PAC-Bayes Inspired NAS Without Training

James Bowden, Steven Csaposs, Thomas Hoffmann

June 2021

Abstract

Many state-of-the-art Neural Architecture Search (NAS) techniques require training steps that are computationally expensive and work exclusively with labeled training data. In this paper we investigate novel NAS objectives inspired by PAC-Bayes theory that require no training and no labels. With our objectives, we can evaluate a neural architecture two orders of magnitude faster than a training based approach while still finding high-performing architectures. We show that PAC-Bayesian objectives are promising to quickly evaluate neural architecture performance.

1 Introduction

It is unlikely that a single architecture can outperform all others on every task. Traditionally, finding a high performing neural architecture for a particular task involves human intuition, experimentation and effort. Recent advances in the field of Neural Architecture Search (NAS) show that automated procedures for generating neural architectures often results in a state-of-the-art architecture for a given task. NAS is typically done in a task-agnostic manner that requires minimal knowledge of the task by the search algorithm. However, a drawback of NAS is that it tends to be slow and resource intensive, consuming several GPU hours on the low end and many GPU days on high end.

Many NAS techniques use a controller to propose neural architectures, as was done in the seminal work by Zoph and Le[10]. After the controller proposes an architecture, it is scored according to some objective. Typically the objective is simply the accuracy (or other target) of the model after some training, but recent work shows that an objective can result in good architectures if it is just correlated with accuracy[6,8]. This score is used to update the controller such that future architecture proposals are more likely to result in high scores. We approach NAS with a Bayesian Optimization controller, though our objectives are general and can be readily extended to other commonly used controllers.

We investigate PAC-Bayes evidence motivated objectives to perform NAS without training, with the goal to improve NAS efficiency. PAC-Bayes theory suggests that the performance of a neural network improves with a larger solution space. We apply different objective functions that seek to measure this solution space to evaluate performance of different architectures without training. We find success with both a contrastive objective, based on unsupervised triplet loss, and a perturbation objective that measures a neural network's robustness to dropout.

2 Approach

2.1 Overview

On a high level, we run NAS via Bayesian optimization in order to identify high-performing convolutional neural network (CNN) architectures on the MNIST dataset[5]. We aim to learn effective CNN architectures. This is more challenging than NAS over simpler neural networks, like feedforward networks, but much more representative of the kind of flexibility we would like a neural architecture search to have.

We simplify the NAS by only optimizing a block of CNN operations, which receive the output of a 3×3, 64 filter convolution followed by a 2×2, stride 2 max pooling layer as an input. The output of the optimized block is followed by global average pooling and a softmax layer. We represent the block as a list of layer dimensions and an accompanying lower-triangular adjacency matrix which describes the connections between individual neurons and thus can account for irregular architectures including skip connections while disallowing cyclic connections. When a particular convolution receives multiple inputs via skip connections, the inputs are concatenated along the filter-dimension. Each convolution operation in the CNN is followed by a ReLU activation. We allow up to 5 convolution operations in our block design.

Bayesian optimization aims to find the best possible set of layer dimensions and adjacency matrix, as evaluated on an objective function. We use a variety of objective functions, to be described below. At each step of the optimization, we pass a suggested architecture along with a small training dataset of 100 unlabeled images into the objective function, and use the returned score to continue Bayesian optimization. At the end, we sample 50 architectures that trained to completion for 10 epochs. We then evaluate the networks on the MNIST test dataset and compare correlation with our objective scores in order to analyze how predictive a certain metric is.

Our objective metrics are motivated by the PAC-Bayes evidence, defined as

$$e = \frac{\text{Vol(solutions)}}{\text{Vol(weight space)}}$$

PAC-Bayes theory suggests that networks having a larger evidence correlates with better generalization. In particular, because we're looking at the volume of solutions with respect to the weight space, it's not necessarily important that networks are trained out to calculate their evidence. In fact, we would expect random solutions (weights) from a network architecture with more evidence to generalize better to a given dataset. This is because the evidence is the proportion of solutions to all possible weights, and this is effectively the probability of drawing a solution at random. This motivates our general method:

1. Generate a network architecture via Bayesian optimization.
2. Randomly initialize network weights (untrained).
3. Compute objective metric given weights, training data (without training weights).
4. Repeat steps 2-3 n times (e.g. 100).

5. Aggregate scores and return.

The key part of this algorithm is that we only sample n weight initializations from the weight space. For large neural networks, as is the relevant application, the weight space is prohibitively large and it quickly becomes intractable to calculate evidence. As such, we aim to approximate the evidence in an effective but computationally tractable manner. Given that random solutions pulled from an architecture with more evidence should generalize better, we generally use performance of these randomly sampled weights on the training data as a proxy for evidence, which we then use as a predictor for overall test performance of a network.

2.2 Related Work

Mellor et al.[8] finds that training is not necessary for discovering neural architectures that perform substantially better than random. Their approach linearizes neural networks with respect to an input, and scores a neural network during NAS based on the correlations between the slopes of the linearization, arguing that lower correlations tend to produce higher scoring networks. While Mellor et al. argues that their approach is largely insensitive to weight initialization, they do not explicitly control for the effect of weight initialization in their objective. In contrast, our approach of averaging a score over many weight initializations should reduce the impact of random weight initializations and better ensure that our objective measures the property of just the neural architecture rather than a property of a particular weight initialization.

Liu et al. find that unsupervised objectives can perform similarly to supervised objectives in neural architecture search, and that an architecture's unsupervised score is highly correlated with its test score after supervised training[6], but their approach still relies on expensive training steps. We use a similar evaluation of architecture during neural architecture search using our metric learning NAS objective, but our approach does not require any expensive training step during NAS.

There has also been some work on calculating the PAC-Bayes evidence in a principled way. In particular, we examine three approaches suggested in lecture. The first attempts to estimate evidence by looking at the flatness of the weight space–in theory, a weight space that is flatter is more likely to generalize well[4]. Here, we define flatness to be the maximum angle α that we can arbitrarily rotate every neuron through without causing any misclassifications[7]. This is essentially a measure of how much we can perturb a given solution without it giving different predictions. Here, it is suggested that we calculate this α for each distinct solution, which is not very feasible computationally. We revisit the main idea of this approach though as a motivation for our perturbation objective. It is unclear how best to perturb a high-dimensional weights vector in a principled manner for the sake of determining flatness. Liu et al. introduce a neuronal rotator algorithm that constrains the weight vector of each neuron to the unit hypersphere[7]. The idea here is that a small perturbation is then a rotation of the vector by a small angle. They show that networks can actually be trained using this procedure, and propose it as an optimizer. We're more interested in using this method to determine α-robustness (i.e. how flat the weight space is around a given solution). The second approach acknowledges that we cannot determine how many distinct solutions

exist in the weight space, and instead focuses on finding PAC-Bayes bounds for many priors of varying size in order to select the best one[2]. The final approach notes that the structure of the solution manifold in function space for binary classification is an orthant, and thus uses the neural network - Gaussian process correspondence to estimate the evidence as an infinitely wide neural network[1]. However, it is less clear how to extend this on non-binary classification problems such as MNIST, which we would like to consider here.

2.3 Naive PAC Bayes Objectives

We began by defining objectives based on models with small amounts of training to establish a baseline for our search. For each architecture, the weights were randomly initialized and the model was trained on a small set of 100 labeled images. Each model was then tested. This process was repeated 100 times for each architecture, and average training loss and testing scores were recorded. From these scores, we defined an objective that is the average cross entropy training loss during training. We also define a second untrained pseudo-relabeled objective which is the untrained accuracy at classifying whether an image is of a digit is below 5 or above 4. We apply a "pseudo-relabeling" to this objective, where the image class can be swapped to maximize accuracy. This pseudo-relabeling exploits a symmetry that if the 1st neuron performed better for the 2nd class and vice-versa, then we can swap the weights of the 1st and 2nd output neurons to improve the loss.

With these objectives, we implicitly assume that training loss of a lightly trained or untrained architecture is a good measure for the overall quality of the architecture. These objectives will also provide a baseline to compare against our objectives that do not require training or labeled data.

2.4 Contrastive Objective

The contrastive objective performs triplet loss with cosine distance on an anchor, positive and negative image. We generate a positive and anchor image by performing two different augmentations on the same image, and generate a negative image by performing an augmentation on another image sampled from the same dataset. The cosine distance triplet loss then scores how similar the output of the network is for the anchor and positive samples, and how dissimilar the negative sample is from the anchor. This triplet loss is computed 100 times on different weight initializations per NAS iteration, and the loss is averaged over each of the 100 runs to calculate the final value of the contrastive objective. We use this objective in NAS to select for networks that are robust to augmentations of the same image, but still can differentiate between distinct images.

The augmentations performed are a random rotation up to 35 degrees, randomly zooming into a section of the image as small as 75% of the original, and randomly erasing a block (no larger than 1/3 the original size) of the image with a 50% probability.

2.5 Perturbation Objective

This objective takes inspiration from the aforementioned flatness approach to calculating evidence. It is difficult to calculate α and practically impossible to calculate the number

of distinct solutions in the weight space, so we instead devise an alternative measure of α-robustness that captures the same idea. We sample weight vectors from the weight space (and take these to be representative of the probability of a solution being drawn), and for each weight vector, perturb it. We would like to perturb the weight vector by some small angle, and given that we are working in very high-dimensional space, we decided to use Monte Carlo dropout (MC dropout) with $p = 0.25$ to enact these perturbations[3]. This boils down to applying dropout layers on the network during prediction time, and is justified as zeroing some of the weights is equivalent to perturbing the weights vector by some angle, just less principled than some sort of neuronal rotation method[7]. Dropout is chosen for its intuitive and computational simplicity. After performing inference with the initial random weights and their perturbations, we return the mean cosine distance. Cosine distance is employed to avoid penalizing networks with more weights, since those are often the most successful in practice. The key idea is that the smaller the distance between the predictions of a set of weights and its perturbations, the flatter the weight space is in that region, which we use as a proxy for estimating the evidence of the network.

3 Results

3.1 Naive Training Accuracy

Figure 1 shows the relationship between the training loss and performance of the neural network architecture. We observed a strong correlation between the average training loss of the networks trained on 100 images and the accuracy of the fully trained architecture. Figure 2 shows the relationship between the testing error of these networks and the accuracy of an untrained network. We found that the testing error of the untrained networks has no clear relationship with the quality of the architecture.

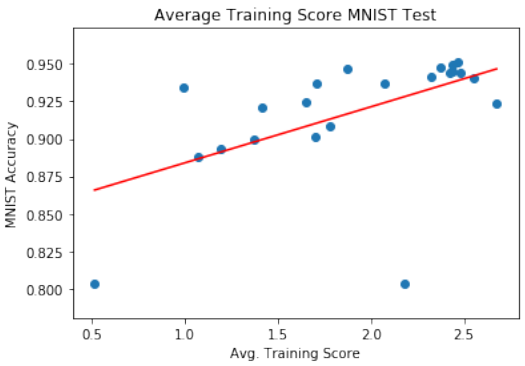

Figure 1: Average training score plotted against MNIST accuracy for 25 architectures evaluated during NAS.

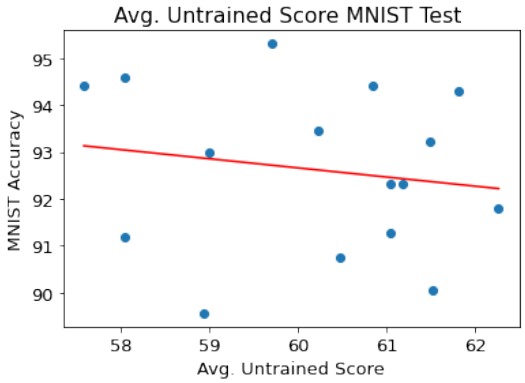

Figure 2: Untrained accuracy over 100 images plotted against MNIST Test accuracy for 20 architectures evaluated during NAS.

We suspect that the error metric of an untrained network was too random to produce a useful objective, even once performing a pseudo-relabeling where we map an output neuron

of the network onto the classes that would minimize overall error. However, the training score objective is able to predict the success of fully trained neural architectures ($0.299R^2$). This is sensible because a network architecture that trains effectively across many different weight initializations likely fits the data well. Because this objective requires training, we would like to explore other options for objectives that will result in a faster NAS, though it serves as a supervised baseline on the small dataset.

3.2 Contrastive Objective

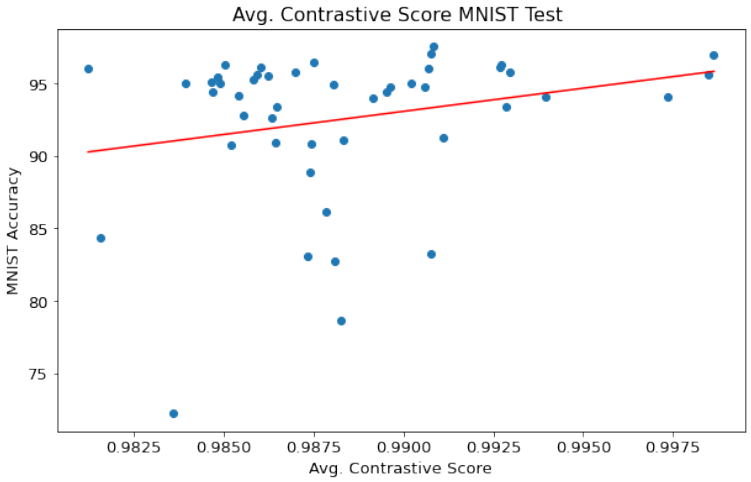

Figure 3: Contrastive score plotted against MNIST test accuracy for 47 of the 500 architectures evaluated during NAS. Of 50 selected for test evaluation, 3 were disconnected network architectures.

We evaluate 500 architectures during neural architecture search with the contrastive NAS objective, and select 1 in 10 (in sequential order) for supervised training on MNIST data over 10 epochs and testing on the MNIST test set. Of these 50, 3 were not trainable due to a lack of connectivity in the NAS block. We then regress a line over remaining contrastive scores and MNIST test accuracy and find that our contrastive score explains 5.6% of the variance (0.056 R^2) in test accuracy (Figure 3). Within our the sample of 50 networks, only 2 architectures had higher test accuracies than the highest-scoring architecture according to the contrastive score (architecture shown in Figure 4, Right).

3.3 Perturbation Objective

For the perturbation objective, we randomly sampled 30 weights vectors, and for each, applied MC dropout with $p = 0.25$ 5 times. We then measure the average cosine distance of these predictions (meaning lower corresponds to more evidence), and invert the score such that higher scores correspond to more evidence instead (shown on the x-axis of Figure 5). On the y-axis we have MNIST test accuracy for the fully trained network architectures. We see a pretty good correlation between our distortion score and true test accuracy with $0.170R^2$.

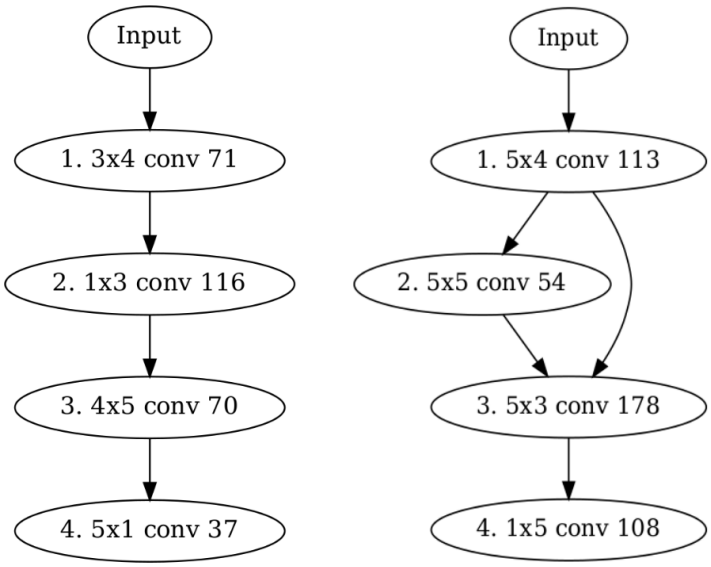

Figure 4: Block architecture with the highest contrastive score over the 500 architectures evaluated during the neural architecture search (Left) and the architecture with the highest contrastive score on the 50 sampled architectures for scoring (Right).

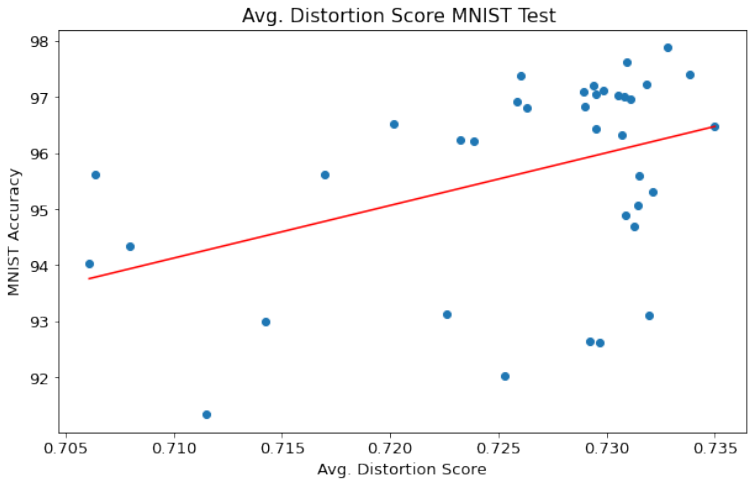

Figure 5: Distortion score plotted against MNIST test accuracy for 37 of the 500 architectures evaluated during NAS. Of 50 selected for test evaluation, 4 outliers with low distortion score were removed, and 9 disconnected architectures were removed.

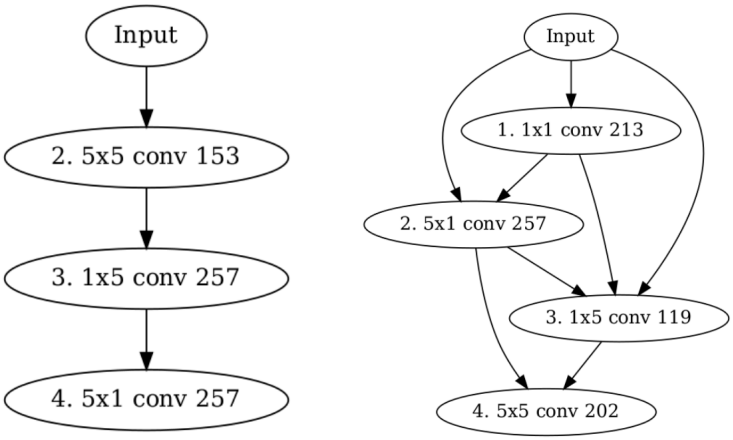

Figure 6: Block architecture with the highest perturbation score over the 500 architectures evaluated during the neural architecture search (Left) and the architecture with the highest perturbation score on the 50 sampled architectures for scoring (Right).

In other words, networks that score highly via this metric tend to perform better in practice. This objective yielded the strongest correlation of those we tested.

The top scoring networks according to the perturbation objective tended to reflect some human intuition on neural architectures. In particular, top scoring architectures tended to use $N \times 1$ and $1 \times N$ convolutions, similar to the use of these filter sizes in the Inception V2 architecture introduced by Szegedy et al.[9] The reasoning given by Szegedy et al. for this is that by applying an $N \times 1$ convolution followed by a $1 \times N$ convolution (or the reverse order), the network obtains an $N \times N$ receptive field with fewer parameters than a single $N \times N$ convolution.

4 Discussion

Of the objectives considered, we were able to obtain positive correlation between our scores and MNIST test accuracy with two of our objectives. The contrastive objective showed significant correlation, and the perturbation objective has even better predictive accuracy. Although the metrics we've developed don't perform as well as fully-fledged state-of-the-art NAS algorithms, they're more than 50 times faster to evaluate than methods using training (wall-time) and thus have potential as NAS objectives. Moreover, our approximation of the PAC-Bayes evidence for the two objectives we describe indicates that PAC-Bayes evidence can indeed be a good predictor for test accuracy of a network when trained. This has the potential to be very useful as a work-around for training networks out and using the training accuracy as a metric because computing scores on untrained networks is much less expensive.

5 Further Work

As this has been a relatively short project, there are many interesting future directions. The most obvious one is trying to devise other evidence-inspired objective functions to test in a NAS framework. We are interested in looking into metrics that are both intuitive (in lieu of being rigorously principled) and tractable as evidence proxies. With regards to the perturbation objective, a number of obvious questions emerge. How does this method perform when we use a more principled method of perturbation, or perturb the weights more thoroughly? Would a method like the neuronal rotator give better results[7]? Can we design an objective with an adversarial bent, that takes into account perturbations on the data side as well? It would be interesting to try different methods of measuring flatness in the weight space/function space and compare their performance.

On a more practical note, we would like to try to use our objective functions to filter poor architectures before moving on to a fully-fledged NAS procedure. This would reduce runtime significantly for a traditional NAS. An important next step would be to perform statistical analysis to ascertain reasonable confident bounds on the correlation of our score and actual test performance. This would prevent us from filtering high performing architectures with our objectives. However, we don't need to be overly concerned with throwing out the occasional good architecture since there are often many good architectures. Our ultimate goal is to identify architectures that are likely to perform well while removing those that are likely to perform poorly.

References

[1] Jeremy Bernstein and Yisong Yue. "Computing the Information Content of Trained Neural Networks". In: *CoRR* abs/2103.01045 (2021). arXiv: 2103.01045. URL: https://arxiv.org/abs/2103.01045.

[2] Gintare Karolina Dziugaite and Daniel M. Roy. "Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data". In: *CoRR* abs/1703.11008 (2017). arXiv: 1703.11008. URL: https://arxiv.org/abs/1703.11008.

[3] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: (2016). arXiv: 1506.02142 [stat.ML].

[4] Geoffrey Hinton and Drew van Camp. "Keeping Neural Networks Simple by Minimizing the Description Length of the Weights". In: *ACM Conference on Computational Learning Theory* (1993).

[5] Yann LeCun. "The MNIST database of handwritten digits". In: *http://yann. lecun. com/exdb/mnist/* (1998).

[6] Chenxi Liu et al. "Are Labels Necessary for Neural Architecture Search?" In: *CoRR* abs/2003.12056 (2020). arXiv: 2003.12056. URL: https://arxiv.org/abs/2003.12056.

[7] Yang Liu et al. "Learning by Turning: Neural Architecture Aware Optimisation". In: *CoRR* abs/2102.07227 (2021). arXiv: 2102.07227. URL: https://arxiv.org/abs/2102.07227.

[8] Joseph Mellor et al. *Neural Architecture Search without Training*. 2021. arXiv: 2006.04647 [cs.LG].

[9] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[10] Barret Zoph and Quoc V. Le. "Neural Architecture Search with Reinforcement Learning". In: *CoRR* abs/1611.01578 (2016). arXiv: 1611.01578. URL: http://arxiv.org/abs/1611.01578.