Neural Architecture Search for Tiny Visual Wake Words

Suyash Singh
Data Science Engineering
Politecnico di Torino
s307798@studenti.polito.it

***Abstract*—In the recent years, TinyML has become a field of focus with the aim to deploy deep neural networks (DNNs) on small and efficient devices. This idea of adopting DNNS on resource-constrained devices such as Raspberry-Pi is a challenging task. Our project is based on working with the Visual Wake Words dataset, with focus TinyML related issues. The idea is to train a DNN that can fit and make predictions on an edge device while being within the hardware specification range. This project is based on review of paper namely FreeREA [8] which represents training-free evolution-based architecture search. For this purpose we implemented random search as well as evolutionary algorithm in order to find a neural network that is better performing. As a variation, we proposed to apply mobile net v3 on the Visual Wake Words dataset, a model that performs significantly well on tinyML problems. Finally we compared the results from testing these different models. The Code is available in the following github repository.**

I. Introduction

Mobile applications are increasingly using efficient neural networks to enable whole new on-device experiences. They also play a significant role in enabling personal privacy by enabling users to benefit from neural networks without having to upload their data to a server for analysis. The effectiveness of neural networks has improved thanks to higher accuracy and quicker response times. However, research is being done in order to improve the performance of these algorithms. The problem is that, these enhancements come at the of increased processing and memory usage. This constitutes a significant barrier to the deployment of research results in actual environments, where factors like cost, energy use, and framework complexity are critical. The designer should look for models that maximize performance while minimizing their footprint to address this problem. A typical approach is to apply Neural architecture search (NAS), these methods rely on algorithms that must train and test each candidate during the search to evaluate their performance. Because NAS pipelines typically require a lot of computational time and resources, regular Machine Learning users without access to supercomputers find it difficult to take advantage of them. The energy consumption of a typical training-based NAS algorithm is another disadvantage that should be mentioned, since it would render the effort put into designing small models useless. There for in order to avoid heavy computation, we used random search and evolutionary algorithm with training free metrics. We managed to achieve the required accuracy based on the target specifications.

II. Related Works

Training-free metrics have emerged as a promising solution in Neural Architecture Search (NAS) to address the computational challenges associated with training and evaluating numerous candidate architectures. These metrics aim to score and evaluate architectures at the initialization stage, without the need for complete training and evaluation. The introduction of training-free metrics in NAS began with the proposal of Linear Regions (NASWOT) [1]. This metric measures the expressivity of an architecture, providing a quick estimation of its competitiveness within seconds. However, architectures identified using this metric were found to be less accurate compared to those discovered through traditional NAS algorithms. As a result, its usage was limited to the initialization phase of the search process. Subsequent research aimed to enhance the performance of training-free metrics. GA-NINASWOT [2] combined Linear Regions with a genetic algorithm, while EPE-NAS [3] proposed improvements to the base metrics. However, these methods did not demonstrate competitiveness compared to training-based methods. Inspired by NASWOT [4], researchers explored a hybrid approach that combined a variation of Linear Regions with the Neural Tangent Kernel (NTK) metric [5], as described in [6]. This hybrid metric aimed to capture not only the expressivity of architectures but also their trainability. However, computing the NTK metric proved to be computationally intensive. To overcome this challenge, NASI [7] introduced an approximation technique that significantly accelerated the metric computation process. Training-free metrics offer an alternative way to evaluate and score candidate architectures in NAS, bypassing the resource-intensive training and evaluation stages. Although they reduce computational requirements, they generally exhibit lower performance compared to methods that rely on complete training and evaluation. Nonetheless, ongoing research in this area aims to refine training-free metrics and bridge the performance gap with traditional NAS algorithms.

III. Metrics

Training-free algorithms place a significant emphasis on measurements that are a reliable substitute for model per-

formance. Unfortunately, these do not exactly match the test accuracy, hence it is crucial to take into consideration a combination of several metrics that may take different aspects like trainability and expressivity into account. The original paper suggests that in order to evaluate the validity of a number of indicators that serve as proxies for test correctness provides information on the correlation between several measures. Moreover based on research it proposes that the Linear Regions and a modified version of Synflow, which are dubbed LogSynflow, are the metrics that perform the best. The normalized scores of the individual measures are summed up to integrate the many indicators, which produces better empirical findings compared to a typical cumulative ranking score [2]. A model that is both trainable and expressive is more likely to demonstrate good performance. The fitness function f for a certain model i is implemented as follows:

$$f_i = \frac{LS_i}{maxLS_j} + \frac{LR_i}{maxLR_j} + \frac{Skip_i}{maxSkip_j} \tag{1}$$

where J is the set of the investigated networks, LS is an acronym for LogSynflow, LR stands for Linear Regions, Skip for Skipped Layers.

A. Linear Regions

With the intention of evaluating an architecture's early expressiveness, Linear Regions was introduced in . In fact, the activation values divide the tensors into active (positive values) and inactive (negative values) areas when one training sample is fed into a ReLU network, creating binary masks. The number of continuous regions of the input space that are mapped to various network activations, given an input space of fixed dimension, offers a measure of the model's ability to distinguish between various input values. In fact, the model finds it difficult to differentiate between two input items that produce identical masks when they are different. As a result, an expressive model has the ability to map various activation tensors to reasonably close input space values. Using Hamming distance, we may assess the variations in the activation patterns. The score s is calculated as follows once a Kernel matrix KH has been created from the Hamming distances:

$$s = log|det(K_H)| \tag{2}$$

The higher the score s, the more expressive the network is and the better it is able to tell the samples apart.As a result, increasing the Linear Regions score results in models that are more expressive and probably more test-accurate.

B. LogSynflow

Synflow was first presented in as a pruning metric for the choice of weights to be reduced in order to compress an architecture while maintaining the performances to the fullest extent possible. Its initial version evaluated the value of individual weights locally before being expanded by [1] to score an entire architecture by adding up the weights' individual contributions. The formal definition of the global metric is the scalar product of the weight vector and gradient vector:

$$S(\Theta) = \theta.\frac{\partial R}{\partial \theta} \tag{3}$$

It is important to note that since batch normalization layers obstruct gradient flow, they must be muted in order to compute Synflow. Even in very small architectures, this is likely to cause a gradient explosion. Because the term associated with the gradients may be several orders of magnitudes higher than the corresponding weight, the metric ignores the relevance of the weight values. To solve this problem,[8] suggests LogSynflow, which first sums the contributions of each network weight before scaling down the gradients with a logarithmic function:

$$S(\Theta) = \theta.log(\frac{\partial R}{\partial \theta} + 1) \tag{4}$$

Thus, finding designs with high LogSynflow scores leads to results that are probably more test-accurate.

IV. Search Algorithms

A. Evolutionary algorithm

Evolutionary algorithms (EAs) are computational methods inspired by the process of natural selection and evolution. They are used to solve optimization and search problems by imitating the principles of survival of the fittest and the generation of new offspring. Evolutionary algorithms operate based on a population of individuals, each representing a potential solution to the problem at hand. The individuals are encoded as a set of parameters, often referred to as a chromosome or genotype. These parameters can be binary strings, real numbers, or other data structures.The evolutionary process begins with an initial population of individuals, typically generated randomly or using specific initialization methods. Each individual is then evaluated based on a fitness function that quantifies its quality or performance regarding the problem being solved.

Through a series of iterative steps, the algorithm simulates evolutionary processes such as selection, crossover, and mutation. Selection involves choosing individuals from the current population for reproduction, typically based on their fitness. Higher fitness individuals are more likely to be selected, mimicking the principle of "survival of the fittest". During the crossover process, genetic material from selected individuals is combined to create new offspring. This is often done by exchanging or recombining segments of their chromosomes. Mutation is also applied, introducing random changes or variations to the offspring's genetic makeup. These mechanisms enable exploration of the search space and introduce diversity into the population. The new offspring replace or coexist with some of the existing individuals, forming the next generation. The process of selection, crossover, and mutation is repeated for multiple generations, with the hope that the population evolves towards better solutions over time. As the algorithm progresses, the population tends to converge towards optimal or near-optimal solutions, driven by the improvement

of individuals through natural selection and the exploration of the search space through crossover and mutation.
Evolutionary algorithms are versatile and can be applied to a wide range of problems, including optimization, machine learning, scheduling, and design. They have been successful in solving complex problems where traditional methods struggle, and they provide a flexible framework for finding solutions in a variety of domains. By running the search algorithm we found the following architectures. We used a block based search space for the evolutionary search algorithm.

Block Type	Output channels	Kernel size	Stride	Expansion Factor
Inverted Residual	64	5	1	6
Depthwise Separable	320	3	2	0
Depthwise Separable	320	7	1	0
Inverted Residual	32	3	1	6
Depthwise Separable	32	5	2	0
Inverted Residual	32	7	1	2
ConvNeXt	32	7	1	2
Inverted Residual	32	5	1	6
Inverted Residual	32	5	1	4
Inverted Residual	16	3	1	4
Depthwise Separable	16	3	1	0
Inverted Residual	64	3	1	4
Depthwise Separable	16	5	1	0
Inverted Residual	16	5	2	6
Depthwise Separable	16	5	2	0
Depthwise Separable	96	7	1	0
Inverted Residual	96	7	1	2
Depthwise Separable	16	5	1	0
Inverted Residual	16	7	1	2
Inverted Residual	96	7	1	2

Fig. 1. Architectures using Evolutionary algorithm search

B. Random search algorithm

The random search algorithm is a simple yet effective optimization technique that explores the solution space by randomly sampling points and evaluating their performance. The random search algorithm begins by defining a search space, which is the range of possible solutions to the problem at hand. This search space can be defined by a set of variables or parameters that characterize the problem.
In each iteration of the algorithm, a random point within the search space is generated. This point represents a potential solution to the problem. The selection of the random point can be uniform across the entire search space or follow a specific distribution. After generating a random point, it is evaluated using a predefined objective or fitness function. This function measures the quality or performance of the solution. The evaluation can involve running simulations, performing calculations, or any other method to assess the solution's effectiveness. The algorithm keeps track of the best solution found so far, typically based on the highest fitness value achieved. If the evaluated solution outperforms the current best solution, it is updated as the new best solution. The process of generating a random point, evaluating it, and updating the best solution is repeated for a predetermined number of iterations or until a stopping criterion is met. The stopping criterion can be based on the number of iterations, reaching a specific fitness threshold, or other termination conditions. Random search does not incorporate any prior knowledge about the problem or the search space. It explores the solution space by randomly sampling points, making it a simple and easy-to-implement optimization approach. However, it does not exploit any structure or gradient information of the problem. Due to its simplicity, random search can be computationally efficient and suitable for problems with large or complex solution spaces. It is particularly useful when the fitness landscape is rugged, discontinuous, or noisy, where traditional gradient-based methods may struggle.
Although random search does not guarantee finding the global optimum, it can often provide good results or serve as a baseline for more sophisticated optimization techniques. It can be combined with other methods, such as local search or evolutionary algorithms, to improve the overall search performance. In summary, the random search algorithm explores the solution space by randomly sampling points and evaluating their performance. It is a simple and versatile optimization technique that can be applied to a wide range of problems, particularly when prior knowledge or gradient information is limited. For random search algorithm, we exploited block based search space. We managed to obtain the following architectures using random search.

Block Type	Output channels	Kernel size	Stride	Expansion Factor
Inverted Residual	16	5	1	4
Depthwise Separable	32	5	2	0
Depthwise Separable	16	5	1	0
Inverted Residual	96	3	1	4
ConvNeXt	16	7	1	2
Depthwise Separable	96	7	1	0
Inverted Residual	64	5	2	4
Inverted Residual	160	7	2	4

Fig. 2. Architectures using random search

C. Mobile net V3

MobileNetV3 [10] is a convolutional neural network architecture designed for efficient and lightweight deep learning on mobile and embedded devices. The network introduces several key innovations to improve upon its predecessors. It leverages a combination of techniques, including efficient network design, advanced building blocks, and neural architecture search (NAS) to optimize both accuracy and efficiency. One of the major advancements in the network is the introduction of the "MobileNetV3 inverted residual blocks." These blocks utilize a combination of depthwise separable convolutions, linear bottlenecks, and a modified version of the Swish activation function. These design choices help reduce computational complexity and improve feature representation capabilities.It also incorporates the use of NAS to automatically search for the optimal architecture. By employing a reinforcement

learning-based search algorithm, the network's building blocks and their configurations are optimized to achieve a good trade-off between accuracy and efficiency. This NAS approach allows the network to adapt to different resource constraints and tasks. Furthermore, The network introduces two versions: MobileNetV3-Large and MobileNetV3-Small. The "Large" variant focuses on achieving higher accuracy, making it suitable for tasks where precision is crucial. On the other hand, the "Small" variant aims for even greater efficiency, making it well-suited for applications with strict computational constraints or limited resources. MobileNetV3 has demonstrated impressive performance on various computer vision tasks, including image classification, object detection, and semantic segmentation. It achieves competitive accuracy on standard benchmark data sets while requiring significantly fewer parameters and computations compared to larger and more complex networks. The following figure reveals the architecture of the network is given as follows: The architec-

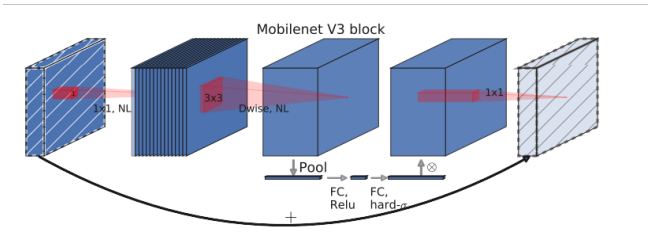

Fig. 3. Mobile Net V3 Architecture

ture's compact size and computational efficiency make it particularly well-suited for deployment on resource-constrained devices, such as smartphones, edge devices, and embedded systems. It enables real-time inference and on-device processing, reducing the reliance on cloud-based services and providing faster and more privacy-preserving solutions. The network is overall an efficient and lightweight neural network architecture specifically designed for mobile and embedded devices. It incorporates innovative design choices, advanced building blocks, and neural architecture search to achieve a good balance between accuracy and computational efficiency. The network has proven to be highly effective in various computer vision tasks and is well-suited for deployment on resource-constrained platforms.

V. The Data Set

For experiments we used Visual Wake dataset which is designed to simulate a typical scenario in vision micro controllers where the task is to determine whether a person is present in an image. It serves as a reliable benchmark for evaluating the effectiveness of small-scale vision models. The dataset comprises 115,000 samples for training and validation, carefully selected from the larger COCO dataset. Each image is labeled as either "Person" if it contains individuals or "Not-person" if there are no people present.
We had the following hardware constraints to be satisfied:

- Maximum parameters to be less than 2.5 M.
- Maximum flops no more than 200 M.
- Accuracy upto 80%.

VI. Proposed Strategy

In this project we implemented and compared different DNNs based on accuracy score. Initially we used above discussed metrics to select best performing algorithm in the search space. The search space was based on constraints to be satisfied as mentioned earlier. Table given in figure 4 reveals the comparison of the metric score of these networks.

Algorithm	FLOPS	Parameters	Synflow score	NASWOT score
Random Search	1.9e8	1.49e5	2.49e5	473.6
Evolutionary Algorithm	1.9e8	1.94e5	7.05e5	478
Mobile Net V3	1.74e8	2.18e5	2.94e5	454.96

Fig. 4. Algorithm constraints

In the next phase we trained neural network obtained from random search that was best performing according to the metrics discussed above. Using the above mentioned metrics we obtained the best performing mode. For instance, following figure reveals best performing models evaluated on the metrics mentioned above. Then we moved on to train neural network

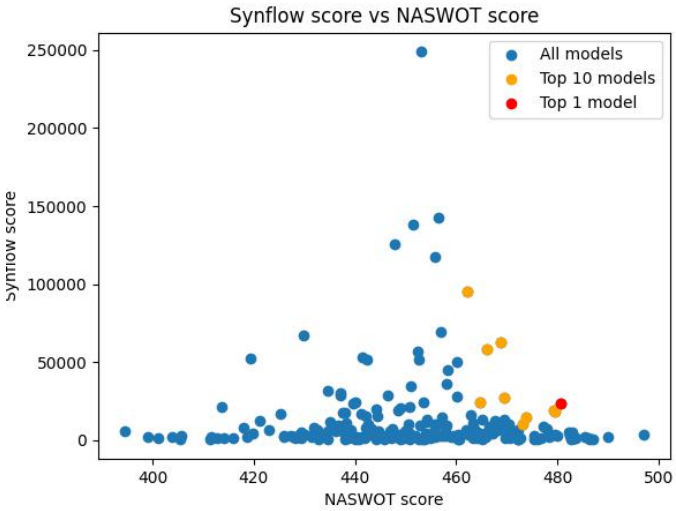

Fig. 5. Random Search model performance

obtained using evolutionary algorithm. Finally we trained mobile net V3 on the same data set and compared the results. We trained the models for 40 epochs. Following figure reveals the training accuracy for the neural networks.

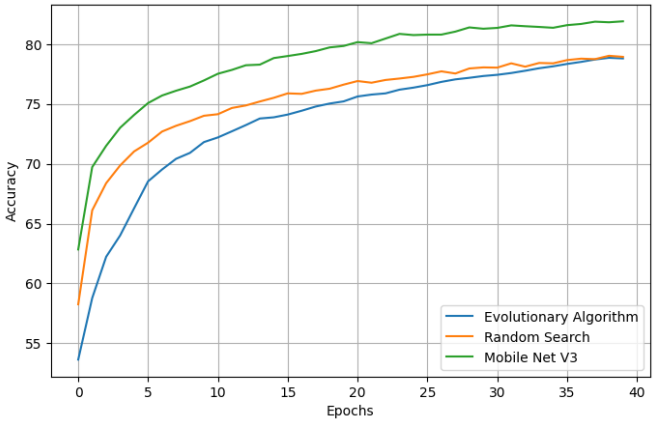

Fig. 6. Training results

The results of the model accuracy are given in figure 7.The final accuracy achieved by these models on the test set looks promising. Especially accuracy reached by mobile net v3 is way above the threshold of 80%

Algorithm	Test Accuracy
Random Search	83.81
Evolutionary Algorithm	80.68
Mobile Net V3	87.07

Fig. 7. Test results

VII. Conclusion

In conclusion we worked with Visual wake words dataset to deal with TinyML. We used train free metrics to obtain best performing models from the search space restricted by constraints in terms of parameters and flops. We used both random search and evolutionary algorithm along with metrics such as synflow and NASWOT score. Finally we trained mobile net v3 on the same dataset and compared the accuracy of all the trained models. Based on our analysis These mobile net v3 performed better than the other models, however the train free metrics proved to be helpful during model selection in the search space.

References

[1] Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In International Conference on Machine Learning, pages 7588–7598. PMLR, 2021.

[2] Meng-Ting Wu, Hung-I Lin, and Chun-Wei Tsai. A trainingfree genetic neural architecture search. In Proceedings of the 2021 ACM International Conference on Intelligent Comp

[3] Vasco Lopes, Saeid Alirezazadeh, and Lu´ıs A Alexandre. Epe-nas: Efficient performance estimation without training for neural architecture search. In International Conference on Artificial Neural Networks, pages 552–563. Springer, 2021.

[4] Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In International Conference on Machine Learning, pages 7588–7598. PMLR, 2021.

[5] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neu- ´ ral tangent kernel: Convergence and generalization in neural networks. Advances in neural information processing systems, 31, 2018.

[6] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In International Conference on Learning Representations (ICLR), 2021.

[7] Yao Shu, Shaofeng Cai, Zhongxiang Dai, Beng Chin Ooi, and Bryan Kian Hsiang Low. Nasi: Label-and data-agnostic neural architecture search at initialization. In International Conference on Learning Representations, 2021.

[8] Niccolo Cavagnero, Luca Robbiano, Barbara Caputo, Giuseppe Averta. FreeREA: Training-Free Evolution-based Architecture Search

[9] Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In International Conference on Machine Learning, pages 7588–7598. PMLR, 2021.

[10] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, Hartwig Adam. Searching for MobileNetV3