Redefining Neural Architecture Search of Heterogeneous Multinetwork Models by Characterizing Variation Operators and Model Components

Unai Garcíaarena, Roberto Santana, and Alexander Mendiburu, Member, IEEE

Abstract—With neural architecture search (NAS) methods gaining ground on manually designed deep neural networks—even more rapidly as model sophistication escalates—the research trend is shifting toward arranging different and often increasingly complex NAS spaces. In this conjuncture, delineating algorithms which can efficiently explore these search spaces can result in a significant improvement over currently used methods, which, in general, randomly select the structural variation operator, hoping for a performance gain. In this article, we investigate the effect of different variation operators in a complex domain, that of multinetwork heterogeneous neural models. These models have an extensive and complex search space of structures as they require multiple subnetworks within the general model in order to answer different output types. From that investigation, we extract a set of general guidelines whose application is not limited to that particular type of model and are useful to determine the direction in which an architecture optimization method could find the largest improvement. To deduce the set of guidelines, we characterize both the variation operators, according to their effect on the complexity and performance of the model; and the models, relying on diverse metrics which estimate the quality of the different parts composing it.

Index Terms—Generative modeling, heterogeneous multitask learning (MTL), neural architecture search (NAS), supervised learning.

I. INTRODUCTION

The lack of interpretability of the operating mechanisms of deep neural networks (DNNs) has not deterred researchers from studying their applications in many research fields, attracted by their impressive performance in several different domains [22]. This is particularly noticeable in image-related tasks: classification [28], [50], captioning [52], or segmentation [3]; albeit other areas, such as natural language processing [26] or data generation [23], have also been benefited.

Their adaptability to different problem specifications has resulted in an enormous spike in DNN research in the last decade. Initially, little or no attention was paid to the DNN structure (a term that we use to comprise the architecture of a DNN as well as other hyperparameters, such as the training specification), and default or baseline designs were used. However, several researchers noticed that finding the right structure proved to be essential to obtain a top-performing DNN for the particular problem at hand [28], [49], [50]. This fact promoted the usage of hand-made designs, guided by expert knowledge. This approach, to this day, is still very popular [43], [48]. However, the popularization of DNNs and their application to more and more challenging problems has proportionally escalated the necessity for increasingly complex DNN structures, to the point where designing them, has become a process, which is too time-consuming and difficult to be carried out by hand.

In order to overcome this issue, several approaches aiming at neural architecture search (NAS) have been proposed in the last few years, their main goal being to extricate humans from the duty of manually designing neural models, as well as being able to obtain structures [15], [32], [62], which fit a given problem exceptionally well. These techniques have evolved from relying on modest operators whose scope merely resulted in an enormous spike in DNN research in the last decades. Their adaptability to different problem specifications has resulted in an enormous spike in DNN research in the last decade. Initially, little or no attention was paid to the DNN structure (a term that we use to comprise the architecture of a DNN as well as other hyperparameters, such as the training specification), and default or baseline designs were used. However, several researchers noticed that finding the right structure proved to be essential to obtain a top-performing DNN for the particular problem at hand [28], [49], [50]. This fact promoted the usage of hand-made designs, guided by expert knowledge. This approach, to this day, is still very popular [43], [48]. However, the popularization of DNNs and their application to more and more challenging problems has proportionally escalated the necessity for increasingly complex DNN structures, to the point where designing them, has become a process, which is too time-consuming and difficult to be carried out by hand.

In order to overcome this issue, several approaches aiming at neural architecture search (NAS) have been proposed in the last few years, their main goal being to extricate humans from the duty of manually designing neural models, as well as being able to obtain structures [15], [32], [62], which fit a given problem exceptionally well. These techniques have evolved from relying on modest operators whose scope merely enabled them to perform small changes [46], to directly adding complete neural cells that could be considered DNNs themselves [32]. Some representative examples of NAS techniques employ reinforcement learning [16], evolutionary algorithms (EAs) [39], or local search methods [14].

The lack of efficiency is a flaw often held against NAS algorithms, as, commonly, assessing the quality of a DNN structure involves weight optimization procedures, which tend to be rather costly [11], [53], [56]. Hence, our main concern when conceiving NAS algorithms is sharpening the usage of every component of the search so that we make the most out of every evaluation. The efficiency of NAS algorithms is largely dependent on the effectiveness of the operators they employ. Moreover, that effectiveness could fluctuate depending on when or where it is applied, i.e., a productive
operator can prove useless if applied in the wrong circumstances.

In this context, we identify a large potential for improvement in these NAS methods, as most of them apply modification operators randomly, hoping for an improvement in the resulting model. Developing criteria to select the most suitable choice from a pool of operators, and the part of the model to be the target of that modification, would improve the efficiency of these NAS algorithms.

Structural specialization is not the only route, in which DNNs have made progress. As a matter of fact, research areas, which establish their basis on a completely antipodal intuition have also received their fair share of attention. A good example of this is multitask learning (MTL) [5], a learning paradigm, which employs one single model for predicting similar targets from pieces of data, which follow similar paradigm, which employs one single model for predicting similar targets from pieces of data, which follow similar distributions. This approach has proven to have more benefits apart from the obvious parameter reduction in the DNN, as it has been demonstrated that the multiple tasks coexisting in a single-model act as regularizers of each other by introducing an inductive bias since the model cannot focus on learning a single task. Furthermore, it has been theoretically proven [34] that the value of the loss function of a task within an MTL framework tends to the same value that it would have reached had it been learned separately, as the number of training observations increases. Although not as popular as those introduced at the beginning of this section, problems in which multiple outputs have to be addressed simultaneously also exist, and DNNs (of elevated complexity) have recently been the model of choice to approach them [20], [31], [59]. This does nothing but further intensify the need for automatic DNN structure designing algorithms. Furthermore, while extending the complexity of the networks results in an increase in the time required for researchers to develop new structures that outperform the previous ones, the process made regarding the hardware in which DNN training can be parallelized pushes the time elapsed by NAS techniques in the opposite direction [42]. One particular variant of MTL, heterogeneous MTL (HMTL) was proposed in [20]. Unlike traditional MTL, this new research line focuses on developing models which

are capable of dealing with several problems simultaneously (potentially of different natures, e.g., classification, regression, and data generation) and in a collaborative way. This is achieved by designing a single and complex DNN, susceptible to being incrementally extended for new data inputs and/or tasks, as different requirements arise.

The diagram in Fig. 1 shows a perspective on the directions in which the research of DNNs has evolved over the last few years according to two criteria: the structure type of the data they are specialized for, and the number and types of tasks they are designed for.

Due to their increased complexity, designing these models by hand poses an even harder challenge than designing single-task models. Moreover, the lack of hand-made models expected to fulfill several assignments at the same time (unlike the case of single-task models) results in a lack of references toward which the searches could be guided. Additionally, approaches based on reinforcement learning or gradient descent would require nontrivial adaptations in order to deal with the intricacies of operating in such a complex environment as that of HMTL. A clear example of these complexities is the necessity to satisfy data type constraints. Consequently, designing refined incremental NAS techniques that can cope with these restrictions becomes a necessity. More specifically, defining effective operators and smart approaches that maximize the efficiency of their application would be essential for obtaining powerful HMTL models.

This work aims at opening a new research line in the direction of making NAS more efficient. This has recently been identified as an area for potential improvement for these methods [4], [36] in general, and particularly for HMTL models. To that end, we attempt to illustrate the effectiveness of an intelligent NAS approach by reducing the random component that characterizes some of the current structural search algorithms. With that goal in mind, we define a set of guidelines that can help a NAS algorithm to make an informed choice between all the variation operators at its disposal. These guidelines rely on a first step in which the model status is diagnosed using a set of metrics, which dictate the variation operator to be applied to improve the model in the second step.
We attempt to make these guidelines as problem-independent as possible so that they ideally can be applied to any problem domain (e.g., classification, regression, generative modeling, or even combinations of them), regardless of the network architectures or variable dependencies involved. In that regard, we identify the areas in which the modeling difficulty is high: problems that fit the HMTL paradigm [20], [31]. To illustrate the effectiveness of these guidelines in complex domains, we evaluate them on NAS algorithms for models that satisfy two particular characteristics: 1) they comprise multiple subnetworks that interact with each other. 2) They solve multiple heterogeneous machine learning problems (e.g., classification, regression, and data generation simultaneously). More specifically, we employ the VALP [20] (a recently introduced HMTL model) as a testbed to demonstrate the effectiveness of the proposed strategy.

This article is organized as follows. In Section II, the literature relevant to this work is covered. The problem in which the approach is tested is introduced in Section III. In Section IV, the ideas which are the main contribution of this work are described. These ideas are then materialized into mechanisms for improving NAS runs, which are described in Section V. The experiments designed for showing the potential of the proposal are presented in Section VI, and the obtained results are summarized and discussed in Section VII. Finally, Section VIII contains the conclusions drawn from this work, as well as some future research lines.

II. RELATED WORK

As stated in Section I, the approach presented in this work consists of the smart application of variation operators in order to increase the efficiency level of NAS algorithms, particularly—but not limited to—when applied to HMTL. In this section, we first review relevant work on the NAS area from the perspective of the type of search and operators they employ (so that the beneficial aspects of each type are identified), as well as some works on the efficiency of these algorithms. By integrating multiple perspectives on the NAS problem, an algorithm with access to multiple options from which to choose when making decisions could be designed. We next discuss some work performed on DNN model diagnosis, as the metrics proposed in these works will be useful to make an estimation of the relevance of each component within a neural model. This estimation can ultimately lead to informed decisions among all the possibilities imported from the different NAS approaches. Finally, we cover some different approaches to the HMTL problem, a problem definition to which the efforts in this article are devoted, and which is more general than the commonly addressed single-task scenarios. By proving the proposal of this article in such a complex environment as HMTL, it seems fair to assume that it could also be successfully applied to other simpler scenarios—such as single-task ones.

A. Neural Architecture Search

As previously mentioned in Section I, NAS methods have been designed inspired by multiple techniques. We limit this review to the approaches that have the largest influence on the proposed set of guidelines: neuroevolution (EAs which usually rely on neural variation or mutation operators) and network morphism (NM).

1) Neuroevolution: In the last couple of decades, several works have used evolutionary approaches for NAS with the ultimate goal of producing DNNs, which are able to cope with different tasks. Although these works have particularly focused on image-related tasks, many of the proposals could be applied to other types of tasks.

The traditional approach to neuroevolution (NE) commonly considers relatively low-parametrized networks, both regarding the number of layers and the number of neurons in them. However, as the hardware supporting DNNs has improved, these methods have shifted from performing low-level modifications, e.g., the addition of one neuron or connection [45], [47], to more complex operations, such as the concatenation of full neural cells to the DNN [32], [36]. This second kind of evolution has proven competitive against hand-crafted structures and is the most popular approach considering the amount of recent work devoted to it. Currently, these two scopes of variation operators are known as microsearch (modifications limited to the small cells or subnetworks within the model) and macrosearch (altering the general structure of the neural model) [56].

The work in [41] presents an NE approach that adopts the NASNet search space (originally designed for reinforcement learning-based NAS) [62]. The authors propose incorporating an age property for all individuals involved in the NE procedure, in order to favor individuals of recent creation at the time of performing the tournament selection.

Some other approaches, while still being framed in the image treatment scheme, have variations, which are especially relevant to our work. For example, in [36], NSGA-Net is proposed. This population-based algorithm permits the inclusion of (potentially conflicting) objectives as opposed to the classic single error metric-minimization scenario. This way, the authors address the problem of low efficiency on state-of-the-art NE algorithms by introducing a second objective which seeks the minimization of the computational complexity of the models. The work presented in [7] introduces ModuleNet. This NE algorithm is largely inspired by [36], although new mutation operators are introduced. This NE algorithm is based on connecting subnetworks found in top-performing DNNs proposed in the literature.

The previously reviewed works (as well as many others not included here) mostly follow the same pattern: They introduce a framework different from those that have already been proposed, and design ad hoc operators for it. In this work, we aim at forming a set of guidelines, which are able to operate in different schemes in terms of model structure and problem domain by encompassing different types of operators and applying them when their positive impact on the search can be maximized.

2) NAS Efficiency: The growth in interest in the efficiency of NAS methods has garnered a significant amount of attention recently, due to the current trend of increasingly complex and expensive models and algorithms.
The work reported in [11] presents broad NAS, an extension of Efficient NAS. The authors argue that, in current reinforcement learning-based NAS methods, a considerable proportion of the time is spent in training and evaluating the DNNs, and this is due to the depth of the models. As an alternative to orienting the search toward depth, the authors propose orienting it toward broadness. This approach yields state-of-the-art results (or comparable) in the medium-sized models category (0.5 M parameters), while significantly reducing the time spent optimizing the CNN (less than half of the total time).

Wang et al. [53] propose EffPNet, which tackles this issue from a different perspective. EffPNet incorporates different aspects from the DenseNet and NASNet approaches and combines them with a surrogate model which is learned from a surrogate dataset and particle-swarm optimization to efficiently evolve CNN blocks. A set of experiments shows that no other approach is able to outperform EffPNet in all three of: 1) error rate in CIFAR-10; 2) the number of parameters; and 3) computational cost. They also show the transferability of the found models, as they also report better or comparable to the state-of-the-art results when transferring the evolved models to CIFAR-100, SVHN, and Imagenet.

The approach reported by Zhou et al. [60] pursues a goal with a similar inspiration to ours (selecting the best possible operation), as they propose a variation of a gradient-based NAS algorithm (GDAS). In the common definition of GDAS, candidate operations are added to the final network depending on the weight assigned to them during training. We work to propose a different variant, in which the operation is not only chosen by its weight but also the accuracy of the model and the number of epochs it has been trained for. As a result, the proposed EoiNAS outperformed other state-of-the-art approaches in CIFAR-10 and CIFAR-100 and offered comparable results to other approaches when transferred to ImageNet. These results show the importance of performing the correct operation in the adequate section of the model.

3) Network Morphism: Another research subfield that has grown separately, but is related to NE due to being based on structural variation operators, is NM. It consists of a special set of operations for extending DNN structures in such a way that the performance of the network is not altered.

The work described in [6] proposes two operators for expanding DNN architectures, Net2Net, applicable to both MLP and CNN architectures. Their effectiveness is tested in a framework in which, initially, a relatively shallow and narrow teacher network is trained for the objective task. Next, one of the two operators: Net2WiderNet (enlarging the size of a layer) or Net2DeeperNet (introducing a new layer to the DNN), are applied to increment the number of parameters of the model. By conveniently initializing the added weights and/or the activation functions, the newly created student network is able to produce the same result as the teacher network. However, because its modeling power has increased, better results can be expected with further training.

These operators are first employed to gradually transform a shallow teacher into an inception network. Results show higher accuracy and faster convergence than training the same structure from scratch.

Wang et al. [55] resolved some of the inherent limitations of [6]: e.g., the inclusion of nonidempotent activation functions in the network modifications. Besides, the subnet adding operator was also presented, which is equivalent to adding several layers at once. The authors finally used the NM term to name the framework containing this kind of operator. This extension of Net2Net is able to outperform the original proposal both in learning speed and final accuracy.

The work in [14] takes full advantage of the framework defined in [55] and uses it as a tool for a NAS by hillclimbing (NASH), using a simple structure as the starting point.

To the best of our knowledge, the research carried out using these operators is rather limited, considering the complementary role they can play for the more common operators usually employed for NAS. Therefore, we integrate them into the NAS framework governed by the guidelines proposed in this work, along with more traditional operators for NAS algorithms.

B. Model Internal Diagnosis

Studies attempting to understand the way DNNs operate have yielded many interesting approaches [1], [2], [44]. Shwartz-Ziv and Tishby [44] propose the diagnosis of a neural model by computing the mutual information between the representations of the information found in the different layers of a DNN, and the input and output of the network during the training of the model. They concluded that, trained with the common combination of stochastic gradient descent (SGD) and backpropagation, the weight optimization procedure of a DNN consists of two different phases, the information compression phase and the error minimization period.

In [1], a similar approach is presented. The information representation in each layer of a classification DNN is extracted for a set of observations, and a linear classifier is fit between each of these representations and the original classes, independently. The errors reported by the classifiers from the different layers can serve as a measure of the quality of the information representation at each level of depth in a DNN. Because linear classifiers are rather limited and require rich representations of the data to perform well, it can be expected that, the deeper the layer—and therefore, the richer the representation—the better a linear classifier will perform.

In both previous approaches [1], [44], comparing the values given by the metrics (mutual information and classification error) between layers can help to understand the level of importance of each layer within the general model context.

In an attempt to identify the origin of an issue in the differentiable architecture search (DARTS) [33] search space, Wang et al. [54] propose the deletion of different parts of a DNN and using the observed performance decrease of the overall DNN as an estimate of the relevance of that part to the overall model.

Although the explainability of the decisions made by DNN models is not among the objectives of this work, trying to estimate the relevance of a subnetwork (named sub-DNN indistinctly in this work) within a neural model composed
C. Heterogeneous Learning

In [20], a framework for HMTL was defined, called VALP (more details in Section III). One particular characteristic of the VALP is the fact that the structure of the graph is not fixed, and it is, therefore, optimizable. That work makes an initial exploration of the search space of VALPs by employing a random search and comparing its results to a fixed regular MTL model. That experiment showed to which extent finding the right structure is an important aspect of the performance of the HMTL models. This observation later motivated an extension of this work in [19], where a VALP structural search algorithm was proposed. The method consists of four different operators which modify the structure of a VALP used as a base for a hill climbing (HC) algorithm. All four operators are based on connection modification, namely, add and delete connection, and insert and delete network. Conducted experiments showed that even though the operator to be applied was randomly chosen, the HC was able to find better-performing structures than a simple random search.

After discovering the potential of a search algorithm in an HMTL environment, in this article, we go one step further in this area by exploiting NAS methods in an HMTL framework. Our main goal is—as we recognize the difficulty of dealing with multiple inputs, outputs, loss functions, sub-DNN types, and so on—to present advances to the NAS field that can make the structural search feasible even in such convoluted search spaces by presenting a set of guidelines that can be used by NAS algorithms. Although its initial presentation is directed toward the HMTL framework, the application of these guidelines is not bounded to that environment, as it can be exported to many other NAS formulations.

Multiloss function joint learning (JL) [31] was initially defined over the person reidentification problem, which consists of matching identity classes in detected person-bounding boxes from nonoverlapping camera views. To that end, several different features are learned for each person in multiple images and are afterward used to match persons across images according to the similarity of these features. The authors choose to employ a JL model to test the hypothesis of whether learning these features can yield better results if done together, rather than using separate models for each one. With that goal in mind, the authors manually build a CNN in which the first part is shared before the network is divided into two branches, each of which has a specific design for the different tasks and is trained using a separate loss function.

DG-Net [59] introduces a new dimension to the JL framework as it was designed to fulfill tasks, which are different from each other. In contrast to [31], DG-Net is able to both classify and generate new samples similar to those in the training set. DG-Net is designed as a module-based neural model, in which each module has a specific task to fulfill, varying from encoding or decoding image features or structures to discriminating the images based on the mentioned features. Because each part of the model has a predefined role, the structure of the model cannot be changed, as it could lead to a change in the role played by each subnetwork of the model. The training of the model for performing such diverse tasks requires, of course, the management of different loss functions. The components and loss functions are arranged in a predefined structure to maximize the modeling capabilities of each of the subnetworks within DG-Net. In order to combine the losses in a single expression, a weighted average of them is computed by hand-picking weights.

III. HETEROGENEOUS, MODULE-BASED MODEL

We aim at defining a set of guidelines as general as possible, capable of guiding a NAS algorithm. Consequently, we choose an application benchmark, which shares that characteristic: a model which is able to handle multiple data inputs, can make predictions for more than one output at a time (homogeneous or heterogeneous), and is composed of sub-DNNs. The inputs, outputs, and modules within the model are interconnected with a scheme of connections, which indicates which information (either from a model input or a subnetwork) is redirected to which place (either another subnetwork or a model output), forming a directed graph. Each output corresponds with a prediction required for the model. Because the performance of a model can be assessed by (at least) as many metrics as outputs it has, this problem has a multiobjective nature.

The recently proposed VALP [20] is an example of a module-based model used for solving HMTL problems. Its structure can be defined as a directed graph, \( G = (V, A) \), in which the sub-DNNs within the model are represented by the vertices \( V \), and the flow of information produced and received by them is directed by the arcs \( A \). We name the combination of these two sets as the set of components. The vertices in \( V \) can be categorized in three different disjoint subsets: \( I \cup N \cup O = V \). \( I \) is composed of the source nodes \( i_j \), and, in the network topology, these are the sources of information, the data. \( N \) contains the internal nodes \( n_k \), which are the actual subnetworks within the VALP, each of which can be defined by different architectures and hyperparameters. Finally, \( O \) contains the set of sink nodes \( o_l \), the final components in which the final predictions (regarding the data present in the source nodes) of the VALP can be collected.

The VALP can be trained using backpropagation, a gradient descent algorithm, and a set \( L \) of loss functions (which, in this work, are combined by addition), containing at least one loss function for each item in \( O \).

In a VALP (as in any other module-based model with multiple inputs, outputs, and components), the subnetworks can be grouped according to different criteria. We define the following subgraphs of \( G \): output subgraph, and output exclusive subgraph. The subgraph of an output \( o_l \) consists of all the components that, upon modification, alter the prediction in \( o_l \). The output exclusive subgraph consists of a similar subgraph, although in this case, the components that affect multiple outputs are not included in either output subgraph.
In this work, we choose the second option. We understand a structurally correct VALP as one which guarantees to comply with a set of characteristics, which could try to evaluate solutions, which fall outside the constraints of the problem, which would be poor exploitation of computational resources. Although techniques for detecting and repairing this kind of solutions exist [9], a common approach when facing such complex search spaces is to rely on operators, which guarantee that their product is going to be feasible. In this work, we choose the second option. We understand a structurally correct VALP as one which guarantees to comply with a set of characteristics, which necessarily include:

1) The model has as many model outputs as the problem has targets, and there is a correspondence in terms of data type between each target variable and each model output. For example, a classification and a regression output do not share the same characteristics (the values in the output neurons in a classification problem have to amount to one, and are, therefore, activated by the softmax function).

2) Every sub-DNN and model output must receive data from at least another component, and the data coming from every subnetwork and model input must go to at least another component.

3) This instance of the VALP model does not consider the possibility of recurrent connections, despite this limiting the capacity of the model to deal with temporal data. However, the proposal of this work would still be valid if recurrent connections were contemplated.

IV. INTELLIGENT SEARCH

Due to the costly nature of some NAS algorithms caused by the magnitude of the search space, an efficient structural search of module-based models is crucial, especially when dealing with HMTL problems. This efficiency is mainly dependent on the operators integrated into the search algorithm, and, most importantly, how they are employed. In this section, critical aspects of different search methods are identified, before reflecting on how to exploit these characteristics in order to improve the efficiency of the search algorithms.

First, we categorize the search methods by the number of neural models being taken into account at any given moment.

1) Single Model Search: In this instance, the search algorithm consists of improving a single model at a time, as in a local search (e.g., HC).

2) Population-Based Search: This second formulation considers several models at each time during the search.

A. Model Internal Diagnosis

The first key question in the proposed intelligent structural search is identifying which component or part of the structure (in our case, a subnetwork) should be improved at a given point. To that end, we identify diverse sources of information depending on the type of search, which could help to make the right decision in this matter.

In a single-model scenario, the main sources of information consist of the following.

1) Comparisons with the performance of models evaluated in previous iterations of the algorithm.

2) The relevance of the different components within the model to the final predictions made by the model.

3) The effectiveness of the training procedure to improve the prediction.

In the population-based search, along with these three sources, other ones are also available. These can be used to gauge the performance of a given model with respect to its peers, providing a more accurate idea of which component of the model, when modified, can provide a larger gain in terms of model performance or loss function optimization.

B. Metrics

With the information sources identified, the next step is determining how they are going to be measured. Focusing on the single model scenario, we formalize four different metrics.

1) Historic Subloss Information: Performance metrics extracted from the loss functions associated with the subnetwork. The performance of a sub-DNN (or a set of them) along time can be estimated by comparing metrics—e.g., the loss function values—at each iteration.
2) **Module Intervention:** Inspired by the techniques of intervention for causal discovery [13] and neural architecture selection methods [54], here we modify one or multiple characteristics of a module (e.g., setting the weights to random values) and use the gain/loss in performance of the model to estimate the importance of the module.

3) **Input Intervention:** Similarly, it is possible to modify input values (i.e., a subset of the features of the data) and estimate their relevance with respect to the predictions.

4) **Dependency Measures:** Metrics (such as the mutual information or the classification error [1]) between the output of each subnetwork and the model output(s) it contributes to, would ideally improve compared with the output of the sub-DNN preceding it [44]. Otherwise, it could be interpreted that the component is not helpful. For population-based approaches, we define an additional metric based on comparisons between models in population-based searches (although it could also be applied to the isolated model search by comparing the current model with other models in previous stages of the search).

5) **Relative Performance:** several rankings—at least one per output—can be arranged, according to the performance of the model in each output, relative to the rest of the models. The position of a model in the ranking of a given output determines the quality of the subgraph of that output.

### C. Variation Operator Types

The third step is to define variation operators that cover the different needs that the models can present at different points during their development.

In this work, we categorize the variation operators according to two attributes: their aggressiveness and the effect they have on the complexity of the model. Regarding aggressiveness, we distinguish two types of operators:

1) We consider an operator to be aggressive when it performs drastic alterations to the model structure, in such a way that the performance of the model can be severely changed in at least one of the objectives (e.g., operators commonly used in NAS).

2) On the contrary, an operator is considered as gentle when the performance of the model does not vary after its application (i.e., morphism operators).

When discriminating operators by their effect on the complexity of the model, we also divide the set of mutators into two subsets:

1) An operator is considered a reducer when its application decreases the number of weights in the model, and thus, theoretically, the modeling capacity.

2) Alternatively, an operator is an extender when the model sees its number of weights increased.

Because we have two categories for each characteristic, we can define four operator types. First, an aggressive extender operator would increase the number of weights of a model at the same time as the performance of the model is altered. For example, integrating a new random subnetwork to a subgraph of output could alter the performance of the model in that output.

Second, a gentle extender operator would increase the modeling capacity of the model without modifying the performance of the model, e.g., by modifying other components already present in the model and cautiously designing and placing the new component.

Third, an aggressive reducer would decrease the modeling capacity and have the collateral effect of altering the model performance, e.g., by deleting a sub-DNN or a connection that was relevant to the overall model.

Finally, a gentle reducer would delete certain parts of a model, without affecting the performance of the model. The deleted parts would need to be irrelevant to the model.

### D. Donation Operator

In population-based searches, mutation operators are not the only method to perform alterations to models. In this case, although they have been widely omitted by the NE community [17], [51], we define a special version of the crossover, traditionally referenced as the conjugation operator [24]. In this method, a donor model donates a part of itself (e.g., the output exclusive subgraph) to a host model.

### E. Principles for Using the Metric Information

In this section, we propose a set of criteria for optimizing NAS procedures for HMTL models (although their application is not limited to those models), exploiting the metrics defined in Section IV-B to guide the selection of the variation and donation operators, as defined in Sections IV-C and IV-D.

1) **Historic Subloss Information:** This metric can be used to observe the behavior of one or more model outputs by fitting a linear regression model which attempts to predict the subloss value of an output, given the training step. This way, the slope of the loss function can be approximated with a line and, depending on that value, different approaches can be taken:

   1) If the slope is ∼0 or positive, it can be concluded that the output has converged. In that case, an aggressive operator could move the model away from those local optima.

   2) When the slope is slightly smaller than 0, it can be interpreted that the output is still improving, although a major improvement is unlikely. Here, a gentle extender could add modeling power, helping the model perform another significant gain without losing the current performance.

   3) In the case in which the slope is considerably smaller than 0, the output is still in the early phase of improvement and should be left as it is until a certain level of convergence is reached, i.e., the previous two scenarios.

2) **Module Intervention:** This metric can be used to determine the relevance of a given subnetwork to the overall model by measuring the performance loss after resetting the weights of that sub-DNN.

   1) If the performance loss is not great for any output, the importance of the sub-DNN to the model is low, and a reducing operator could be advised.
2) On the contrary, if the drop-off is significant, the component is assumed to be relevant and should either remain intact or be expanded using a gentle operator.
3) Finally, if the subnetwork is connected to an output, which was not affected, a connection deletion would reduce the model complexity without deteriorating the overall model performance.

3) Input Intervention: Similar to module intervention, this metric would estimate the importance of a given input to the final prediction of the model. This could be done by observing the performance change in the different outputs when randomly changing a subset of the features of the data.
1) If the performance loss is not great, then the input is not very relevant to the output, and deleting a connection that connects the path between the input and the prediction would be advisable. The model would also graphically represent that independence.
2) If the performance loss is significant, then the input is relevant to the output, and no connection should be deleted.

4) Dependency Measures: As was the case for the module intervention, this metric serves the purpose of measuring the importance of a component for the model. In this case, the metric values (e.g., the mutual information or the error of a linear estimator) achieved by a model output and the components on its subgraph are compared with the predecessors.
1) When the measure indicates a larger dependency between the values, it can be assumed that the component is performing satisfactorily, and should be either gently expanded or left unchanged.
2) If the value does not improve, the component is not performing as expected, and a reducer operator can be applied without losing much potential.

5) Relative Performance: By constructing rankings of models according to their performance in the different outputs, it would be possible to estimate the relative performance of a model in that output. A model with all but one output in the higher part of their corresponding rankings could become the host of the exclusive subgraph of a model with a high rank in that specific output ranking. This vision is closely related to multiobjective optimization, as one model can be viewed as valuable or useless depending on different factors, such as the output being evaluated, or the current state of the search.

V. SEARCHING FOR OPTIMAL VALP STRUCTURES USING VARIATION OPERATORS

Section IV presented a general approach and guidelines for an intelligent structural search. In order to show its utility, this theoretical framework is implemented into the VALP NAS context. In what follows, we introduce variation operators which can be applied to the VALP but could, generally, be applied to any other neural model based on sub-DNNs (or neural cells). We decided that the defined operators must comply with the characteristic of having to produce structurally valid VALPs. The operators are classified according to the characteristics described in Section V-B (aggressiveness and effect over the complexity of the model) and the scopes of application.

A. Subnetworks

We start with the operators with reduced performance scope (microsearch): layerwise modifications of a subnetwork in a VALP. Three different mutation operators have this scope.
1) Add_layer: This extender operator adds a layer in the network. Depending on how the weights are initialized and where the layer is added, this operator can be aggressive (e.g., by randomly initializing the weights) or gentle (e.g., by using the morphism approach).
2) Remove_layer: This operator deletes a layer from the network. The rest of the layers remain the same. As a reducer, this operator is aggressive.
3) Extend_layer: This operator adds neurons to a layer from the network. The remaining layers stay the same. This extender operator can be either aggressive or gentle.

B. General Model Structure

The next set of operators is capable of affecting the VALP structure at its higher level (macrosearch), i.e., the interconnections between the different sub-DNNs in a VALP. We define five modifiers with this capacity.
1) Add_connection: Given two unlinked subnetworks of a VALP, this operator links them by creating a new connection. In other words, the second sub-DNN receives the output of the first subnetwork as additional input. This extender operator can be either gentle or aggressive.
2) Delete_connection: Given a connection of a VALP, this aggressive and reducer operator deletes it.
3) Insert_network: Given a connection of a VALP, this operator inserts a network in the middle of the connection. For example, if a connection c₀ that links n₀ to n₁ is chosen, a connection c₁ between n₀ and the newly created nₘ, and a connection c₂ between nₘ and n₁ are created, and c₀ is deleted. This expander operator can be either aggressive or gentle.
4) Delete_network: Given a network nₘ of a VALP, this operator deletes it. Each subnetwork providing data to nₘ switches to supplying data to each and every sub-DNN nₙ provided data to. This operator is a reducer, and, as determined by an additional experiment reported in the Supplementary Material of this article, aggressive.
5) Clone_network: Given a network of a VALP, this operator duplicates that network and all the connections related to it. This operator is an expander and can be either aggressive or gentle—by applying a 0.5 factor to the outputs of both the original and the clone networks, which neutralizes the immediate effect of the operator. These last five methods will be applied only if structural correctness is guaranteed. For example, delete_connection will not, under any circumstances, delete a connection.
Employing crossover-like operators enables parts of models to have some exclusive sub-DNNs (and, therefore, weights) of each other to some degree, as each output will normally only raise the model to a certain point, as the rest of the model components need to be synchronized to obtain optimal performance. This is the case of the loss functions related to training, such as the learning rate and batch size, or delete_network will never suppress a subnetwork when it is the only one between a model input and a model output.

The gentle operators defined in this work depend entirely on reusing and adequately modifying the weights optimized in the previous training epochs. We reuse the weights learned by a model before it is altered, i.e., we apply weight inheritance, whenever it is viable (when a random sub-DNN is added to a model, we apply weight inheritance, if it is viable). Graphical examples of these operators are shown in Fig. 3.

C. Hyperparameters

Searching for the optimal model architecture (the subnetworks and how they are interconnected within the model) would only raise the model to a certain point, as the rest of the model components need to be synchronized to obtain optimal performance. This is the case of the loss functions used to optimize the weights of the neural model and other hyperparameters, such as the SGD algorithm. Other aspects related to training, such as the learning rate and batch size, also have to be properly set. With this in mind, we define the following variation operators, all of which are gentle:

1) **Change_lr** changes the learning rate of model output. If convergence is detected, the learning rate can be decreased, aiming at improving the effectiveness of training.

2) **Change_sgd** changes the SGD algorithm used to optimize the model weights with respect to model output.

3) **Change_bs** changes the size of the batch used at each training epoch.

D. Crossover Operator

In this multiobjective scenario, the objectives are independent of each other to some degree, as each output will normally have some exclusive sub-DNNs (and, therefore, weights). Employing crossover-like operators enables parts of models to be cherry-picked for constructing other models with the best parts of each one. We define a crossover operator based on the donation between models:

1) **Exclusive Subgraph Crossover:** This aggressive operator can be applied when, based on the relative performance measure, a model behaving adequately in multiple tasks fails at another one. A model with a top performance in that last task is selected as the donor of the exclusive subgraph of that output for the first model, the host, which has its exclusive subgraph replaced by the donation.

VI. Experiments

We have designed a set of experiments in order to validate some of the general guidelines for the NAS framework proposed in this article.

Several works have reported that starting from simple neural models with relatively few parameters and allowing them to evolve toward more complex structures yields positive results [35]. The experiments described in this section consist of the employment of the proposed search guidelines with this same mindset. We consider a model with a number of components close to the minimum (roughly one subnetwork per model output) to provide the required output to be in its initial stages, whereas a mature model would consist of a more complex structure with more subnetworks and connections.

A. Test Benchmark

For the different parts of the experimentation, we have built two artificial problems. Both of them consist of extensions of two widely known problems, MNIST [30] and fashion MNIST [57]. The two datasets are composed of images of the same number of pixels (784) arranged in the same manner (28 × 28), although the former is composed of handwritten digits, and the latter consists of pictures of clothing pieces. We define the multiobjective version of both problems [21].

1) **Classification Objective:** This is the original definition of the problem. It consists of correctly classifying the observations in the dataset into one of the ten classes.

2) **Histogram Prediction Objective:** This objective consists of correctly predicting the histogram of the pixels in the images. For the images being grayscale, a single histogram (of eight bins) can be computed and associated with each one of the examples.

3) **Image Sampling Objective:** This last objective consists of sampling images similar to those in the dataset.

This way, we can test the performance of the operators when acting in an environment where the outputs are related to a single data input.

We define the two separate problems to simulate the scenario, in which the rules are inferred from one set of experiments, and are then applied to another, more complex problem. The two problems having very similar characteristics in terms of the number of examples and features as well as data type and the number of classes are purely coincidental, as this approach could be tested in problems of varying data inputs, outputs, and characteristics of both.
B. Initial Experimentation

The first step consists of testing the proposed metrics and operators isolated from the NAS framework. This way, we will be able to extract valuable information about how to use the metrics with the final goal of deciding which operator to use and where it should be applied in a NAS process.

In order to assess the impact that each operator can have in different scenarios (these being described by the values obtained from the different metrics), we perform an exploratory search over the space of medium-sized VALPs (i.e., twice as many subnetworks as model outputs). In this experimentation, we will be able to observe the difference between applying gentle mutation operators over their aggressive counterparts.

Additionally, and this is the main goal of this experimental section, we aim at setting the grounds for the set of rules which will improve the efficiency of future NAS runs. To that end, we attempt to identify which operators offer the largest improvement potential. Because the rules we are looking for should not be tied to the particular problem used in this instance, we rely on the metrics defined in Section IV-B instead of the common metrics for assessing the performance of a prediction model (e.g., accuracy for a classification model).

Choosing the MNIST problem, we test the effect of the mutation operators defined in Section V-B. To that end the following holds.

1) Hundred VALPs are randomly created (as described in the Supplementary Material of this work and in [20]) and trained for \( \sim 67 \) epochs (20,000 batches of size 200).
2) Every operator is applied to different clones of each VALP. The operators are applied to each component of the VALPs only if structural correctness is guaranteed.
3) Every VALP is retrained to adjust the weights of the model to the variation for \( \sim 17 \) additional epochs (5000 more batches).

The quality of the VALPs is determined by evaluating them before and after the modification and secondary training.

C. Main Experimentation

In this second step, we want to employ the knowledge obtained in the first step on a NAS procedure. With that goal in mind, we propose a common HC algorithm (Fig. 4 contains a pseudocode form of the method) with two different implementations: the common approach, in which the operators isolated from the NAS framework. This way, we will be able to observe the difference between applying gentle mutation operators over their aggressive counterparts.

Additionally, and this is the main goal of this experimental section, we aim at setting the grounds for the set of rules which will improve the efficiency of future NAS runs. To that end, we attempt to identify which operators offer the largest improvement potential. Because the rules we are looking for should not be tied to the particular problem used in this instance, we rely on the metrics defined in Section IV-B instead of the common metrics for assessing the performance of a prediction model (e.g., accuracy for a classification model).

Choosing the MNIST problem, we test the effect of the mutation operators defined in Section V-B. To that end the following holds.

1) Hundred VALPs are randomly created (as described in the Supplementary Material of this work and in [20]) and trained for \( \sim 67 \) epochs (20,000 batches of size 200).
2) Every operator is applied to different clones of each VALP. The operators are applied to each component of the VALPs only if structural correctness is guaranteed.
3) Every VALP is retrained to adjust the weights of the model to the variation for \( \sim 17 \) additional epochs (5000 more batches).

The quality of the VALPs is determined by evaluating them before and after the modification and secondary training.

C. Main Experimentation

In this second step, we want to employ the knowledge obtained in the first step on a NAS procedure. With that goal in mind, we propose a common HC algorithm (Fig. 4 contains a pseudocode form of the method) with two different implementations: the common approach, in which the operators are chosen randomly, and the smart approach, in which the most promising operator is chosen. The pseudocode makes use of the following functions.

1) Random VALP(): This function randomly initializes a VALP, with a limited number of components.
2) Evaluate(model): Given a VALP, this function evaluates it and returns one value per model output. In this work, it consists of a triple, as the problem has three objectives.
3) Select operator(model): Given a VALP, this function selects the operator to be applied. The difference between the random and the smart HC approach resides in the implementation of this function.

VII. Results

A. Initial Experimentation

First, we want to investigate whether the gentle operators consistently perform better than their aggressive counterparts. To that end, we have computed the improvement observed in the VALPs between the end of the first training session and after it has been modified and retrained. With the improvement measured—performance after the second training divided by performance after the first training, both measurements in logarithmic scale—we subtract the improvement observed due to the application of gentle operators to the improvement caused by their corresponding aggressive counterparts. This metric \( G \) serves as a measure of the gain or advantage of using one class of operator over the other. Fig. 5 shows the frequency (y-axis) of the \( G \) difference values (x-axis). The more positive they are, the bigger the difference in favor of the gentle operator. Any difference superior to one is cut to that value to improve the visualization of Fig. 5.
As can be observed in Fig. 5, the gentle operators have outperformed the aggressive ones considerably more frequently than the other way around, especially taking into account extreme differences (values over 1). Gentle operators are, in general, conservative variations when it comes to altering the performance of the model. In closer comparisons, the gentle operators also tend to produce a bigger improvement in model quality. However, although less frequent, there are many cases in which the aggressive operators had a more positive impact than their gentle counterparts. The noise these operators introduce into the model, in the form of random weights, appears to be able to shake the model away from local optima (an issue at which the gentle operator failed). This is especially visible in the cases in which extreme improvements were achieved by the aggressive operators.

The presence of these last cases suggests that the employment of aggressive operators is not only viable but advisable in some scenarios. This theory is also backed up by statistical testing. After the null hypothesis of all mutations producing the same effects was rejected by the Kruskal–Wallis statistical test [29], the Dunn post hoc test [12] found significant differences between all pairwise comparisons between mutations—p-value < 0.0007—except for one, the comparison between the aggressive and gentle version of the connection adding operator (probably due to the high number of extreme differences in improvements). In other cases, although many of the measurements fall close to zero, there is a significantly larger amount of differences on the positive side, including a substantial amount of extreme values over one.

We now address the question of how to create the set of rules which helps NAS algorithms to correctly identify the best operator, given one model and its state.

Regarding the second part of this experimentation, we attempt to define the metrics that will eventually guide future NAS runs. With that goal in mind, we aim at observing, given the metric values (from those defined in Section IV-B), which operator(s) produced the largest gains. The two metrics which have produced the most significant differences among the analyzed mutation operators were the historic subloss information and the module intervention.

Regarding the loss function slope, Fig. 6 shows the percentage of improvement observed in a VALP after it was modified by the gentle operators and their aggressive counterpart (in the y-axis, in logarithmic scale), regarding the historic subloss function of the regression output of the VALP (x-axis). The improvement percentages (the lower, the larger the improvement) have been cut to the [−0.5, 0.5] range. The loss slope also only considers a minimum value of −0.00015. As can be seen in Fig. 6, most improvements are marginal, just below the 0 mark. However, note that the goal of this analysis is to observe in which cases the operators can produce a significant improvement to the search, rather than how often they are able to do so. The random application of an operator, as can be seen in the distributions in Fig. 6, would very unlikely result in a drastic improvement. Because of this, some interesting insights can be extracted from Fig. 6.

For example, because of the lack of existence of large performance drop when applying the clone net operator [Fig. 6(a)] if the loss function is still decreasing (left-hand side), we can conclude that this is usually a beneficial mutation. The connection-adding operator [Fig. 6(b)] was also able to produce large performance gains when the loss function of output is still steeply decreasing. This means that these kinds of changes are beneficial, especially in the gentle form, when the loss function is still decreasing.

The mutation that places a network in the middle of a connection [Fig. 6(c)] was able to produce significant changes (both improvement and deterioration) when the slope of the loss function is smaller. More specifically, a significant number of drastic improvements were observed compared with the gentle version. This means that applying it to an output, which has saddled in a poor local optimum can dramatically improve the model, while a drastic performance loss would not hurt the search, as the local optima were not desirable anyway.

A similar set of figures has been generated for the network relevance metric. Relevance consists of the change observed between the two stages of the model, before and after being affected by the module intervention approach, and it is also measured in percentage points. This way, if no change was observed in a model output after being affected, 1 is recorded. If the performance was halved (e.g., only half of the observations which were previously correctly classified are correctly classified after modifying the model), 2 is recorded.

In the case of Fig. 7, because the performance of a regression output can decrease indefinitely, the relevance has been cut to 0.4 (in logarithmic scale). As can be observed in the top right corner of Fig. 7(a)–(c), when modifying a network relevant to an output, the result, as expected, can be very bad if the mutation is an aggressive one. Gentle network cloning appears to be a conservative choice when it comes to a relevant net, given the few cases in which performance declines have been observed. A similar effect can be observed with the insert network operator, as it produced more improvements when applied to relevant networks. The connection-adding operator in this case is not advisable with relevant networks.

B. Operator per Network Characterization

With the insights made in Section VII-A, we have defined the following set of rules to display the potential of this kind of guided search.
Fig. 6. Percentage improvement observed over the regression output of the VALP (in the y-axis, the lower the value, the larger the improvement) in logarithmic scale, by different mutation operators, with respect to the slope of the historic subloss function evolution (x-axis). The color darkness represents the number of mutations that registered the improvement in the y-axis. The subfigures on the left-hand side represent data relative to the aggressive version of the operators, whereas the ones on the right-hand side show information about the gentle ones. (a) Clone net mutation operator. (b) Add connection mutation operator. (c) Insert network mutation operator.

Fig. 7. Percentage improvement observed over the regression output of the VALP in logarithmic scale (in the y-axis), by different mutation operators, with respect to the relevance of the network affected by the operator, also in logarithmic scale (x-axis, the larger, the more relevant a network to the output). (a) Clone net mutation operator. (b) Add connection mutation operator. (c) Insert network mutation operator.

1) When a network involved in a loss function registers a steep descent, the gentle version of the network cloning mutation can be applied.
2) If a network is part of an output that is moderately descending, the add connection operator can be applied.
3) When a network is part of an output in a local optimum, the insert network can be applied.
4) If a network is not relevant for some outputs it is connected to but is for other ones, the delete connection, the insert network, or the aggressive version of the clone network can be applied.

These rules, along with these additional ones, hold the following.
1) When a network is not relevant for an output it is connected to, the delete connection operator can be applied.
2) If a network is not relevant for any output, the network deletion operator can be applied.

The earlier have been compiled into the mutation-selection guidelines, which are going to be used in the HC algorithm. A more sophisticated usage of the metrics and variation operator characterization could be carried out. Defining a metamodel that is able to capture the patterns between model states and operator applications for an efficient search of the DNN structure could improve the obtained results [18].

C. Main Experimentation

The threshold values for determining whether a loss function is descending or not, or how relevant a subnetwork is, are parameters of the NAS algorithm. In this case, they are estimated from the initial experimentation. A loss slope larger than $-10^{-10}$ is considered to be stuck, and if smaller than $-2 \times 10^{-5}$, it is determined to be steeply descending. Anything in between these two values is considered to be moderately descending. A network is considered to be relevant to an output if the performance of the VALP in that output decreases by 20% or more when it is intervened.

These values could be used as a reference for setting these threshold values in the future, taking into account the magnitudes of the problems being dealt with in each case.

D. Operator Selection

With these defined criteria, all networks within a VALP can be modified by several operators at each stage. Therefore, we define a hierarchy in which the operators are organized according to the priority they are given to modify the models.

1) Reducers: Because we pursue efficient models, any network or connection, which is not valuable for the overall performance should be deleted.
2) Aggressive Expanders: Any network which, according to the rules defined in Section VII-B, can be affected by an aggressive expander operator, is not using its full modeling potential, and this is the second priority.
3) Gentle Expanders: Giving more modeling power to a model only makes sense when all its current resources are being effectively used, and therefore, this is the last type of operator to be taken into account.

At the time of selecting the operator to be applied, within the set of operators with the highest priority, one is chosen at random. When a selected operator is not able to improve the
Fig. 8. Boxplot showing the number of points (y-axis) in the PFs generated from combining all HC runs (blue for random and orange for guided), per step (x-axis). The larger the number of points in the PF, the better the performance of the algorithm.

Fig. 9. Barplot showing the number of points (y-axis) in the combined PF from each approach (blue for random and orange for guided), per step (x-axis). The larger the number of points in the PF, the better the performance of the algorithm.

current model, it is not included among the candidate operators to generate a neighbor of the model in the next step. If no operator from those selected by this method is able to create a candidate model that improves the current solution, a random gentle operator is applied to a random network in the model.

Since we deal with multitask problems where multiple objectives have to be simultaneously optimized, the question of deciding what model is the better one is not trivial, and therefore, neither is it when comparing search algorithms. We, thus, resort to using two Pareto front-based approaches to compare the quality of the VALP structures found. Further analysis of these results can be found as part of the Supplementary Material.

For the first comparison, we take each of the 30 pairs of runs separately, considering as pairs those runs which start from the same random VALP structure and use the random and guided HC approaches. At each step, all the structures found (across the whole search) by a pair of runs are compiled into a single set, and a Pareto front is computed, considering the three outputs of the model. This way, in, for example, the fourth step, we have 30 different PFs, each being composed of at most eight points, four from each of the corresponding runs (one per completed step). Next, the points in the PF from each HC approach are counted. In Fig. 8, boxplots are presented, which display the number of points in the PFs (y-axis) by each approach (orange for the guided HC and blue for the random version), in each step (x-axis).

As can be seen in Fig. 8, in the initial 20 steps, both versions of the algorithm work similarly, with a slight advantage for the random HC. This trend changes after the 20th step, where, although the median remains similar, the top results are clearly produced by the guided version of the algorithm.

Interestingly, both the random and the guided versions have produced one run each which generates a number of points in their corresponding PFs far superior to the rest. These outliers are also higher in the guided version.

Second, we consider all 30 runs together, in order to know what algorithm is able to obtain the best results, overall. In this case, instead of constructing one PF per step and pair of runs, we simply construct a single PF from all the points found across runs limited only by the step. Again, all the found structures until a certain step is considered in that certain step. The results are shown in Fig. 9.

Although Figs. 8 and 9 look dissimilar, the information shown coincides. During the initial stages of the search (the initial 16 and 17 steps), the algorithms are searching for the best area to exploit, at which the random HC seems to outperform the guided version. This comes as no surprise, as the randomized approach does not focus on a search path to follow. Because it can perform modifications in any place within the model structure, the model can improve or lose performance continuously in different outputs. This helps a larger presence of points generated by the random HC in the PFs shown in Fig. 9, as opposed to the guided version, which focuses on improving certain aspects of the model—the efficiency of the subnetworks—before starting to seek performance improvements. That first phase ends near the 18th step, as one of the guided runs achieves one VALP configuration capable of dominating all the ones found during all searches. Slowly, other points start to form the PF, most of which belong to the guided runs. This shows the benefit of the guided search over the randomized one in the long term when performing intelligently chosen moves.
VIII. CONCLUSION

This work is framed in the NAS field. Our efforts are focused on compiling a set of guidelines, which aims at maximizing the effectiveness of the application of variation operators to model structures during a structural search procedure, illustrated using a complex scenario, that of HMTL. More specifically, we first identify several metrics which can be used to determine the level of importance of different sub-DNNs in the overall performance. Second, we compile a set of variation operators previously used in NAS procedures described in the literature and classify them according to the effect they have on the complexity and performance of the model. Next, we conduct an extensive exploratory search on how these operators affect the performance of a medium-sized model, in order to identify patterns that relate the defined metrics and the improvements in the models. These patterns are later transformed into a set of guidelines for enhancing the efficiency of future NAS searches. These guidelines add one level of sophistication to current NAS algorithms, as, contrary to the common practice of randomly selecting a variation operator, a more informed choice is made, which can save the need to evaluate DNN structures affected by the wrong operator. Finally, a comparison of the performance of the two variants of the NAS search—blind versus guided by the introduced rules—is presented as an illustration of the gains that could be obtained in NAS efficiency.

The main contribution of this work is the methodology for diagnosing the state of a model (in this case, HMTL) and identifying the relevance of its different components, and application of these metrics for more efficient NAS algorithms. One key for this goal is the set of metrics defined with this purpose, although others, which complement those introduced in this work could result in more valuable information about the model, ultimately making the processes more efficient.

The experiments conducted in this work serve as a blueprint for implementing the presented ideas to other problems and domains, as they have already served the purpose of efficiently exploring a complex search space. Although the conducted NAS runs can be considered as simple, the defined methodology is not restrained to be applied in such scenarios.

The study and application of these (or similar) methodologies to other problem definitions; e.g., NAS types (e.g., neuroevolutionary algorithms), or DNN types (single objective, convolutional DNNs, and so on); is left as future work, as are the employment of some of the metrics and operators described in this work, which were not tested in the experiments; and studying the possibility of adaptation of reinforcement learning- and gradient-based NAS approaches to the HMTL framework.

ACKNOWLEDGMENT

The authors would like to thank the support of NVIDIA Corporation for providing a Titan X Pascal GPU, which is used to train the models used in this work.

REFERENCES

[1] G. Alain and Y. Bengio, “Understanding intermediate layers using linear classifier probes,” 2016, arXiv:1610.01644.

[2] L. Arras, G. Montavon, K.-R. Müller, and W. Samek, “Explaining recurrent neural network predictions in sentiment analysis,” 2017, arXiv:1706.07206.

[3] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A deep convolutional encoder–decoder architecture for image segmentation,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 12, pp. 2481–2495, Dec. 2017.

[4] G. Bender, P. J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, “Understanding and simplifying one-shot architecture search,” in Proc. 35th Int. Conf. Mach. Learn., Jul. 2018, pp. 550–559.

[5] R. Caruana, “Multitask learning,” Mach. Learn., vol. 28, no. 1, pp. 41–75, 1997.

[6] T. Chen, L. Goodfellow, and J. Shlens, “Net2Net: Accelerating learning via knowledge transfer,” 2015, arXiv:1511.05941.

[7] T. Chen, R. Gao, F. Liu, and D. Zhao, “ModuleNet: Knowledge-inherited neural architecture search,” 2020, arXiv:2004.05020.

[8] K. Cho et al., “Learning phrase representations using RNN encoder–decoder for statistical machine translation,” 2014, arXiv:1406.1078.

[9] C. A. C. Coello, “Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art,” Comput. Methods Appl. Mech. Eng., vol. 191, pp. 1245–1287, Jan. 2002.

[10] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” Math. Control, Signals Syst., vol. 2, no. 4, pp. 303–314, Dec. 1989.

[11] Z. Ding, Y. Chen, N. Li, D. Zhao, Z. Sun, and C. L. P. Chen, “BNAS: Efficient neural architecture search using broad scalable architecture,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 9, pp. 5004–5018, Sep. 2022.

[12] Q. J. Dunn, “Multiple comparisons using rank sums,” Technometrics, vol. 6, no. 3, pp. 241–252, 1964.

[13] C. Eberhardt and R. Scheines, “Interventions and causal inference,” Philos. Sci., vol. 74, no. 5, pp. 981–995, 2007.

[14] T. Elsken, J.-H. Metzen, and F. Hutter, “Simple and efficient architecture search for convolutional neural networks,” 2017, arXiv:1711.04528.

[15] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey,” J. Mach. Learn. Res., vol. 20, no. 55, pp. 1–21, 2019.

[16] T. V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and V. François-Lavet, “Exploring the limits of language modeling,” 2016, arXiv:1312.6114.

[17] T. V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and T. Paine, “An introduction to deep reinforcement learning,” Found. Trends Mach. Learn., vol. 11, nos. 3–4, pp. 219–354, 2018.

[18] E. Galván and P. Mooney, “Neuroevolution in deep neural networks: Current trends and future challenges,” 2020, arXiv:2006.05415.

[19] U. Garciarena, N. Lourenço, P. Machado, R. Santana, and A. Mendiburu, “On the exploitation of neuroevolutionary information,” in Proc. Genetic Evol. Comput. Conf. Companion, Jul. 2021, pp. 279–280.

[20] U. Garciarena, A. Mendiburu, and R. Santana, “Automatic structural search for multi-task learning VALPs,” in Proc. Int. Conf. Optim. and Learn. Cham, Switzerland: Springer, 2020, pp. 25–36.

[21] U. Garciarena, A. Mendiburu, and R. Santana, “Towards automatic construction of multi-network models for heterogeneous multi-task learning,” ACM TKDD, vol. 15, no. 2, pp. 1–23, 2021.

[22] U. Garciarena, R. Santana, and A. Mendiburu, “Expanding variational autoencoders for learning and exploiting latent representations in search distributions,” in Proc. Genetic Evol. Comput. Conf., Jul. 2018, pp. 849–856.

[23] L. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, Cambridge, MA, USA: MIT Press, 2016.

[24] I. J. Goodfellow et al., “Generative adversarial nets,” in Proc. 27th Int. Conf. Neural Inf. Process. Syst., vol. 2. Cambridge, MA, USA: MIT Press, 2014, pp. 2672–2680.

[25] I. Harvey, “The microbial genetic algorithm,” in Proc. Eur. Conf. Artif. Life Cham, Switzerland: Springer, 2009, pp. 126–133.

[26] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 47, no. 260, pp. 583–621, 1997.

[27] D. P. Kingma and M. Welling, “Auto-encoding variational Bayes,” 2013, arXiv:1312.6114.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. NeuIPS, 2012, pp. 1097–1105.

[29] W. H. Kruskal and W. A. Wallis, “Use of ranks in one-criterion variance analysis,” J. Amer. Stat. Assoc., vol. 47, no. 260, pp. 583–621, 1952.

[30] Y. LeCun, C. Cortes, and C. Burges. (2010). MNIST Handwritten Digit Database. ATT Labs. [Online]. Available: http://yann.lecun.com/exdb/mnist
[31] W. Li, X. Zhu, and S. Gong, “Person re-identification by deep joint learning of multi-loss classification,” 2017, arXiv:1705.04724.

[32] J. Liang, E. Meyerson, and R. Miikkulainen, “Evolutionary architecture search for deep multitask networks,” in Proc. Genetic Evol. Comput. Conf., Jul. 2018, pp. 466–473.

[33] H. Liu, K. Simonyan, and Y. Yang, “DARTS: Differentiable architecture search,” 2018, arXiv:1806.09055.

[34] T. Liu, D. Tao, M. Song, and S. J. Maybank, “Algorithm-dependent generalization bounds for multi-task learning,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 2, pp. 227–241, Feb. 2017.

[35] J. Lu, W. Ma, and B. Faltings, “CompNet: Neural networks growing via the compact network morphism,” 2018, arXiv:1804.10316.

[36] Z. Lu et al., “NSGA-NET: A multi-objective genetic algorithm for neural architecture search,” 2018, arXiv:1810.03522.

[37] C. White, S. Nolen, and Y. Savani, “Local search is state of the art for NAS benchmarks,” 2020, arXiv:2005.02960.

[38] E. Meyerson and R. Miikkulainen, “Beyond shared hierarchies: Deep convolutional neural networks,” in Proc. IEEE Int. Conf. Comput. Vis. Workshops (ICCVW), Oct. 2017, pp. 2174–2182.

[39] E. Meyerson and R. Miikkulainen, “Beyond shared hierarchies: Deep multitask learning through soft layer ordering,” 2017, arXiv:1711.00108.

[40] R. Miikkulainen et al., “Evolving deep neural networks,” in Artificial Intelligence in the Age of Neural Networks and Brain Computing, Amsterdam, The Netherlands: Elsevier, 2019, pp. 293–312.

[41] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell, “A dual-stage attention-based recurrent neural network for time series prediction,” 2017, arXiv:1704.02971.

[42] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized evolution for image classifier architecture search,” in Proc. AAAI Conf. Artif. Intell., 2019, pp. 4780–4789.

[43] K. Rupp. (Aug. 2016), CPU, GPU and MIC Hardware Characteristics Over Time. [Online]. Available: http://www.karlrupp.net/

[44] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 4510–4520.

[45] R. Shwartz-Ziv and N. Tishby, “Opening the black box of deep neural networks via information,” 2017, arXiv:1703.00810.

[46] K. O. Stanley, “Compositional pattern producing networks: A novel abstraction of development,” Genet. Program. Evolvable Mach., vol. 8, no. 2, pp. 131–162, Jun. 2007.

[47] K. O. Stanley and R. Miikkulainen, “Efficient reinforcement learning through evolving neural network topologies,” in Proc. GECCO, 2002, pp. 569–577.

[48] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” Evol. Comput., vol. 10, no. 2, pp. 99–127, Jun. 2002.

[49] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, Inception-ResNet and the impact of residual connections on learning,” in Proc. AAAI Conf. Artif. Intell. Palo Alto, CA, USA: AAAI Press, 2017, pp. 4278–4284.

[50] C. Szegedy et al., “Going deeper with convolutions,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2015, pp. 1–9.

[51] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 2818–2826.

[52] T. Uriot and D. Izzo, “Safe crossover of neural networks through neuron alignment,” in Proc. Genetic Evol. Comput. Conf., Jun. 2020, pp. 435–443.

[53] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, “Show and tell: A neural image caption generator,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2015, pp. 3156–3164.

[54] B. Wang, B. Xue, and M. Zhang, “Surrogate-assisted particle swarm optimization for evolving variable-length transferable blocks for image classification,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 8, pp. 3727–3740, Aug. 2022.

[55] R. Wang, M. Cheng, X. Chen, X. Tang, and C.-J. Hsieh, “Rethinking architecture selection in differentiable NAS,” in Proc. ICLR, 2021, pp. 1–18.

[56] T. Wei, C. Wang, Y. Rui, and C. W. Chen, “Network morphism,” in Proc. ICML, 2016, pp. 564–572.

[57] C. White, S. Nolen, and Y. Savani, “Local search is state of the art for NAS benchmarks,” 2020, arXiv:2005.02960.

[58] Y. Zhang and D.-Y. Yeung, “A regularization approach to learning task relationships in multitask learning,” ACM Trans. Knowl. Discovery Data, vol. 8, no. 3, p. 12, Jun. 2014.

[59] Z. Zheng, X. Yang, Z. Yu, L. Zheng, Y. Yang, and J. Kautz, “Joint discriminative and generative learning for person re-identification,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 2138–2147.

[60] Y. Zhou, X. Xie, and S.-Y. Kung, “Exploiting operation importance for differentiable neural architecture search,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 11, pp. 6235–6248, Nov. 2022.

[61] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” 2016, arXiv:1611.01578.

[62] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 8697–8710.

Unai Garciarena received the Ph.D. degree in computer science from the University of the Basque Country (UPV/EHU), Donostia-San Sebastian, Spain, in 2021.

He has been a Lecturer with the Department of Computer Architecture and Technology, UPV/EHU, since 2022. His current research interests include generative modeling, supervised classification, and optimization.

Roberto Santana received the Ph.D. degree in mathematics from the University of Havana, Havana, Cuba, in 2005, and the Ph.D. degree in computer science from the University of the Basque Country, Donostia-San Sebastian, Spain, in 2006.

He is currently a researcher with the University of the Basque Country. His research interests include machine learning, evolutionary computation, probabilistic graphical models, and neuroscience.

Alexander Mendiburu (Member, IEEE) received the Ph.D. degree from the University of the Basque Country, Donostia-San Sebastian, Spain, in 2006.

Since 1999, he has been a Lecturer with the Department of Computer Architecture and Technology, University of the Basque Country. His current research interests include evolutionary computation, probabilistic graphical models, and parallel computing.