hyperparameters. Bayesian optimization, like Sequential Model-Based Optimization (SMBO) with Gaussian model is known to perform well to optimize continuous hyper-parameters [47] [22]. Tree-based models are more adapted to the discrete hyperparameters like Tree Parzen Estimator (TPE) [3] and Sequential Model-based Algorithm Configuration (SMAC) [24]. Despite that SMBO are inherently sequential methods, a parallel version have been proposed [47] based on successive populations of trials to explore the hyperparameter space by leveraging multi-cores.

Evolutionary methods [46] [57] are naturally parallelizable algorithms running successive populations (called "generation") of trials in parallel. By running them on a large scale on GPU clusters for several days, authors discovered those methods can converge very late and very high with a high robustness.

1.3 AutoML with ensembling

AutoML which not only selects the best model but combines them is valuable for domains that require the best possible accuracy. Several benchmarks were performed and AutoML+ensembling is today considered as the big AutoML challenge series winner on data points like AutoML challenges [18] and Kaggle challenges on image recognition. Previous researches on non-deep machine learning ensembles shows that over-fitted machine learning algorithms predictions can be averaged out to get more accurate results [48]. This phenomenon is mainly explained by the Law of Large Numbers which claims that the average of the results obtained from many non-biased trials should be close to the expected value. These results are especially interesting for deep learning models. They are the most affected models to random effects (over-fitting) due to their huge number of parameters.

We observe two main trends in AutoML with ensembling. First, *AutoML+Ad-hoc ensembling* [52] [30] [16] [8] which consists in searching directly ensembles. The hyperparameter space describes an ensemble of fixed size. At the end,

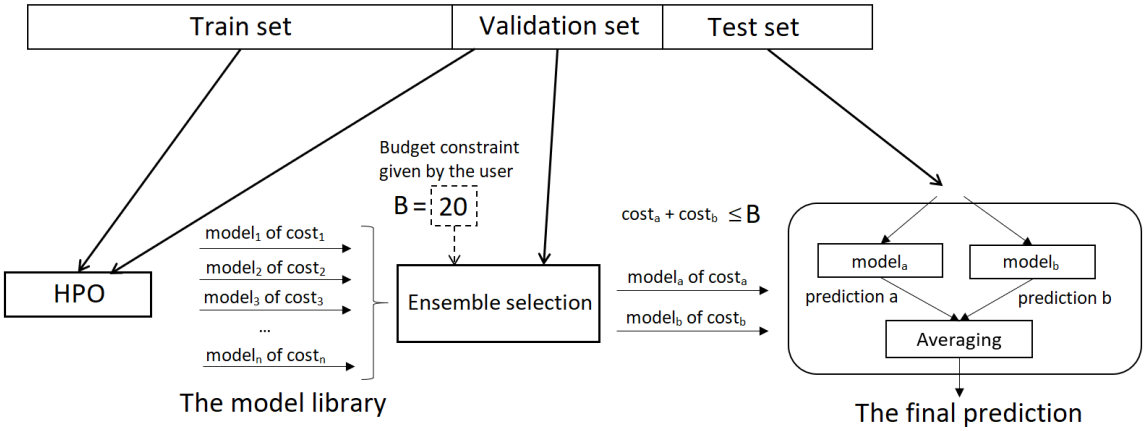

Fig. 1. The proposed workflow runs 3 steps 1) The HPO algorithm generates a library of models. The trials are distributed on several GPUs synchronized by a master process. We recommend asynchronous Hyperband based on experimental results. 2) We propose a new multi-objective Ensemble Selection algorithm to search the most efficient ensemble honoring a budget given by the user (here B=20). It is based on a parallel greedy algorithm evaluating hundreds of combinations per second on multi-core CPUs. 3) The returned ensemble predicts by combining (averaging) DNNs predictions on new data. The deployment of the ensemble into a server is not shown in the figure.

the HPO algorithm keeps the best ensemble and wastes all the others. It suffers from a lack of flexibility because the number of models in an ensemble is fixed before build all DNNs. In other words, changing the number of DNNs in the ensembles require a new AutoML runtime. Second, *AutoML+Post-hoc ensembling* [6] [53] [14] runs a standard HPO algorithm providing a library of trained models, then constructs ensembles from the library based on a greedy algorithm. This approach is flexible because we can produce several ensembles with different numbers of models with the same library of DNNs.

2 PROPOSED WORKFLOW

In this section, we give an overview of the proposed workflow 1. Then, we will go into further details on the distributed architecture that accelerates these three steps of our AutoML workflow: HPO to construct a library of models, Ensemble Selection to construct ensembles based on their validation accuracy and computing cost, and ensemble deployment.

2.1 Detail of the workflow

Hyperparameter optimization (HPO). The choice of the HPO algorithm has several impacts in terms of the number of produced models at a given time horizon, the accuracy of models, and their diversity. After several experiments, we recommend Hyperband based on our experiments.

In this regard, Hyperband produces the biggest library of models among tested algorithms because most models do not reach the maximum number of epochs. It is also a lock-free distributed algorithm until the termination of the algorithm, spending most of its runtime to train models occupying all GPUs and storing them on the library. Then, Hyperband produced also very diverse models due to the initial random sampling of hyper-parameters which is desirable in terms of final accuracy. Finally, because more training iterations are given to the best models, Hyperband performs explore-exploit in the time (not in the hyperparameter space) and we observe it builds a better distribution of individual DNNs (regarding the top 10 % and top 25 %) compared to random search on all our runtimes. But we

cannot expect that Hyperband outperforms or underperforms other exploratory-exploitative HPOs in all cases such as Bayesian or genetic algorithms [54]. A large number of models, hyperparametric diversity, and efficiency are three reasons explaining why Hyberband is robust to build a library of models but we still ignore in which proportion they are important.

Ensemble selection. An ensemble selection algorithm finds the best combination possible honoring the budget given by the user. We propose SMOBF greedy standing for "Scalarized Multi-Objective with Budget Forward greedy".

Forward greedy [6] starts with the empty ensemble and successively adds the best available model to improve the ensemble target metric. The algorithm stops when no available model can improve accuracy or respect the budget.

We propose new equation 2 to inform the greedy algorithm to favor accurate and cheap DNNs before consuming the overall budget. The current ensemble is a with its computing cost C_a and its predictions y_a. Penalty P_a returns 0 when the budget B is honored ($C_a \leq B$) or an arbitrary large number when it is not to exclude a from the possible ensembles.

The weight w allows controlling the nature of the solution found by the greedy algorithm by placing greater or lesser emphasis on the objectives. The greedy algorithm is run multiple times with different values $w = 0.1$, $w = 0.01$, and $w = 0.001$. Scalarization is a convenient way to handle multi-objective problems by reducing them to a single objective problem, so a simple mono-objective optimization can be performed.

$$score_a = (1 - w) * E(y_a, y) + w * C_a + P_a \tag{2}$$

Then, the best ensemble is picked according to its validation cross-entropy loss and respecting the given budget. To improve the robustness of the method on new applications C_a is standardized and it is the rate between the sum of computing cost of base models (time to predict 2000 images) and the budget B.

Ensemble selection algorithm computes the validation loss of candidate ensembles to evaluate how well a solution will generalize on the test database. Since the data used for validating is taken away from training the individual models, keeping the validating set small is important. Smaller validating sets are however easier to overfit. Contrary to common AutoML on data points datasets, due to the cost of training and evaluating one model on images datasets, we do not repeat the experience with K-fold cross-validation.

In the library of models, some models diverge or have such poor performance compared to other models that they are unlikely to be useful to improve any ensemble building [5]. Eliminating these models from the library should not reduce the performance and facilitates the ensemble selection task by decreasing the number of non-promising combinations. Pruning works as follows: models are sorted by their performance on the validation metric and only the top X% of them are used for ensemble selection. After pruning, the predictions on the validation set are cached before running the ensemble selection algorithm. This allows handling only predictions vector and not models during the ensemble selection process.

Ensemble combiner rule. We use the simple average as a combiner rule. More advanced methods exist such as "ensemble selection with replacement" [6], weighted averaging, and stacking. Those methods are calibrated on a validation set and thus prone to over-fitting.

Now that the workflow is developed, the final accuracy depends on two settings. The tuning effort of the library of models and the budget given by the user to generate a new ensemble.

2.2 Distributed HPO with GPU clusters

To assess large numbers of hyperparameter trials in a reasonable amount of time, a distributed framework to use one GPU cluster or several GPU clusters is required. Several trials are distributed with a middleware containing one

scheduler and multiple workers, each worker being associated with heavy computing resources. Typically, the scheduler sends a hyperparameter set to the available workers with the number of training epochs to perform. Then, it gathers the scores of the trials from the workers and finally computes the next hyperparameters to send and so on.

This distributed framework suits the need of most optimization by leveraging GPUs and clusters. Each trial is run independently without slowdown among themselves. Workers can connect or disconnect to the master at run time; however, a disconnection interrupts the current trial. Many HPO algorithms benefit from this middleware such as random search, Hyperband, parallel SMBO, evolutionary algorithms, and also some greedy algorithms for discrete optimization such as SMOBF.

Modern clusters have evolved into a hybrid machine that contains both CPUs and GPUs on each node. These heterogeneous CPU-GPU clusters are particularly useful to accelerate the training loop of one trial. One trial is implemented with two processes, the *preprocessor* which loads and preprocesses the data on multi-cores, while the *trainer* trains the DNNs on the GPU. In case the entire training dataset fits into memory, the preprocessor does not need to load multiple time the data, it is shared between preprocessors of the cluster to avoid copies and useless I/O.

2.3 Distributed Ensemble Selection

Ensemble selection is a greedy algorithm evaluating all neighbors of a current ensemble at each iteration. For each ensemble a, the equation 2 is performed. This procedure takes one second on one CPU with 100 elements per prediction (100 classes), 2000 validation data samples, and a relaxed budget.

In the case of semantic segmentation images, the class predictions are at pixel scale making this procedure much more computing-intensive. A typical dataset for autonomous vehicle applications [10] contains 256x256 input images, 30 classes per pixel and 500 validation samples. Linear scaling indicates that the computing cost of this procedure would take 1h20 (256x256x30x500 predictions elements compared to 100x2000 predictions elements in CIFAR100 taking 1 second). Because SMOBF is an optimization algorithm we may use a similar distributed framework (previous section) to distribute neighbors evaluation on CPUs or GPUs to alleviate this computing cost.

2.4 Multi-GPU inference

After getting an ensemble of DNNs, we need to serve it efficiently on the available computing resources. We design an efficient inference pipeline illustrated in figure 2. All DNNs into the ensemble predict asynchronously. We store the input and the outputs into shared data to avoid slowing copies. The orchestrator asynchronously runs in one CPU-core the cumulative averaging of all predictions to avoid slowing down the entire pipeline.

Algorithm 1 aims to return an available solution in terms of memory while algorithm 1 speeds up the allocation settings found based on iterative updates.

Algorithm 1 balances the memory requirement between devices to fit into memory in only n steps, with n the number of models in the ensemble. It is basically a variant of the worst-fit-decreasing algorithm to solve a bin packing problem except we give a priority on the GPU because we make the assumption it is faster. If it is not possible, it allocates on the CPU because more memory is present. We decided to not write it for space reasons and because it is similar to the already known worst-fit-decreasing.

Algorithm 1 starts with a correct allocation and at each iteration, it assesses all neighborhood settings distant from 1 update. One update consists of either updating the batch of any DNN or changing its device. The algorithm is stopped when no neighborhood improves the target metric or when the max number of iterations is reached. The number of possible combinations is $(M+B)^n$ with M devices, B batch size values, and n DNNs in the ensemble. Algorithm 1 breaks

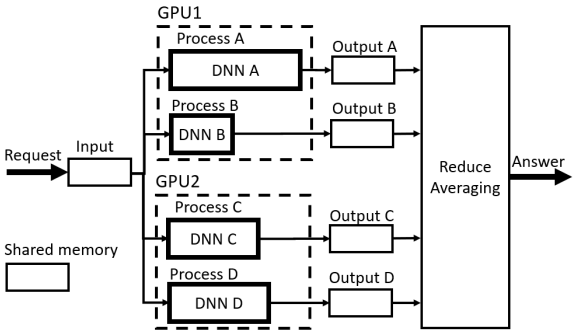

Fig. 2. Toy example of allocation of 4 heterogenous DNNs into 2 GPUs for inference. There are 5 processes, the orchestrator containing and its 4 children A, B, C and D. The orchestrator receives data to predict and serve predictions as a service. Shared memory are buffers in the RAM.

down this complexity into a succession of $Mn + Bn$ combinations (or "neighbors") to assess. We have no guarantee that this greedy algorithm finds the optimal allocation and we cannot verify if it is obtained (except if we brute-force all possibilities). However, we have the guarantee that in the worst-case scenario, a solution as good as the starting one is returned (line 10).

We formulate an inference setting like a $\{B, G, C\}$ set containing three lists. We use ordered structures because it makes it easier to write the algorithm. B_i is the batch size of the i^{th} DNN. G_j is the set of DNNs contained in the j^{th} GPU, in the same way, C_k contains DNNs into the k^{th} CPU. Again, a DNN is placed into one single device but a device can contain multiple DNNs.

In algorithm 1 I are randomly generated images to benchmark and calibrate the inference server. The number of fake images must be chosen high enough to smooth random effect and not too high to reduce the duration of the overall algorithms. *bench* function instantiates the pipeline with the given allocation settings (first argument) on the fake images (second argument) and returns the performance metrics (e.g. the number of predictions per second). The same algorithm can generalize well to other performance metrics such as latency consisting in reducing the time between one input data sample and its prediction.

3 EXPERIMENTS AND RESULTS

We experiment and discuss our workflow by varying the three steps of the AutoML workflow of deep neural networks on CIFAR100 and the microfossils datasets. More details is given in appendix A.

3.1 The infrastructure

Experiments were done on IBM Power9 architecture, containing 2 sockets. In each socket, there are 18 cores of CPU with a maximum frequency of 3.8Ghz and 256G of RAM. There are also 6 GPUs by node and are Nvidia Tesla V100-SXM2 with 16G of memory. Hyperparameter optimization framework Tune [38] was used. It runs above the Ray framework [40], it schedules and spreads experiments to run on GPUs and store results into files. Deep Learning training loop and data augmentation was coded with the framework Keras [17] with TensorFlow 1.14.0 [1] backend.

Algorithm 1 Refine GPUs allocation to speed up

input: D the list of DNNs in the ensemble, PB possible batch size values, max_combi maximum number of assessed combinations, G and C are preliminary GPU and CPU allocation, B preliminary batch sizes
output: G, C, B
start
$I \leftarrow fake\_images()$ // generate fake data to calibrate the allocation
$current\_score \leftarrow$ bench(D,B,G,C,I)
while $trials < max\_combi$ **do**
  better_allocs $\leftarrow$ []
  better_scores $\leftarrow$ []
  $i\_allocs \leftarrow$ update_i_alloc(i,B,G,C)
  for all $\{B2, G2, C2\}$ in i_allocs **do**
    if $\{B, G, C\} \neq \{B2, G2, C2\}$ **then**
      $score2 \leftarrow$ bench(D,$B2$,$G2$,$C2$,I)
      if $score2 > current\_score$ **then**
        $better\_scores$.append($score2$)
        $better\_allocs$.append($\{B2, G2, C2\}$)
      end if
      if $trials \geq max\_iter$ **then**
        break
      end if
    end if
    $trials \leftarrow trials + 1$
  end for
  if length($better\_allocs$) > 0 **then**
    id=argmax($better\_scores$)
    $\{B, G, C\}$ = $better\_scores[idbest]$
    $current\_score$=$better\_scores[idbest]$
  end if
end while
return $\{B, G, C\}$

3.2 step 1 - HPO

3.2.1 Comparison of hardware allocation. Our CNN framework based on ResNet can take between 15 seconds and 11min30 regarding the complexity of the neural architecture assessed (number of filters per convolution, number of convolutional blocks, ...). Our benchmark 1 reveal that Random Search and Hyperband occupy more the GPUs than algorithms based on a sequence of population algorithms (such as GA) which is explained by the fact that no intermediate stopping of all GPUs is required. Those benchmarks also confirm that Hyperband terminates earlier than Random Search because the less promising DNNs have been stopped before the maximum number of epochs is reached. The scalability of HPO algorithms and saving useless computations are two important algorithmic characteristics to explore a large number quantity of DNNs.

At the end of any HPO algorithm, the last few DNNs free the GPUs but they take various amounts of time to terminate, this explains why we do measure not a perfect usage of GPUs of Random Search and Hyperband. Hyperband emphasis this phenomenon because the early stopping increases the variability of time took between DNNs.

	9 trials 100 epochs					30 trials 30 epochs				
	acc (%)	duration	speed up	#epochs	gpu (%)	acc (%)	duration	speed up	#epochs	gpu (%)
RS 6GPUs	67.08 ± 3.85	10h13	6.2	900	85	64.90+-0.92	11h40	5.5	900	82
HB 6GPUs	62.58 ± 4.73	5h48	4.4	710	49	62.94 ± 5.28	1h40	5.2	388	62
HB 4nodes*6GPUs		2h00	8.8	710	33		0h48	10.9	388	34
GA 6GPUs	61.85 ± 2.55	14h29	2.8	900	27	65.33 ± 2.33	10h01	4.1	900	57

Table 1. Limited scale benchmarks of some HPO algorithms. Hyperband was set up with a halving of 3 and the genetic number of trials is divided into 3 successive generations. The columns from the left to the right are: The accuracy error which is not dependant on the number of resources, the duration of the HPO process, the speed up measured compared to the single GPU version, the #epochs is the number of train iteration performed, the mean percentage of occupied GPU during all the entire HPO process.

Data	Budget	RS	HB	BOGP	BOHB	SMAC	TPE	GA
C100	20	31.26	27.8	31.58	29.24	**24.44**	29.97	24.61
	40	24.24	**22.87**	28.2	25.6	22.97	26.01	22.91
	60	22.27	21.85	26.91	23.53	22.65	25.44	**21.51**
	80	22.69	21.73	26.2	22.58	22.17	24.25	**21.07**
	120	21.63	**21.55**	24.78	21.86	22.17	23.03	21.95
	160	**20.11**	**21.11**	24.24	21.64	21.87	22.47	21.91
	240	20.7	**20.46**	23.76	21.2	21.87	22.55	21.91
	320	20.64	**20.42**	23.76	21.1	21.87	22.46	21.91
Micro	125	13.45	11.93	14.44	12.18	**10.27**	13.91	10.33
	250	10.98	9.82	10.93	11.71	**9.63**	10.66	9.71
	375	10.84	**9.32**	9.93	10.5	9.45	10.49	9.5
	500	10.23	**8.91**	9.93	9.84	9.27	9.06	9.28
	750	9.7	**8.87**	9.21	9.28	9.18	9.3	9.18
	1000	9.69	**8.46**	8.77	8.8	9.09	9.21	9.07

Table 2. Our workflow error (%) by comparing seven HPO algorithms was run on both datasets for 6 days on 6 GPUs. Each HPO generates a library of models for a given dataset. The budget is expressed as "sum of DNNS times (seconds) to predict 2K images on 1 GPU"

3.2.2 HPO to produce ensembles. We compare in table 2 our presented workflow by varying the HPO algorithm to generate a different library of models. Random Search (RS) [2], asynchronous Hyperband (HB) [34], Bayesian Optimization with Gaussian Processes (BOGP) [22], BOHB, Sequential Model-based Algorithm Configuration (SMAC) [24], Tree Parzen Estimator (TPE) [3], Genetic Algorithm (GA). To combine generated models, we benchmark them with the SMOBF greedy ensemble selection algorithm. We benchmark three times each algorithm and we show the median value of each computing. When the budget is relaxed we observe the standard deviation is generally lower than 0.1%. When the budget is lower the standard deviation is lower than 1.5%.

Not surprisingly, when the budget increases the ensemble selection can find better ensembles from any library. This is explained by the fact that the number of available good combinations between them increases. In the table 2 some algorithms converge and do not found better ensembles after high budgets such as the Bayesian method and genetics ones. Different run time shows similar conclusion. On both datasets, the best ensemble is found with Hyperband. 79.44% accurate on CIFAR100 and 91.54% accurate on microfossils. We do not show results of AutoML without ensembling compared to AutoML+ensembling due to a lack of space, but AutoML+ensembling is Pareto dominant for all budgets and all performed run times.

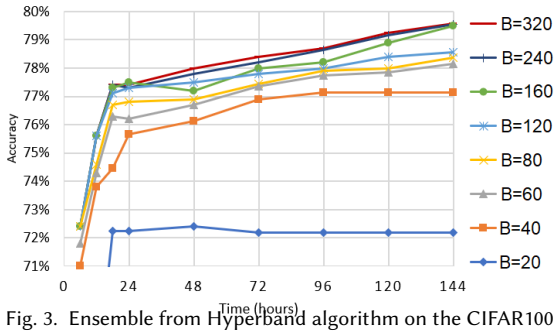

Fig. 3. Ensemble from Hyperband algorithm on the CIFAR100 dataset

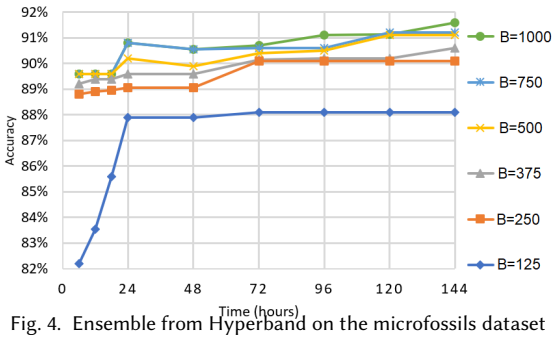

Fig. 4. Ensemble from Hyperband on the microfossils dataset

3.2.3 Effect of tuning time. In AutoML we often want to see the evolution of performance over the tuning time, not only the final performance after a time horizon. Therefore, we assess the workflow varying the budgets until it does not affect the ensemble construction anymore and at different tuning time snapshots. Those figures are presented in figure 3 4.

First, regarding the benefit of the tuning time, we observe two main trends. When the budget is very small (B=20) models accuracy converges early to 18, 24, or 48 hours reaching the limit of the hyperparameter optimization with a little (or without) ensembling. When the budget is bigger the exploding number of available combinations of ensembles leads to the discovery of better ensembles. The post-hoc ensembling is a promising line of research that deserves more attention and more understanding of how DNNs interact with each other.

Then, we observe that increasing the budget systematically leads to an increasing accuracy (colored lines are rarely crossed) but this trend decline. For example, in the figure 3 we show that when Hyperband is finished, the benefit is obvious from 1 to 2 models: +5 point of accuracy percentage, but the improvement is small from B=120 to B=320: +1 point of accuracy percentage.

3.3 step 2 - Ensemble Selection

3.3.1 Ensemble selection pruning. We try multiple pruning factors X such X% only the top X% are kept. It reduces the size of the library of models and thus helps the ensemble selection algorithm. When the pruning factor is above 20% it does not reduce accuracy and reduce the ensemble selection time, while the threshold under 15% reduces the target metric in some experiments. For all experiments, the pruning factor is set to 20%.

3.3.2 Ensemble selection methods under budget. The literature [48] indicates the diversity is of high importance to increase ensemble machine learning. The relationship between models diversity and the accuracy of their combination is not fully understood [29] but the methods exploring automatically the ensemble space from a library of models and returning the most promising ensemble have been proposed [6] [53] but as far we know we are the first to propose a multi-objective with a budget to suit with a heterogeneous ensemble of DNNs.

3.3.3 SMOBF greedy compared to Forward greedy (baseline). The ensemble selection algorithm often used by most advanced AutoML software [6] [14] is *forward greedy with fixed number of models* that we pick as baseline. We perform three runtimes of the overall AutoML workflow and observe that SMOBF is Pareto dominant or equivalent to the baseline each time.

The figures 5, 6 compares those two algorithms in function of the cost (vertically) and the error rate (horizontally) of produced ensembles with SMOBF greedy (blue) and the baseline (orange) on one runtime. On figures the mention "BX" means a budget of X and "#Y" means an ensemble of size Y.

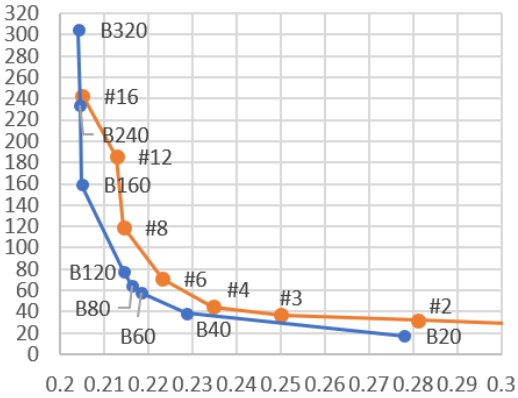

Fig. 5. Ensembles generated from Hyperband algorithm on the CIFAR100 dataset

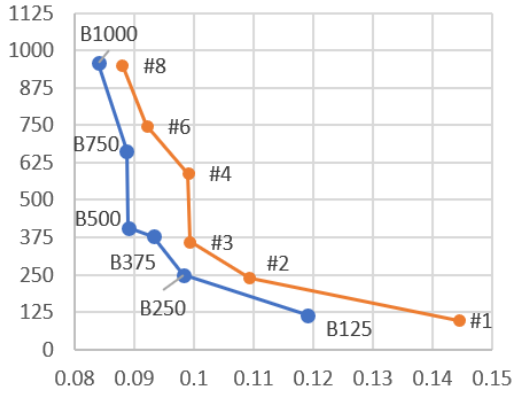

Fig. 6. Ensembles generated from Hyperband algorithm on the microfossils dataset

The gain of SMOBF is particularly obvious when the budget is small on both criteria, but it is reduced when the budget is relaxed. Indeed, when we target the best ensemble at any cost or any size, the objective of the two algorithms converges. On the opposite when the budget is small, SMOBF informs the greedy algorithm to go toward efficient models before the budget is consumed.

3.3.4 Assessing ensemble combiners. During the ensemble selection process, we try three ways to best combine a candidate ensemble: majority voting, averaging, and weighted averaging (or equivalently Ensemble Selection with replacement). Results are not shown due to the lack of space but discussed.

On 6 ensembles of different sizes and different libraries of models, majority voting has shown to be inferior each time. It appears that naively averaging predictions is a simpler method and performs as well as weighted averaging them. We can explain this result by the fact that the validation information is already used during the ensemble construction, so the weights tuned on this validation set are over-fitted.

3.4 step 3 - Ensemble allocation for serving

Table 3 shows the performance in terms of prediction performance of 2 DNNs ensembles. The first one was generated with B=80 on CIFAR100 datasets it contains 7 DNNs. The second one with B=375 on the microfossils dataset containing 14 DNNs.

In addition to those benchmarks, we analyzed how DNNs are allocated. We observe in both ensembles, that when we increase the number of GPUs, the CPU is quickly not used anymore. Indeed, the CPU is only used for its greater memory capacity. We observe also that bigger models are often put alone in one GPU but smaller ones are co-localized, it is quite intuitive in terms of memory and GPU cores. Furthermore, when multiple models are in the same GPU, the batch size is chosen smaller. We observe also a bigger batch size on the CPU than on the GPU. Moreover, the second algorithm produces always a speed up but this speed is strongly dependent on the ensemble size and the available resources (from factor 10 to a few %) confirming the usefulness of the second algorithm. The ensemble of 7 models

#gpus (+1cpu)	7 DNNs on CIFAR100 algo1	algo1+algo2	14 DNNs on microfossiles algo1	algo1+algo2
0	11	13	20	28
1	35	79	40	59
2	40	411	150	199
3	290	788	167	213
4	325	789	176	232
5	375	791	198	244
6	355	791	203	252

Table 3. We benchmark the predictions/seconds of an ensemble of 7 models and another of 14 models. The first one was generated with B=80 and the second one B=375

shows that the increase of GPUs plateaus after 3 GPUs, in the second ensemble we do not see yet a plateau with 6 GPUs. Those results teach us that to serve efficiently an ensemble of DNNs we do not need systematically as much as GPUs as DNNs.

4 BASELINE COMPARISON

We report in figure 4 results of authors running until convergence multiple AutoML methods with diverse theoretical backgrounds [31], [46], [57]. We run the Auto-Keras framework version 1.0.12 with one single GPU and default settings, except the max_trials set to 10. We notice no improvement when max_trials is set to 20. We report our proposed workflow results and some intermediate scores over time.

The comparison between AutoML methods is known to be difficult because authors explore different search spaces, with different initial conditions, different data processing, and ensembles, however, we observe two main trends. The first block of rows AutoML methods converging fast in several GPU hours which often explore the neural architecture space sequentially on one GPU. The second block is evolution-based algorithms that converge later, their population-based approach allows to easily leverage GPU clusters. Asynchronous Hyperband is still not widely used.

As evolutionary methods, our method can benefit from high computing power. In comparison, asynchronous Hyperband does not require intermediate stopping of all GPUs which makes it more suitable to use GPU clusters compared to a sequence of generations. Also, in the large-scale case where the number of available GPUs is superior to the number of trials, Hyperband can run all trials at the same time, while genetics is limited by running all trials of the current generation.

Furthermore, Hyperband has low settings requirements. It uses early stopping which reduces the sensibility to the #trials/#epochs dilemma. Also, the exploration/exploitation of the hyperparameter space is balanced by stopping a fraction of less promising trials, so only the halving factor is needed. Evolutionary methods require much more initial settings such as the number of generations, the crossover operation, the mutation operation. They make this algorithm sensitive to the initial choices and so the settings calibrated for an application cannot be suited for a new application.

We try our best to do a fair comparison by using the same dataset. However, different data processing, different hardware, different initial settings can have an important impact on experimental results. We do not perform cutout and yet it seems effective processing on CIFAR100. Finally, we recall that with the given time horizon of 36GPU/hours visible in figure 5 and figure 6, Hyperband+SMOB greedy has not yet converged.

Method	#GPU	hours	Cum h	GPU name	#DNNs	Test(%)
RSPS [35]	1	2	2	GTX1080TI	1	52.31±5.77
DARTS-V1 [39]	1	3	3	GTX1080TI	1	15.03±0.00
DARTS-V2	1	10	10	GTX1080TI	1	15.03±0.00
GDAS [13]	1	9	9	GTX1080TI	1	71.34±0.34
SETN [12]	1	10	10	GTX1080TI	1	58.86±0.06
ENAS [44]	1	4	4	GTX1080TI	1	13.37±2.35
Auto-Keras [25]	1	2	2	Tesla V100	1	69.57±0.53
LSE [46]	250	264	66K	?	?	77.0
CNN-GA [57]	3	320	960	GTX1080TI	1	77.97
CNN-GA+cutout [57]	3	320	960	GTX1080TI	1	79.47
Ours with B=320	6	6	36	Tesla V100	4	72.39
Ours with B=320	6	24	144	Tesla V100	18	77.35
Ours with B=320	6	144	864	Tesla V100	34	79.44

Table 4. The comparison between AutoML algorithms in terms of the classification accuracy (%) and GPU hours on CIFAR100 benchmark dataset. The "?" mention means the information is missing in the paper. Column from left to right are: the name of the method, the number of GPUs used, duration of the algorithm (hours), cumulated time, GPU name, number of models (1=no ensembling) , mean test accuracy

5 FUTURE WORKS

To speed up the inference service of an ensemble, they are two ways, either running SMOBF with a lower budget to build another light ensemble or running post-training network optimization on each DNN. Post-training optimization is an active field of research, such as weights pruning [32] and weights quantization [20], [41]. All these methods may have a low or no impact on the accuracy.

In those experiments, we use the Tensorflow inference engine (".pb" file format), which is both compatible with GPUs and CPUs. Some inference engine frameworks perform post-training optimization and platform-specific optimizations such as OpenVINO [15] for Intel CPUs and TensorRT [11] for Nvidia GPUs.

Those lines of research should again increase the effectiveness and popularize the automatic construction of a heterogeneous ensemble of DNNs with a smart allocation strategy.

CONCLUSION

Due to the increasing number of new Deep Learning applications and datasets, Auto Machine Learning (AutoML) methods are an important line of research. We propose an AutoML workflow capable to tune, train, ensembling, and deploy DNNs automatically but that runs a heavy workload at each stage. We aim to fill the gap between Machine Learning researches, the new GPU clusters, and the end-user application quality of service. To go toward this direction, we formulate the problems by aiming at the accuracy, the inference speed, and the flexibility of the underlying heterogeneous infrastructure.

First, we presented the experimental results demonstrating that asynchronous Hyperband is suitable for parallelism and generates the best library of models to ensemble them. We then propose a new Ensemble Selection strategy that allows controlling the final ensemble computing cost of heterogenous DNNs. When the budget is relaxed, our algorithm offers high and robust accuracy compared to other AutoML workflows. Finally, we propose a solution to the complex allocation problem of DNNs into GPUs to democratize heterogeneous ensembles even if the number of DNNs is larger than the number of GPUs.

The history of Machine Learning is correlated to the available computing power. Since the emergence of multi-core processors in the 2000s allowed to stride from simple statistical models (e.g., decision tree) to machine learning based on ensemble (e.g., Random Forest). Then, GPGPU allows to stride from non-deep machine learning using a few cores to deep learning using hundreds of cores. GPU clusters are undoubtedly the dawn of a new era for future deep learning methods such as AutoML with ensembling.

Acknowledgement. We would like to thank TotalEnergies SE and its subsidiaries for allowing us to share this material and make available the needed resources.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 265–283.

[2] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* 13 (2012), 281–305. http://dblp.uni-trier.de/db/journals/jmlr/jmlr13.html#BergstraB12

[3] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2546–2554. http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

[4] Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. 2005. Diversity Creation Methods: A Survey And Categorisation. *Information Fusion* 6 (03 2005), 5–20. https://doi.org/10.1016/j.inffus.2004.04.004

[5] Rich Caruana, Art Munson, and Alexandru Niculescu-Mizil. 2006. Getting the Most Out of Ensemble Selection. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 828–833. https://doi.org/10.1109/ICDM.2006.76

[6] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. 2004. Ensemble Selection from Libraries of Models. In *Proceedings of the Twenty-First International Conference on Machine Learning* (Banff, Alberta, Canada) *(ICML '04)*. Association for Computing Machinery, New York, NY, USA, 18. https://doi.org/10.1145/1015330.1015432

[7] María V. Sainz de Cea, David Gruen, and David Richmond. 2021. Pneumoperitoneum Detection In Chest X-Ray By A Deep Learning Ensemble With Model Explainability. In *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)*. 1637–1641. https://doi.org/10.1109/ISBI48211.2021.9434122

[8] Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, and Hod Lipson. 2018. Autostacker: A Compositional Evolutionary Learning System. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Kyoto, Japan) *(GECCO '18)*. Association for Computing Machinery, New York, NY, USA, 402–409. https://doi.org/10.1145/3205455.3205586

[9] François Chollet. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), 1800–1807.

[10] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. 2016. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[11] Pooya Davoodi, Chul Gwon, Guangda Lai, and Trevor Morris. 2019. "TensorRT inference With TensorFlow". GPU Technology Conference.

[12] Xuanyi Dong and Yi Yang. 2019. One-Shot Neural Architecture Search via Self-Evaluated Template Network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[13] Xuanyi Dong and Yi Yang. 2019. Searching for a Robust Neural Architecture in Four GPU Hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[14] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 2962–2970. http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf

[15] Yi Ge and Monique Jones. 2018. Inference With Intel. AI DevCon 2018.

[16] Hui Guan, Laxmikant Kishor Mokadam, Xipeng Shen, Seung-Hwan Lim, and Robert Patton. 2020. FLEET: Flexible Efficient Ensemble Training for Heterogeneous Deep Neural Networks. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 247–261. https://proceedings.mlsys.org/paper/2020/file/ed3d2c21991e3bef5e069713af9fa6ca-Paper.pdf

[17] Antonio Gulli and Sujit Pal. 2017. *Deep learning with Keras.* Packt Publishing Ltd.

[18] Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michèle Sebag, Alexander Statnikov, Wei-Wei Tu, and Evelyne Viegas. 2019. *Analysis of the AutoML Challenge Series 2015–2018.* Springer International Publishing, Cham, 177–219. https://doi.org/10.1007/978-3-030-05318-5_10

[19] Shahzeb Haider, Adnan Akhunzada, Iqra Mustafa, Tanil Bharat Patel, Amanda Fernandez, Kim-Kwang Raymond Choo, and Javed Iqbal. 2020. A Deep CNN Ensemble Framework for Efficient DDoS Attack Detection in Software Defined Networks. *IEEE Access* 8 (2020), 53972–53983. https://doi.org/10.1109/ACCESS.2020.2976908

[20] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[22] Matthew Hoffman, Eric Brochu, and Nando de Freitas. 2011. Portfolio Allocation for Bayesian Optimization. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence* (Barcelona, Spain) *(UAI'11)*. AUAI Press, Arlington, Virginia, USA, 327–336.

[23] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2261–2269. https://doi.org/10.1109/CVPR.2017.243

[24] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*. Springer, 507–523.

[25] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-Keras: An Efficient Neural Architecture Search System. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Anchorage, AK, USA) *(KDD '19)*. Association for Computing Machinery, New York, NY, USA, 1946–1956. https://doi.org/10.1145/3292500.3330648

[26] Travis Johnston, Steven R. Young, David Hughes, Robert M. Patton, and Devin White. 2017. Optimizing Convolutional Neural Networks for Cloud Detection. In *Proceedings of the Machine Learning on HPC Environments* (Denver, CO, USA) *(MLHPC'17)*. Association for Computing Machinery, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/3146347.3146352

[27] Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations* (12 2014).

[28] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.

[29] Ludmila Kuncheva and Chris Whitaker. 2003. Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy. *Machine Learning* 51 (05 2003), 181–207. https://doi.org/10.1023/A:1022859003006

[30] Erin LeDell and Sebastien Poirier. 2020. H2O AutoML: Scalable Automatic Machine Learning. *7th ICML Workshop on Automated Machine Learning (AutoML)* (July 2020). https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf

[31] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, Cong Hao, and Yingyan Lin. 2021. {HW}-{NAS}-Bench: Hardware-Aware Neural Architecture Search Benchmark. In *International Conference on Learning Representations*. https://openreview.net/forum?id=_0kaDkv3dVf

[32] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning Filters for Efficient ConvNets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rJqFGTslg

[33] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. 18, 1 (1 2017), 6765–6816.

[34] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. 2020. A System for Massively Parallel Hyperparameter Tuning. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 230–246. https://proceedings.mlsys.org/paper/2020/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf

[35] Liam Li, Mikhail Khodak, Nina Balcan, and Ameet Talwalkar. 2021. Geometry-Aware Gradient Algorithms for Neural Architecture Search. In *International Conference on Learning Representations*. https://openreview.net/forum?id=MuSYkd1hxRP

[36] Liam Li and Ameet Talwalkar. 2020. Random Search and Reproducibility for Neural Architecture Search. In *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference (Proceedings of Machine Learning Research, Vol. 115)*, Ryan P. Adams and Vibhav Gogate (Eds.). PMLR, 367–377. http://proceedings.mlr.press/v115/li20c.html

[37] Yuansong Liao and John Moody. 1999. Constructing Heterogeneous Committees Using Input Feature Grouping: Application to Economic Forecasting. (1999), 921–927.

[38] Richard Liaw. 2019. A Guide to Modern Hyperparameters Turning Algorithms. In *PyData Los Angeles*.

[39] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1eYHoC5FX

[40] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (Carlsbad, CA, USA) *(OSDI'18)*. USENIX Association, USA, 561–577.

[41] Deniz Oktay, Johannes Ballé, Saurabh Singh, and Abhinav Shrivastava. 2020. Scalable Model Compression by Entropy Penalized Reparameterization. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=HkgxW0EYDS

[42] Christopher Olston, Fangwei Li, Jeremiah Harmsen, Jordan Soyke, Kiril Gorovoy, Li Lao, Noah Fiedel, Sukriti Ramesh, and Vinu Rajashekhar. 2017. TensorFlow-Serving: Flexible, High-Performance ML Serving. Workshop on ML Systems at NIPS 2017.

[43] Sudipta Pathak, Xingyu Cai, and Sanguthevar Rajasekaran. 2018. Ensemble Deep TimeNet: An Ensemble Learning Approach with Deep Neural Networks for Time Series. In *2018 IEEE 8th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*. 1–1. https://doi.org/10.1109/ICCABS.2018.8541985

[44] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameters Sharing. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 4095–4104. https://proceedings.mlr.press/v80/pham18a.html

[45] Lutz Prechelt. 1998. Early Stopping-But When?. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. Springer-Verlag, Berlin, Heidelberg, 55–69.

[46] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. 2017. Large-Scale Evolution of Image Classifiers. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, NSW, Australia) *(ICML'17)*. JMLR.org, 2902–2911.

[47] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc., 2951–2959. https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf

[48] Peter Sollich and Anders Krogh. 1995. Learning with ensembles: How overfitting can be useful. 8 (01 1995), 190–196.

[49] Gang Sun, Jianqiao Liu, Wei Mengxue, Wang Zhongxin, Zhao Jia, and Guan Xiaowen. 2020. An Ensemble Classification Algorithm for Imbalanced Text Data Streams. In *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*. 1073–1076. https://doi.org/10.1109/ICAICA50127.2020.9182576

[50] Qingqiang Sun and Zhiqiang Ge. 2021. Deep Learning for Industrial KPI Prediction: When Ensemble Learning Meets Semi-Supervised Data. *IEEE Transactions on Industrial Informatics* 17, 1 (2021), 260–269. https://doi.org/10.1109/TII.2020.2969709

[51] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 6105–6114. https://proceedings.mlr.press/v97/tan19a.html

[52] Chris Thornton, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. 2012. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. *KDD* (08 2012). https://doi.org/10.1145/2487575.2487629

[53] Grigorios Tsoumakas, Ioannis Partalas, and I. Vlahavas. 2008. A Taxonomy and Short Review of Ensemble Selection. *ECAI 2008, Workshop on Supervised and Unsupervised Ensemble Methods and Their Applications* (01 2008).

[54] D. H. Wolpert and W. G. Macready. 1997. No Free Lunch Theorems for Optimization. *Trans. Evol. Comp* 1, 1 (4 1997), 67–82. https://doi.org/10.1109/4235.585893

[55] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated Residual Transformations for Deep Neural Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5987–5995. https://doi.org/10.1109/CVPR.2017.634

[56] Tianhao Xu. 2020. "Deep into Triton Inference Server: BERT Practical Deployment on NVIDIA GPU". GPU Technology Conference.

[57] Sun Yanan, Xue Bing, Zhang Mengjie, Yen Gary G., and Lv Jiancheng. 2020. Automatically Designing CNN Architectures Using the Genetic Algorithm for Image Classification. *IEEE Transactions on Cybernetics* 50, 9 (2020), 3840–3854. https://doi.org/10.1109/TCYB.2020.2983860

[58] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide Residual Networks. In *Proceedings of the British Machine Vision Conference 2016 (BMVC 2016)*, Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith (Eds.). BMVA Press. http://www.bmva.org/bmvc/2016/papers/paper087/index.html

[59] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. 2016. Accelerating Very Deep Convolutional Networks for Classification and Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 38, 10 (Oct. 2016), 1943–1955. https://doi.org/10.1109/TPAMI.2015.2502579

A EXPERIMENTAL DATA SET AND HYPERPARAMETER SETTINGS

This section describes machine learning experiments for reproducibility purposes.

A.1 The two datasets used

The CIFAR100 dataset. CIFAR100 [28] consists to 60,000 32x32 RGB images in 100 classes. For each class, there are 580 training images, 20 validation images and 100 testing images.

The Microfossils dataset. Microfossils are extremely useful in age dating, correlation, and paleo-environmental reconstruction to refine our knowledge of geology. Microfossil species are identified and counted on large microscope images and thanks to their frequencies we can compute the date of sedimentary rocks.

To do reliable statistics, a big number of objects needs to be identified. That is why we need deep learning to automate this work. Today, thousands of fields of view (microscopy imagery) need to be shot for 1 rock sample. In each field of view, there are hundreds of objects to identify. Among these objects, there are non-fossils (crystals, rock grains, etc...) and others are fossils that we are looking for to study rocks.

Our dataset contains 91 classes of 224x224 RGB images (after homemade preprocessing). Microfossils are calcareous objects took with polarized light microscopy. The classes are unbalanced, we have from 50 images to 2500 images by class, with a total of 32K images in all the datasets. The train/validation/test split is as following: 72% 8% 20%. The F1 score was used and labeled as 'accuracy' on all benchmarks.

A.2 Hyperparameter configuration space

Table 5 shows all hyperparameters properties in this workflow. We use ResNet-based architectures due to their simplicity to yield promising and robust models on many datasets. We explore different residual block versions: "V1", "V2" [21] and "next" [55]. Regarding the optimization method, we use Adam optimizer [27] due to its well-known performance and its lower learning rate tuning requirement.

The simplicity of the ResNet architecture makes this work easy to test by a scientist on its image dataset [58]. Exploring other convolutional block type like VGG [59], Xception [9], DenseNet [23] and EfficientNet [51] are potential improvement which could increase the degree of liberty in DNNs construction and improve again the accuracy of ensembles.

The CIFAR100 dataset contains 32x32 images while usually ResNet is adapted to be used on ImageNet (224x244 images). Those different resolutions need some adaptation. Therefore, on the CIFAR100 case, the first convolutional network is replaced from the 7x7 kernel size with a stride of 2, to a 3x3 kernel size with a stride of 1. With equivalent settings, our CNN framework produces nearly the same number of weights between the CIFAR100 and microfossils dataset, but the time complexity is factor 11 different because of different signal resolutions flowing through the layers.

Category	Name	Type	Range
Optimization	Learning rate	Continuous	[0.001; 0.01]
	Batch size	Discrete	[8; 48]
	L2 regularization factor	Continuous	[0; 0.1]
NN architecture	Convolution type	Categorical	{v1,v2,next}
	Activation function	Categorical	{tanh,relu,elu}
	Number of filters in the first convolutional layer	Discrete	[32; 128]
	Multiplier of filters in the 4 blocks	Discrete	[32; 128]
	Number of convolutional block in the first stage	Discrete	[1;11]
	Number of convolutional block in the second stage	Discrete	[1;11]
	Number of convolutional block in the third stage	Discrete	[1;11]
	Number of convolutional block in the fourth stage	Discrete	[1;11]
Data augmentation	Max zoom	Continuous	[0; 0.6]
	Max translation	Continuous	[0; 0.6]
	Max shearing	Continuous	[0; 0.3]
	Max channel shifting	Continuous	[0; 0.3]
	Max rotation measured in degrees	Discrete	[0; 90]

Table 5. The hyperparameter space experimented based on the ResNet neural architecture framework