Genetically Mutated Network with Dynamic Evaluation for Tiny Devices

Joseph Laganà, Davide Monaco, Matteo Scanu
Politecnico di Torino
Via Duca degli Abruzzi, 24

Abstract

Creating a Deep Learning architecture from scratch is a time-consuming and error-prone process; even though a human can tailor a specific model for its own needs, the increasing complexity in recognition tasks is pushing experts and researchers towards a more automatic process, like Neural Architecture Search (NAS). Considering the severe constraints that small electronic devices can bring, creating an hand-made neural network can become excruciatingly hard.

Here we will discuss a complete pipeline that firstly adapts the images of dataset in a reduced and common size, then creates models using basic blocks from popular neural architecture, changing its parameters in the span of limited search space. After that, the pipeline ranks them with a training-free metric based on FreeREA that is capable of evaluating the potential accuracy of the new architecture without training and a computation that returns the total FLOPs of the given architecture, and lastly tries to improve the best-score model introducing some random mutations, which will be described in full detail later.

The generation process is capable of creating an architecture and then converting it into a unique fixed-size string, in order to preserve the limited amount of memory we have. This paradigm was made having in mind the scope of finding an architecture that satisfies both the target accuracy and the computational constraints of a tiny device with the lowest requirement in terms of computational resources that is used to find the previously quoted architecture.

In order to conduct this experiment we used the Visual Wake Word (VWW from now on) dataset [1], a collection of images made for tiny machines that has two classes, "Person" and "Not-Person". The results show that the genetic mutation of the best found network, even though is subject to heavy constraints in terms of FLOPs and parameters, can achieve an accuracy of 82%. Considering also that the result is obtained in a dataset that has not been used as much as, for example, the CIFAR, it can be said that this result can compete with the actual state of the art.

1. Introduction

In order to create a brand new NAS it is required to possess a large amount of computational power to both build the networks and train them [4]; the difficulty furthermore increases if the scope of the NAS is to find an architecture for tiny devices, such as smartphones [3] or similar. There are various efforts in the direction of ranking architectures, such as the NASWOT [7] approach and one of the successors, the FreeREA [10] approach, without the actual need of training every potential architecture, with the help of different training-free metrics. NAS, even considering its severe requirements, has the potential to precisely fullfill the requests of the clients [4]. In the course of the years, the development of NAS framework(s) has taken another direction, the Benchmarks [11]. The latter fills the hole created by the necessity to find architectures having in mind a specific topology and size, two of the most important parameters in order to classify an architecture. Genetic algorithms have a special role in this research [5], thanks to their relatively easy implementation and maintenance, it is possible to achieve better accuracy with little to no extra computational cost. The actual state of the art has achieved a solid and robust accuracy and stability, but the overall knowledge is fragmented among the different fields of this vast and heterogenous setting. The different works presented here and later in this paper achieve different and optimal results within their scope but is yet to be done in regards of creating an entire framework that combines the creation of a brand new NAS with its own custom search space, the training free metrics and the necessities of keeping to a specified amount the total FLOPs of the given architecture. In this work it is presented the entire creation process of a pipeline that, in this given order:

- adapts a dataset, in this case the VWW, to the specified requirements;
- builds a NAS, defines a whole new search space based on the MobileNetV2 [3] with the scope of creating CNNs (short for convolutional neural networks) with identical architectures but with different parameters;

- ranks them all with already known metrics [7, 10], computing the number of parameters and FLOPs, too;
- genetic algorithm is deployed to further increase the potential accuracy of the unconstrained networks, then the evaluation process is re-done;
- trains and tests the best candidate net among all of the one we considered. Code at `https://github.com/JosephLagana/TinyML/blob/main/Complete_Pipeline.py`.

2. Related Work

Nowadays, NAS approach is widely discussed, in order to solve the issue of hand-building from scratch CNNs. NAS approach alone, as previously stated, is costly in term of both training time and computational resources. After the discovery of this architecture emerged new ways that are capable of studying NAS candidates without having to train all of them, as mentioned in the framework of NASWOT [7]. In terms of ranking architectures for a precise task without the training, and using a new point of view on the topic, NATS Benchmark [11] analyzes the results of a given and already trained NAS for specific purposes. The search space is the first important aspect of the design. A search space, defined as the layers or blocks that will be explored, can be hand built from scratch or can be tailored around different and already known architectures. Since the scope of the pipeline presented here is to find the best candidate for the binary person recognition in images, MobileNetV2 [3] become mandatory in order to aim for the state of the art accuracy while keeping inside the number of FLOPs constrains. The formed quoted architecture is built with the scope of recognizing objects, the latter is built mainly for image classification with the specific purpose of considering a low computational power for the usage. The search space can be built with a block-like approach, e.g the block-QNN [2]; this research will not be discussed in this paper. The ranking method utilized in this pipeline is directly inspired by the FreeREA method [10], that is also built around the concept of the framework NASWOT [7]. The best candidate is further improved with different methods, such as a Genetic Algorithm [5]. This work has the goal of creating a complete pipeline that is made with recent approaches and new techniques in order to further improve the research towards the miniaturization of architectures.

3. Method

In this paper, as previously stated, it is introduced a complete pipeline that begins with the acquisition and standardization of the VWW dataset, and ends with the training and testing, of the best given candidate that is generated through the pipeline. This section is divided in 7 subsection, one for every step of the pipeline.

3.1. Dataset adaptation

The first step in the pipeline involves the acquisition of the dataset, we are referring to VWW dataset, then is loaded into a format suitable for PyTorch library. To preserve space, dataset was stored virtually in a cloud server. Subsequently, we introduced a function responsible for resizing each image in the dataset, to make them lighter, and converting them into tensors. Moreover, the tensors are normalized using empirically obtained values of the mean and standard deviation, given by the matrix of our dataset. This preprocessing step ensures the compatibility with the downstream CNN design process.

3.2. CNN creation and Search Space definition

The Search Space used to find the best CNN is based on MobileNetV2, in which is able to be more reconcilable on our target. The layers that are present in this CNN are mainly Conv2D, Batchnorm2D and ReLU6.

The main idea is to use the Bottleneck structure of MobileNetV2 (a very light and easy-to-tune network) in order to find the one with the best accuracy just by changing its parameters, in a grid-search fashion: the chosen parameters were:

- structure, which tells us how many times each block is being repeated;
- expansion factors for each block;
- output channels.

Please note that each set of parameters is written as a row vector; each of them refers to a singular net, and every element of every vector is referring to a singular block. Lastly, for each structure, expansion factor and output channel combination, the pipeline generates a net using the data given in input and ranks it using the zero-cost metrics, in order to have a lot of CNNs to operate with: at the beginning of this pipeline, so before applying the genetic algorithm, we have 512 of them, with 129 respecting the constraints already (just the 25%).

3.3. Metrics

The main issue related to the timing of training phase is mitigated by the use of zero-cost metrics that allow us to rank all the nets with no training process required. For this work we considered three of the best metrics known until now, in an attempt to increase the probabilities of our success: Snip, Synflow, Jacob Covariance [8].

3.3.1 Snip

The Snip metric [9], short for 'Single-Shot Network Pruning', is used to evaluate the sensitivity of connections inside the network, especially regarding the change of loss with respect to parameter. Mathematically, the formula is the following:

$$\text{snip}: \mathcal{S}_p(\theta) = \left| \frac{\partial \mathcal{L}}{\partial \theta} \odot \theta \right|,$$

where $\mathcal{L}$ is the loss function, θ is the parameter vector and $\odot$ is the Hadamard product; the latter is just ad element-wise product for two matrices. Please note that Snip, as Synflow, computes a score for each parameter of each CNN considered: so, to compute a global score for a neural network with N parameters, we need to sum over all the parameters:

$$\mathcal{S}_n = \sum_{i=0}^{N} [S_p(\theta)]_i.$$

3.3.2 Synflow

The Synaptic Flow metric [6] is a generalization of Snip, in which it is calculated the dependency of a new loss function with the parameters. The following formula represents it mathematically:

$$\text{synflow}: \mathcal{S}_p(\theta) = \frac{\partial \mathcal{L}}{\partial \theta} \odot \theta.$$

3.3.3 Jacobian Covariance

The Jacobian covariance (we are going to call it Jacob) is designed in order to evaluate the correlation between the activations of the layers of untrained network within a mini-batch of data: the lower the correlation, the better the network. Mathematically, given a certain kernel $\mathbf{K}_H$ computed as in [7], we have the following score:

$$s = \log |\mathbf{K}_H|$$

3.4. Ranking

Following the deployment of the described metrics, a comprehensive ranking procedure is executed. The initial step involves the systematic evaluation of each untrained neural network using these metrics. Subsequently, the acquired metric scores undergo a normalization process utilizing the Z-score transformation.

The aggregation of the normalized scores culminates in the determination of a custom score, which is calculated through a sample mean without assigning explicit weights to the normalized metrics: this "vote" metric will be called AVG for the rest of this work. During the ranking process, adherence to the task's specifications entails consideration of auxiliary factors, such as the network's parameter count and the required FLOPs. The best candidate is found considering the maximum overall score in combination with the constraints on the parameters count and FLOPs. These supplementary considerations contribute to the overall ranking process, facilitating a comprehensive evaluation of neural network performances in alignment with the given task.

3.5. Genetic Algorithm

In the context of the machine learning classification framework, post-ranking selection procedures are employed to enhance the unconstrained performances of the candidates. This process is delineated into two distinct stages about the definition and deployment of the mutation function. The mutation function and cycle are largely inspired by [5], and so we need to divide two structure in which the nets are saved: *population* and *history*. We will work on the first one for the moment. The cycle is written as follows: after having defined and ranked all the desired CNNs, we considered a sample drawn uniformly. The dimension S of this sample is quite important, since a big value will allow us to have more exploitation of the nets (i.e. we will modify a good network more often than not, hoping to obtain a better one), a little value will allow us to have more exploration (i.e. we will modify a bigger number of nets, but the one we will pick at each cycle will not always be that good: this does not mean that the modified nets will not perform as effectively, though). After the sample has been drawn, we consider the best-AVG-scoring net, and we apply one random change to its structure, its expansion factors or its output channels.

The core of the genetic mutation involves in the application of the mutation function, which takes as input the network with the best score among the sampled, a data-loader, the pertinent features and labels for classification, and the loss function. In the case of structural mutations, only a designated (randomly chosen) layer is modified, rather than the entire network. Analogously, the same principle applies to mutation within the expansion factor and channel dimensions. Concerning the structure mutation, the space of different choices is made by five different types.

Concerning the expansion factor, instead, the choice space is made with seven different possible factors. Concerning channel modifications, a new value for the 'i-th' channel is selected from a range spanning from one to nine, with the chosen value serving as the exponent of two.

This calculated result is then incorporated into the 'i-th' parameter. Regardless of the score of this new net, this one will substitute the oldest (i.e. the first one we generated) net in the population, and the cycle will be repeated for a number of times (we put it equal to 500). Besides, the one we just obtained will be saved in history, as per every other net we will get using the mutation function. In the end we

Network	AVG	Params.	FLOPs
BM C	0.61	1.91	183.36
AF C	0.91	1.56	198.39
BM NC	1.37	3.99	1292.83

Table 1. Features of the networks

will consider the history data-frame, that will count as many as 1012 nets: among them, more than half respect the constraints: 572 (to be more precise, the 56.5%).

In a way, the mutation cycle has allowed us to have a lot more nets to consider, even relatively to the total number. But the real question regards the effectiveness of this cycle. The answer is in the best constrained (in the summary table C) net before (BM) and after (AM) the mutation: the best AVG score before the mutation was equal to 0.61, the best one after it was equal to 0.91, which is a considerable improvement while keeping inside the constraints. Just to have a comparison, the best overall net (NC) before and after the mutation cycle (which was not so effective in this case, probably because we prioritized exploration over exploitation) has an AVG score equal to 1.37. We will see in a minute what it means when evaluating training and test accuracy. The results before and after the mutation cycle are summarized in Table 1. Remember that in any table in this paper values for parameters and FLOPs are expressed in millions.

4. Results and discussion

After all of this, we can finally train the best (constrained and unconstrained) models we found: this way we can have a comparison of the two, and we can really assess how the constrained net performs with respect of the overall best.

We used the following parameters for each run:

```
batch_size = 64
learning_rate = 0.1
# after a few runs we noticed it
# was the value that yield the best
# results
optimizer = torch.opt.SGD()
```

The results after the training of the best constrained net that has been found are quite promising. With a relatively small requirements in terms of both time and resources, around two hours (40 epochs: it took more or less five minutes per epoch) and with 15 GB of T4 GPU, run thanks to Google Colab, the results are consistently around the 85% of training accuracy with 83% percent of train accuracy before the actual over-fitting issue comes into play. The epoch training accuracy shows the accuracy of a model on the training data as it goes through multiple epochs of training, while epoch test accuracy shows the accuracy on a separate test dataset (Table 1). As the epochs progress, test accuracy reaches a lower plateau and even decreases after the 25^{th} epoch. This suggests that the model may be over-fitting the training data.

Regarding the convergence speed, instead, the two vectors demonstrate that both training and testing accuracies improve step by step with the number of epochs. Training accuracy shows a consistent upward trend, while testing accuracy tends to stabilize and even slightly decrease after a certain point, around 22^{th} epoch.

This indicates that the model converges relatively quickly in terms of training accuracy, but its performance on the test data plateaus or even degrades slightly after a certain point.

Considering that for the VWW dataset the documentation and the benchmark are relatively poor, this result shows that the true potential of tiny devices for the image recognition has still to be discovered.

Nonetheless, considering also the constraints that are not part of the task but are relative to the availability of resources, this point has to be considered not as the conclusion but as the starting point of a new approach to tiny devices machine learning algorithms.

Different tests have been conducted also to the non-constrained networks for the same amount of epochs in order to have a more general perspective about the compromises of accuracy ad resources available for the deployment of the software.

The hypothesis that have been made during the construction of the pipeline regarding the power of the unconstrained net have not been rejected, in fact the unconstrained net performs better, but with a massively larger cost in terms of parameters flops and time, as the best overall network takes ten minutes (double the time!) for each epoch. In 3, the Nets are divided into two sections, pre mutation and after mutation, namely 5.1 and 5.2, and in Non-Constrained (NC) and Constrained (C). In The graphs below are plotted the networks' accuracies for training and test both for the constrained net and the unconstrained net. As shown in Figure 2, the training accuracy sits around the 90%, but the test accuracy is heavily impaired, it can be said that there is a serious issue of over-fitting in the unconstrained net.

Despite that, we can be very satisfied: we have demonstrated that we can achieve results that are almost as good as the overall nets just by using a net that is less complex and much, much more efficient.

5. Conclusions

Concluding, it is important to say that the results presented here can be improved with a high margin in the future, both with a better mutation function and a more suited search space. It's fascinating to observe that the heavily constrained net, compared to the unconstrained one, performs marginally better and can be deployed with minimal consumption on a low-middle end device. The state-of-the-art for both VWW and NAS using MobileNetV2 is still to be found, but the path has been marked, hoping somebody will decide to follow it.

6. Tables and plots

Table 2. Accuracy and loss of the best constrained net per epoch

heightEpoch	Training Loss	Training Acc.	Test loss	Test Acc.
1	0.01163	61.63	0.01009	65.05
2	0.00877	71.29	0.00940	70.79
3	0.00808	74.55	0.00824	73.50
4	0.00763	76.29	0.00763	76.50
5	0.00732	77.72	0.00859	75.03
6	0.00706	78.70	0.00753	77.07
7	0.00687	79.48	0.00747	76.96
8	0.00669	80.12	0.00721	77.44
9	0.00652	80.65	0.00669	79.92
10	0.00639	81.17	0.00660	80.01
11	0.00622	81.86	0.00684	79.97
12	0.00610	82.32	0.00667	79.95
13	0.00597	82.76	0.00643	81.11
14	0.00587	83.10	0.00663	80.37
15	0.00578	83.39	0.00659	80.94

Table 3. Comparison between Constrained (C) and Unconstrained (NC) Networks after 20 epochs

heightNet	Training Accuracy	Test Accuracy
C	84.45	82.71
NC	93.03	83.45

Table 4. Scores, parameters, FLOPs for different nets

Net	Jacob	Synflow	Snip	Average	Params	FLOPs
5.1 NC	0.454	-0.003	3.665	1.372	3.990	1292.838
5.1 C	-0.112	2.221	-0.271	0.612	1.915	183.36
5.2 NC	0.454	-0.003	3.665	1.372	3.99	1292.834
5.2 C	0.654	2.294	-1.094	0.618	1.520	197.151

7. Acknowledgements

We would like to thank Niccolò Cavagnero for his never ending patience, for his help and for his advices: they were really precious and without them, we would not have been able to make this work and this report.

Also we would like to thank professor Barbara Caputo for having introduced us to this complicated but amazing world: we hope we will be able to apply everything we learnt (and more!) in our academic and working careers.

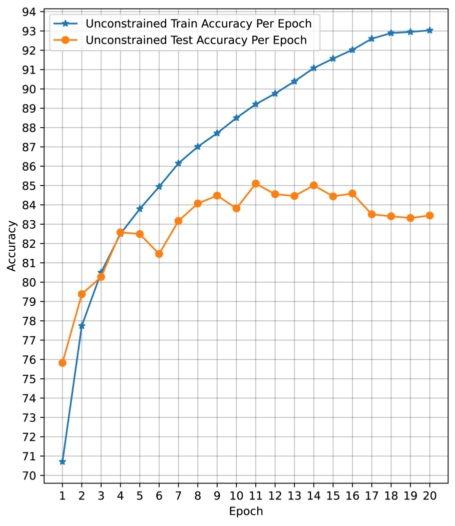

Figure 1. Accuracy for Unconstrained Net

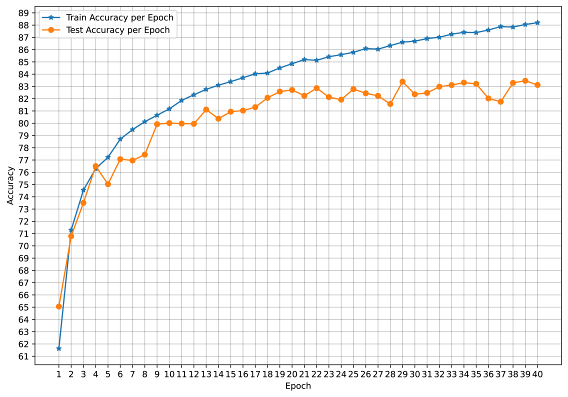

Figure 2. Accuracy for Constrained Net

References

[1] Visual wake word dataset. Accessed: 2010-09-30. 1

[2] Grace Chu Liang-Chieh Chen Bo Chen Mingxing Tan Weijun Wang Yukun Zhu Ruoming Pang Vijay Vasudevan Quoc V. Le Hartwig Adam Andrew Howard, Mark Sandler. Practical block-wise neural network architecture generation. 2019. 2

[3] Grace Chu Liang-Chieh Chen Bo Chen Mingxing Tan Weijun Wang Yukun Zhu Ruoming Pang Vijay Va-

sudevan Quoc V. Le Hartwig Adam Andrew Howard, Mark Sandler. Searching for mobilenetv3. 2019. 1, 2

[4] Quoc V. Le Barret Zoph. Neural architecture search with reinforcement learning. 2019. 1

[5] Yanping Huang Quoc V Le Esteban Real, Alok Aggarwal. Regularized evolution for image classifier architecture search. 2018. 1, 2, 3

[6] Daniel L. K. Yamins Surya Ganguli Hidenori Tanaka, Daniel Kunin. Pruning neural networks without any data by iteratively conserving synaptic flow. 2020. 3

[7] Amos Storkey Elliot J. Crowley Joseph Mellor, Jack Turner. Neural architecture search without training. 2020. 1, 2, 3

[8] Łukasz Dudziak Nicholas D. Lane Mohamed S. Abdelfattah, Abhinav Mehrotra. Zero-cost proxies for lightweight nas. 2

[9] Thalaiyasingam Ajanthan Philip H. S. Torr Namhoon Lee. Snip: Single-shot network pruning based on connection sensitivity. 2019. 3

[10] Barbara Caputo Giuseppe Averta Niccolò Cavagnero, Luca Robbiano. Freerea: Training-free evolution-based architecture search. 2023. 1, 2

[11] Katarzyna Musial Bogdan Gabrys Xuanyi Dong, Lu Liu. Nats-bench: Benchmarking nas algorithms for architecture topology and size. 2020. 1, 2