ProboNAS: A Training-Free Evolutionary Approach for NAS

Dario Avalle, Francesco Marigioli, Michela Lucia Saraceno
Politectino di Torino, Italy
{dario.avalle, francesco.marigioli, michelalucia.saraceno}@studenti.polito.it

Abstract

In recent years, a significant portion of Machine Learning research has focused on enhancing existing models, particularly neural networks, to improve their performance in various tasks. However, these advancements often result in higher memory usage and computational requirements, posing challenges for practical deployment considering factors such as cost, energy consumption, and framework complexity. To address this, designers aim to find models that balance high performance with minimal resource demands. The approach commonly used is Neural Architecture Search (NAS), which has evolved over the years, achieving remarkable results in terms of performance and execution times. Recent studies have shown that a notably efficient solution is to utilize training-free metrics to achieve comparable accuracy results in much less time compared to traditional NAS algorithms. In our case, we specifically tackled the task of human detection using the VisualWakeWords dataset, which is part of the COCO-Dataset and labels the presence or absence of a "Person". In this paper, we propose ProboNAS, a block-based evolutionary NAS algorithm that combines training-free metrics, customized block search space, and an efficient evolutionary algorithm. It achieves remarkable results by quickly finding architectures that perform excellently while respecting the computational and memory limits of modern devices, without the need for network training. Our experiments demonstrate that ProboNAS is a rapid, efficient, and successful approach for automated model design, representing a valuable solution for the task of human detection on the evaluated dataset. The project GitHub is available at www.github.com/FrancescoMarigioli98/ProboNAS.

1. Introduction

In the last decade, the advancement in the world of machine learning is at its peak, so much so that insiders are going as far as calling it "The Golden Age of Machine Learning" [8] [19]. Indeed, the growing computational capabilities of ML-oriented hardware – like GPUs and TPUs – and the widespread adoption of open-source machine learning tools – like TensorFlow and PyTorch, – have brought a drastic increase in the possible applications of machine learning models. Proof of this statement can be found in the striking number of papers posted daily in the machine learning–related categories of arXiv, a popular paper pre-print hosting service.

As machine learning models become more sophisticated, they can take on more complex tasks. Consequently, they require more computational capabilities. Just note the evolution of the number of GPT transformers parameters: the first model (2018) [28] had 117 million, GPT-2 (2019) [29] had 1.5 billion, and GPT-3 (2020) [3] had 175 billion. The number of parameters of GPT-4 [27] is not known but is estimated to be in the order of trillions. However, this need strongly collides with the drastic reduction of the dimension of everyday devices, given by the constant development of technology. As a consequence, the world of machine learning is eagerly trying to adapt to the drastic constraints that need to be respected. Nowadays, the challenge is finding an equilibrium between performance and the limits of the smaller devices.

Furthermore, the task becomes even trickier due to the modeling process of a Neural Network, which often relies on empirical experiments and experts' knowledge. That's why we turn to Neural Architecture Search (NAS) [21] in order to acquire models that can fully exploit the available resources, providing us with the optimum.

To be precise, we propose ProboNAS, a NAS algorithm that has two main characteristics:

- It's an evolutionary algorithm, which means that at each iteration it dynamically explores the Search Space combining the best models found previously, looking for an even better model that must also be constraints complaint.

- It's training-free, which means that the models discovered by the Search Algorithm are not trained. This is achieved thanks to metrics correlated to the accuracy of the model. In particular, LogSynflow [4] and NASwot [26].

After comparing the performances of different design choices, these two properties allow us to quickly explore the Search Space, increasing the probability of finding a model that guarantees high performance and accuracy, even in hardware-constrained scenarios.

2. Related Works

The paper [40], published in 2017, introduced the first pioneering method for neural architecture search (NAS) using reinforcement learning (RL). The paper addresses the challenge of automating the design of neural network architectures by formulating it as a sequential decision-making process. Instead of relying on human-designed architectures, the authors propose using RL to discover architectures that achieve high performance on a given task.

A crucial point for this specific problem is the choice of the search space. The search space refers to the space of all possible neural network architectures that can be explored to find an architecture that achieves good performance on a given task. The search spaces can be layer-based, cell-based and block-based.

[30] introduced layer-based search spaces, which allow the exploration and optimization of individual layers, each independently selected and composed to form the architecture.

NASNet [41] introduced cell-based search spaces, which are implemented trying to find the best structure for each cell. A cell represents a substructure, repeated multiple times to construct the complete neural network. [23] expanded the concept of a cell-based search space with a differentiable architecture search algorithm that jointly learns the weights of the operations and the architecture itself.

In block-based search spaces, the architecture is composed of blocks, which are composite modules built from a predefined structure that can contain multiple operations. Blocks are combined to form the neural network architecture, and each block can have different operations and connections, sharing the structure with the other blocks. [39] introduces a block-wise architecture generation approach that reduces the search space and makes the architecture search process more efficient. [34] incorporates hardware constraints while exploring a block-based search space. The authors aim to optimize architectures for both spatial and channel dimensions, taking into account computational efficiency and model accuracy.

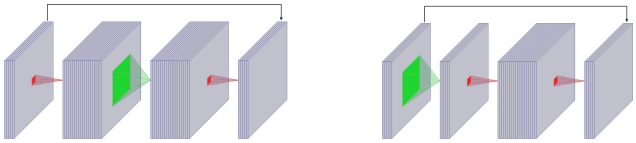
Over the years, various blocks have been proposed for convolutional neural networks (ConvNets), especially in the context of Neural Architecture Search (NAS) algorithms for computer vision tasks. Efficient convolutional blocks, like MobileNet [17] and ShuffleNet [38], achieved reasonable accuracy while reducing computational complexity. They often employ techniques such as depth-wise separable convolutions or group convolutions to decrease the number of parameters and operations involved. Residual blocks, initially introduced in ResNet [14], improved gradient flow and enabled the training of deeper networks. NAS algorithms frequently incorporate variations of residual blocks, including DenseNet [18] and ResNeXt [35] blocks, which further enhance the connectivity and representational power of the networks. Inverted Residual blocks, utilized in MobileNetV2 [31] and EfficientNet [32], have become popular due to their efficiency in terms of parameter count and computational cost. These blocks leverage depth-wise separable convolutions followed by point-wise convolutions to efficiently capture both spatial and channel-wise information. Among the variants derived from the Inverted Bottleneck concept introduced in MobileNetV2 [31], ConvNeXt [25] and MobileNetV3 [16] have showcased remarkable performance in visual recognition tasks. An enhancement to architectural designs like Deep Pyramidal Residual Networks involves utilizing the zero-padding technique [13], crucial for preserving spatial information. It enables the construction of deeper network architectures, and ensures comprehensive information capture from the input. In the specific case of DeepPyramidNet [13], a novel architecture for deep ConvNets, pyramid-like skip connections are incorporated, building upon the concept of residual networks (ResNets [14]). These skip connections facilitate the flow of information across different network levels, enabling more effective gradient propagation and addressing the challenge of vanishing gradients during deep network training.

In order to develop a search algorithm that converges towards an optimal solution, it is necessary to evaluate the neural networks built from exploring the defined search space. Due to the high computational cost associated with training neural networks, researchers often rely on benchmarks, like NAS-Bench-101 [36], NAS-Bench-201 [11] and NATS-Bench [10], to evaluate the performance of Neural Architecture Search (NAS) algorithms. Typically, a benchmark includes datasets, evaluation metrics, and evaluation protocols that establish a common basis for evaluating the efficiency and effectiveness of various NAS methods.

Recent studies have shown that the evaluation of neural networks using training-free metrics produces comparable results to evaluations conducted after the training phase. [1] provides a comprehensive analysis of metrics that allow for the efficient evaluation of architectures without the need for computationally expensive training. Other research papers, such as [26] and [4], further delve into these training-free metrics, offering a more detailed exploration, improvement, and consolidation of these metrics to form a highly efficient and effective set.

3. Method

This paper presents ProboNAS, a novel method for Neural Architecture Search (NAS) that utilizes training-free



(a) MobileNet block (b) ConvNeXt block

Figure 1. The two types of blocks used in ProboNAS. Skip connections, point-wise convolutions (in red), and depth-wise convolutions (in green) are shown. For graphical reasons, normalization layers and activation functions are omitted.

metrics to achieve high-performance architectures while adhering to constraints on FLOPs (floating-point operations) and the number of parameters. By carefully selecting metrics, optimizing the evolutionary search process, and implementing an efficient search space, our approach demonstrates the capability to generate accurate models for the human detection task on the Visual WakeWords database [6]. Importantly, ProboNAS achieves these results with significantly reduced search time compared to conventional NAS methods that do not utilize training-free metrics [26].

3.1. Search space

Before proceeding to define the evolutionary algorithm, it was necessary to determine the search space, which refers to the collection of all possible architectures to be explored in the search for a suitable candidate to be evaluated on our dataset. We opted for a block-based search space to take advantage of the inherent modularity offered by blocks, enabling easy combination and reuse. This approach also reduces the computational burden of the search algorithm by focusing on selecting and combining the most effective blocks, rather than exhaustively exploring a wide range of configurations. ProboNAS employs two distinct types of blocks, implemented in MobileNetV2 [31] and ConvNeXt [25], that leverage the concepts of depth-wise separable convolution and point-wise convolution. These blocks are shown in Figure 1 and are explained below.

3.1.1 MobileNet Block

As demonstrated in MobileNetV1 [17], depth-wise separable convolutions offer an efficient alternative to conventional convolutions. They achieve this by splitting the spatial filtering process into a single-channel 3×3 convolution that doesn't increase the number of channels, and a heavier 1×1 point-wise convolution responsible for feature generation [17]. This separation of operations significantly improves efficiency compared to traditional convolutions. If we denote with D_F the spatial width and height of a square input feature map, $D_K \times D_K$ the kernel size, M the number of input channels and N the number of output channels, the ratio between the computational cost of a depth-wise separable convolution and of a traditional convolution can be expressed by:

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2} \quad (1)$$

where the first part of the numerator is the cost of the depth-wise convolution, while the second part represents the cost of the point-wise [17]. This proves that the computational cost reduces a lot using this kind of convolution.

This concept is adopted from the MobileNetV2 [31] paper, which introduces the linear bottleneck and inverted residual structures to convolutional blocks. It achieves this by adding an expansion convolution 1×1 block followed by depth-wise convolution. Additionally, it incorporates a residual connection only when the input and output have the same number of channels. This design strategy ensures that the input and output remain compact while internally expanding to a higher-dimensional feature space. This expansion enhances the impact of non-linear transformations, such as ReLU, on the channels without compromising information. Instead, it effectively represents the manifold of interest, which is low-dimensional, in a subspace of the higher-dimensional activation space [31]. The depth-wise and the first point-wise convolutions are followed by a BatchNorm and ReLU activation function, whereas the last pointwise is followed only by a BatchNorm.

3.1.2 ConvNeXt block

Based on [25], we implemented the ConvNeXt block. Similar to the MobileNet block, this block introduces an inverted bottleneck, with a few differences. The authors of the original paper [25] designed the model by imitating some features of vision Transformers such as Swin Transformer [24]:

- The depth-wise layer, being the computationally more complex one, is moved before the inverted bottleneck. This is also present in vision Transformers since the multi-head self-attention block (the most complex module) precedes the MLP layers [25]. The result of this shift is that the depth-wise convolution is performed on a smaller feature map, causing both a reduction in performance but also the number of FLOPs.
- Many transformers (e.g. BERT [9], GPT-2 [29], ViT [12]) use the GELU [15] as activation function. This is a smoother version of the more famous ReLU. Furthermore, observing that in transformer blocks the activation function is only present once between the two MLP layers [24], the authors of ConvNeXt [25] decided to stick with the same strategy and to use only one GELU activation function between the two pointwise layers.

- In addition to reducing the number of activation functions, the same reason the authors of ConvNeXt [25] decided to reduce the number of normalization layers, leaving only one after the depth-wise convolution. Furthermore, instead of using a BatchNorm, they used a LayerNorm [2], again motivated by the fact that this normalization layer is widely used in transformers.

3.1.3 Zero Padding

The skip connection is originally designed to work when the input and output channels of a block are the same. However, in our case, we wanted the network to learn and capture more complex features and patterns as information passes through the layers. To achieve this, we chose to increase the number of output channels in each block compared to the input channels [37]. By doing so, we enable hierarchical feature extraction, allowing each layer to focus on learning higher-level and abstract features by leveraging the richer representations from previous layers [13]. To accommodate both channel expansion and the skip connection, we utilized the technique of zero-padding [13]: this consists of adding zero-value channels to the input feature map, so that it has the same dimensionality as the output. This allows to make the skip connection in a block, even though its input and output sizes are different, and the number of channels can increase as the depth increases.

3.1.4 Genetic code

The genetic code is a list that defines the structure of our Neural Network. It is composed by:

- A random value, which states the number of output channels of the stem block's convolution layer. The stem block is implemented so as to quickly downsample an input image with strided convolutions, so that further layers can effectively do their work with much less computational complexity. In this paper, it is comprised of the above-mentioned convolution layer, a GELU activation function [15], and a BatchNorm layer.
- A list (1) of lists (2) of lists (3):
    1. The outer list is a representation of the configuration of the whole Neural Network and is composed of 4 middle lists.
    2. Each middle list is a representation of one of 4 stages. The number of elements in it is the number of blocks inside a stage and it depends on the structure of the network. The global structure is $[3, 3, 9, 3]$, but we also evaluated the performance using the structure $[2, 2, 6, 2]$. In the test phase, the two structures showed similar performances, but we chose the first one because the correct predictions for the two classes were more balanced.
    3. Each inner list is a representation of a single block and it's composed of the genes that characterize the block.

In order to avoid introducing bias in our model as much as possible, the genes are chosen with a random approach. In each block, we randomly sample 4 genes:

- The type of block, which can be 'i' for the MobileNet block or 'c' for the ConvNeXt block.
- The number of output channels, which is created in a way that it increases proportionally along the depth of the Neural Network [37].
- The kernel size, which varies based on the block; in particular, MobileNet blocks' and ConvNeXt blocks' possible values are $[1, 3, 5, 7]$ and $[3, 5, 7, 9]$, respectively. As we can notice, the second allows for bigger kernel sizes. This design choice is due to the fact that in the ConvNeXt paper [25], the authors demonstrated the effectiveness of using larger kernel sizes, relying on the non-local self-attention mechanism of Vision Transformers [12], which allows all layers to have a global receptive field.
- The expansion ratio of the channels inside the block, which can be one of the values $[2, 3, 4]$.

Once the genetic code is constructed, it is passed to a script that has the task of actually building the Neural Network based on its instructions, also placing a downsampling block between the stages. The downsampling block scales down the size of an input image with minimal effect on features extracted; it is composed by a depth-wise convolution (kernel 3×3, stride 2×2), a point-wise convolution, a BatchNorm layer and a ReLU activation function. Finally, our Neural Network is completed with a fully connected layer. In Figure 2 and in Table 1, we can observe the network of a given individual.

3.2. Training-free metrics

As mentioned earlier, we opted for a training-free approach in our research. Such algorithms heavily rely on the choice of metrics, which serve as reliable indicators of the model's performance. However, it's important to note that these metrics may not directly align with the observed testing accuracy. Therefore, it becomes crucial to assess a range of metrics that take into account various factors like the model's learning capacity and its ability to handle complex information. After examining performance reports from [1], which provide a comprehensive overview of

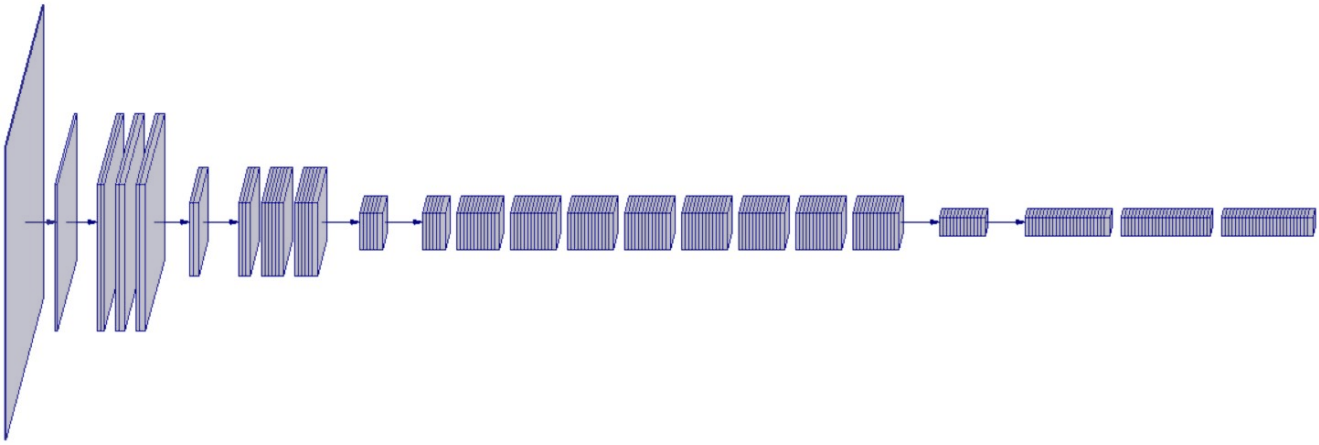


Figure 2. Example of the structure of a possible individual of the search space. It is shown how the feature map varies while forwarding the network. You can see how the number of channels increases with depth. For graphical reasons, the final fully connected layer is omitted.

the primary training-free metrics used in various NAS algorithms, we decided to utilize the SynFlow [33] metric as one of our reference points. According to the research paper, SynFlow has the most substantial influence on the performance of training-free algorithms. Specifically, we utilized a variant called LogSynflow, as implemented in [4].

However, no single metric is comprehensive enough to fully capture the complexity of a training-free model's performance. Each metric offers unique insights and highlights different aspects of the model's performance. To address this, we also incorporated another metric from NASwot [26], which evaluates network characteristics beyond what LogSynflow [4] considers.

To evaluate each individual network, we devised a scoring system that took into account both of these metrics. We ranked each metric and assigned scores based on their respective rankings. Finally, we combined the scores to generate a final score that represents a weighted and normalized aggregate of the metrics [5].

3.2.1 LogSynflow

Synflow [33] is built upon the idea of "synaptic importance," which quantifies the contribution of synaptic connections (or weights) to a neuron's output. This metric calculates the importance of each connection in the network and uses backpropagation to determine the significance of the neurons themselves. In its original form, it performed local evaluation to determine the relevance of individual weights and was subsequently expanded by [1] to evaluate the entire architecture by aggregating the contributions of the individual weights. The metric involves taking the scalar product of the weight vector $\boldsymbol{\theta}$ and the gradient vector $\frac{\partial R}{\partial \boldsymbol{\theta}}$, where R is a scalar loss function based on the output y of a feed-forward network parameterized by $\boldsymbol{\theta}$ [1] [4]:

$$S(\Theta) = \boldsymbol{\theta} \cdot \frac{\partial R}{\partial \boldsymbol{\theta}} \tag{2}$$

To compute the Synflow metric, the network is first initialized with the absolute values of its original weights. Then, an all-ones tensor is fed forward through the network, and the output is backpropagated to calculate the gradients. It is important to note that BatchNorm layers need to be removed during this process as they can interfere with the gradient flow. However, removing these layers can lead to gradient explosion, even in relatively small architectures. Consequently, the metric overlooks the significance of weight values because the term associated with gradients can be orders of magnitude higher than the corresponding weight value.

To address this issue, we employed LogSynflow, which is a variant of Synflow introduced in [4]. LogSynflow scales down the gradients using a logarithmic function before summing up the contributions of each network weight:

$$S(\Theta) = \boldsymbol{\theta} \cdot log\left(\frac{\partial R}{\partial \boldsymbol{\theta}} + 1\right) \tag{3}$$

In [4] it is demonstrated the increase of the correlation between the test accuracy and the metric score, with respect to the original implementation.

3.2.2 NASwot

NASwot [26] proposed a novel metric that leverages Linear Regions to assess the expressiveness of a neural architecture during initialization. By analyzing the activation values produced by forwarding a training sample through a ReLU network, binary masks are generated to categorize the tensors into active regions (positive values) and inactive regions (negative values). The underlying idea is that inputs

	BLOCK	KERNEL	EXP	OUT SIZE
	Stem			64x64x5
1	MobileNet	3x3	2	64x64x13
	MobileNet	7x7	2	64x64x17
	MobileNet	1x1	2	64x64x17
	Downsapling			32x32x17
2	ConvNeXt	9x9	2	32x32x21
	MobileNet	1x1	4	32x32x39
	MobileNet	3x3	2	32x32x40
	Downsapling			16x16x40
3	ConvNeXt	3x3	2	16x16x40
	ConvNeXt	5x5	3	16x16x74
	MobileNet	7x7	2	16x16x80
	ConvNeXt	5x5	3	16x16x80
	MobileNet	5x5	4	16x16x80
	MobileNet	3x3	2	16x16x80
	MobileNet	5x5	4	16x16x80
	MobileNet	5x5	4	16x16x80
	ConvNeXt	5x5	2	16x16x80
	Downsapling			8x8x80
4	ConvNeXt	5x5	4	8x8x149
	ConvNeXt	3x3	4	8x8x157
	MobileNet	7x7	3	8x8x160
	Fully Connected			2

Table 1. Example of the structure of a possible individual of the search space. The various convolutional blocks are shown. "KERNEL" represents the kernel size of the depth-wise convolution of each block, while "EXP" represents the expansion factor of the inverted bottleneck. The corresponding genetic code is the following: `(5,[[[i,13,3,2],[i,17,7,2],[i,17,1,2]],[[c,21,9,2],[i,39,1,4],[i,40,3,2]],[[c,40,3,2],[c,74,5,3],[i,80,7,2],[c,80,5,3],[i,80,5,4],[i,80,3,2],[i,80,5,4],[i,80,5,4],[c,80,5,2]],[[c,149,5,4],[c,157,3,4],[i,160,7,3]]])`

with similar binary codes pose a greater challenge for the network to learn how to differentiate them. When two inputs share the same binary code, they reside within the same linear region of the network, making them particularly difficult to untangle. Conversely, inputs that are well separated facilitate easier learning. The NASwot metric measures the dissimilarity between two inputs by calculating the Hamming distance $d_H(\mathbf{c}_i, \mathbf{c}_j)$ between their binary codes, which are induced by the untrained network. By constructing a kernel matrix, the metric examines the correspondence between binary codes across a whole mini-batch of data:

$$\boldsymbol{K}_H = \begin{pmatrix} N_A - d_H(\boldsymbol{c}_1, \boldsymbol{c}_1) & \dots & N_A - d_H(\boldsymbol{c}_1, \boldsymbol{c}_N) \\ \vdots & \ddots & \vdots \\ N_A - d_H(\boldsymbol{c}_N, \boldsymbol{c}_1) & \dots & N_A - d_H(\boldsymbol{c}_N, \boldsymbol{c}_N) \end{pmatrix} \tag{4}$$

where N_A is the number of rectified linear units [26].

In high-performing networks, there are fewer off-diagonal elements that exhibit significant similarity. With this assumption, the score s is computed as [26]:

$$s = log \mid det(\boldsymbol{K}_H) \mid \tag{5}$$

A higher score indicates greater network expressiveness and its capacity to differentiate samples. Generally, this leads to improved final accuracy following training.

3.3. Search Algorithm

To discover the most suitable model, we employed an evolutionary approach, specifically modifying a Tournament selection algorithm [4]. This involved starting with a random population of size N, where each individual was defined by a genetic code representing the network's structure. During each iteration, a subset of the surviving population, consisting of n individuals, was randomly chosen. The two best individuals from this sample were selected as parents for reproduction. Through independent mutations, two children were generated from the parents, while a third child was produced through a crossover operation. The crossover involved randomly selecting each gene from one of the two parents, enhancing the search algorithm's exploration capability by combining different genotypes to create diverse individuals that differed significantly from their parents. Both the mutation and crossover operations were designed to ensure the growth of the number of output channels in the network's blocks. If the extracted gene failed to meet the condition, it was replaced with either the number of channels from the preceding block if it was lower or the output channels from the following block if higher.

To enforce constraints during the search, we set limits on the FLOPs (floating point operations) and the number of parameters considered for the models. These constraints were independent of the specific deployment architecture. Feasible individuals satisfying the constraints were added to the population, and offspring generation continued until a feasible child was discovered. If, even after pruning based on the constraints, the population exceeded N, individuals were further pruned based on their scores in the evaluation metrics. By employing this method, the algorithm identifies the top N architectures in each iteration. Upon completion of the iterations, the algorithm provides the code of the best architecture from the population, which is evaluated based on training-free metrics.

Throughout our experiments, we fixed the values of N and n to 25 and 5, respectively.

4. Experiments

4.1. Dataset

Our solution was evaluated using the Visual Wake Words dataset [6], which is commonly used for vision microcontroller applications. This dataset focuses on detecting the presence or absence of a person in an image, making it an ideal benchmark for assessing the effectiveness of small-scale vision models. Comprised of approximately 123k images, the dataset consists of 82k training samples and 40k validation samples, filtered from COCO [22]. Each image is labeled as either "Person" if individuals are present or "Not-person" otherwise.

In order to ensure uniformity in input dimensions for all images, we resized each image in the dataset to 224×224 pixels. Subsequently, we compressed the training and validation images into tar files, enabling us to accommodate the memory constraints of the testing environment.

4.2. Experimental setup

The execution of the ProboNAS algorithm, aimed at delivering the optimal Neural Network, and its subsequent training took place in the Google Colab© environment utilizing a 12GB Test P100 GPU. Before executing the entire pipeline, we further compressed the images to 128×128 pixels to reduce the number of parameters and FLOPs, which represented our computational constraints for the running process. The maximum FLOPs were set at 200M, while the maximum number of parameters was limited to 2.5M. During the training phase, we experimented with different combinations of batch size and learning rate, while the number of epochs was set to 10. For all the experiments, we used an Adam Optimizer with parameters $\beta_1 = 0.9, \beta_2 = 0.99$ [20] and we implemented an early stopping technique based on the test loss.

4.3. Results

The outcomes of our study, presented in Table 2, showcase the Neural Network generated by our ProboNAS algorithm within a mere 15-minute time-frame. For this purpose, we started with an initial set of 25 feasible individuals, and 5 individuals sampled at each iteration involved. Our analysis primarily focused on the impact of different learning rates and batch sizes.

Specifically, we trained the Neural Network using batch sizes of $[8, 16, 32]$ and learning rates of $[0.01, 0.001, 0.0001]$. Looking at Figure 3, the case learning rate 0.001 and batch size 32 performed the best testing loss. However, it can be seen that it does not converge, indicating the potential for even better results with more epochs.

Nonetheless, these results demonstrate the effectiveness of capturing valuable features and achieving an accuracy above 80%, all while adhering to the limitations of the selected constrained scenario, reaching a utilization percentage of 92% of the allowed FLOPs and less than 37% of the maximum achievable parameter count.

ACCURACY				
		Batch size		
		8	16	32
LR	0.01	0.538	0.538	0.538
	0.001	0.749	0.803	**0.823**
	0.0001	0.646	0.698	0.672

Table 2. Accuracy obtained with different batch sizes and learning rates (LR).

4.4. Data Augmentation

To address the issue of the network entering an oscillation phase in the test loss for certain combinations of learning rate and batch size, we considered the possibility of partial overfitting. In order to mitigate this potential problem, we decided to explore the application of Data Augmentation techniques. Specifically, we conducted experiments using the RandAugment [7] library, which offers a streamlined approach to selecting effective augmentation policies.

Traditional data augmentation methods often require manual selection and fine-tuning of various augmentation operations, which can be both time-consuming and subjective. RandAugment [7] simplifies this process by reducing the search space for augmentation policies. It achieves this by applying a set of predefined augmentation operations to the input images. Rather than exhaustively searching through all possible operations, RandAugment [7] utilizes a smaller set of augmentation primitives that can be combined randomly. This approach significantly reduces the search space while still generating diverse and effective augmentation policies.

The RandAugment [7] library employs two parameters to control the augmentation process. The first parameter, M, determines the magnitude of the individual data augmentation policy and influences its impact on the image. The second parameter, N, defines the number of policies that are combined and applied to each image.

The results of our exploration obtained with 15 epochs can be found in the provided Table 3. As evident from the results, there was no increase in accuracy for the best case (learning rate 0.001 and batch size 32), probably due to the limited number of epochs. However, it is noteworthy that certain training combinations, like the one with a batch size of 16 and learning rate 0.001, which previously exhibited a minimum in the testing loss graph, actually demonstrated performance improvement, as it is shown in Figure 4. This suggests that in this kind of scenario, where the training is likely entering overfitting, data augmentation can be a viable solution to further enhance the algorithm.

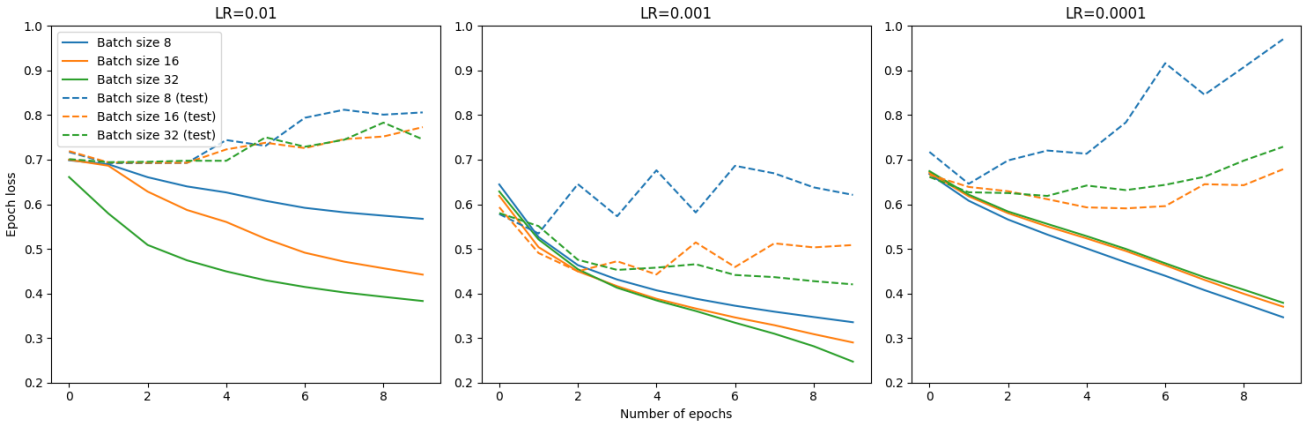


Figure 3. Train and test losses of the network during the training. Each graph represents a different combination of learning rate and batch size.

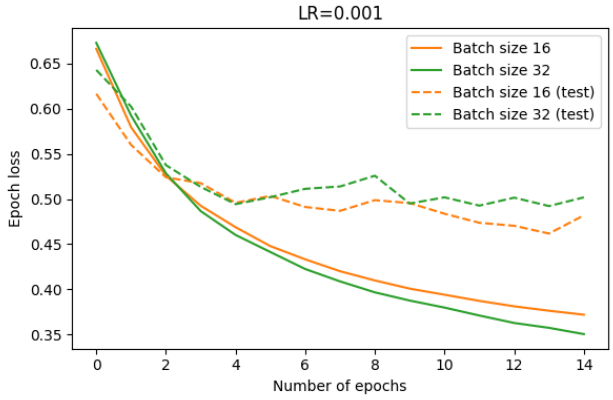


Figure 4. Train and test losses of the network during the training once we implemented the data augmentation.

ACCURACY			
		Batch size	
		16	32
DA	N=2 M=14	0.803	0.809
	N=3 M=14	0.771	0.697
	N=3 M=8	0.812	0.815

Table 3. Accuracy obtained with different batch sizes and data augmentation techniques (DA), where N is the number of transformations applied and M is the magnitude of the transformations. The models were trained using $LR = 0.001$

5. Conclusion

In this paper, we introduced ProboNAS, an innovative training-free approach for Neural Architecture Search (NAS) specifically designed for tiny models. Our initial step involved identifying an appropriate combination of training-free metrics that effectively capture the model's test accuracy. These metrics served as a dependable proxy for evaluating different networks during the search process. Additionally, we devised a novel search space by incorporating the blocks of MobileNetV2 [31] and ConvNeXt [25] into a block-based model and enhancing them using the zero-padding technique [13].

Our approach, guided by an evolution-based policy, yielded highly competitive results in significantly shorter time-frames compared to existing methods. This enabled us to strike a balance between speed, efficiency, and accuracy in NAS for tiny models. We validated our approach on the Visual Wake Words dataset [6], which consisted of 82k training images and 40k testing images. Our findings demonstrate that ProboNAS effectively performs Neural Architecture Search without training any candidate models, achieving an accuracy of over 80% with an architecture extracted in a time-frame of 15 minutes.

Furthermore, we explored the potential of incorporating Data Augmentation techniques [7], which showed promise in improving the process under specific training conditions.

Future research could focus on exploring novel training-free metrics that easily integrate into our scoring system. Alternatively, different training methods for the network could be explored. Another avenue for investigation involves experimenting with different blocks or testing alternative search spaces, such as cell-based or layer-based approaches.

References

[1] Mohamed S. Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D. Lane. Zero-cost proxies for lightweight nas, 2021. 2, 4, 5

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. 4

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. 1

[4] Niccolo Cavagnero, Luca Robbiano, Barbara Caputo, and Giuseppe Averta. FreeREA: Training-free evolution-based architecture search. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, jan 2023. 1, 2, 5, 6

[5] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective, 2021. 5

[6] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. Visual wake words dataset, 2019. 3, 7, 8

[7] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space, 2019. 7, 8

[8] Jeffrey Dean. A Golden Decade of Deep Learning: Computing Systems
amp; Applications. *Daedalus*, 151(2):58–74, 05 2022. 1

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. 3

[10] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. NATS-bench: Benchmarking NAS algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021. 2

[11] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search, 2020. 2

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. 3, 4

[13] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks, 2017. 2, 4, 8

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 2

[15] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. 3, 4

[16] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019. 2

[17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. 2, 3

[18] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018. 2

[19] Okyay Kaynak. The golden age of artificial intelligence. *Discover Artificial Intelligence*, 1(1):1, 2021. 1

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 7

[21] George Kyriakides and Konstantinos Margaritis. An introduction to neural architecture search for convolutional networks, 2020. 1

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. 7

[23] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search, 2019. 2

[24] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021. 3

[25] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022. 2, 3, 4, 8

[26] Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. Neural architecture search without training, 2021. 1, 2, 3, 5, 6

[27] OpenAI. Gpt-4 technical report, 2023. 1

[28] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018. 1

[29] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. 1, 3

[30] Esteban Real, Chen Liang, David R. So, and Quoc V. Le. Automl-zero: Evolving machine learning algorithms from scratch, 2020. 2

[31] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019. 2, 3, 8

[32] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020. 2

[33] Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow, 2020. 5

[34] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search, 2019. 2

[35] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks, 2017. 2

[36] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search, 2019. 2

[37] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017. 4

[38] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices, 2017. 2

[39] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation, 2018. 2

[40] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2017. 2

[41] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition, 2018. 2