WILDA: Wide Learning of Diverse Architectures for Classification of Large Datasets

Rui P. Cardoso[1], Emma Hart[4], David Burth Kurka[2], and Jeremy Pitt[3]

[1] Imperial College London, rui.cardoso@imperial.ac.uk
[2] d.kurka@imperial.ac.uk
[3] j.pitt@imperial.ac.uk
[4] Edinburgh Napier University, e.hart@napier.ac.uk

Abstract. In order to address scalability issues, which can be a challenge for Deep Learning methods, we propose Wide Learning of Diverse Architectures — a model that scales horizontally rather than vertically, enabling distributed learning. We propose a distributed version of a quality-diversity evolutionary algorithm (MAP-Elites) to evolve an architecturally diverse ensemble of shallow networks, each of which extracts a feature vector from the data. These features then become the input to a single shallow network which is optimised using gradient descent to solve a classification task. The technique is shown to perform well on two benchmark classification problems (MNIST and CIFAR). Additional experiments provide insight into the role that diversity plays in contributing to the performance of the repertoire.

Keywords: Diversity · MAP-elites · Machine Learning · Ensemble

1 Introduction

Deep Learning (DL) techniques have revolutionised the field of Machine Learning (ML) in recent years, providing high-performing neural network models for solving a number of complex tasks such as Computer Vision (CV) and Natural Language Processing (NLP). However, the best performing networks often have very large numbers of layers and millions of parameters, exhibiting a type of scaling that has mostly been vertical. Some DL algorithms run for days [19, 18]. Amongst other things, this raises important questions regarding the environmental sustainability of DL methods [16]. As an alternative, *distributed* learning techniques have been proposed as a way of enacting *horizontal* scaling [22]. Often such methods involve training an *ensemble* of networks whose results can be aggregated in some manner. This has the obvious advantage that the ensemble can be trained in parallel, with the added benefit not only that ensembles are often much more accurate than the individual classifiers they encompass [3], but also that the individual members of ensembles do not themselves have to be complex or high-performing [6].

In this paper, we propose an algorithm called WILDA (Wide Learning of Diverse Architectures) that integrates ML and Evolutionary Computation (EC)

to train a classification model applicable to the type of datasets used in DL. WILDA falls under the broad paradigm of Wide Learning (WL) [23, 12, 15] as a viable complementary alternative to DL. The approach first trains an ensemble of low-complexity artificial neural networks (ANNs), each of which extracts a feature vector from each data point. In a second phase, the features extracted from the ensemble are then aggregated in a single shallow model to solve the classification task. It is well understood that a necessary condition for good ensemble performance is that its members are both *accurate* and *diverse* [3]: although the former can be addressed through judicious training of each member in the ensemble using standard ML methods, the latter provides more challenge. To address this, we turn to a relatively recent class of algorithms from EC known as *quality diversity* (QD) algorithms [14]. These algorithms return an archive of diverse, high-quality solutions to a problem in a single run, where diversity is defined by the user with respect to features of interest. Here, we exploit the QD approach to generate an ensemble of shallow neural networks which are *architecturally diverse* yet each optimised with respect to the classification task. The features extracted from each network in the first phase are then used in the second phase to train a single shallow network to output the final classification. The resulting architecture is easily parallelised and quickly trained, unlike a typical DL model, which often uses a complex model on a single machine. Running the full method takes around 50 minutes.

The major contribution of the work is to describe the novel WILDA algorithm, which exploits the latest developments in EC to generate an ensemble of networks which are diverse w.r.t their *hyperparameters* and uses ML methods to optimise their *parameters*. The main goal is to develop a general model that can be easily applied to large datasets without specialisation and executed on standard computer architectures. As such, we do not expect to obtain state-of-the-art results that compete with methods that are specialised to specific datasets and extensively tuned. Instead, we aim to show that the method produces reasonable results without any specialisation to a dataset and gain insight into the factors that influence the performance of the method.

2 Background

Deep neural networks typically consist of large numbers of stacked layers of neurons. Layers can be different *types*, for example convolutional, dense, or pooling [8], and each layer can have a variable number of neurons. Network design is thus an optimisation problem, and as such, Evolutionary Algorithms (EAs) have naturally been applied to this task. For example, the evolutionary-algorithm-based Automatic Evolving CNN (AE-CNN) algorithm [19] achieves high performance on the CIFAR-10 dataset, however consuming 22 days of computational time on three Graphics Processing Units (GPUs). Using a surrogate model with an EA considerably reduces this time [18], but still requires 8 GPU days. Genetic Programming (GP) is employed by Suganuma et al. [17] to evolve a CNN, again

for image processing, showing promising results although resulting in networks with over 1.5 million learnable weights.

The above approaches all use an EA to evolve a single high-performing network. In contrast, *ensemble methods* consist of building a set of models and then aggregating their outputs to form a collective prediction, rather than relying on the predictions made by a single model [3]. Although the use of ensembles is well known in classical ML, it is less used in DL. An ensemble method for image classification using deep networks [5] has been shown to provide excellent performance; the networks within the ensemble contain up to 152 layers each, contrasting to classical ML ensemble approaches which tend to consist of multiple but low-complexity models (e.g. Random Forest [1]).

The *diversity* of an ensemble is crucial to its performance [3] and hence is a key factor in its design. The relatively recent paradigm of *quality diversity* (QD) [14] algorithms within the EA field — which aim to find a maximally diverse but high-performing collection of individuals for a given optimisation task — thus appears ideally suited to this goal. Specifically of interest to this paper is the ability of QD methods to produce a diverse repertoire of optimised solutions in a single run. The methods have received much attention in the Evolutionary Robotics (ER) literature to evolve behaviourally or structurally diverse robots [2, 10] but much less attention elsewhere[5]. A single example within ML [20] uses a QD algorithm (Novelty Search) as an approach to unsupervised feature learning in a method that continually accumulates features that make novel discriminations amongst a training set (with no regard to the classification task), showing that after generating approximately 3000 features, a simple two-layer network performed well compared to other shallow architectures. In this paper, for the first time we propose to use the MAP-Elites algorithm [10] to evolve a set of optimised architectures that are diverse w.r.t to their structure, as explained in the next sections.

3 Methodology

WILDA (Wide Learning of Diverse Architectures) uses a two-step approach to classification in which a diverse ensemble of shallow ANNs is trained in the first step (section 3.2) and the features extracted by the ensemble are used in the second step (section 3.4) to train a small feedforward ANN to provide the final result. This is based on the conjecture that a set of diverse features can be extracted from an architecturally diverse repertoire of ANNs, which can then be used to efficiently train a single shallow network in the aggregation phase. The process can be summarised as follows:

1. Apply MAP-Elites to discover a *set* of neural network classifiers which are architecturally diverse, each optimised for accuracy using gradient descent (section 3.1, algorithm 2)

[5] with the exception of a handful of papers in the combinatorial optimisation domain, for example [21]

2. Repeat step 1 r times, and then merge the r sets into a single archive (section 3.3)
3. Extract a single feature from each network in the merged archive for each data point (section 3.2)
4. For each data point, concatenate the features extracted in step 3 to form a single input vector (section 3.4)
5. Train a single shallow network to output the desired classification using the input vectors from step 4 as input (algorithm 5)

WILDA uses a hybrid method that combines an evolutionary approach (MAP-Elites [10]) with a traditional ML approach (gradient descent) for training each ANN (Algorithm 1). MAP-Elites (Multi-dimensional Archive of Phenotypic Elites) explores a low-dimensional projection of the space of *hyperparameters* which describe the architecture of the networks and returns an archive of structurally diverse networks. A gradient descent procedure optimises the *parameters* of each ANN discovered. The MAP-Elites algorithm can be run simultaneously on multiple nodes, resulting in `n_nodes` archives at the end of the feature-extraction phase. These archives are merged before running the aggregation phase, which provides the final classification. We first give an overview of the MAP-Elites algorithm before describing each phase in detail.

Algorithm 1 WILDA algorithm (high-level view)

```
procedure WILDA
    for n = 1 → n_nodes do
        MAP_n ← MapElites()                          ▷ generate n archives of diverse networks
    end for
    mergedMap ← merge(MAPs)                          ▷ merge n archives into a single archive
    features ← extractFeatures(data, mergedMap)      ▷ for each data point, extract a feature vector from each network in the archive
    aggregatedModel ← trainShallowNetwork(features)  ▷ train single network to classify data
end procedure
```

3.1 MAP-Elites

Fundamentally different to a traditional search algorithm, the MAP-Elites algorithm provides a holistic view of how high-performing solutions are distributed throughout a feature space [10]. The method creates an archive of high-performing solutions at each point in a space defined by dimensions of variation chosen by a user, according to characteristics of a solution that are of interest. The resulting archive enables the user to gain specific insight into how combinations of characteristics of solutions correlate with performance. As the approach encourages

diversity, it has often been shown to be more capable of fully exploring a search space, outperforming state-of-the-art search algorithms which are given a single objective, and can be particularly helpful in overcoming deception [14].

The standard algorithm is given in Algorithm 2. This is adapted for our purposes as follows. A solution consists of a PyTorch [13] representation of an ANN. We select three dimensions to characterise an architecture, namely the number of convolutional layers, the number of dense layers, and the maximum size (number of outputs) of any dense layer in the network, which together comprise the `featureDescriptor`. The algorithm begins by generating random solutions which are mapped to a grid that is discretised in each dimension into a fixed number of cells (representing possible values of each feature). The grid thus contains $|C| \times |D| \times |S|$ cells, where these values represent the total number of values permitted for the convolutional, dense and size dimensions, respectively. Following an initialisation phase, solutions are randomly selected from the grid, after which a *variation* operator is applied to generate new solutions. Child solutions are evaluated according to a performance metric and then mapped back to the grid according to their descriptor: a child solution replaces an existing solution in any cell if it is better according to its performance metric or may simply occupy an empty cell. The search process aims to fill the entire grid with solutions, each of which represents the best performing solution for a given feature descriptor. The precise implementation of each of the above steps is described in the next section.

Algorithm 2 MAP-Elites Algorithm, taken directly from [10]

procedure MAP-ELITES ALGORITHM
  $(\mathcal{P} \leftarrow \emptyset, \mathcal{X} \leftarrow \emptyset)$
  for $iter = 1 \rightarrow max\_iterations$ **do**
    if $iter < initialise\_iterations$ **then**
      $x' \leftarrow$ randomSolution()
    else
      $x \leftarrow$ randomSelection($\mathcal{X}$)
      $x' \leftarrow$ randomVariation(x)
    end if
    $b' \leftarrow$ featureDescriptor(x')
    $p' \leftarrow$ performance(x')
    if $\mathcal{P}(b') = \emptyset$ or $\mathcal{P}(b') < p'$ **then**
      $\mathcal{P}(b') \leftarrow p'$
      $\mathcal{X}(b') \leftarrow x'$
    end if
  end for
  return feature-performance map ($\mathcal{P}$ and $\mathcal{X}$)
end procedure

3.2 Feature-Extraction Phase

During the extraction phase, MAP-Elites attempts to find a set of diverse ANN architectures, each of which is optimised on a subset of the data towards solving a classification task of interest. At the end of this phase, a feature vector is extracted from each network for each data point, corresponding to the output of the second-to-last layer of each network as explained below.

Network Representation An individual uses a list representation to describe a variable-length sequence of convolutional layers followed by a variable-length sequence of dense layers. Each layer has a random number of neurons, selected from a list of discrete values. Each dense layer uses a hyperbolic tangent [11] activation function. The last hidden layer is designated as the *feature layer*: the output of this layer is a binary vector which represents a feature extracted by the network to be used in the second phase (section 3.4). As a result, this layer always has a fixed number of neurons, *feature_size*, set by the user according to the desired size of the feature vector. Finally, an output layer is added which provides the classification of the data point.

Variation Operators Three new individuals are generated at each iteration by the *crossover* and *mutation* operators. Two children are generated by applying *crossover* to a pair of randomly selected individuals. The third child is generated by applying *mutation* to a single randomly selected parent. Crossover randomly picks two individuals, selects random crossover points among their dense layers, and swaps them accordingly. Mutation randomly picks a mutation point among the dense layers of an individual and either adds or removes a layer at that position. Note that crossing over two sequences of dense layers or removing a layer from such a sequence will, in the general case, require changing the input and/or output sizes of layers at the crossover/mutation point; when adding a layer at a mutation point, its size is given by the output and input sizes of the previous and following layer, respectively. For simplicity, crossover and mutation only operate over the dense layers of an ANN; since convolutional layers tend to have a non-decreasing number of channels, these operations would require modifying all layers beyond the crossover/mutation point, defeating their purpose [8]. However, as the convolutional layers generated in the initialisation process will be paired with different combinations of dense layers as a result of these two operations, this still ensures a diverse search process.

Performance Evaluation To evaluate each individual, the single network encoded by the individual is *trained* for a fixed number of iterations (`eval_iters`) on a sample training set using a standard gradient descent procedure which minimises cross-entropy loss [11]. Its classification accuracy is then calculated on a sample test set, and this value assigned as its fitness, as described in algorithm 3. The sample train and test sets are drawn randomly from the training data at each iteration; they are both 20% the size of the complete training data, which

encompasses 60000 examples in the two datasets we tested (MNIST and CIFAR-10). For this reason, each node uses 12000 examples at each evaluation and may therefore only ever have a partial view of the data required to solve the task.

3.3 Distribution of Computation

As described in the introduction, one of the goals of WL is to be able to distribute the computation over multiple nodes to enable the model to be run in parallel. One approach to achieving this would be to segment the $|C| \times |D| \times |S|$ grid into sub-partitions and run each sub-partition on a separate node. However, here we adopt an approach described in [4], which proposes a fully distributed implementation of MAP-Elites designed to be run on a robot swarm. In this approach, each node runs its own instance of the MAP-Elites algorithm 4. At the end of the extraction phase, all the maps returned are merged into a single map referred to in the QD literature as a global map of elites. In previous work [4], we evaluated multiple options for performing the merge step which inform our choice of two strategies:

1. merging without overlap: for each cell in the map, select the highest-performing ANN model found in that cell from any of the `n_nodes` individual maps returned
2. merging with overlap: for each cell in the map, return all of the ANNs found in that cell across all `n_nodes` maps. This means that a maximum of `n_nodes` $\times |C| \times |D| \times |S|$ neural networks is returned

In both cases, the maximum number of models passed to the learning phase via the global map is N, where N is the size of the map (i.e. $|C| \times |D| \times |S|$). When merging without overlap, the procedure returns a maximum of N networks, maximising diversity. On the other hand, the merge with overlap procedure can return $>> N$ networks. In this case, the procedure selects the top N networks according to their fitness metric. This strategy can return multiple networks which map to the same cell, therefore favouring the quality of solutions over their diversity.

Algorithm 3 Calculating the fitness of an individual neural network and adding it to the map of elites

procedure TRAIN_AND_EVAL(m, *sample_train*, *sample_test*, ME)
  $c, l, s \leftarrow$ architectural features of m
  train(m, *sample_train*)
  if $ME[c, l, s] = \emptyset$ **OR**
    accuracy(m, *sample_test*) $> ME[c, l, s]$.fitness **then**
    $ME[c, l, s] \leftarrow m$
  end if
end procedure

Algorithm 4 Training loop for each node in the extraction phase

create empty map of elites ME
draw *sample_train* and *sample_test* from training set $\mathcal{D}$
for *initial_size* **do**
  $m \leftarrow$ generate random ANN model
  train_and_eval(m, *sample_train*, *sample_test*, ME)
end for
for *extraction_epochs* **do**
  draw *sample_train* and *sample_test* from training set $\mathcal{D}$
  draw individuals x, y from map of elites ME
  $x', y' \leftarrow$ crossover(x, y)
  train_and_eval(x', *sample_train*, *sample_test*, ME)
  train_and_eval(y', *sample_train*, *sample_test*, ME)
  draw individual z from map of elites ME
  $z' \leftarrow$ mutate(z)
  train_and_eval(z', *sample_train*, *sample_test*, ME)
end for

3.4 Learning/Aggregation Phase

The learning phase uses the information learnt by the ANNs contained in the repertoire resulting from the first phase to train a single model to solve the classification task. This single model is a fixed-structure shallow ANN that has a single intermediate layer with `n_hidden_agg` neurons and a hyperbolic tangent activation function. The node where this model is trained is called the *root node*. Note that, even though there is a global merged repertoire at the end of the feature-extraction phase, as described in section 3.3, this repertoire contains only references to the models which were generated and trained in separate nodes and each of these models will still be running in its corresponding node.

Algorithm 5 Learning phase of the procedure for the **root node**

$all\_MEs \leftarrow$ gather_all_maps() ▷ Root node receives all repertoires
$global\_ME \leftarrow$ merge(all_MEs). ▷ Repertoires merged into a global map
send $global\_ME$ to the other nodes ▷ Global map known by all nodes
initialise model M
for *learning_epochs* **do**
  for batched *data* and *labels* **do**
    $all\_features \leftarrow$ gather_all_features($data$) ▷ Each node sends its feature vector extracted from *data*
    concatenate $all\_features$ into intermediate representation $\underline{f}$
    M.train_step $(\underline{f}, labels)$
  end for
end for

The phase begins with an extraction step: each data point in the training set is passed through each of the n networks contained in the merged map from the previous phase. This returns n binary vectors, each representing a feature (as described in section 3.2), which are concatenated to form the input layer of the new model. This model is then trained with a standard gradient descent procedure by minimising cross-entropy loss. Algorithm 5 shows pseudocode for the learning phase specific to the root node and algorithm 6 shows pseudocode for all nodes.

Algorithm 6 Learning phase of the procedure for all nodes (including root node)

```
send(ME, root)                                  ▷ Sends own repertoire to root node
global_ME ← receive(root)                       ▷ Receives global map from root node
own_models ← get_own_models(global_ME)
for learning_epochs do
    for batched data do
        own_features ← get_features(own_models, data)
        send(own_features, root)                ▷ Sends feature vector to root node
    end for
end for
```

4 Experiments

Experiments have been conducted to: (1) evaluate the performance of WILDA as a classifier on two datasets providing varying levels of challenge; (2) explore the effects of encouraging diversity vs. quality within an ensemble; (3) explore the influence of the size of the ensemble used to execute the centralised learning step.

Two well-known benchmark datasets are used: MNIST [8] and CIFAR-10 [7]. MNIST is a set of 60000 hand-written digits, while the CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. MNIST is known to be relatively straightforward for ANN architectures, while the latter poses a significant challenge to "off-the-shelf" models; state-of-the-art DL models for CIFAR-10 require significant customisation and tuning. Four sets of experiments are conducted as described below.

As a baseline for comparing the quality of the ensemble-based solutions from WILDA, we use a single shallow ANN, trained in a similar fashion to the aggregated single-layer model used in the learning phase of the algorithm (section 3.4). This ANN is the one that has achieved the best performance after a run of the feature-extraction phase 3.2. Note that this is not a true "baseline" in the sense that we do not choose a random or otherwise uninformed architecture, but rather one that has been found to be the best. This is because we wish to understand the benefits of the ensembles built by our diversity-driven approach;

outperforming individual neural networks with the highest fitness values is therefore a more interesting challenge. Out of interest, the shallow network trained with diverse features mentioned in section 2[20] achieves an accuracy of 98.75% on MNIST, while a shallow CNN is reported to obtain 75.86% on CIFAR-10 in [9].The relevant parameter values are set as per table 1, which lists all the parameter values that are used throughout the experiments. All experiments, including the runs of the feature-extraction phase to get the baseline results, are repeated 30 times in order to evaluate statistical significance. Two-tailed Mann-Whitney significance tests are applied to compare experimental results, and noted as significant if the resulting p-value is < 0.01.

Table 1: Parameter settings

Parameter	**Description**	**Value(s)**
`n_nodes`	Number of distributed nodes	8
`initialise_iterations`	Number of ANNs in the initial maps	20
`C`	set of possible # of convolutional layers	{1,2}
`D`	set of of possible # of dense layers	{2,3,4}
`S`	set of possible values for layer size	{100,110,120, 130,140,150,170}
`max_iterations`	Number of iterations in the extraction phase	30
`eval_iters`	Number of iterations in gradient-descent training in extraction phase	5
`feature_size`	Size of binary vector produced by last hidden dense layer in extraction phase	100
`aggregation_iters`	Number of iterations in the learning phase	10
`n_hidden_agg`	Number of neurons in the intermediate layer of the centralised model (learning phase)	50
`merge`	Strategy used to merge the maps of elites evolved by each node	*with/without overlap*
`n_models`	Maximum number of models to use in the learning phase ($\leq$ the size of the map)	48

4.1 Comparison of Different Merge Strategies

This set of experiments compares the two merge strategies (section 3.2) to understand how the construction of the merged archive impacts the performance of the aggregated model. For reference, a full run of WILDA with the parameters of table 1 takes around 50 minutes on the machine upon which the algorithm was tested. Recall that the two strategies represent different trade-offs between diversity and quality of the solutions. The size of the maps evolved by each node,

as well as the global map, is $2 \times 3 \times 8 = 48$; this is also the maximum number of ANN models used in the learning phase (`n_models`).

Table 2 presents the median test set accuracy for the two merge strategies and compares them to the baseline. Recall that the architecture of the baseline network is that of the best network found in a run of the feature-extraction phase, as explained earlier in this section. Both methods significantly outperform the baseline individual best network for both datasets ($p << 0.01$). There is no significant difference between the two merge strategies, however. A possible reason for this is that these two merge strategies actually lead to similar global maps of elites. On the one hand, each node might be finding high-performing ANNs in different regions of their individual maps, thus leading to few overlaps at a same cell when merging. On the other, it is possible that networks which are mapped to the same cell are still significantly diverse. Further investigation is required to answer these questions. We also include for interest the result obtained by Szerlip et al. [20] from first evolving 3000 divergent discriminative features, but note that the training procedure used in that paper differs from ours, which runs a two-phase procedure that first trains on a small training set before shifting to the full example set, using a single-layer network to classify. Our evolved ensemble of features obtained from 48 diverse networks outperforms both the single high-performing learner and the previously obtained result.

Table 2: Median test set accuracy for the two merge strategies considered

	MNIST	**CIFAR-10**
Baseline	0.9899	0.646
Merge without overlap	0.99175	0.6982
Merge with overlap	0.9919	0.6983
Divergent Discriminative Feature Accumulation [20]	0.9875	*n/a*

4.2 Investigating the role of architectural diversity

The global map of elites which is constructed from the individual maps and used in the aggregation phase is essentially an ensemble of the best ANNs found for different types of architecture. This naturally raises the question of how useful it is to promote *architectural diversity* amongst the networks in the ensemble, and how the performance of such a diverse ensemble compares with ensembles which do not have *architectural* diversity, but are diverse in terms of their optimised *weights* due to training on different samples of the dataset. Thus, we compare the performance of architecturally diverse ensembles evolved by WILDA with two kinds of ensembles that lack architectural diversity:

– an ensemble of networks in which every individual has the *best* architecture found in the extraction phase but is trained using a different sample of the training data

– an ensemble of networks in which each individual has the *worst* architecture found in the extraction phase but is trained using a different sample of the training data

Table 3 shows the accuracy results for ensembles trained with a fixed set of architectures (the best and worst architectures found in the extraction phase). All differences are significant compared to both the baseline and to both merge strategies ($p << 0.01$). It is clear that ensembles that do not have architectural diversity perform significantly worse than the results obtained by WILDA on both datasets. They also show that the fixed-architecture ensembles perform significantly worse than the baseline case. This is perhaps surprising given that the baseline case uses a feature vector from a single network obtained from the extraction phase. It appears that combining the predictions made by the best architecture trained on different subsets of the data leads to overall poorer performance than training an individual network on all of the data. This could be a particular characteristic of our procedure for aggregating the features extracted from the data by the ANNs in the ensemble during the learning phase, as described in section 3.4. However, it is a clue that the ensembles may be accumulating and reinforcing prediction errors when there is a lack of architectural diversity, i.e. errors made on the same data or on data from which similar features have been extracted. The performance of an ensemble depends on the *diversity of errors* made by each of its learners [3]; the results of this section enable us to suggest that promoting architectural diversity among the ANNs in the repertoires built during the extraction phase of the algorithm drives diversity of features extracted from the data and diversity of prediction errors, which in turn leads to higher test set accuracy. This observation is of the utmost importance in informing future research into how to increase the performance of the diverse ensembles evolved by WILDA.

Table 3: Median test set accuracy for ensembles without architectural diversity

	MNIST	**CIFAR-10**
Ensemble of instances of the best architecture	0.98685	0.62835
Ensemble of instances of the worst architecture	0.97815	0.557

4.3 Investigation of the influence of ensemble size

After constructing the merged map, one question that arises is how to use it to solve the task. We can simply pick the single best-performing architecture, as per the baseline case, or use an ensemble selected from the map. This raises the question of how many networks to include in the ensemble. This set of experiments assesses the relevance of fine-tuning the number of ANN models used in the ensemble by comparing the test set performance of ensembles of

different sizes. We vary `n_models` (the size of the ensemble) in the range {10, 20, 30, 40, 48}, selecting the best `n_models` ANNs in each experiment. All the other parameters are fixed as per table 1.

Figure 1 presents the performance results when only the `n_models` top-performing models from the global map of elites are used in the learning phase. In all cases the algorithm significantly outperforms the baseline and changing this parameter only produces small variations in accuracy. For MNIST, using only 10 models outperforms all other cases. The difference is statistically significant when compared with cases using 30 or more models. These observations suggest that using fewer models in the learning phase leads to better performance on the MNIST dataset. This could be because the simplicity of MNIST leads to smaller error diversity among different learners, which would cause the reinforcement of errors in larger ensembles. On the other hand, using only 10 models leads to significantly worse performance on the CIFAR-10 dataset than all other cases. This disparity in the observations for both datasets suggests that the choice of number of models that brings optimal performance is domain-dependent and must therefore be fine-tuned to the problem being tackled.

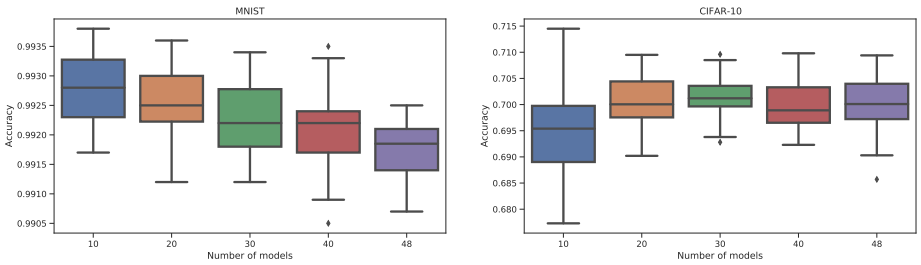

Fig. 1: Test set accuracy for each number of models added to the ensemble in the learning phase

4.4 Search Space Illumination

Figure 2 shows an example of how the extraction phase of WILDA, which runs a version of MAP-Elites, can *illuminate* the search space of architectures. For each combination of number of dense layers and maximum size of any dense layer, the diagram shows the fitness (accuracy on sample test set) — averaged out along the other dimension, which is the number of convolutional layers — of the best-performing ANNs on the CIFAR-10 dataset which map to that cell after merging all individual maps into a global map *without overlap* (section 3.2). Note that more runs of the extraction phase would be required in order to draw conclusions about which architecture leads to the best performance.

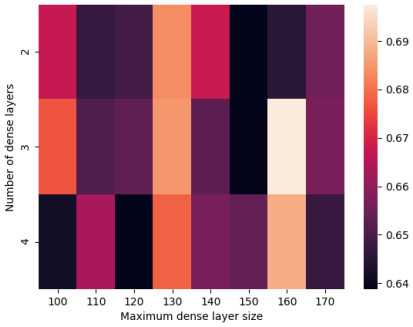

Fig. 2: Illuminating the architectures' search space for the CIFAR-10 dataset - the colour shading indicates the accuracy of the best network found for each cell

5 Conclusions and Future Work

We have presented an innovative diversity-driven distributed algorithm dubbed WILDA for training classification models within a WL paradigm. The basic idea is to extract a representation for the input in a way that scales horizontally rather than vertically. The algorithm first trains a repertoire of *architecturally diverse* ANNs in parallel: each node constructs a repertoire of high-performing, architecturally diverse networks, accessing different subsets of the data, which are then merged together into a global map of elite networks, each of low complexity. The features extracted by each network from the data are then aggregated and fed to a centralised model which will solve the classification task. The approach relies on the assumption that networks constructed with diverse architectures and trained on diverse samples of data will extract diverse features from a dataset, ultimately improving classification.

The overriding goal of the paper is to show that a general method that is easily distributed and does not require either vast amounts of computational power or expert knowledge to design a network is capable of reasonable performance. Experimental results show that this technique performs well on the MNIST and CIFAR-10 datasets and that a diverse ensemble performs better than the best individual model found in the extraction phase. We also show that architectural diversity is key to improving performance: ensembles using fixed architectures have been found to perform worse than an individual model, likely due to the accumulation and reinforcement of the same kind of errors. The results suggest that architectural diversity promotes error diversity, which in turn increases the performance of the ensembles evolved by the algorithm. Note that only basic tuning of the method was conducted and there remains considerable scope for improvement, for instance in investigating greater ranges of values for each of the dimensions over which diversity is defined, or adding additional dimensions

of diversity within MAP-Elites; both would lead to an increase in the size of the map and therefore a larger potential space of networks.

We note that the algorithm can also be a useful tool for exploring and illuminating the space of hyperparameters in that it exposes correlations between the characteristics of different architectures and their performance. Moreover, we believe that the WL paradigm can complement current DL techniques, particularly when addressing issues of scaling and distribution. Finally, further work will focus on using QD methods to explicitly create an archive of diverse features, rather than implicitly relying on diverse architectures to do so.

References

1. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
2. Cully, A., Clune, J., Tarapore, D., Mouret, J.B.: Robots that can adapt like animals. Nature **521**(7553), 503 (2015)
3. Dietterich, T.G.: Ensemble methods in machine learning. In: International workshop on multiple classifier systems. pp. 1–15. Springer (2000)
4. Hart, E., Steyven, A.S.W., Paechter, B.: Evolution of a functionally diverse swarm via a novel decentralised quality-diversity algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 101–108. GECCO '18, ACM, New York, NY, USA (2018)
5. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
6. Hong, L., Page, S.E.: Groups of diverse problem solvers can outperform groups of high-ability problem solvers. Proceedings of the National Academy of Sciences **101**(46), 16385–16389 (2004). https://doi.org/10.1073/pnas.0403723101
7. Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images. Tech. rep., University of Toronto, Science Dept. (2009)
8. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE (1998)
9. McDonnell, M.D., Vladusich, T.: Enhanced image classification with a fast-learning shallow convolutional neural network. In: Proceedings of the International Joint Conference on Neural Networks (2015). https://doi.org/10.1109/IJCNN.2015.7280796
10. Mouret, J.B., Clune, J.: Illuminating search spaces by mapping elites. arXiv preprint arXiv:1504.04909 (2015)
11. Murphy, K.P.: Machine learning: a probabilistic perspective) (2012)
12. Pandey, G., Dukkipati, A.: To go deep or wide in learning? (2014)
13. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., Facebook, Z.D., Research, A.I., Lin, Z., Desmaison, A., Antiga, L., Srl, O., Lerer, A.: Automatic differentiation in PyTorch. In: Advances in Neural Information Processing Systems 32 (2019)
14. Pugh, J.K., Soros, L.B., Stanley, K.O.: Quality diversity: A new frontier for evolutionary computation. Frontiers in Robotics and AI **3**, 40 (2016)
15. Shafaei-Bajestan, E., Baayen, R.H.: Wide learning for auditory comprehension. In: Proc. Interspeech 2018. pp. 966–970 (2018). https://doi.org/10.21437/Interspeech.2018-2420, http://dx.doi.org/10.21437/Interspeech.2018-2420

16. Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in nlp. arXiv preprint arXiv:1906.02243 (2019)
17. Suganuma, M., Shirakawa, S., Nagao, T.: A genetic programming approach to designing convolutional neural network architectures. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 497–504 (2017)
18. Sun, Y., Wang, H., Xue, B., Jin, Y., Yen, G.G., Zhang, M.: Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor. IEEE Transactions on Evolutionary Computation (2019)
19. Sun, Y., Xue, B., Zhang, M., Yen, G.G.: Automatically designing cnn architectures using genetic algorithm for image classification. arXiv preprint arXiv:1808.03818 (2018)
20. Szerlip, P.A., Morse, G., Pugh, J.K., Stanley, K.O.: Unsupervised feature learning through divergent discriminative feature accumulation. In: Proceedings of the National Conference on Artificial Intelligence (2015)
21. Urquhart, N., Hart, E.: Optimisation and illumination of a real-world workforce scheduling and routing application (wsrp) via map-elites. In: International Conference on Parallel Problem Solving from Nature. pp. 488–499. Springer (2018)
22. Xing, E.P., Ho, Q., Xie, P., Wei, D.: Strategies and Principles of Distributed Machine Learning on Big Data (2016). https://doi.org/10.1016/J.ENG.2016.02.008
23. Zagoruyko, S., Komodakis, N.: Wide residual networks (2016)