Applied Intelligence (2022) 52:17103–17117
https://doi.org/10.1007/s10489-021-02679-7

Designing convolutional neural networks with constrained evolutionary piecemeal training

Dolly Sapra[1] · **Andy D. Pimentel**[1]

Accepted: 9 July 2021 / Published online: 30 July 2021

Abstract

The automated architecture search methodology for neural networks is known as Neural Architecture Search (NAS). In recent times, Convolutional Neural Networks (CNNs) designed through NAS methodologies have achieved very high performance in several fields, for instance image classification and natural language processing. Our work is in the same domain of NAS, where we traverse the search space of neural network architectures with the help of an evolutionary algorithm which has been augmented with a novel approach of piecemeal-training. In contrast to the previously published NAS techniques, wherein the training with given data is considered an isolated task to estimate the performance of neural networks, our work demonstrates that a neural network architecture and the related weights can be jointly learned by combining concepts of the traditional training process and evolutionary architecture search in a single algorithm. The consolidation has been realised by breaking down the conventional training technique into smaller slices and collating them together with an integrated evolutionary architecture search algorithm. The constraints on architecture search space are placed by limiting its various parameters within a specified range of values, consequently regulating the neural network's size and memory requirements. We validate this concept on two vastly different datasets, namely, the CIFAR-10 dataset in the domain of image classification, and PAMAP2 dataset in the Human Activity Recognition (HAR) domain. Starting from randomly initialized and untrained CNNs, the algorithm discovers models with competent architectures, which after complete training, reach an accuracy of of 92.5% for CIFAR-10 and 94.36% PAMAP2. We further extend the algorithm to include an additional conflicting search objective: the number of parameters of the neural network. Our multi-objective algorithm produces a Pareto optimal set of neural networks, by optimizing the search for both the accuracy and the parameter count, thus emphasizing the versatility of our approach.

Keywords Neural networks · Neural architecture search · AutoML · Evolutionary search

1 Introduction

Neural Networks often demonstrate excellent proficiency in performing complex and challenging problems such as image classification, speech recognition and natural language processing [4, 35]. To design an optimal neural network architecture is usually considered to be a manual exercise, relying heavily on human experience and expertise. Nevertheless, recent work in Neural Architecture Search (NAS), to automate the exploration of neural networks architectures, has exhibited some highly effective methodologies. These methodologies diligently search for an efficient neural network architecture for the specific task, while demanding low human expertise and interference. The NAS algorithms that have been proposed over the last few years, cover a wide range of domains and search techniques. They vary in their definition of the search space, search strategy, or performance estimation technique [17]. Moreover, the discovered neural network architectures often outperform the manually formulated ones [7, 37, 58].

This article belongs to the Topical Collection: *Emerging topics in Applied Intelligence selected from IEA/AIE2020*
Guest Editors: Hamido Fujita, Philippe Fournier-Viger and Moonis Ali

✉ Dolly Sapra
d.sapra@uva.nl

Andy D. Pimentel
a.d.pimentel@uva.nl

[1] University of Amsterdam, Amsterdam, Netherlands

Popular NAS methodologies use evolutionary algorithms [33, 38] and reinforcement learning [37, 57], however, they need numerous GPUs consecutively, to prepare, run and converge the search process, which can consume tens to thousands of days. Many of these approaches rely heavily on resource-intensive training to direct the search algorithm, which is usually considered to be an isolated and separate task, to estimate the model performance. NAS is examined from a different viewpoint in our work, by looking at the possibility of searching for efficient architectures during a modified and extended training process, contradicting the conventional training methodology as a distinct function, required for accuracy evaluation.

Moreover, recent research in NAS approaches is chiefly focused on datasets in the image classification domain, specifically CIFAR-10 [26] and Imagenet [14], which has contrived novel and intricate search spaces typically matched to the vision based tasks. These innovative search spaces are sketched from previous hand-crafted architectures, for example, residual connections [22], cell based designs such as inception [47], dense net [24] or generated in a graph-like fashion [46]. Even though these innovative search spaces exhibit ingenuity as well as efficiency, they are complex to understand, design and train. Many tasks and domains that are of medium complexity such as human activity recognition [49], earth sciences [28, 36] and astronomical studies [11] utilize plain Convolutional Neural Networks (CNNs) in their research. They are considered sufficient and are well understood by scientists and researchers who do not come from a background in Artificial Intelligence (AI).

Therefore, a tool, which is convenient to be used by non-AI experts from various domains, and one which finds an efficient CNN in a timely manner, is vital to simplify the design process of neural networks and subsequently democratizing AI. For example, a neural network for breast cancer classification has been proposed in [55], which was manually designed and further optimized. For a non-AI expert, in the absence of relevant tools, limited knowledge about the neural network architecture design, can be a hindrance in effectively utilizing AI models in their respective domains. In this direction, the chief contribution of this work is a novel algorithm for NAS, called Evolutionary Piecemeal Training. This paper is an extension of our earlier work presented in [41], and the methodology has been appended with the ability to define multiple objectives for the search process in this paper.

Our algorithm explores the constrained design space of CNNs for the selected task and attempts to discover an efficient architecture while converging in a restrained amount of GPU hours. The CNN models that are created and altered during the search are constrained by a minimum and maximum value for each of the architecture parameters. These bounds have been set up to ensure that the size of all CNN architectures is regulated, and at the same time, it limits their potential to outgrow the availability of hardware resources. This is one of the crucial consideration for models intended to be implemented on embedded systems, for instance wearables in the Human Activity Recognition (HAR) domain [31, 32]. High computational and memory demands by large neural networks may result in inefficient utilization of the limited resources on the embedded device.

The proposed NAS technique in this work is based on a population based computation method which allows a group of neural networks to train simultaneously. During the training process, random CNNs from the population are chosen and evolutionary operators applied to them. These evolutionary operators are designed in such a way that they lead to small architecture modifications and hence guide the exploration of the larger search space. Every new architecture derived through modification is invariably partially trained, since the parent architecture was already undergoing the training. In every subsequent iteration, CNNs continue to train while some of them are subject to architecture modifications. When the algorithm converges, the best individual models (with high accuracy) are chosen from the population. These selected CNNs can be processed and trained for more epochs.

In particular, we use a genetic algorithm in our methodology, which was chosen after considering various factors from amongst different meta-heuristic algorithms, such as, Simulated Annealing and Particle Swarm Optimization. The most important factor was the ability of the algorithm to simultaneously allow the training to continue, while searching for an appropriate neural architecture. To this end, the genetic operators (mutation and crossover), can be defined in such a manner, that there is minimum disruption to the training process, while the large design space of architectures is explored. Additionally, these algorithms are well studied in the multiple-objective search domain. At the onset of this research, the potential to extend to multi-objective search was taken into consideration. Moreover, other NAS methodologies, which are based on evolutionary algorithms, prominently use the genetic algorithm in their research [33, 39].

In the current work, we perform experiments with the PAMAP2 [40] dataset, for the HAR domain, in which data is measured from body worn sensors on a person's body to anticipate the activity being performed by the human wearer. The versatility of this approach is demonstrated through the use of the CIFAR-10 dataset in the domain of image classification, markedly different from the HAR domain in terms of both input data type and format.

The methodology is further extended to consider multiple objectives for the search. To prove the efficacy of the extension to original methodology, we consider the

reduction of the number of parameters of the neural network as an additional search goal. The accuracy maximization and parameter minimization can be conflicting objectives for an efficient neural network. Smaller CNNs tend to have lower accuracy and high accuracy is generally obtained by larger neural networks. However, too many parameters tend to cause over-fitting, which may lead to poor generalization, and therefore, highlight the importance of a constrained search process not only for hardware resource usage, but also to avoid over-fitting. With multiple objective based search, selection of the best candidates is concluded through *Pareto optimization*, where any objective cannot be improved without worsening some of the other objectives. The set of candidates selected in such a fashion are collectively called a *Pareto Front*. We apply this proposed extension to the PAMAP2 dataset and perform the multi-objective search for efficient architectures. The Pareto Front obtained upon convergence sets forth the various possible CNNs to deploy on the wearable embedded device. It allows the designer to be aware of the architecture choices available in terms of which CNNs provide a trade-off between the size of CNN versus the accuracy. One of these CNNs can be strategically deployed depending upon the desired functional goals and available resources on the device.

The remainder of this paper is structured as follows. Firstly, in Section 2, recently published related works for NAS are discussed. Subsequently, Section 3 presents our methodology and concepts, and further outlines the algorithm in detail. Next, the experimental setup is explained and the results from evaluations and validations are described in Section 4. Lastly, Section 5 concludes the paper.

2 Related work

Various research works have been published recently demonstrating proficiency of NAS techniques. They can be partitioned mainly into three categories, namely Reinforcement Learning (RL) based, evolutionary algorithms and one-shot architecture search. Both reinforcement learning and evolutionary based algorithms, mandate the complete training of the neural network, at each iteration, for performance evaluation. In RL based methods [5, 37, 57], the validation performance, or accuracy, of the trained model guides the reward towards the RL agent. When the agent is continuously rewarded for finding better architectures, the search is slowly steered towards neural networks with higher performance. Any RL approach resolutely demands a suitable agent, which frequently happens be a complicated model or perhaps another neural network. The construction of the agent and its optimization involve substantial effort towards designing and subsequent fine tuning.

Evolutionary methodologies [9, 30, 38, 39, 52] utilize genetic algorithms to discover the efficient neural architecture in a large search space. Evolutionary algorithms have also been successfully deployed for CNN optimizations, such as ,for compression [25], pruning [19] and hyper-parameter optimization [45]. Evolutionary NAS algorithms work with a population of possible CNN candidates, where each one of them is trained and evaluated at every iteration. With subsequent iterations, the models in the population get selected, rejected and modified depending on their accuracy and other control variables of the algorithm. The aim is to improve the population's average performance with time, and eventually when the algorithm converges, it has discovered an architecture for a high performing neural network. Our work also utilizes an evolutionary algorithm for architecture modification, where the key difference is in the manner training and architecture modifications are interwoven in the algorithm to conduct joint search for both weights and architecture.

Unfortunately, most of these approaches demand intensive computational resources to train hundreds and thousands of neural network architectures. For example, the RL method in [57] trained more than 10,000 models, involving over a thousand GPU days, while another adept evolutionary search [16] required 56 GPU days to finally converge. Other works have utilized proxy tasks, for example, hyper-networks [6], predictors [5, 13] and controllers [37] to fasten up the search process. However, they still continue to demand abundant planning and time to be implemented before the actual search commences. In direct contrast, our algorithm does not require helpers and proxy tasks and still converges in a reasonable time.

One-Shot NAS methodologies are based on the concept of a trained super-network, which constitute all possible sub-networks within. The entire super-network may have to fit in the GPU memory during the NAS execution, which results in a highly restricted architecture size, and it typically results in a small cell with a limited amount of operations. DropPath [58] is an example of one-shot search approach, where a path is dropped out with a fixed probability, and by randomly removing different paths, a new sub-network is formed. The pre-trained super-network is then used to evaluate and eventually discover the best sub-network architecture. DARTS [29] additionally proposed an architecture parameter for every path and by employing the standard gradient descent to train the weight and architecture parameters together. Other approaches attempt to be more efficient by utilizing other proxy tasks, for instance [7] proposed a memory-efficient algorithm to update fewer paths while searching. The cell that is discovered through one-shot search, needs to be sufficiently repeated and connected in an appropriate manner, to eventually form a neural network that will perform the task

efficiently. Aside from posing a meta-architecture design challenge, the models based on replication of single cells, may not be suitable to various domains.

The deployment of neural networks in a wide range of settings requires a multi-objective perspective these days with the advent of connected and intelligent edge devices [42, 56]. These devices do not have large computation or storage capabilities. Keeping this in mind, most of the recent NAS methodologies have started to focus on other objectives other than the accuracy of the neural network, such as, number of parameters, number of mathematical operations, hardware usage (power, memory), latency, etc. The multi-objective NAS based on evolutionary algorithms have been widely explored for multi-objective search as their ability to easily incorporate an extra objective is well known [10]. LEMONADE [16] is an evolutionary based multi-objective algorithm, which utilizes the Lamarckian inheritance mechanism. NSGA-Net [30] utilizes the Non-dominated Sorting Genetic Algorithm II (NSGA-II) to construct the Pareto Front with the aim of learning the trade-off between model classification error with its computational complexity.

Other approaches for multi-objective search have also been researched in recent times, for instance, MONAS [23] and MnasNet [48] are Reinforcement Learning based approaches for multi-objective NAS. While MONAS uses validation accuracy and energy expenditure on the target model to create a pareto front, MnasNet explicitly incorporates the latency information in the main search objective to discover models with a good trade-off across accuracy and latency.

In order to deploy the one-shot architecture search in a multi-objective scenario, the objectives are incorporated into the loss function of the training process. For example, FBNet [50] and DenseNAS [18] are multi-objective one-shot differentiable NAS frameworks. In the former method, the loss function is the weighted product of cross-entropy loss, incurred during training, with the latency of the target device. Whereas in the latter work, loss function is the weighted aggregation of cross-entropy loss along with the latency-based regularization. However, it is not possible to add all objectives to the search in this manner, thereby making them inflexible to embrace an additional objective on the go.

3 Methodology

In this section, we go into the details of our proposed methodology, Evolutionary Piecemeal Training, describe its key concepts and the complete algorithm. Piecemeal training makes a reference to the training of a CNN with a small 'data-piece' randomly taken from the whole dataset, the size of which, referred to as δ, can vary from 5% to 20% of the dataset. The conventional training of a neural network is regularly interrupted through an evolutionary operator, at intervals determined by δ. The operator modifies some parameters in the model architecture, and subsequently permits the continuation of the training process. Numerous CNNs begin this training in parallel, creating a population that is subject to architecture modifications after each iteration of piecemeal training. The models that do not perform as well as other models are removed from the population. Conceptually, in the context of neural network training, this can be envisioned as the early termination of candidates showing no promise in their ability to reach a high accuracy.

3.1 Search space

All possible neural network architectures with their configurations and constraints constitute the search space for our work. More specifically, we focus on linearly connected plain CNNs, where layers are only connected to their consecutive layer, and do not have complex connectivity through residual connections or branches. We anticipate that the search methodology can be extended to more complicated search spaces. However, this particular research is focused on plain CNNs, which are used by many non-AI experts in their respective domains and may be considered sufficient for the given task [11, 36, 49].

Formally, a neural network $\widetilde{nn}$ consists of its architecture T and weights $\omega \in \mathbb{R}$.

$$\widetilde{nn} = \{T, \omega\} \tag{1}$$

Here, T is a sequence of layers, which can further be grouped into clusters of consecutively similar layers. Fig. 1 illustrates the concept of cluster formation for a simple CNN. All subsequent layers that are of same type

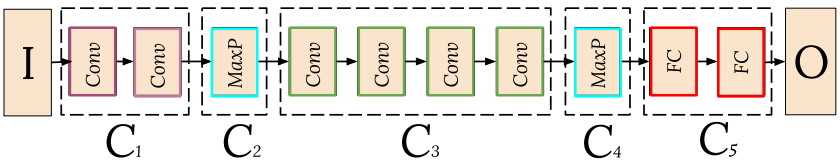


Fig. 1 Plain CNN architecture where similar consecutive layers are grouped into clusters. Conv is convolutional, MaxP is max-pool and FC is fully-connected layer respectively

are grouped in the same cluster, for instance, the first two convolutional layers are in the cluster C_1 and cluster C_5 contains only fully-connected layers. A general cluster based architecture T with l clusters, and with I and O as input and output layers respectively, can now be defined as:

$$T = \{I, C_1, C_2 ... C_l, O\}, \tag{2}$$

Figure 2 illustrates a general cluster based architecture. Additionally, every cluster in the architecture places constraints on its number of layers, on the number of units per layer, and on other layer specific parameters such as the kernel size in a convolutional layer. A cluster C_k of type C_k^{type} consisting of n layers can be represented by:

$$\begin{aligned} C_k &= \{L_{k1}, L_{k2} ... L_{kn}\} : \ \beta_k^{min} \leq n \leq \beta_k^{max} \\ \text{where, } L_{ki} &= \left\{ L | \, L \in [C_k^{type}, \ \eta_{ki}, \ p_{ki}] \right\} : \ \eta_k^{low} \leq \eta_{ki} \leq \eta_k^{up} \\ &\ni \left\{ \beta_k^{min}, \ \beta_k^{max}, \ \eta_k^{low}, \eta_k^{up} \in \mathbb{N}^+ \right\} \text{ and } p_{ki} \in \pi_k \end{aligned} \tag{3}$$

Where $(\beta_k^{min}, \beta_k^{max})$ are the bounds on the number of layers possible in the cluster, and the number of neurons in a layer are bounded by $(\eta_k^{low}, \eta_k^{up})$. Moreover, the hyper-parameters that are dependant on the layer type, p_{ki}, such as kernel size and stride, are defined by π_k in the cluster. These constraints are specific to each cluster and are independent from bounds of the other clusters. Every layer in a cluster is of same type (e.g., convolution, pooling, fully connected), and its hyper-parameters conform to the constraints placed by its parent cluster.

For our experiments, all clusters and their boundary definitions are construed before the start of the search algorithm. All possible permutations of layers and their hyper-parameters together represent the whole search space. This architecture search space is usually not a trivial space to navigate. For example, the search spaces for experiments in Section 4.2 have 10^8 (CIFAR-10) and 10^5 (PAMAP2) possible architectures. This search space definition is encoded in the form of a genotype, which ensures that not only the factory to generate new neural networks, but also the evolutionary operators, adhere to the cluster constraints.

3.2 Population based training

As mentioned before, the training process employed in this work is based on a population based approach. A population of CNN models is randomly picked from the defined search space and subsequently randomly initialized. With each iteration, every individual model in the population is piecemeal-trained with $delta$ data points. Afterwards, their performance is evaluated on the validation set, which is then engaged by the algorithm to define the next generation of the population through selection, replacement and modification functions.

While a neural network is being trained, its weights are constantly changing, and in that sense both the weights and the neural network may be considered as functions of time (i.e., iterations): $\omega(t)$ and $\widetilde{nn}(t)$. $\widetilde{nn}(0)$ is then the initial neural network at the beginning of its training with randomly initialized weights $\omega(0)$. The architecture of this neural network remains unchanged during the training. Hence,

$$\widetilde{nn}(t) = \{T, \omega(t)\} \tag{4}$$

A neural network $\widetilde{nn} \in \widetilde{NN}$, where $\widetilde{NN}$ is the set of all possible neural networks with an architecture $T \in \widetilde{T}$, where $\widetilde{T}$ is the set of all architectures defined in the search space along with its constraints. The population of neural networks may also be seen as a function of time. The population $\widetilde{P}(t)$ of CNNs at any given time can then be defined as a set of neural networks at that point,

$$\widetilde{P}(t) = \{\widetilde{nn}_1(t), \widetilde{nn}_2(t), \widetilde{nn}_3(t) ..... \widetilde{nn}_s(t)\} : \ s \in \mathbb{N}^+ \tag{5}$$

The population size, s, is constant throughout the duration of the algorithm. If a neural network is dropped from the population because it is not performing as well as the other models, then it has to be replaced by another neural network. Moreover, the population size is required to be large enough so that enough diversity is maintained among the CNN models in the population.

The algorithm runs for τ_{max} iterations, the value of which is dependent on the nature and complexity of the task. τ_{max} can be defined at the beginning of the search or can be updated during the iterations based on the rate of change of the evaluation metric, such as prediction accuracy. When the change in the evaluation metric slows down, the algorithm

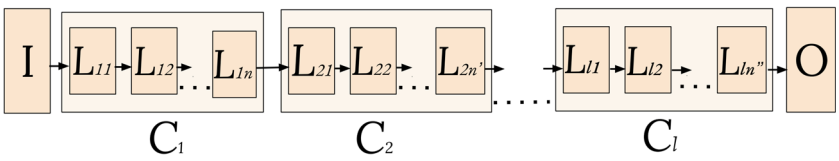


Fig. 2 A general cluster based architecture with l clusters, where each cluster defines its layer type along with the constraints it enforces on member layers

is highly likely to be near convergence and therefore the algorithm can halt further iterations.

At any time t, the population is then defined as,

$$\forall\, 1 \leq t \leq \tau_{max},\ \widetilde{P}\,(t) = f_{ept}(\widetilde{P}\,(t-1)) \tag{6}$$

The function $f_{ept}()$ is applied to the population during every iteration. In absence of the evolutionary architecture modifications, $f_{ept}()$ consists of only the training function $f_{train}()$, which trains a CNN with a small subset of training data. It is possible to train all neural networks in an iteration, in any order of sequential and parallel executions, to best utilize the computational resources available.

3.3 Evolutionary operators

The evolutionary algorithm needs to sufficiently explore the huge search space to ensure that a good model with high performance can be discovered in a reasonable time. To this cause, at each evolutionary step, architectures of some of the models in the population are modified through one of the evolutionary operators, i.e. the recombination or mutation operator. While mutation does small changes to one layer at a time, recombination exchanges some layers in one model to another to create significantly different models. In this respect, the mutation operator explores the search space closer to the existing population, and in contrast, the recombination operator explores a wider design space by generating distant architectures. The number of evolutionary operators executed in each iteration is controlled through a pre-defined mutation rate (P_m) and recombination rate (P_r).

Mutation operates on a CNN and randomly selects one layer to change one of its hyper-parameters, such as the number of kernel units in the layer or the kernel size. We employ the Net2Wider operator from [8] to broaden the layer by increasing the number of kernel units. On the other hand, to shrink the layer, we use a pruning process [27], which reduces the number of units by removing the least significant kernels in terms of their activation weights. Kernels are radially zero-padded or cropped from the outer edge, when their size changes because of mutation.

Furthermore, the mutation operator is devised to be function-preserving in nature to make sure that mutation does not disrupt the ongoing training of the neural networks. Any change to the architecture will invariably cause an additional loss in the training process. The functions in mutation operator were particularly chosen since these are either totally, or at the minimum, partially function-preserving, implying that the loss drawn from these operators is as minimum as possible and recovers quickly during later piecemeal-training iterations.

In direct contrast to mutation, the recombination operates on two neural networks and swaps all their layers in a cluster. The swap is carried out for only one randomly selected cluster position. Figure 3 shows an example of the recombination operator, which swaps different numbers of layers from the clusters C_2 in two different neural networks. Since all models have exactly the same number of clusters, it follows that the layers that are exchanged are approximately in the same stage of neural computation, and hence the new models need minimum repair to the architecture to remain valid.

It is important to note that the recombination is not a function preserving operator. However, they are required in the algorithm to introduce and maintain diversity [43] by introducing significantly varied models into the population. This is achievable due to the fact that the total number of layers being swapped is not the same. To diminish the adverse effect of loss incurred by the recombination operator, a cooling-down approach is applied to the recombination rate. During the early iterations, when the training loss has not yet started to converge and is at a high value, more swaps are allowed as compared to later iterations, when the training loss is low.

Together, these evolutionary operators are responsible for traversing the large design space of neural network architectures in an efficient manner. Additionally, these operators are responsible for making sure that the cluster constraints are always adhered to. The mutation operator never allows a layer to expand or narrow beyond the cluster defined boundaries, and it also ensures that other layer hyper-parameters conform to the cluster specifications as well. The recombination operator swaps clusters which

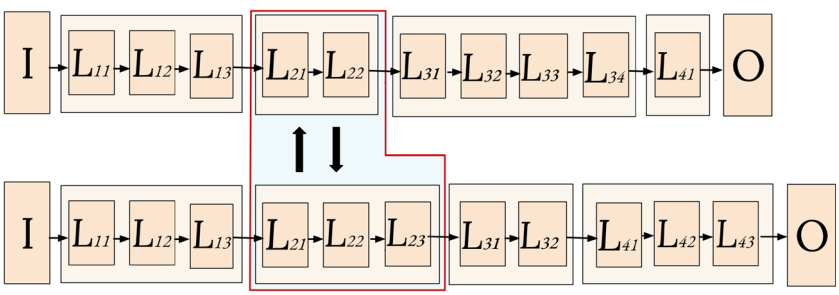


Fig. 3 Recombination operator applied to two neural networks, where the cluster C_2 gets swapped. The cluster has a different number of layers in each model

are already within their bounds, thus maintaining the constraints.

3.4 Selection and replacement

One of the most important features of the population based evolutionary approach is that every subsequent population attempts to be better than the one in the previous iteration. This is achieved through selection and replacement policies geared towards retaining the better performing candidates at every step. A remove-worst strategy is employed to select the next generation of the population. However, the rejection rate is kept relatively low, around 2-5% of the total population. To keep the population size constant, individuals are selected and put back in the population using a non-elitist random selection policy. This means that every neural network in the population has an equal chance of being selected to replace the worst performing model.

To summarize, if $f_{evo}()$ represents the evolutionary operator function and $f_{select}()$ represents the population selection function then, $f_{ept}()$ from (6), for population $\widetilde{P}(t)$ at any iteration t, can be defined as

$$f_{ept}\left(\widetilde{P}(t)\right) = f_{train} \circ f_{select} \circ f_{evo}\left(\widetilde{P}(t)\right) \tag{7}$$

The function $f_{ept}()$, as composition of other mentioned functions, defines the transition function of the population from one iteration to the next. $f_{train}()$ trains every CNN in the population, $f_{select}()$ evaluates the population and selects (or rejects) suitable candidates, some of which then go through evolutionary operators in $f_{evo}()$.

As the iterations continue, the population keeps gradually changing, due to architecture alterations, along with appropriate selection and replacement of models.

3.5 Optimization objective

The main objective of our algorithm is to find a neural network with maximum accuracy possible. Let $Acc(\widetilde{nn}(T, \omega(t))$ represent the accuracy of a neural network $\widetilde{nn}$. Given $\widetilde{T}$ as the set of all possible architectures and $\widetilde{NN}$ as the set of all possible neural networks, the objective is to find neural network $\widetilde{nn}'$ ($\widetilde{nn}' \in \widetilde{NN}$) with architecture ($T' \in \widetilde{T}$), such that

$$max_{\widetilde{nn}' \in \widetilde{NN}}\ Acc(\widetilde{nn}'(T', \omega(t)) \tag{8}$$

The optimization objective helps to formulate the $f_{select}()$ function to which the whole population is subject to after every iteration. To maximise accuracy, $f_{select}()$ chooses the best performing CNNs, able to reach higher accuracy in the population.

It is possible to extend our algorithm for multi-objective search, and we demonstrate the concept by adding minimization of the number of parameters as another objective. Let $Params(\widetilde{nn}(T, \omega(t))$ represent the number of parameters of a neural network $\widetilde{nn}$. Then, the objective is to find neural network $\widetilde{nn}''$ with architecture ($T'' \in \widetilde{T}$), such that

$$min_{\widetilde{nn}'' \in \widetilde{NN}}\ Params(\widetilde{nn}''(T'', \omega(t)) \tag{9}$$

When the optimization objectives are conflicting with each other, as is the case with accuracy maximization and parameter minimization, $f_{select}()$ can not be as simple as selecting the best candidates based on linear sorting on one specification. The selection function now has to consider a sorting based on non-domination of any single objective and select the best candidates. We use the NSGA-II selection algorithm [12] in our work to make sure that both optimization objectives are catered for during the search.

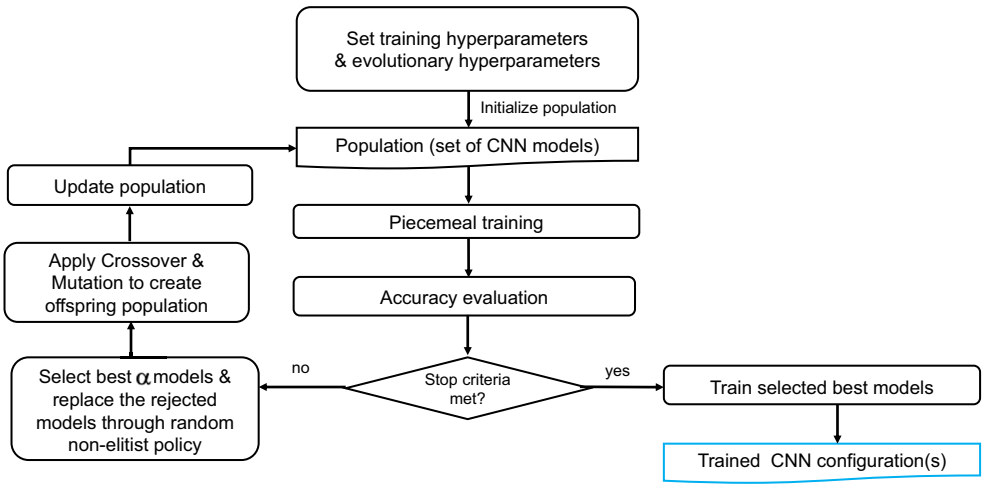


Fig. 4 Workflow for the evolutionary piecemeal training

3.6 Workflow and algorithm

We assemble all the concepts described and consolidate them to present an overview of the workflow (Fig. 4), together with the complete algorithm (Algorithm 1). The algorithm begins with the initial set up of the configuration parameters for both training and evolutionary operators. The evolutionary inputs are, N_g: the number of iterations, N_p: the population size, P_r: Recombination rate, P_m: Mutation rate and α: the selection policy for each iteration. Additionally, the training parameters τ_{params} including optimizer choice, learning rate, batch size etc., and δ to determine the size of the subset of data to be used for piecemeal training are provided. All the evolutionary and training parameters are empirically selected after conducting a small number of initial experiments, in order to fine-tune the algorithm. The selection of evolutionary parameters is mainly guided by available computing resources, along with the complexity of the task. Whereas the training parameters are determined through a small grid search by training a few architectures from the population prior to commencement of the whole algorithm. These parameters stay constant throughout the algorithm, unless specifically stated.

The population is generated using the genotype, which represents the search space, and contains all the information about clusters and their constraints. $InitializePopulation()$ creates the population $\wp_o$, of N_p neural networks, and their initialization can be random or through training for a few epochs.

Algorithm 1 Evolutionary piecemeal training.

Evolutionary Inputs: $N_g, N_p, P_r, P_m, \alpha$
Training Inputs : τ_{params}, δ
$\wp_o \leftarrow InitializePopulation(N_p)$
for $i \leftarrow 0 .... N_g$ **do**
  $\wp_i \leftarrow PiecemealTrain(\wp_i, \tau_{params}, \delta)$
  $E_v \leftarrow EvaluateAccuracy(\wp_i)$
  $\wp_{best} \leftarrow BestSelection(\alpha, \wp_i, E_v)$
  $\wp_r \leftarrow random((1-\alpha) * \wp_i)$
  update $\wp_i \leftarrow \wp_{best} + \wp_r$
  $\wp_{rc} \leftarrow Recombine(\wp_i, P_r)$
  $\wp_{mu} \leftarrow Mutate(\wp_i, P_m)$
  update $\wp_i \leftarrow \wp_{mu} + \wp_{rc} + \wp_{remaining}$
end
$E_v \leftarrow EvaluateAccuracy(\wp_{N_g})$
return $BestCandidates(E_v)$

Once the start set-up is complete, the iterative core of the algorithm is initiated and this iterative algorithm runs for N_g generations. The population at every i^{th} iteration is referred to as $\wp_i$. Firstly, the function $PiecemealTrain()$ trains all individuals in the population $\wp_i$, with the random subset of the data. Next, $EvaluateAccuracy()$ evaluates the accuracy of every model in the population, based on which, $BestSelection()$ selects α best individuals found so far ($\wp_{best}$), from the whole population . To keep rejection rate low, α is chosen to be a high ratio of $> 0.95 * N_p$, which is important to keep the focus on removing the poor performing architectures gradually from the population. This approach discourages the promotion of a model that is able to learn fast but is unable to finally reach a high accuracy. Afterwards, to keep the population size constant, $1-\alpha$ neural networks ($\wp_r$), are randomly selected from survivors. The population $\wp_i$ is updated by replacing the population with $\wp_{best}$ and $\wp_r$. Random selection makes sure that subsequent generations do not get crowded with only one parent architecture, which got higher accuracy by chance due to the stochastic nature of training. The evolutionary operators, $Recombine()$ and $Mutate()$, select individuals ($\wp_{rc}$ and $\wp_{mu}$), with probability of P_m and P_r respectively, from the population to alter some of the neural network architectures. P_r is linearly cooled to ≈ 0 from its initial value. The population is then updated with modified neural networks from $\wp_{rc}$ and $\wp_{mu}$, while the part of the population not undergoing any modification ($\wp_{remaining}$) remains unchanged for the next iteration of the algorithm.

After the iterations conclude, the algorithm evaluates all the remaining models in the final population and returns the best neural networks determined. These best models are processed and modified, if needed, and further trained to achieve final CNN configurations. Other hyper-parameter optimization techniques [53] can be utilized to find the optimal training parameters, in order to train the CNNs at this stage.

Furthermore, this methodology is extended to be suitable for multiple objectives besides the accuracy. Figure 5 highlights the changes done to the primary workflow, for the multi-objective search. Initialization and piecemeal-training steps remain the same, however, additional evaluations are performed to deduce the number of parameters of the model. Parameter minimization now becomes the second search objective, to be considered along with accuracy maximization for the search. The selection policy is updated to be based on non-dominated sorting [12], which takes into account all the objectives to sort the models in the order of their relative quality of performance. Replacement policy, evolutionary operator application and population update steps are unchanged in the extended algorithm.

After the iterations are complete, a Pareto optimal set is selected from the population based on all evaluated objectives. The CNNs in this Pareto optimal set, also called as Pareto Front, are selected to be eventually completely trained. All the models in the Pareto set are considered to be equally adequate to be marked as the best model. In this scenario, the final selection lies in the hands of system

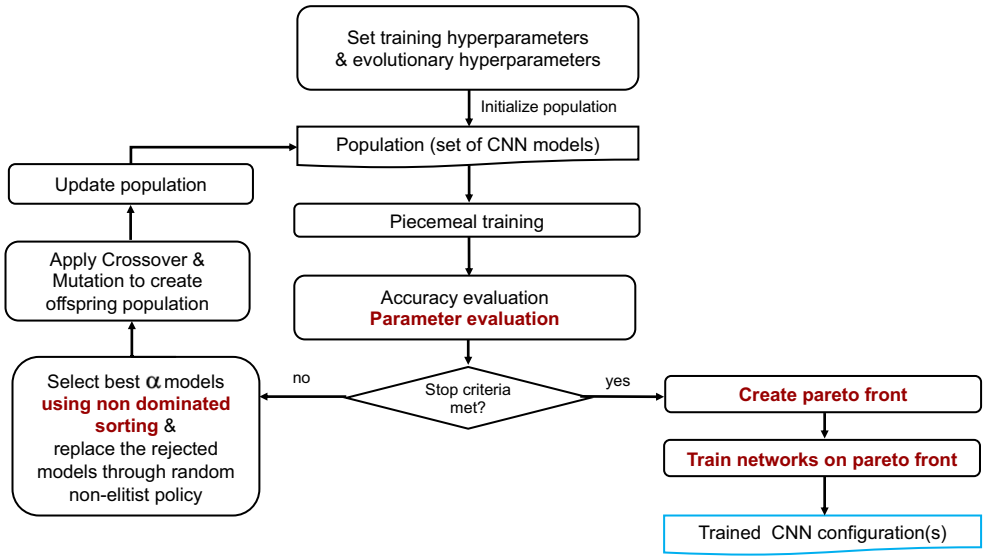


Fig. 5 Workflow for the Evolutionary Piecemeal Training, extended version to include multiple objectives

designer, and may also be based on higher priority placed on one of the objectives.

We outline the modified and extended algorithm for multi-objective search in Algorithm 2. The changes to the original Algorithm 1, reflect the same changes that were highlighted in the multi-objective workflow in Fig. 5. $EvaluateParameters()$ evaluates the number of parameters of CNNs in the population. $NSGA2Selection()$ replaces the previous selection algorithm with α still kept at very high value.This function is based on the popular multi-objective selection algorithm, NSGA-II [12], which takes all objectives into account when selecting the best individuals. This algorithm returns the *Pareto Front* from the population.

Algorithm 2 Multi-objective evolutionary piecemeal training.

Evolutionary Inputs: $N_g, N_p, P_r, P_m, \alpha$
Training Inputs : τ_{params}, δ
$\wp_o \leftarrow InitializePopulation(N_p)$
for $i \leftarrow 0 .... N_g$ **do**
  $\wp_i \leftarrow PiecemealTrain(\wp_i, \tau_{params}, \delta)$
  $Ev_{acc} \leftarrow EvaluateAccuracy(\wp_i)$
  $Ev_{params} \leftarrow EvaluateParameters(\wp_i)$
  $\wp_{best} \leftarrow NSGA2Selection(\alpha, \wp_i, Ev_{acc}, Ev_{params})$
  $\wp_r \leftarrow random((1-\alpha) * \wp_i)$
  update $\wp_i \leftarrow \wp_{best} + \wp_r$
  $\wp_{rc} \leftarrow Recombine(\wp_i, P_r)$
  $\wp_{mu} \leftarrow Mutate(\wp_i, P_m)$
  update $\wp_i \leftarrow \wp_{mu} + \wp_{rc} + \wp_{remaining}$
end
$Ev_{acc} \leftarrow EvaluateAccuracy(\wp_{N_g})$
$Ev_{params} \leftarrow EvaluateParameters(\wp_{N_g})$
return $ParetoFront(Ev_{acc}, Ev_{params})$

4 Experiments

In this section, the experimental setup is presented in addition to the algorithm's evaluations using two datasets. We describe features and format of both the datasets, their respective search spaces with the constraints and the results achieved. We have utilized the Java based Jenetics library [1] for evolution based operators and computation, while the Python based Caffe2 [3] library was used for the training and accuracy evaluation. The neural networks were represented in the ONNX [2] format, which combines architecture and weights in one file, and facilitates the storage and transfer of the CNNs across different modules. All our experiments were performed on a single GeForce RTX 2080Ti GPU.

4.1 Datasets

In the experiments, the CIFAR-10 dataset for image classification and the PAMAP2 dataset of human activity recognition have been used. CIFAR-10 is a labelled set of 60,000 images, bifurcated into train and test sets in the ratio of 5 : 1. The images are of size $32 \times 32 \times 3$ and are divided into 10 classes. We reserved 5,000 images from the training set, to use them for validation during the search process. The test set was eventually used to evaluate the final accuracy of the neural network, which is what we report in this paper. Standard data augmentation techniques were deployed [20, 44], which include small amounts of translation, crop, rotation and horizontal flip. During the piecemeal-training process, an image set of size δ was randomly chosen from the training set, out of which approximately 50% were subjected to the augmentation pipeline.

Table 1 CIFAR-10 architecture search space

Cluster type	Layers		Units/layer		Kernel-size		Stride
	β^{min}	β^{max}	η_{low}	η_{up}	K_{min}	K_{max}	S_t
Convolution	2	5	48	96	3x3	7x7	1
MaxPool	1	1	1	1	2x2	2x2	2
Convolution	2	7	80	320	3x3	7x7	1
MaxPool	1	1	1	1	2x2	2x2	2
Convolution	2	7	256	640	3x3	7x7	1
MaxPool	1	1	1	1	2x2	2x2	2
Fully Connected	2	3	128	1024	–	–	–
SoftMax	1	1	1	1	–	–	–

The markedly different PAMAP2 dataset compiles recordings from body-worn sensors. The input is organised as time-series data from a total of 40 channels from three Inertial Measurement Units (IMU) along with a heart rate monitor. The person wearing these sensors performed one of the twelve different activities in everyday life. We ignored some of the optional activities provided in the dataset. For a fair comparison with other papers, the validation and test sets were same as in [21, 51], i.e. the recordings from participants 5 and 6 were used as validation and test sets respectively. To prepare the data, firstly, the recordings from all IMUs were downsampled to 30 Hz and secondly, the data was segmented through a sliding window approach, with a window size of 3s (100 samples) and step size of 660ms (22 samples). For appropriate data augmentation, we moved the sliding window using different step sizes while the window size was kept the same at 3s.

The experiment for the extended version of the algorithm with multiple objectives for the search has been performed on the PAMAP2 dataset only. The definition of the architecture search space, and initialization of hyper-parameters for both evolutionary operators and training is exactly the same as the original algorithm.

4.2 Search space

We outline the search space specifications for CIFAR-10 and PAMAP2 in Tables 1 and 2 respectively. In the CIFAR-10 experiments, the number of kernel units per layer were multiples of 16, which brings the total number of models in the search space to the order of 10^8. For PAMAP2, the number of kernels per layer are multiples of 8 and the total design points are to the order of 10^5.

4.3 Training setup

The CIFAR-10 dataset was trained for 80 generations, while the population size was kept at 80. The data size, δ, was set to 4,000 images to be used by the piecemeal-training. Every convolution and fully connected layer was appended by ReLu activations. Training was performed with the Adam optimizer, while the batch size was set at 80. Initial learning rate was set to $5e^{-4}$, with a step learning rate decay policy where the learning rate was reduced by $1e^{-4}$ at the interval of 20 iterations. The evolutionary selection probabilities P_m and P_r were both set to 0.3 at the beginning. P_m stayed constant, whereas P_r was linearly reduced to reach 0.01 at the last iteration.

For the second dataset, PAMAP2, training was done during 30 generations with a population size of 50. The data size, δ, for piecemeal-training was set to 20,000 samples. ReLu activations follow every convolution and fully connected layer. The training was performed with the Adam optimizer, the batch size of 100 , and a constant learning rate of $1e^{-4}$. Evolutionary selection probabilities P_m and P_r were both initialized as 0.3. Similar to the CIFAR-10 experiment, P_m is kept constant, whereas P_r is linearly reduced.

Table 2 PAMAP2 architecture search space

Cluster type	Layers		Units/layer		Kernel-size		Stride
	β^{min}	β^{max}	η_{low}	η_{up}	K_{min}	K_{max}	S_t
Convolution	2	4	64	128	3x1	7x1	1
MaxPool	1	1	1	1	2x1	2x1	2
Convolution	2	5	96	256	3x1	7x1	1
GlobalMaxPool	1	1	1	1	2x1	2x1	2
Fully Connected	1	3	128	512	–	–	–
SoftMax	1	1	1	1	–	–	–

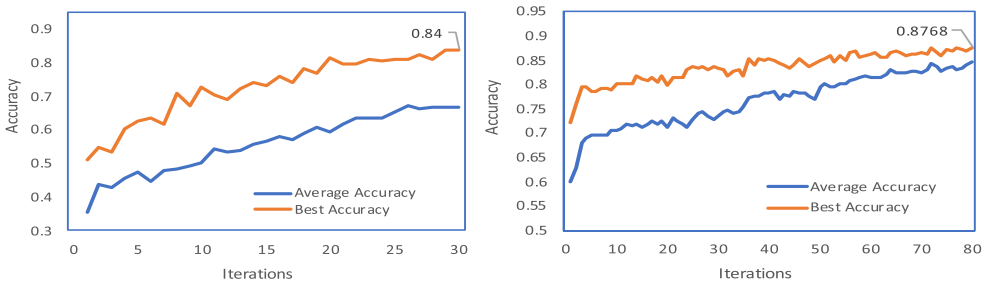


Fig. 6 Training curves. Average accuracy refers to the average performance of the whole population at the given search iteration. Best accuracy refers to best performance of any individual model in the population

(a) Training curve for PAMAP2

(b) Training curve for CIFAR-10

The CIFAR-10 models are substantially more memory consuming than the PAMAP2 models, which limits the amount of parallelism for training, on a single GPU with 11 GB memory. For CIFAR-10, 4 parallel training threads could execute, while 7 simultaneous threads for PAMAP2 could run. The limit on the level of parallel executions was governed by the GPU memory available. Additionally, no Batch Normalization was used in order to fasten up the search, since it consumes more memory and therefore reduces the parallelism. Once the search finished, the best model found was altered, to have a batch normalization layer following every convolutional layer and was trained for 100 epochs more.

4.4 Results

This section presents the results obtained through the algorithm execution, which we then compare with the state-of-the-art. We show the results of the extended multi-objective search algorithm for PAMAP2 dataset afterwards.

The training curves for experiments on PAMAP2 and CIFAR-10 are depicted in Fig. 6, where accuracy maximization is the only objective in consideration. As the iterations continue, it can be observed that the average accuracy of the whole population generally increases, despite architecture modifications interrupting the training. The best accuracy of any individual in the population is similarly increasing gradually with each iteration. The best model in one iteration may be different from the best one in the next iteration. The best model(s) discovered at completion was trained further for more epochs to achieve the accuracy that is reported in Tables 3 and 4.

The first experiment using the CIFAR-10 dataset consumed 2-GPU days and reached the best prediction accuracy of 92.5% on the test set. Table 3 compares our results with other evolutionary based NAS approaches. We understand that when we compare the accuracy of 92.5% to other published works, it ranks slightly lower than the other efforts, however, the key difference is that the architecture space is defined for plain CNNs. There is a marked omission of architectural enhancements such as residual connections and cells in our architecture search space. In addition, advanced data augmentation like mixup [54] or cutout [15] were not deployed either. Other approaches commonly use a hybrid search space, which may include different cell modules or architecture block along with arbitrary residual connections.

Despite the lower accuracy, we emphasize the shorter convergence time, of only 2 GPU-days, when compared with other evolutionary NAS methodologies. The best CNN found by the search algorithm had 13 convolutional layers with addition of 2 fully connected layers as summarized in Fig. 7a.

The PAMAP2 dataset is an unbalanced dataset, which means that some of the classes are over-represented in the data set. For this reason, we report not only the classification

Table 3 CIFAR-10 accuracy comparisons with evolutionary approaches

Model	Search space	GPU-days	Accuracy(%)
CoDeepNeat [33]	hybrid	–	92.7
GeneticCNN [52]	hybrid	17	92.9
EANN-Net [9]	hybrid	–	92.95
AmoebaNet [38]	cell	3150	**96.6**
NSGANet [30]	hybrid	8	96.15
Evolution [39]	hybrid	1000+	94.6
EPT (ours)	plain CNN	**2**	92.5

Bold entries highlight the best value in the respective column (low GPU days are better and high accuracy/F1 is better)

Table 4 PAMAP2 accuracy comparisons

Model	Accuracy(%)	$\mathbf{F1_w}$(%)	$\mathbf{F1_m}$(%)
Hand Designed [34]	93.13	93.21	–
Grid Search (CNN) [21]	–	–	93.7
D^2C [51]	–	–	92.71
D^2CL [51]	–	–	93.2
EPT (ours)	**94.36**	**94.17**	**94.36**

Bold entries highlight the best value in the respective column (low GPU days are better and high accuracy/F1 is better)

accuracy, but also the weighted F1-score ($F1_w$) and mean F1-score ($F1_m$). These scores are computed using precision and recall for each class, and weigh the classification of each class based on the ratio of class represntation in the dataset. These are computed as:

$$F1_w = \sum_i 2 \times \frac{n_i}{N} \times \frac{precision_i \times recall_i}{precision_i + recall_i}$$

$$F1_m = \frac{2}{N} \times \sum_i \frac{precision_i \times recall_i}{precision_i + recall_i}$$

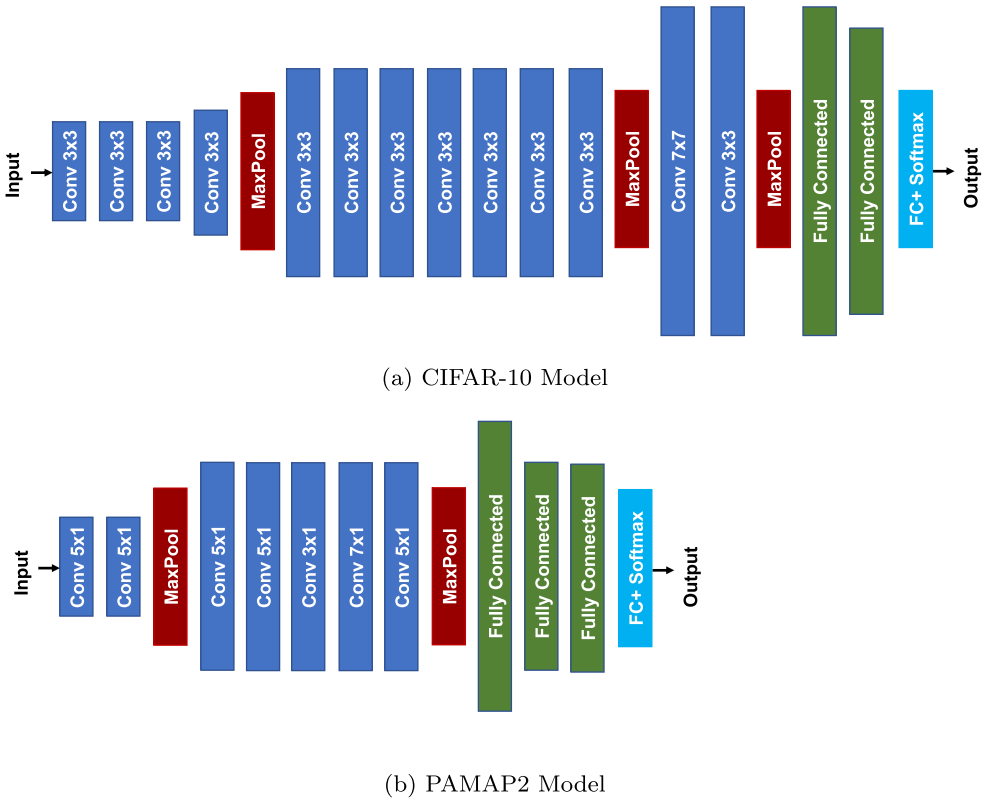
Here, n_i is the number of samples per class and N represents the number of data points in the whole dataset. In comparison to the classification accuracy, especially for unbalanced datasets, the F1-scores provide a better overview about the performance of a neural network. We compute both presented F1-scores in order to compare the result with other published works.

Our algorithm was able to achieve impressive results on the PAMAP2 dataset. The search took 10 GPU-hours, while the best neural network discovered after complete training was able to reach a prediction accuracy of 94.36%. Table 3 compares our algorithm's results against other published works. In direct comparison, the grid search [21] on neural networks for PAMAP2 was able to reach their best at 93.7% and another hand-crafted model [34] achieved 93.21%. These results clearly demonstrate that our methodology is more effective than the naive algorithms that involve simple approaches such as random search or grid search. The best neural network found had 7 convolutional layers and 3 fully connected layers as shown in Fig. 7b.

Additional experiments were performed with the extended multi-objective search algorithm for the PAMAP2 dataset. The conflicting search objectives were to maximize accuracy while simultaneously minimize the number of parameters of the CNN model. Once the algorithm has finished all the iterations, we plot the graph for accuracy versus the number of parameters for all models in the population, as shown in Fig. 8a. The Pareto Front as selected by the algorithm is marked in red. The candidates that lie on the Pareto Front exhibit the trade-off between two objectives and one cannot be considered better over the other w.r.t both the objectives.

Subsequently, all the candidates on the Pareto Front were processed and trained further, after the addition of the Batch Normalization layers. Figure 8b now presents two Pareto Fronts, first one being the Pareto Front determined by the algorithm (in red), and the second curve (in green) depicts the associated "trained" Pareto Front. The latter curve plots the accuracy for the same neural networks on the former curve, but as completely trained models.

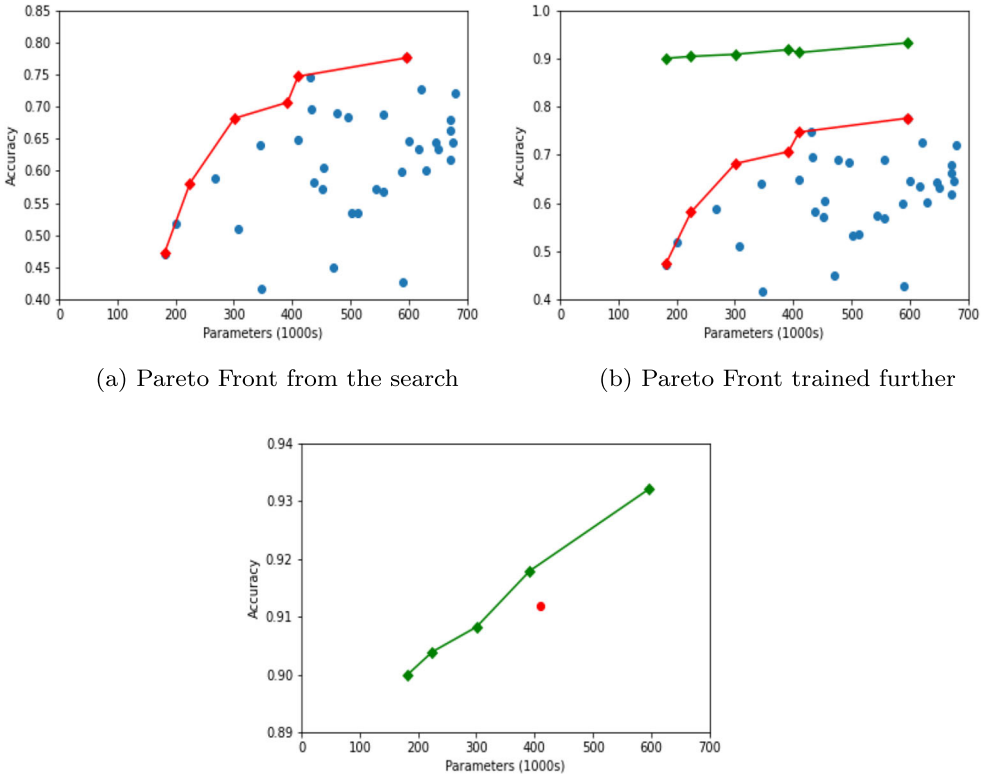
However, once the training is complete, it is highly probable that some of the models do not follow the rules of a Pareto optimal set anymore. Figure 8c shows a closer look at the "trained" Pareto Front, where the point marked in red clearly does not belong to the Pareto Front any longer. Those



(a) CIFAR-10 Model

(b) PAMAP2 Model

Fig. 7 Best neural networks found for (a) CIFAR-10 dataset and (b) PAMAP2 dataet. Every Convolutional layer is followed by Batch Normalization and ReLu activation layer



(a) Pareto Front from the search

(b) Pareto Front trained further

(c) Updated final Pareto Front

Fig. 8 Pareto Fronts for accuracy vs parameters of CNNs. Figure 8a shows the Pareto Front created during the search with EPT. The scattered points represent CNNs from the final population upon convergence. The candidates from Pareto Front are further trained and Fig. 8b shows the "trained" Pareto Front. The Pareto Front is finally updated (Fig. 8c) to remove points that do not fall on it anymore

points are then removed from consideration and eventually a final Pareto Front is deduced.

The Pareto Front that is finally achieved gives a good indication of the trade-off between size of a neural network versus the performance. In the Pareto Front for PAMAP2, there were 5 points, ranging from 89.99% accuracy to 93.34% accuracy with models consisting of $\approx$ 200k to $\approx$ 600k parameters. This gives a designer an important tool to pick the most suitable neural network to deploy on the wearable device. When the memory is limited on the hardware platform, a designer may consider it to be acceptable to use a model with slightly lower accuracy, specially where lower memory footprint may also result in better response time of the device.

5 Conclusion

A novel methodology for NAS, Evolutionary Piecemeal Training, was presented in this paper. For a given task, our algorithm traverses the search space of plain CNNs with the aim of discovering an efficient neural network architecture with reasonable constraints.The methodology was validated on two markedly different datasets, to demonstrate that our approach is versatile and can be adapted to suit distinct domains. Moreover, we illustrated that for moderate complexity tasks, for example the PAMAP2 dataset, our algorithm outperforms random or grid search methodologies. The flexibility of our algorithm was further demonstrated, by extending it to perform multi-objective optimization, which allows the neural network to be optimized simultaneously for multiple task specific objectives, such as parameter minimization and accuracy maximization.

In the future, we aim to modify this search algorithm to incorporate hardware metrics and optimize the resource usage of the neural network on the designated hardware. We envision that the Pareto Front obtained for multiple hardware-specific objectives will allow the designer to have better design choices and more flexibility, in switching from one CNN to another systematically, for both the given task and the hardware. Additionally, we plan to modify the search space to include complex and cell-based neural architectures for further research.

Acknowledgements This work has received funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No. 780788.

References

1. (2019) Jenetics library. https://jenetics.io/
2. (2019) Onnx: Open neural network exchange formet. https://onnx.ai/
3. (2019) Pytorch: An open source deep learning platform. https://pytorch.org/
4. Alam M, Samad MD, Vidyaratne L, Glandon A, Iftekharuddin KM (2020) Survey on deep neural networks in speech and vision systems. Neurocomputing 417:302–321
5. Baker B, Gupta O, Raskar R, Naik N (2017) Accelerating neural architecture search using performance prediction. arXiv:1705.10823
6. Brock A, Lim T, Ritchie JM, Weston N (2017) Smash: one-shot model architecture search through hypernetworks. arXiv:1708.05344
7. Cai H, Zhu L, Han S (2019) ProxylessNAS: Direct neural architecture search on target task and hardware. In: International conference on learning representations
8. Chen T, Goodfellow I, Shlens J (2015) Net2net: Accelerating learning via knowledge transfer. arXiv:1511.05641
9. Chen Z, Zhou Y, Huang Z (2019) Auto-creation of effective neural network architecture by evolutionary algorithm and resnet for image classification. In: 2019 IEEE international conference on systems, man and cybernetics (SMC). IEEE
10. Coello CAC, Lamont GB, Van Veldhuizen DA et al (2007) Evolutionary algorithms for solving multi-objective problems, vol 5. Springer, New York
11. Davies A, Serjeant S, Bromley JM (2019) Using convolutional neural networks to identify gravitational lenses in astronomical images. Mon Not R Astron Soc 487(4):5263–5271
12. Deb K, Agrawal S, Pratap A, Meyarivan T (2000) A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In: International conference on parallel problem solving from nature. Springer
13. Deng B, Yan J, Lin D (2017) Peephole: Predicting network performance before training. arXiv:1712.03351
14. Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L (2009) Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. IEEE
15. DeVries T, Taylor GW (2017) Improved regularization of convolutional neural networks with cutout. arXiv:1708.04552
16. Elsken T, Metzen JH, Hutter F (2019) Efficient multi-objective neural architecture search via lamarckian evolution. In: International conference on learning representations
17. Elsken T, Metzen JH, Hutter F et al (2019) Neural architecture search: A survey. J Mach Learni Res 20(55)
18. Fang J, Sun Y, Zhang Q, Li Y, Liu W, Wang X (2020) Densely connected search space for more flexible neural architecture search. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition
19. Fernandes Jr FE, Yen GG (2021) Pruning deep convolutional neural networks architectures with evolution strategy. Inform Sci 552:29–47
20. Goodfellow IJ, Warde-Farley D, Mirza M, Courville A, Bengio Y (2013) Maxout networks. arXiv:1302.4389
21. Hammerla NY, Halloran S, Plötz T. (2016) Deep, convolutional, and recurrent models for human activity recognition using wearables. arXiv:1604.08880
22. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition
23. Hsu CH, Chang SH, Liang J, Chou HP, Liu CH, Chang SC, Pan JY, Chen Y, Wei W, Juan DC (2018) Monas: Multi-objective neural architecture search using reinforcement learning. arXiv:1806.10332
24. Huang G, Liu Z, Van Der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition
25. Huang J, Sun W, Huang L (2020) Deep neural networks compression learning based on multiobjective evolutionary algorithms. Neurocomputing 378:260-269. https://doi.org/10.1016/j.neucom.2019.10.053, https://www.sciencedirect.com/science/article/pii/S092523121931433X
26. Krizhevsky A, Hinton G et al (2009) Learning multiple layers of features from tiny images. Tech. rep. Citeseer
27. Li H, Kadav A, Durdanovic I, Samet H, Graf HP (2016) Pruning filters for efficient convnets. arXiv:1608.08710
28. Ling F, Boyd D, Ge Y, Foody GM, Li X, Wang L, Zhang Y, Shi L, Shang C, Li X, e al (2019) Measuring river wetted width from remotely sensed imagery at the sub-pixel scale with a deep convolutional neural network. Water Resourc Res 55(7):5631–5649
29. Liu H, Simonyan K, Yang Y (2018) Darts: Differentiable architecture search. arXiv:1806.09055
30. Lu Z, Whalen I, Boddeti V, Dhebar Y, Deb K, Goodman E, Banzhaf W (2018) Nsga-net: a multi-objective genetic algorithm for neural architecture search. arXiv:1810.03522
31. Meloni P, Loi D, Busia P, Deriu G, Pimentel AD, Sapra D, Stefanov T, Minakova S, Conti F, Benini L et al (2019) Optimization and deployment of cnns at the edge: the aloha experience. In: Proceedings of the 16th ACM international conference on computing frontiers
32. Meloni P, Loi D, Deriu G, Pimentel AD, Sapra D, Moser B, Shepeleva N, Conti F, Benini L, Ripolles O et al (2018) Aloha: an architectural-aware framework for deep learning at the edge. In: Proceedings of the workshop on INTelligent embedded systems architectures and applications
33. Miikkulainen R, Liang J, Meyerson E, Rawal A, Fink D, Francon O, Raju B, Shahrzad H, Navruzyan A, Duffy N et al (2019) Evolving deep neural networks. In: Artificial intelligence in the age of neural networks and brain computing. Elsevier
34. Moya Rueda F, Grzeszick R, Fink G, Feldhorst S, Ten Hompel M (2018) Convolutional neural networks for human activity recognition using body-worn sensors. In: Informatics, vol 5. Multidisciplinary Digital Publishing Institute
35. Otter DW, Medina JR, Kalita JK (2020) A survey of the usages of deep learning for natural language processing. IEEE Trans Neural Netw Learn Syst 32(2):604–624
36. Pan B, Hsu K, AghaKouchak A, Sorooshian S (2019) Improving precipitation estimation using convolutional neural network. Water Resour Res 55(3):2301–2321
37. Pham H, Guan M, Zoph B, Le Q, Dean J (2018) Efficient neural architecture search via parameter sharing. In: International conference on machine learning
38. Real E, Aggarwal A, Huang Y, Le QV (2019) Regularized evolution for image classifier architecture search. In: Proceedings of the AAAI conference on artificial intelligence, vol 33

39. Real E, Moore S, Selle A, Saxena S, Suematsu YL, Tan J, Le QV, Kurakin A (2017) Large-scale evolution of image classifiers. In: Proceedings of the 34th international conference on machine learning-volume 70. JMLR. org
40. Reiss A, Stricker D (2012) Introducing a new benchmarked dataset for activity monitoring. In: 2012 16th international symposium on wearable computers. IEEE
41. Sapra D, Pimentel AD (2020) Constrained evolutionary piecemeal training to design convolutional neural networks. In: Trends in artificial intelligence theory and applications. Artificial intelligence practices. Springer
42. Sapra D, Pimentel AD (2020) Deep learning model reuse and composition in knowledge centric networking. In: 2020 29th international conference on computer communications and networks (ICCCN). IEEE
43. Sapra D, Pimentel AD (2020) An evolutionary optimization algorithm for gradually saturating objective functions. In: Proceedings of the 2020 genetic and evolutionary computation conference, GECCO '20. ACM
44. Springenberg JT, Dosovitskiy A, Brox T, Riedmiller M (2014) Striving for simplicity: The all convolutional net. arXiv:1412.6806
45. Stang M, Meier C, Rau V, Sax E (2019) An evolutionary approach to hyper-parameter optimization of neural networks. In: International conference on human interaction and emerging technologies. Springer
46. Suganuma M, Shirakawa S, Nagao T (2017) A genetic programming approach to designing convolutional neural network architectures. In: Proceedings of the genetic and evolutionary computation conference. ACM
47. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition
48. Tan M, Chen B, Pang R, Vasudevan V, Sandler M, Howard A, Le QV (2019) Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE conference on computer vision and pattern recognition
49. Wang J, Chen Y, Hao S, Peng X, Hu L (2019) Deep learning for sensor-based activity recognition: A survey. Pattern Recogn Lett 119:3–11
50. Wu B, Dai X, Zhang P, Wang Y, Sun F, Wu Y, Tian Y, Vajda P, Jia Y, Keutzer K (2019) Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: Proceedings of the IEEE conference on computer vision and pattern recognition
51. Xi R, Hou M, Fu M, Qu H, Liu D (2018) Deep dilated convolution on multimodality time series for human activity recognition. In: 2018 international joint conference on neural networks (IJCNN). IEEE
52. Xie L, Yuille A (2017) Genetic CNN. In: Proceedings of the IEEE international conference on computer vision
53. Yang L, Shami A (2020) On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing 415:295–316
54. Zhang H, Cisse M, Dauphin YN, Lopez-Paz D (2017) Mixup: Beyond empirical risk minimization. arXiv:1710.09412
55. Zhang YD, Satapathy SC, Guttery DS, Górriz JM, Wang SH (2021) Improved breast cancer classification through combining graph convolutional network and convolutional neural network. Inf Process Manag 58(2):102439
56. Zhou Z, Chen X, Li E, Zeng L, Luo K, Zhang J (2019) Edge intelligence: Paving the last mile of artificial intelligence with edge computing. Proc IEEE 107(8):1738–1762
57. Zoph B, Le QV (2016) Neural architecture search with reinforcement learning. arXiv:1611.01578
58. Zoph B, Vasudevan V, Shlens J, Le QV (2018) Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition

Dolly Sapra is a PhD Candidate in Parallel Computing Systems (PCS) group at the University of Amsterdam, working under Prof. Andy D. Pimentel. She holds an M.E. (Engineering) in Computer Science from BITS-Pilani, India. She turned to academic research after working in the industry as a software developer, for nearly a decade in India and in the U.K. Her current research is mainly focused on neural architectures for embedded devices, which includes multi-objective neural architecture search and adaptive architectures for resource constrained systems. She enjoys supervising graduate and undergraduate projects in the domain of neural networks and edge devices.

Andy D. Pimentel is full professor at the University of Amsterdam where he chairs the Parallel Computing Systems group. His research centers around system-level modeling, simulation and exploration of (embedded) multicore and manycore computer systems with the purpose of efficiently and effectively designing and programming these systems. He has an MSc and PhD in computer science from the University of Amsterdam. He is a cofounder of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS). He has (co-)authored more than 120 scientific publications and is an Associate Editor of Elseviers Simulation Modelling Practice and Theory as well as Springers Journal of Signal Processing Systems. He served as the General Chair of HIPEAC'15, as Local Organization Co-Chair of ESWeek'15, and as Program Chair of CODES+ISSS in 2016 and 2017. Furthermore, he has served on the TPC of many leading (embedded) computer systems design conferences, such as DAC, DATE, CODES+ISSS, ICCD, ICCAD, FPL, LCTES, and SAMOS.