Improving Zero-Shot Neural Architecture Search with Parameters Scoring

Anonymous authors
Paper under double-blind review

Abstract

The exceptional success of deep learning comes at the cost of long training sessions, and a slow iterative process of proposing new architectures that have to be hand-engineered through years of experience. Neural Architecture Search (NAS) is the line of research that tries to automatically design architectures with better performances at a given task. The performance of a network in a task can be predicted by a score, even before the network is trained: this is referred to as zero-shot NAS. However, the existing score remains unreliable for architectures with high accuracy. We develop in this direction by exploring different related scores. We study their time efficiency and we improve on their dependence with the final accuracy, especially for high values of the score. We propose a monotonicity metric to evaluate the adequate relative scoring of the architectures, as a way to avoid imposing a linearity assumption too early. We find that our use of noise improves the score, but a more substantial improvement comes when the evaluation of the score is done in the parameter space. We hope this effort will help clarify promising directions to speed up automatic discovery of good neural architectures without training.

1 Introduction

Machine learning in the deep learning era has proven to be able to solve impressively complex tasks (Krizhevsky et al., 2012; Silver et al., 2018; Badia et al., 2020). However the process of finding the best architecture for a given task is still cumbersome. This process typically involves training multiple networks, and different research groups proposing incremental improvements, which can take years to develop. Given the inefficiency associated with finding the best architecture for solving a task, a line of research has been focused on improving the speed at which the optimal architecture can be found for a given task.

A recent significant step forward has been taken in (Mellor et al., 2020), where a score has been proposed to predict how well an architecture will perform in a task before training. The score relies exclusively on the Jacobian of the architecture with respect to the input, which represents the task. A drawback of such score is that it seems to be linearly correlated to the final accuracy for low values of the score, but fails to discriminate architectures for higher values of the score. The computational cost of their approach is dominated by an eigenvalue decomposition that has a complexity of $O(B^3)$, where B is the number of samples in the batch. This can be problematic when larger inputs are considered, or if the metric has to be repeated at different layers in the architecture or with different noise conditions to improve its dependence with the final accuracy.

Our contributions are therefore:

- a new set of scores to predict how an architecture is going to perform, that are more discriminative at higher levels of accuracy
- we propose a metric based on the degree of monotonicity of a function in order to quantify the quality of each score without assuming a linear dependence

2 Related Work

Neural architecture search can be generalized as three distinct problems: 1. defining a search space, 2. evaluating the networks in the search space and 3. navigating the search space based on a given heuristic. Most works rely on the training or validation accuracy for the second problem, which is expensive to compute. Training a network is time-consuming, especially if some hyperparameters search must be added on top. Standard techniques rely on training with a reduced training set in either size (Xu et al., 2020) or dimension (Chrabaszcz et al., 2017), without hyperparameters tuning or with a limited number of epochs (Baker et al., 2017). Unfortunately, these simplifications can add a significant bias on the reliability of the computed heuristic (Zela et al., 2018).

Reusing trained weights between similar networks in the search space can bring down the computational cost. Network morphism consist of growing a network while keeping the originally trained weights (Wei et al., 2019) in the network, which is a good way of keeping training requirements at a minimum. Similarly, one shot architecture search methods do the opposite. By training a large architecture representing several networks in the search space, one can find sub-graphs with pre-trained weights and therefore avoid subsequent training of the sub-graphs. This method unfortunately will not scale well with larger search spaces. Moreover, it is still unclear if the relative accuracy of the sub-graphs is maintained (Sciuto et al., 2019).

Unlike one shot NAS where one model still needs to be trained, zero shot NAS methods attempt to predict performance before actual training. The networks in the search space are evaluated by a score without access to post-training performances or weights. These methods are the ultimate goal in solving the aforementioned second problem of NAS. To the knowledge of the authors, only (Mellor et al., 2020) has obtained such a metric. While their work is promising, the proposed method fails to distinguish between high-performing network architectures, which is what is going to be addressed in this paper.

3 Methodology

3.1 Datasets

We perform our tests on one batch of 256 samples from CIFAR 10, CIFAR 100 (Krizhevsky, 2009) and ImageNet 16 (Russakovsky et al., 2015). We test our proposed scores on the architectures from the NAS-Bench-201 (Dong and Yang, 2020), which are a collection of $5^6 = 15625$ architectures, built by combining 5 standard blocks for image processing (Zero, Identity, 3×3 convolution, 1×1 convolution, 3×3 average pool), on a graph with 6 fixed edges. All the architectures have been trained on the mentioned datasets by the authors of NAS-Bench-201, to make the final score available, and alleviate the comparability problem faced by the NAS community. The scores we propose below are applied on the untrained networks. The gradients at initialization can then be calculated and used before applying the update rule on the parameters (training).

3.2 Quantifying Future Performance Before Training

Jacobian Score (JS): This score was proposed by (Mellor et al., 2020) and was built on the observation that the Jacobians with respect to the inputs for each sample in a minibatch appeared to be most uncorrelated for the architectures that perform best after training. We first compute the Jacobian of the loss of the network $\mathcal{L}$ with respect to the samples in the input batch $J_B = \left(\frac{\partial \mathcal{L}}{\partial x_1}, \cdots, \frac{\partial \mathcal{L}}{\partial x_B}\right)$, where B is the total number of samples in a batch. We then compute Σ_J, the correlation of those vectors. The eigenvalues $\sigma_{J,i}$ of Σ_J allow us to compute their proposed score:

$$S = -\sum_{i=1}^{B} [\log(\sigma_{J,i} + k) + (\sigma_{J,i} + k)^{-1}]$$

which represents the negative KL divergence between an uncorrelated Gaussian and the Gaussian with kernel Σ_J. k is a small constant chosen for numerical stability as $k = 10^{-5}$. We normalize their metric to have values between zero and one, rescaling and shifting with the maximum and minimum values from their experiments, in order to compare it easily with the scores proposed below. As well, given that the accuracy is bounded between 0% and 100%, it seems desirable that a score that predicts accuracy should be bounded too.

Jacobian Score Cosine (JSC): We designed this score substituting the eigenvalue based similarity performed on the JS score, by a cosine similarity between the Jacobians of each input sample. The goal was to take advantage of the $O(B)$ complexity of the cosine similarity, versus the $O(B^3)$ complexity of calculating the eigenvalues, operation needed in (JS). Our proposed variation is

$$S = 1 - \frac{1}{B^2 - B} \sum_{i=1}^{B} |J_N J_N^T - \mathbb{1}|^{\frac{1}{20}}$$

where Jacobian J is flattened, and normalized into J_N, to have B vectors of module one. We perform the comparison of the Jacobian of every sample by multiplying all the pairs, and we remove the diagonal in $p = J_N J_N^T - \mathbb{1}$. Experimentally we found that our metric performed best when Jacobians in the same direction were given the same value of one as Jacobians in opposite direction, while orthogonal Jacobians had to be considered different, that's why we take the absolute value of p. We found that taking the 20^{th}-root of that number improved the scoring, but there is room for fine-tuning. Finally, we take the mean of the non-diagonal elements. The final one and the subtraction outside the mean, makes sure that orthogonal Jacobians are given the highest value. The calculation of roots makes this score $O(B \log B)$ instead of $O(B)$, which is still an improvement over $O(B^3)$.

Small Noise Score (SNS): With this score we wanted to capture the notion that a good architecture should be robust against noise, and end up making the same decision as it would without noise. We propose a few variants based on defining the noisy version of the batch as $noisy_{B,a} = x_B + a \cdot var(x_B) \cdot \epsilon$, with noise of different amplitudes a and where $\epsilon \sim \mathcal{N}(0, 1)$. We define the Noised Score like $NS_a = 1 - |JS(x_B) - JS(noisy_{B,a})|$, since experimentally we found it performed better than $NS_a = JS(noisy_{B,a})$ and $NS_a = JS(x_B) \cdot JS(noisy_{B,a})$ (results not shown). Therefore we define our first noisy score with a small noise: $SNS = NS_{1/10}$.

Small Noise Score Cosine (SNSC): Equivalent to SNS but changing JS by JSC in the definition, to understand if decreasing the complexity of the similarity operation could bring an advantage.

Large Noise Score (LNS): Equivalent to SNS with a larger noise, $LNS = NS_{10}$, to understand the dependence of the noised score on the strength of the distortion of the input.

Large Noise Score Cosine (LNSC): LNSC is equivalent to LNS, performing cosine similarity instead of the eigenvalue based similarity, to understand again the effect of the complexity of the similarity on speed. Therefore $LNSC = 1 - |JSC(x_B) - JSC(noisy_B)|$.

More Noised Jacobian Score (MNJS): In this line of reasoning, we hypothesized that the architectures should be robust to different degrees of noise. Therefore, we propose the metric $MNJS = JS \cdot NS_{1/10} \cdot NS_{10} \cdot NS_{100}$.

More Noised Jacobian Score Cosine (MNJSC): Equivalent to the previous one but changing JS by JSC in the definition to understand the effect of the choice of similarity operation.

Parameters Jacobian Score (PJS): In a different line of reasoning, we argue that the value of the Jacobian in the parameters space should have a larger impact in the final performance than the Jacobian with respect to the input. Therefore we take the mean of the eigenvalue based score performed on the parameters of each layer. Let $(W_i)_{i=1}^{N}$ be the sequence of parameter tensors, where i starts closer to the input and ends closer to the ouput of the network, N represents the number of parameter tensors in the network, and each tensor has shape $shape(W_i) = (j_1, j_2, \cdots, j_{K_i})$, where $j_k, K_i \in \mathbb{N}^+$. In contrast to JS, the similarity is not evaluated among different samples in the batch, since the gradient update of the parameters is taken over the mean loss over the samples of the batch. Once the Jacobian of the loss with respect to the tensor W_i is computed, it is flattened, keeping intact only the first axis j_1. The good performance of this choice is not yet well understood and it has to be investigated further. We will call the resulting collection of vectors as J_{j_1}. We compute the correlation of this collection of vectors, and performed its eigenvalue decomposition to use the similarity definition of JS and assign a score for each parameter tensor S_i. The final score is the mean over S_i, $S = \frac{1}{N} \sum_{i=1}^{N} S_i$.

Parameters Jacobian Score Cosine (PJSC): An equivalent of the previous score where the eigenvalue-based similarity has been substituted by a cosine similarity.

	JS	JSC	SNS	SNSC	LNS	LNSC	MNJS	MNJSC	PJS	PJSC	WPJS	WPJSI
noise			×	×	×	×	×	×				
parameter space									×	×	×	×
cosine similarity		×		×		×		×		×		

Table 1: **Summary of scores features.**

Weighted Parameters Jacobian Score (WPJS): Trying to capture the intuition that the representations closer to the last layers might benefit more from being disentangled than the layers at the input, we perform again PJS at each layer, but this time weighting the mean by giving higher weight to scores of parameter tensors close to the end. The resulting score is $S = \frac{1}{N} \frac{2}{N(N+1)} \sum_{i=1}^{N} i \cdot S_i$, where the extra normalization term makes sure the score is still bounded between 0 and 1.

Weighted Parameters Jacobian Score Inverse (WPJSI): In case our hypothesis on WPJS was wrong, we tested the opposite hypothesis: early layers will be more influential to the final score. Here we define the new score as $S = \frac{1}{N} \frac{2}{N(N+1)} \sum_{i=1}^{N} (N + 1 - i) \cdot S_i$.

3.3 Quantifying the quality of the score

As the authors of Mellor et al. (2020) noted, JS tends to be more discriminative for low values of the score than for high values. This can be problematic, given that we are interested in finding the architecture with the highest performance. We use the slope of the linear regression between the test accuracy and the score to study the quality of each score. However using the slope relies on the assumption that the dependence is linear. To avoid such a strict assumption on scores that we might not understand well enough yet, we introduce a monotonicity score between sequences X and Y:

$$monotonicity(X, Y) = \frac{2}{N_S^2 - N_S} \left| \sum_{j=1}^{N_S} \sum_{i>j} sign((x_i - x_j) \cdot (y_i - y_j)) \right|$$

where $X = \{x_i\}_{i=1}^{N_S}$ and $Y = \{y_i\}_{i=1}^{N_S}$, N_S is the number of samples, and $sign(\cdot)$ is the sign function. The absolute value makes sure that both extremes of the sign function are good: when X and Y grow together $sign(\cdot) = 1$ and it should be classified as monotonic, and when X and Y grow consistently in opposite directions $sign(\cdot) = -1$ and it should be classified as monotonic too. The non-monotonic case happens when X and Y do not grow consistently together. The score will be one for monotonic functions, and lower for functions that are not consistently monotonic. Our definition is related to the Kendall-tau correlation (Kendall, 1938): $monotonicity(X, Y) = |\tau(X, Y)|$. We performed the metric on several familiar functions in figure 1.

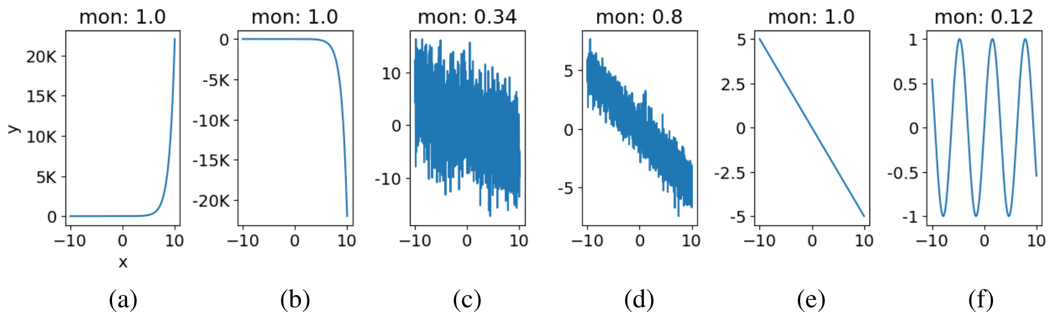

Figure 1: **Monotonicity score on familiar functions.** Panels (a) and (b) show how both $y = \exp(x)$ and $y = -\exp(x)$ are assigned the same monitonicity value. Panel (c), (d) and (e) show how increasing the Gaussian noise on top of $y = -\frac{1}{2}x$, decreases the monotonicity of the dependency. Last panel (f) shows how $y = \sin(x)$ has a very low value of monotonicity, different from zero given that the values at $x = -10$ and $x = 10$ are not at the same height. By design, exponential and linear monotonicity are given the same monotonicity value.

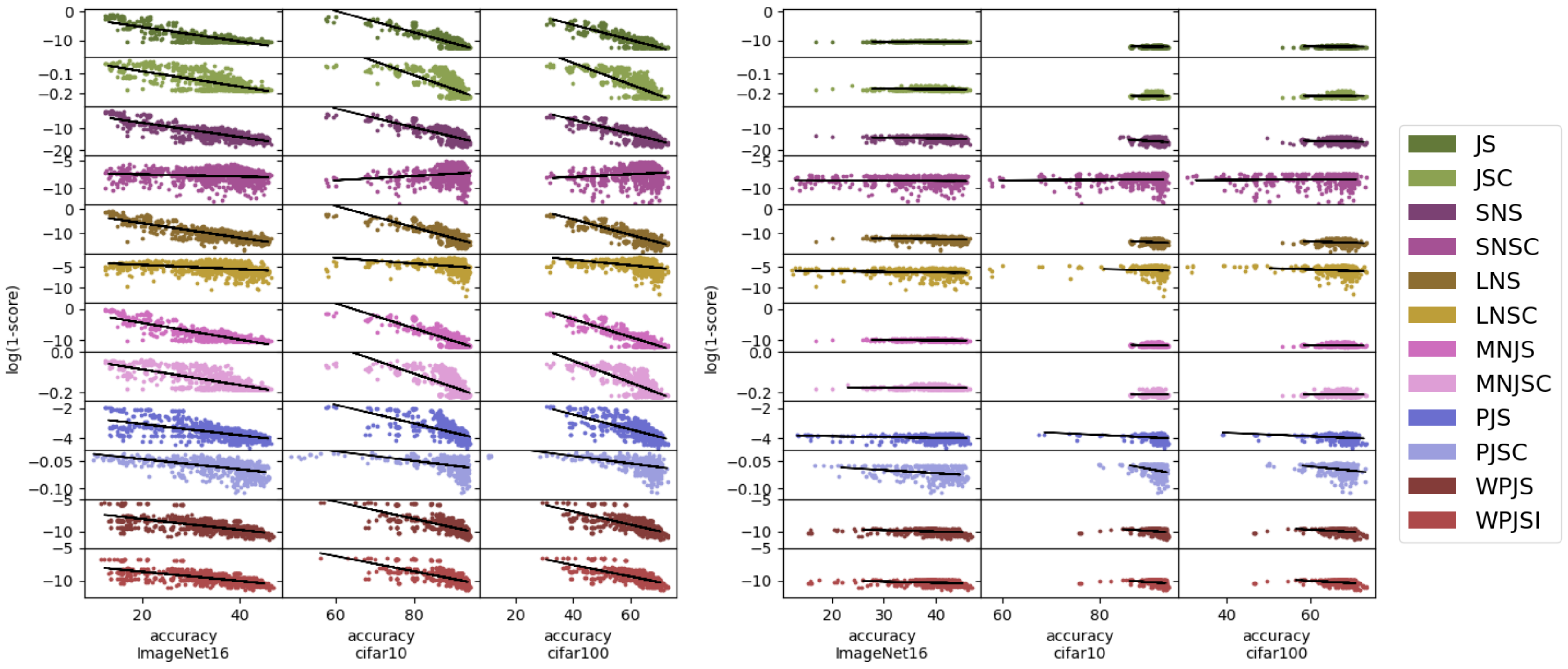

Figure 2: **Metrics score versus accuracy.** We plot $log(1 - score)$, to better display the distribution of points, against the final accuracy of a given architecture. A point in the panel corresponds to a given architecture. On the left all 1000 points drawn randomly from the NAS-Bench-201 dataset are displayed, while on the right only those with a score higher than the median score. We can see that all the metrics seem to have a reasonable linear dependence with the accuracy on the left panel. If we want a reliable metric, it has to be useful for all its range of values. The authors of Mellor et al. (2020) found that their score was not discriminative for high values, which can be observed in the right panel and compared with other metrics. Visually, all scores seem to suffer from the same behavior, we quantify the degree in the rest of the article.

4 Results and Discussion

We show in figure 3 left panel how most of our proposed scores outperform the original JS at finding on average a better architecture. Some of them do so with a lower variance in the decision. It is especially remarkable that our hypothesis that the parameters space was more predictive turns out to be consistent with our findings. Cosine similarity does improve the speed, but it is typically a minor improvement of less than 5% the time, as it can be seen in figure 3 right panel. The gain is expected to improve with the increase of the batch size, and when the similarity is evaluated several times in the score, as it is the case for example in MNJS and MNJSC, where we see a more pronounced gain in speed. Noisy scores seem to stay on par with the original one in terms of accuracy, with a loss in speed, given that they rely on evaluating the original JS or JSC several times: on the clean batch x_B and on several $noisy_{B,a}$.

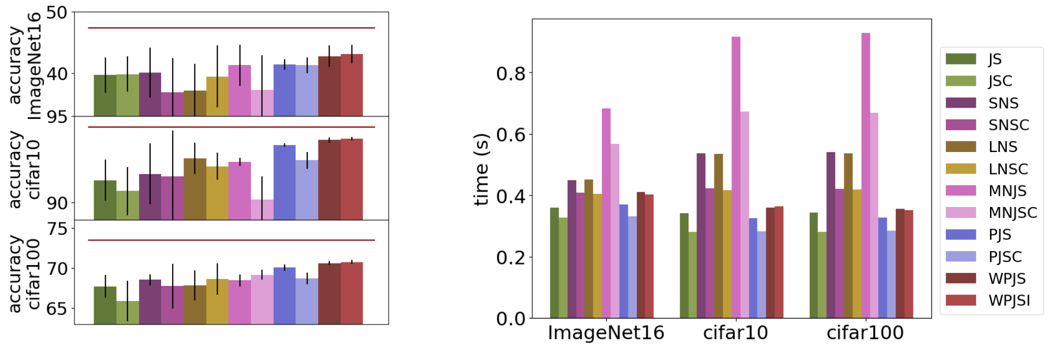

Figure 3: **Best accuracy retrieved by each metric and evaluation time per architecture.** On the left, we performed all our scores on CIFAR-10, CIFAR-100 and ImageNet, and we plot the accuracy after training of the architecture with the highest score. The vertical colored bar represents the mean best accuracy found over 5 runs of 1000 samples, and the black line represents the standard deviation. The dark red horizontal line on top shows the highest accuracy in the NAS-Bench-201 dataset. Cosine similarity based scores perform similarly to eigenvalue-based similarity scores. Scores on the parameter space perform consistently better. On the right, the time each score took to evaluate one architecture. Cosine based scores give a small gain in speed. Parameter space scores have comparable times as input space scores.

We show in figure 2 how each score distributes the architectures with respect to the final test accuracy. On the left for 1000 different architectures and on the right the 500 architectures above the median score. Instead of plotting the score itself, we plot $\log(1 - score)$ to better display the distribution of dots. It can be seen that for high values of the score (low values of $\log(1 - score)$), both JS and JSC tend not to be discriminative anymore, while other metrics such as PJS or PJSC, seem to behave better in that regime. However the improvement is not visually clear.

We quantify this observation in figures 4 and 5. In figure 4 we compute the monotonicity of each score, for 1000 randomly sampled architectures from the NAS-Bench-201 dataset on the left, and for the 500 above the median score on the right. We consider that if the monotonicity is similar in both panels, independently of the region of the score, the score is more reliable. As it can be seen, most scores keep a monotonicity value on the right panel more similar to the one in the left panel, than the original JS, especially the scores in parameter space. To make sure the conclusions hold with more standard metrics, we performed the analysis comparing the slopes of the linear regressions. Similarly as for monotonicity, we consider that if the slope in the left panel is similar to the slope in the right panel, the score is more reliable, since it does not depend on the percentile region considered. A similar conclusion can be drawn when a linear regression is performed on $\log(1 - score)$ and the slopes are compared (figure 5). When all the randomly sampled architectures are considered, JS results in a linear regression with a pronounced slope, but for high values of the score, the slope drops to close to zero, while other metrics such as SS, LS, WPJS and WPJSI seem to be able to maintain a similar slope in both settings.

5 Discussion

Interestingly, both WPJS and WPJSI are the top performers, in terms of best accuracy found with smallest variance, best times and best monotonicities above the median score. The slight systematic improvement of WPJSI against WPJS, are not strong enough to suggests that Jacobians close to the input have more impact on final test accuracy. In fact, we observed that even the score $S = \frac{1}{N}\frac{2}{N(N+1)}\sum_{i=1}^{N} S_i$ improved over PJS, defined as $S = \frac{1}{N}\sum_{i=1}^{N} S_i$, but the reason remains not well understood. A stronger conclusion that can be drawn is that parameter space is more informative of final performance than input space given that PJS, PJSC, WPJS and WPJSI all outperformed the other scores in terms of best architecture selected (figure 3), monotonicity and slope (figures 4 and 5). It is remarkable as well that both WPJS and WPJSI placed the best architecture in the 1000 architectures sample, within the top 1.6 percentile in the score, for the three datasets, while the original JS score placed the best architecture within the 30.8 percentile.

The intuition that led us to investigate parameter space is that perpendicular gradients might be the early signature of the attempt of the network to build disentangled representations, which will ease decision making after training. It is reasonable to assume that disentangled representations are more effective if they happen at every intermediate layer than if they happen only at the input. This intuition seems supported by the present study. The intuition that led us to explore noise at the input is that an effective network should be able to make the same decision for distorted inputs. This intuition seems supported by the present study. The hypothesis that cosine similarity would bring a speed up is supported by the current study, but for the small batches used, it does not provide a significant improvement over the eigenvalue based similarity.

The fact that scores evaluated in the parameter space do not require the input to be differentiable, eases its use for language models. However our preliminary studies suggest that more research is necessary to generalize this line of investigation to models that do not involve convolutions.

Combining this family of scores with approaches such as AutoML-Zero (Real et al., n.d.) by leveraging both techniques strengths: it would make the discovery process (1) more efficient than training all networks, (2) more efficient than scoring all networks randomly generated, and (3) more efficient than training all networks evolved since the score would give an estimate of the performance after training.

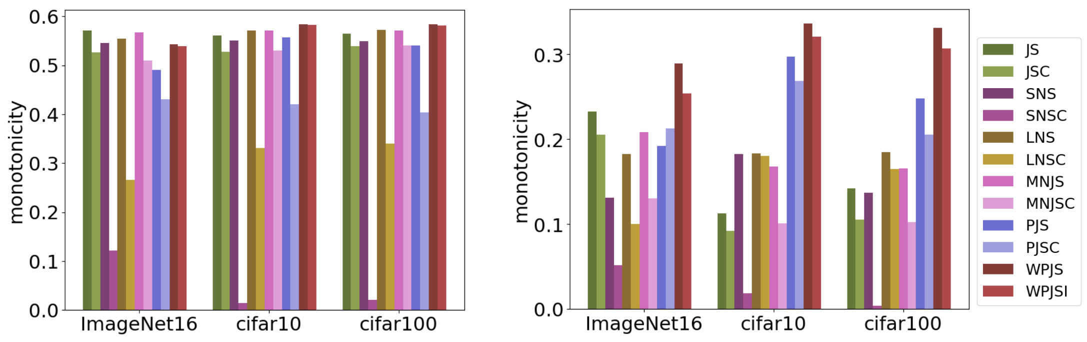

Figure 4: **Monotonicity of each score with respect to the accuracy.** On the left, evaluated on 1000 randomly drawn architectures from the NAS-Bench-201, on the right evaluated on the architectures above the median score. Even if the quality of the monotonicity is high for JS when considering the 1000 architectures, the decrease in monotonic quality is more pronounced for higher values of the JS score than for most of the other scores. Parameter space metrics seem to resist better.

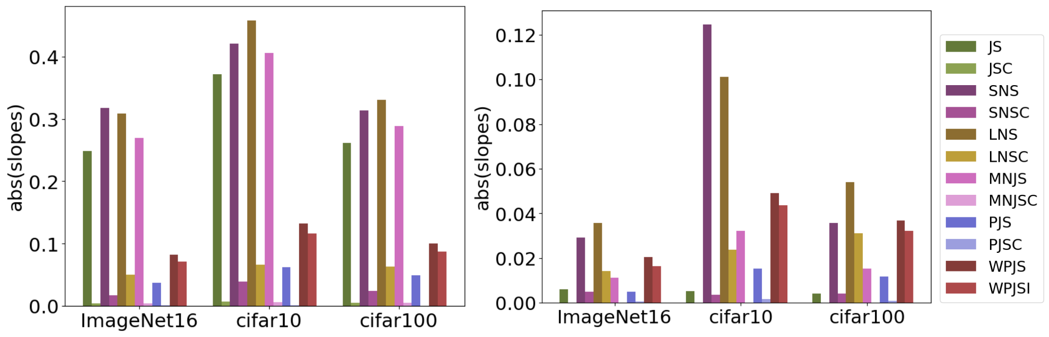

Figure 5: **Slope of each score with respect to the accuracy.** On the left, the slope of the linear approximation of the $\log(1 - score)$ against accuracy dependency of 1000 architectures sampled from the NAS-Bench-201 dataset. On the right the slope of the linear regression performed on the architectures with a score above the median. The absolute value of the slope is shown to ease the comparison with monotonicity. The slope of JS dependency decreases drastically, while metrics such as LS, SS, WPJS and WPJSI keep a consistent slope in both scenarios. A desirable property for a score is that a consistent slope should be maintained throughout its range, without drastic changes for different percentiles, to ease its usability.

6 Conclusion and Future Work

A desirable property for scores that predict the final accuracy of an architecture on a given task is that the score has a monotonic or even linear correspondence with the final accuracy. We proposed a family of scores that derive from the original proposition JS. Final best accuracy architecture retrieved by noised scores was generally better than the original JS, but scores in the parameter space gave a more pronounced improvement of the best architecture retrieved.

We propose a monotonicity metric to measure how each score follows the validation accuracy. We use the monotonicity score and the slope of the linear regression on the data to conclude that scores that evaluate the Jacobian in the parameter space are more robust, and the addition of noise to the original score can improve its behavior. The scores based on adding noise to the input gave an improved monotonicity on CIFAR10, but less so on CIFAR100, and they had worse monotonicity on ImageNet16. Parameter space based scores did seem to improve the monotonicity more consistently. Zero shot Neural Architecture Search holds the promise of drastically speeding up progress in efficient architecture design and therefore deep learning and AI in general.

References

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, D. and Blundell, C. (2020), ‘Agent57: Outperforming the atari human benchmark’, *ArXiv* **abs/2003.13350**.

Baker, B., Gupta, O., Raskar, R. and Naik, N. (2017), ‘Practical neural network performance prediction for early stopping’, *ArXiv* **abs/1705.10823**.

Chrabaszcz, P., Loshchilov, I. and Hutter, F. (2017), ‘A downsampled variant of imagenet as an alternative to the CIFAR datasets’, *ArXiv* **abs/1707.08819**.

Dong, X. and Yang, Y. (2020), ‘Nas-bench-102: Extending the scope of reproducible neural architecture search’, *ArXiv* **abs/2001.00326**.

Kendall, M. G. (1938), ‘A new measure of rank correlation’, *Biometrika* **30** (1-2), 81–93.

Krizhevsky, A. (2009), ‘Learning multiple layers of features from tiny images’, *Technical Report.* .

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012), Imagenet classification with deep convolutional neural networks, *in* ‘Advances in neural information processing systems’, pp. 1097–1105.

Mellor, J., Turner, J., Storkey, A. and Crowley, E. J. (2020), ‘Neural architecture search without training’, *ArXiv* **abs/2006.04647**.

Real, E., Liang, C., So, D. R. and Le, Q. V. (n.d.), ‘Automl-zero: Evolving machine learning algorithms from scratch’, *ArXiv* **abs/2003.03384**.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. et al. (2015), ‘Imagenet large scale visual recognition challenge’, *International journal of computer vision* **115**(3), 211–252.

Sciuto, C., Yu, K., Jaggi, M., Musat, C. and Salzmann, M. (2019), ‘Evaluating the search phase of neural architecture search’, *ArXiv* **abs/1902.08142**.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T. et al. (2018), ‘A general reinforcement learning algorithm that masters chess, shogi, and go through self-play’, *Science* **362**(6419), 1140–1144.

Wei, T., Wang, C. and Chen, C. W. (2019), Stable network morphism, *in* ‘2019 International Joint Conference on Neural Networks (IJCNN)’, pp. 1–8.

Xu, M., Zhao, Y., Bian, K., Huang, G., Mei, Q. and Liu, X. (2020), ‘Federated neural architecture search’, *ArXiv* **abs/2002.06352**.

Zela, A., Klein, A., Falkner, S. and Hutter, F. (2018), ‘Towards automated deep learning: Efficient joint neural architecture and hyperparameter search’, *ArXiv* **abs/1807.06906**.