FEAR : Ranking Convolutional Architectures by Feature Extraction Capabilities

Anonymous authors
Paper under double-blind review

Abstract

The fundamental problem in Neural Architecture Search (NAS) is to efficiently find high-performing architectures from a given search space. We propose a simple but powerful method which we call FEAR , for ranking architectures in any search space. FEAR leverages the viewpoint that neural networks are powerful non-linear feature extractors. First, we train different architectures in the search space to the same training or validation error. Then, we compare the usefulness of the features extracted by each architecture. We do so with a quick training keeping most of the architecture frozen. This gives fast estimates of the relative performance. We validate FEAR on Natsbench topology search space on three different datasets against competing baselines and show strong ranking correlation especially compared to recently proposed zero-cost methods. FEAR particularly excels at ranking high-performance architectures in the search space. When used in the inner loop of discrete search algorithms like random search or local search, FEAR can cut down the search time by 3.8X to 1.33X without losing accuracy on CIFAR10, CIFAR100 and ImageNet16-120. We additionally empirically study very recently proposed zero-cost measures for ranking and find that they breakdown in ranking performance as training proceeds and also that data-agnostic ranking scores which ignore the dataset do not generalize across dissimilar datasets.

1 Introduction

Neural Architecture Search (NAS) (Elsken et al., 2019; Ren et al., 2021) is a sub-field of automatic machine learning (AutoML) (He et al., 2021; Hutter et al., 2018) where the aim is for algorithms to search for high-performing architectures instead of humans manually trying out many possibilities. Given the rapid advances in differentiable operator types and usual application-dependent open questions of number of layers, channels, input resolution, etc this leads to an explosion of combinatorial choices in architecture design space. Indeed, even ubiquitously used search spaces like the DARTS (Liu et al., 2019) search space contain approximately 10^{18} architectures (Siems et al., 2020). The fundamental problem in NAS is to search this combinatorially exploding space of architectures as efficiently as possible.

Current NAS methods conduct this search in one of mainly two different ways: 1. **Discrete methods** use methods that sample architectures from a search space, estimate the final performance of the architectures by training them partially and then update the sampling rule based on that estimate (Zoph and Le, 2017; Liu et al., 2020). These methods can be notoriously computationally expensive due to the cost of training each sampled architecture individually (White et al., 2019; 2020a). 2. **One-shot methods** on the other hand train a single large graph (colloquially known as 'supergraph') that contains all possible architectures in one single graph by sharing weights amongst common edges in architectures. Even though weight-sharing has many issues like optimization gap between training and evaluation and shallow sub-graphs training faster (Xie et al., 2020; Wang et al., 2021; Shu et al., 2020; Zela et al., 2020), by training a single graph the cost of gradient computation via backpropagation can be amortized over exponentially many sub-graphs. Popular approaches include bilevel optimization (Liu et al., 2019), single level optimization approaches (Chang et al., 2019) utilizing the Gumbel-Softmax trick (Jang et al., 2017; Maddison et al., 2017) or even direct optimization without architecture weights (Wang et al., 2021). Note that for both with and without weight-sharing one can use various classes of techniques. For example RL-based (Pham et al., 2018),

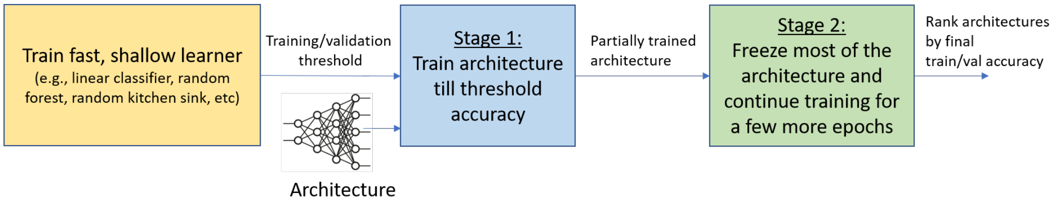

Train fast, shallow learner
(e.g., linear classifier, random forest, random kitchen sink, etc)
Training/validation threshold
Architecture
Stage 1:
Train architecture till threshold accuracy
Partially trained architecture
Stage 2:
Freeze most of the architecture and continue training for a few more epochs
Rank architectures by final train/val accuracy

Figure 1: Overview of FEAR which first trains a fast but shallow learner to get a reasonable training/validation error threshold and then trains the architecture in a two-stage procedure. In the first stage the architecture is regularly trained until it achieves threshold accuracy. In the second stage most of the partially trained architecture from the first stage is frozen and training continues for a few more epochs. All candidate architectures can then be ranked by final training or validation accuracy obtained via this two stage procedure.

evolutionary search (Liu et al., 2020), random search (Li and Talwalkar, 2020), local search (White et al., 2020a) and variants of Bayesian optimization (White et al., 2019).

We focus on the evaluation phase of discrete methods where it is often an ad-hoc choice on how and for how long to evaluate each sampled architecture for a given dataset. As Li and Talwalkar (2020) note, the partial training phase can often be critical in deciding how well a method performs. Crucially, they note that using partial training or validation error after many epochs of training can still not get the correct final rank of architectures. We propose a simple but powerful architecture ranking methodology that enables fast architecture ranking by leveraging the fact that neural networks are powerful feature extractors and the power of an architecture is dependent on how effective it is at extracting useful features from the inputs for the given task. We term this 'recipe' (see Figure 1 for an overview) for ranking architectures FEAR (FEATure-extraction Ranking). At a high level, first trains any architecture regularly till the architecture attains a specified training or validation accuracy threshold in the first stage. In the second stage, most of the architecture is frozen and the final few layers continue training for a bit longer. Architectures can be relatively ranked via their training or validation error after these two stages. We leverage insights in training dynamics of neural networks (Kornblith et al., 2019; Raghu et al., 2017) that show that early layers train very fast and become stable within a few epochs but later layers spend most of their time learning to make decisions from the features extracted by the earlier layers. A crucial aspect of our procedure is that we train architectures up to a specified threshold accuracy instead of committing apriori to fixed number of epochs. This exposes first order dynamics in neural network training where invariably (empirically) weak architectures either take much longer to reach threshold accuracy as opposed to stronger architectures (see Figure 2) or train fast but are unable to extract powerful features and hence lack performance. Such first order training dynamics can be exploited in any sampling-based discrete NAS algorithm. FEAR is discussed at length in Section 3.

We have two main contributions in this work:

- We propose a simple fast architecture evaluation method named FEAR and validate it on a variety of datasets on the Natsbench topological space benchmark against competing baselines.[1]
- We demonstrate how FEAR can be used in the inner loop of any discrete search algorithm to speedup architecture evaluations with minimal modifications.
- As an orthogonal contribution while evaluating strong baselines, we also empirically find that a number of very recently proposed lightweight ranking measures (Abdelfattah et al., 2021; Mellor et al., 2021) degrade in ranking performance as network training progresses and that data-agnostic ranking measures don't generalize across datasets. The performance of an architecture is a function of *both the topology and the dataset* (in addition to the training pipeline).

[1] Reproducible implementation of all experiments submitted herewith and will be made available on Github on publication.

2 Related Work

NAS research has received a lot of attention recently. We refer the reader to continuously updated excellent surveys (Elsken et al., 2019; Ren et al., 2021) and a near-exhaustive list of papers at (Deng and Lindauer, 2016) for an overview of the field. Here we discuss the works that are directly relevant to fast evaluation and ranking of architectures.

Architecture Performance Prediction: Baker et al. (2017) propose training regressors which take in the architecture, training hyperparameters and the first few validation accuracies as features and try to predict the final validation accuracy. The proposed method has a "burn-in" phase where a sampled few architectures are first fully trained to gather data for training the regressor. This regressor is then utilized in the rest of the pipeline. Similarly Rorabaugh et al. (2021) propose fitting curves to the performance numbers of the first few training iterations and extrapolate to later accuracy values. White et al. (2019) study neural network performance prediction in the context of Bayesian Optimization. Orthogonally, White et al. (2020b) conduct an extensive study of architecture encodings for common NAS algorithm subroutines including performance prediction which sheds light on the pros and cons of certain featurizations for this task. `FEAR` is orthogonal to the above body of work on architecture performance prediction and in fact can be used for further speeding up the performance prediction modules as one doesn't have to train the architectures fully.

Lightweight Architecture Evaluation: Zhou et al. (2020) search for combinations of reduced input image resolution, fewer epochs, and number of stem channels to find computationally cheap proxies for evaluating architectures while keeping their relative ranks the same. They find an optimal configuration of resolution, epochs and number of channels on a bag of 50 models. They term their method as EcoNAS. Abdelfattah et al. (2021) note that the configuration found by EcoNAS suffers from degrading performance when evaluated on all 15625 models in Nasbench-201 CIFAR10 dataset (Dong and Yang, 2020). Abdelfattah et al. (2021) conduct their own search and find a different configuration that works better on Nasbench201 CIFAR10 dataset. They caution that such proxies clearly don't work on different search spaces even when the dataset and task are the same and also the importance of measuring actual wall clock run times as reduced flops often don't translate into actual time savings due to different ways of accessing memory.

Cao et al. (2021) propose a lightweight architecture evaluation method based on the viewpoint that neural networks are rich feature extractors which are utilized by the last linear classifier layer. They propose training an architecture for a few epochs and recording the features per datapoint over a window of 'k' last epochs. A linear classifier is trained on each of these feature histories to produce 'k' linear classifiers. An ensemble of these 'k' classifiers is used to predict the class membership of each datapoint. The average error over all datapoints is used to rank architectures. This method is closest in spirit to `FEAR` but differs in a number of important ways. First of all maintaining feature histories for every datapoint is expensive in terms of memory. `FEAR` avoids this step by simply freezing most of the architecture and continuing training (which is much cheaper since gradients have to be computed only for a small part of the architecture). More critically `FEAR` *does not apriori fix the number of epochs*. Instead it trains the network in the first stage to a prespecified threshold training or validation accuracy and then freezes most of the network in the second stage. We describe in detail in Section 3 how to organically set this threshold in a task-dataset dependent manner.

Trainingless Proxies: Mellor et al. (2021) propose a *trainingless* method for ranking architectures based on the KL divergence between an uncorrelated Gaussian distribution and the correlation matrix of local linear operators (for networks with ReLU activation functions) associated with every input data point. If the correlation between such local linear maps is low then the network should be able to model each data point well. Since this score can be computed with just a small sample of the dataset (typically a single minibatch), this takes negligible compute and time. In very recent work Abdelfattah et al. (2021) thoroughly empirically evaluate this trainingless method which they term as `jacob_cov` along with an entire family of pruning-at-initialization schemes which they convert to trainingless architecture ranking methods by simply summing up the saliency scores at each weight of the architecture. The particular methods they evaluate include `snip` (Lee et al., 2019), `grasp` (Wang et al., 2020a), `synflow` (Tanaka et al., 2020) and `fisher` (Turner et al., 2020) in addition to other natural baselines like `grad_norm` which is the sum of Euclidean norm of the gradients using a single minibatch of training data. On Nasbench-201, on all three datasets (CIFAR10, CIFAR100, ImageNet16-120) they find that `synflow` score performed the best with relatively high rank correlations with full training of architectures. `jacob_cov` was second best. A majority vote amongst `synflow` , `jacob_cov` and `snip` termed as `vote` performs the best. They also note that these trainingless methods don't work satisfactorily when evaluated on other search spaces like Nasbench-101 (Ying et al., 2019).

Note that `FEAR` is *not* a trainingless method and does use more computation than the trainingless proxies outlined above. But we empirically show in Section 4 that `FEAR` outperforms these proxy measures as well as the natural baselines of reduced number of training epochs. Furthermore in the process of experimentation we identify some curious properties of trainingless proxies such as degradation in performance as the network trains more which is counterintuitive and also the curious phenomenon of `synflow-based` ranking (which is a data-agnostic scoring mechanism) in particular not generalizing across datasets. This supports the intuition that architecture performance is not an intrinsic property of just its topology (and training procedure) but crucially also dependent on the task and dataset at hand. This has been empirically validated by the very recent work of Tuggener et al. (2021) who show that architectures which perform well on ImageNet (Deng et al., 2009) do not necessarily perform as well on other datasets. In fact on some datasets their ranks are negatively correlated with ImageNet ranks. This further suggests that a data-agnostic scoring mechanism such as `synflow` may not work well at ranking architectures.

3 Approach

Here we describe `FEAR` in more detail. Figure 1 shows a high level schematic of the approach.

Finding training accuracy threshold: `FEAR` first trains a fast but shallow learner on the dataset and task of choice to learn a training or validation accuracy threshold. For example, for the task of image classification one can use a number of fast shallow learners like random forest, linear classifier, with handcrafted visual features such as Histogram-of-Gradients (HoG) (Dalal and Triggs, 2005) or random features such as random kitchen sink (Rahimi and Recht, 2008)[2]. We emphasize that the role of this threshold is to be both non-trivial yet not too difficult to beat with a neural network architecture. We explain the intuition behind this choice below.

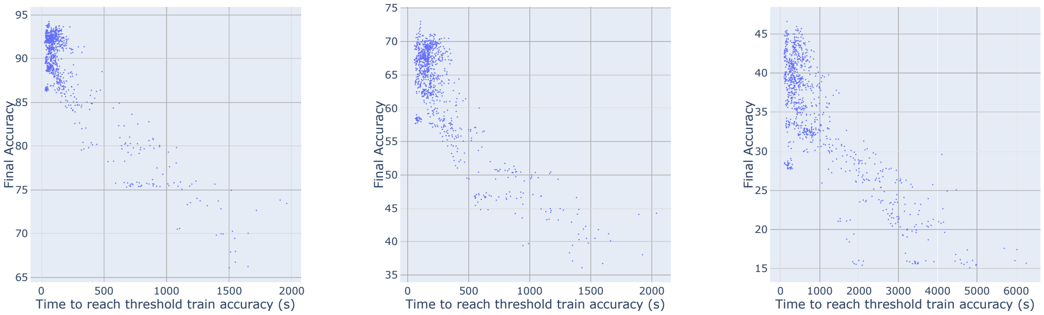
Stage 1: Regular training till threshold accuracy: `FEAR` then trains the candidate architecture till it achieves this threshold accuracy.

Stage 2: Using architecture as feature extractor: `FEAR` then freezes most of the layers of the architecture other than the last few layers and trains it for a few more steps. This *freezing* has the advantage of being several times faster per step than training the entire architecture as gradients don't have to be computed for most of the layers. This stage essentially treats the network as a feature extractor and trains a relatively shallow network utilizing these features for a few more epochs. A pool of candidate architectures are then ranked by their final training or validation accuracies on the dataset under consideration. Intuitively, `FEAR` ranks architectures on their ability to extract useful features from inputs.

A question that may naturally arise is by cutting off training of most of the layers at a relatively early stage of training, are we not hurting the architecture's ability to potentially distinguish itself at feature extraction? Raghu et al. (2017) and Kornblith et al. (2019) dive deep into the training dynamics of neural networks and show that networks train 'bottom-up' where the bottom layers (near the input) train quite fast early-on in training and become stable. As training progresses these bottom layers rarely change their representation and mostly the top layers change to learn the decision-making rules using the bottom layers as rich feature extractors. `FEAR` leverages this insight by first training all layers for a few epochs and then freezing most of the layers but the last few layers.

Role of training till threshold accuracy: We re-emphasize that `FEAR` does not fix the number of epochs apriori. Instead it trains the architecture until a threshold accuracy has been reached. This has a number of advantages. First, it makes architectures *comparable* to each other and makes sure that every architecture gets ample time to learn the best features it can for the task. Secondly, it exposes

[2] See (Mikolajczyk and Schmid, 2005) for a survey of handcrafted visual features.



(a) CIFAR 10

(b) CIFAR 100

(c) ImageNet16-120

Figure 2: Time to reach threshold training accuracy (x-axis) vs. final test accuracy on 1000 uniformly sampled architectures on Natsbench. They show a clear relationship where ultimately worse performing architectures take longer time to reach threshold accuracy than stronger ones.

first-order dynamics of training i.e. weak architectures emperically take longer time to reach the same training or validation error compared to the stronger architectures. See Figure 2 where we plot for three datasets the time taken by 1000 architectures to reach a threshold accuracy (x-axis) against the final test error (y-axis). Invariably, we find that architectures which go on to attain good final test accuracy achieve threshold training accuracy much faster than weaker ones. This alone is not enough for good ranking as some weaker architectures can achieve the threshold accuracy fast as well and hence ranking by training a bit longer (stage 2) to resolve the power of features is necessary. Also note that there are no architectures that train slowly but go on to achieve good final test accuracy after full training ('late-bloomers', these would have been on the upper-right hand part of the plots). This effect can be utilized to early-stop evaluation of candidates that take much longer than the fastest architecture encountered to reach threshold accuracy. This is in fact crucial to obtain large speedup when combining our ranking method with standard discrete neural architecture search techniques.

This phenomenon is also present in other search spaces like the popular DARTS (Liu et al., 2019) search space and the Natsbench Size Search Space (SSS) (Dong et al., 2021). In Appendix A.2, in Figure 4 we used the published training logs of Nasbench301 (Siems et al., 2020) who uniform randomly sampled 21580 architectures from the popular DARTS (Liu et al., 2019) search space as part of the process of building surrogate models. For Natsbench SSS we used FEAR to train 1000 architectures sampled. In both search spaces we find the same phenomenon as above.

Motivation: The motivation for our method comes from the emerging theoretical understanding of gradient learning on neural networks. In a number of works (e.g., (Hu et al., 2020; Chizat and Bach, 2020; Nakkiran et al., 2019; Allen-Zhu and Li, 2020)) it has been observed that at first training happens in the so-called "neural tangent kernel" (NTK) (Jacot et al., 2018) regime, where the network basically uses its initialization as a sort of kernel embedding and performs a kernel regression. Our key hypothesis is that this phase should stop a bit before reaching the fixed threshold accuracy, since this threshold has been obtained with a good kernel method (or something slightly more powerful like a random forest model). To put it differently, in our method, when we stop the training after reaching the fixed threshold accuracy, it should be that the network has already escaped the NTK regime and is currently actively training the features (second phase of learning). Our second (empirical) hypothesis is that the quality of the features learned in the early part of this second phase is predictive of the final quality of the network. We measure the quality of the learned features via the freezing technique, whose extreme case is to only continue training the final layer (i.e., train a linear model on top of the current embedding).

4 EXPERIMENTS

Search space: We uniformly randomly sample 1000 architectures from the 15625 architectures of the topology search space of the Natsbench (Dong et al., 2021) benchmark[3] and hold them constant

[3] Note that Natsbench topology search space is the same as Nasbench-201 (Dong and Yang, 2020).

for all following experiments. The topology search space is similar to that used in DARTS (Liu et al., 2019). All cells have the same topology. Edges are different operator choices (the set of operators are `zeroize`, `skip connection`, 1×1 `convolution`, 3×3 `convolution` and 3×3 `average pooling`). Nodes are tensors. Each cell has 4 nodes. There are three stages in the outer macro skeleton with a downsampling operation after each stage. Each cell is repeated 5 times in each stage. There are respectively 16, 32 and 64 channels in each stage. See Figure 1 in (Dong et al., 2021) for a visualization of the search space. Natsbench topology search space has trained each architecture on CIFAR10, CIFAR100 (Krizhevsky et al., 2014) and ImageNet16-120 (Chrabaszcz et al., 2017) image classification datasets.

4.1 Performance criteria

Spearman's rank correlation vs. evaluation wall-clock time: We report performance of `FEAR` and baselines by first binning architectures into several buckets of increasing size. For example `Top` 10% in Figure 6 shows the average wall-clock time taken by any method (x-axis) vs. Spearman's rank correlation (Spearman, 1904)[4] of the method with the groundtruth rank of architectures (by test accuracy) after full final training procedure over the top 10% of architectures. Similarly the bin of `Top` 20% architectures includes top 20% of candidates and so on. We break-up the performance of methods over such cumulative bins to highlight how methods perform in discriminating amongst high-performing candidates and not just over the entire population. It is crucial for any reduced-computation proxy ranking method to hone-in on good ones and not just the entire population.

Percentage overlap with groundtruth ranking vs. evaluation wall-clock time: While Spearman's rank correlation over cumulative bins of candidates by groundtruth performance is useful for showing the ability of methods to discriminate amongst the top $x\%$ of architectures, it is also important to evaluate what percentage of architectures are common between the top $x\%$ of groundtruth architectures and the architectures that are ranked by a method. When ranking the entire population of candidates, this metric evaluates if the high ranking architectures in groundtruth are also highly ranked by a method (and vice-versa). By definition this is a quantity between $(0, 1)$.

4.2 Baselines

Regular training with reduced epochs: `shortreg` The most natural baseline is to compare rank correlation of `FEAR` against reduced epochs of training. Most NAS methods use a reduced number of training epochs (Li and Talwalkar, 2020) in the inner loop to decide the relative ranks of architectures as a proxy for final performance after undergoing the complete training procedure. We term this reduced training proxy as `shortreg` . We show that `FEAR` consistently outperforms the pareto-frontier of wall-clock time vs. Spearman rank correlation and the ratio of common architectures over cumulative bins of candidate architectures by groundtruth (test accuracy). `shortreg` numbers are obtained by varying **both** batch sizes and number of epochs. Note that one cannot trivially lookup these numbers from the benchmark but has to actually train architectures because different batch sizes and epochs are not part of the benchmark. Learning rate schedules for different number of end-epochs other than a couple of points are not part of the benchmark. For producing the `shortreg` baselines for ImageNet16-120 alone took hundreds of gpu hours of compute![5]

Zero-cost Proxies: As detailed in Section 2, in very recent work (Abdelfattah et al., 2021) a number of nearly negligible cost proxies for ranking architectures are presented. We evaluate these zero-cost proxies for ranking and observe a number of mysterious phenomena where such measures break-down across datasets or as networks are trained. This shows that such measures while exhibiting reasonable prima-facie performance on NAS benchmarks are not generalizing across datasets. We detail our investigation in Section A.6.

[4] Spearman's rank correlation is between $(-1, 1)$ with 1 implying perfect correlation and -1 anti-correlation of the *ranks* of candidate architectures.

[5] Reduced Resolution Proxies: While reduced resolution proxies as proposed in (Zhou et al., 2020) and (Abdelfattah et al., 2021) can significantly speed up architecture evaluation, note that they are orthogonal to our approach as they equally speed-up both `shortreg` and `FEAR` . For use in production pipelines they should be combined with `FEAR` to get even more speedup.

Training procedure hyperparameters and hardware: In all our experiments we use the same hyperparameter settings as Natsbench, specifically cosine learning rate schedule with starting learning rate of 0.1, minimum learning rate of 0.0, SGD optimizer, decay of 0.0005 for both regular weights and batch-norm weights, momentum 0.9 and Nesterov enabled. For `shortreg` baseline we run experiments with varying number of epochs and batch sizes $256, 512, 1024, 2048$ to find a pareto-frontier of wall-clock time vs. Spearman's correlation and common ratio. We especially investigate varying the batch size since that can have a large effect on wall-clock time. When architectures are evaluated by `FEAR` , for the second phase we froze the network up to `cell13` out of 16 total cells for all architectures in the search space. This usually corresponds to $\approx 53\%$ of parameters. All experiments were conducted on Nvidia V100 GPUs with 16 GB of GPU memory.

4.3 Main Results

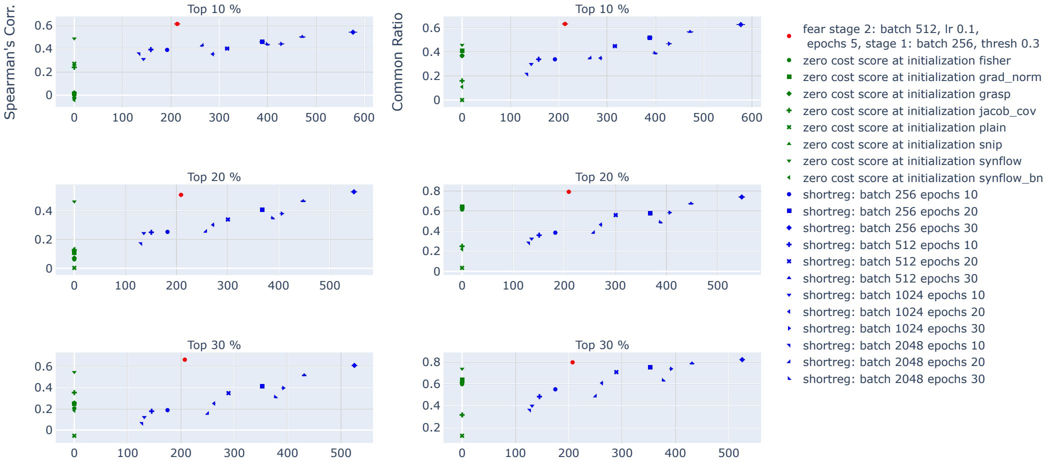


Figure 3: [Left] Average duration per architecture vs. Spearman's correlation and [Right] average duration per architecture vs. common ratio over the top $x\%$ of the 1000 architectures sampled from Natsbench topological search space on CIFAR100. We also show the various zero-cost measures from Abdelfattah et al. (2021) in green. Recall that in Appendix A.6 these measures will be shown to be non-robust to change of task as well as degradation when the training progresses.

Figure 3 shows the ranking efficiency of `FEAR` vs. `shortreg` on the uniform random sample of 1000 architectures on CIFAR100 up to the top 30% of architectures. For elucidation purposes we give out detailed versions in Appendix A.4 (figures 6, 7 and 8) on Natsbench CIFAR10, CIFAR100 and ImageNet16-120 respectively. We sort architectures in descending order of test accuracy, and bin them cumulatively into top 10%, 20%, ... bins. For each bin we report the two performance criteria detailed above of Spearman's rank correlation and the common ratio of architectures in the global ranking. For CIFAR10 and CIFAR100 we find that `FEAR` consistently places above the pareto-frontier of `shortreg` especially at higher ranked architectures. This means that `FEAR` is able to both discriminate better amongst high-performing architectures (better Spearman's correlation) with shorter wall-clock time as well as achieve global ranking which overlaps more with the groundtruth ranking of architectures using test accuracy. Note that as the bin increases to encompass the entire set of 1000 architectures (Top 100%), by construction, `FEAR` will start taking more time as low-performing architectures take more time to reach threshold accuracy (recall Figure 2 and associated explanation) and hence increase the total/average wall-clock time. In practice, this extra time for lower-performing architectures will not be paid since low-performing ones can be simply removed from consideration when they exceed some multiple of the fastest time so far by other architecures to achieve the same threshold accuracy. On ImageNet16-120 the gap between `FEAR` and `shortreg` is not as big but nevertheless it doesn't degrade in performance below `shortreg` and over the high-performance bins is marginally better.

Since figure 3 and corresponding detailed versions 6 7 and 8 contain dense information, for ease of elucidation, in Table 3 for each bin we note FEAR and the nearest point on the pareto-frontier generated by shortreg to rank them amongst each other using Spearman's correlation (denoted as 'spe') and common ratio of architectures in groundtruth ranking against the average time in seconds. Especially on CIFAR10 and CIFAR100 large gaps in performance can be seen at high-performing architecture bins.

Finding the training threshold: We construct a shallow pipeline using Histogram-of-Oriented-Gradients (HoG) (Dalal and Triggs, 2005) as image features and construct a relatively shallow learner by passing the features through two hidden fully connected layers. This simple pipeline achieves 0.6 **training accuracy** on CIFAR10, 0.3 on CIFAR100 and 0.2 accuracy on ImageNet16-120. These numbers were used as the training accuracy threshold for stage 1 of FEAR with respective datasets. We emphasize that similar to Ru et al. (2021) (who use indicators of the speed of training loss) we use the training accuracy both for FEAR as well as for shortreg , which performs much better than validation accuracy in our experiments.

Ablations: FEAR has three main hyperparameters: 1. Training threshold for stage 1 training 2. Number of layers to freeze for stage 2 and 3. number of stage 2 epochs for freeze training. As we detail above, the training threshold for stage 1 is obtained from the performance of a shallow pipeline.[6] For the other two hyperparameters we conduct ablation studies to study how performance evolves. We describe in detail the experiments and results in Appendix A.5 but summarize key findings here.

In Figure 15, 16 and 17 varied the number of cells used for continued training in stage 2. In the legend 'c9 onwards' means cell 9 onwards we finetuned everything (i.e. cell 9 to cell 16 and the last linear classifier layer), 'c13 onwards' means cell 13 onwards and so on. 'nofreeze' means that no layers were frozen in stage 2, 'last layer' means only the last layer was trained. Note that in this search space each architecture has 16 cells. We observe the following trends:

- Performance steadily decreases as too few layers are fine-tuned in stage 2. Finetuning only the last layer or just cell 16 and last layer invariably results in worse ranking performance. (they are always at the bottom of the tall tower of red dots in the figure)
- Not freezing any layers does well in terms of ranking but because gradients must be computed for the entire network it takes a lot more time without significant gains over the closest competitor.
- We find that the sweet spot is always around cell 11 to cell 13 in terms of performance. In terms of parameters this is around 50-60% of total parameters or equivalently around 75% of the layers to be frozen and the rest fine-tuned.

In figures 12, 13 and 14 we changed the number of epochs of training in the second stage where only the last few layers (cell 13 onwards) continue training. We tried values of 5, 10, 15, 20, 25, 30 epochs. We see a clear trend of largely diminishing returns in ranking performance. We can safely say that 5-10 epochs is enough.

Random Search with FEAR: On Natsbench CIFAR10, CIFAR100 and ImageNet16-120 we ran random search (RS) with FEAR (RS-FEAR) 10 times with different random seeds where the search cut-off any architecture which exceeded 4.0 times the fastest time to reach threshold training accuracy encountered so far. Each method got a budget of 500 architectures. As shown in Table 1 RS-FEAR can get similar final accuracy by being $\approx$ 2.4 times faster. See Figure 5 in Appendix A.3 for an intuitive explanation of how FEAR early rejects weaker architectures.

Local Search with FEAR: We also used FEAR in the inner loop of local search White et al. (2020a) with Natsbench CIFAR10, CIFAR100 and ImageNet16-120.

[6] Even though the stage 1 training threshold is obtained from the output of a shallow method we do study how performance evolves as this threshold changes and provide the corresponding plots in Appendix A.5. We do caution that even though these plots are provided, the training threshold should always be determined via the output of a shallow pipeline and not arbitrarily set as it is important to train enough to escape the kernel regime.

	FEAR (4.0 * fastest) top1 (%), duration (s)	shortreg (50 epochs) top1 (%), duration (s)
CIFAR10	$\mathbf{93.97^{\pm 0.08}}$, $\mathbf{90560^{\pm 845}}$	$94.10^{\pm 0.10*}$, $347643^{\pm 2287}$
CIFAR100	$\mathbf{72.04^{\pm 0.29}}$, $\mathbf{142550^{\pm 3106}}$	$72.08^{\pm 0.30}$, $347640^{\pm 1674}$
ImageNet16-120	$\mathbf{45.97^{\pm 0.17}}$, $\mathbf{214824^{\pm 4674}}$	$45.64^{\pm 0.21}$, $528454^{\pm 4901}$

Table 1: When used as the architecture evaluation scheme in random search, FEAR can achieve same or more top-1 accuracy as shortreg while using ≈ 2.4 times less search duration on CIFAR100 and ImageNet16-120. On CIFAR10 it takes ≈ 3.8 times less duration. Each experiment was run 10 times with different random seeds. (*shortreg on CIFAR10 had finished 7 runs at the time of writing).

	FEAR top1 (%), duration (s)	shortreg (20 epochs) top1 (%), duration (s)
CIFAR10 (8.0 * fastest)	$\mathbf{94.08^{\pm 0.05}}$, $\mathbf{36495^{\pm 1076}}$	$94.12^{\pm 0.08*}$, $69757^{\pm 1892}$
CIFAR100 (4.0 * fastest)	$\mathbf{72.9^{\pm 0.15}}$, $\mathbf{50708^{\pm 1628}}$	$72.96^{\pm 0.29}$, $67548^{\pm 2036}$
ImageNet16-120 (2.0 * fastest)	$46.12^{\pm 0.19}$, $68582^{\pm 2792}$	$\mathbf{46.32^{\pm 0.01}}$, $\mathbf{67770^{\pm 2304}}$

Table 2: When used as the architecture evaluation scheme in random search, FEAR can achieve similar top-1 accuracy as shortreg while being ≈ 1.33 times faster on CIFAR100 and ≈ 1.9 faster on CIFAR10. On ImageNet16-120 shortreg was slightly better with slightly less time. Each experiment was run 10 times with different random seeds.

5 Deeper Empirical Analysis of Zero-Cost Measures

The reader will have noticed that Figure 3 shows good ranking performance by various zero-cost measures (Abdelfattah et al., 2021), especially that of synflow . In Appendix A.6 we study these ranking methods in greater detail and summarize out findings here:

1. The performance of synflow suggests that data-agnostic measures are enough and the relative rank of an architecture is data and task independent. By constructing simple synthetic dataset we show that zero-cost measures have almost no rank correlation while FEAR still performs reasonably. This at least provides existence proof of the fact that performance of an architecture is a also a function of the dataset and task. (See Table 4)

2. We also evaluate ranking abilities of various zero-cost measures as training progresses and find that ranking performance drastically degrades as epochs of training increase. This is not surprising since some of the measures like snip , grasp and synflow were originally intended for the task of pruning architectures at initialization. See Figure 18.

Zero-cost measures while being very attractive for NAS, currently warrant further investigation.

6 Conclusion

We have presented a simple but powerful fast architecture ranking scheme (FEAR) which can be used in the inner loop of *any* discrete NAS algorithm to speed-up architecture evaluation. We have shown on standard NAS benchmarks that FEAR is effective and in the inner loop of even random search can drastically speed-up evaluation without loss of accuracy.

In future work we aim to validate FEAR on the state-of-the-art discrete search methods like Bayesian optimization-based techniques (White et al., 2019) or even simpler techniques which have been surprisingly beneficial on NAS benchmarks like (White et al., 2020a). With enough compute we will aim to validate on larger search spaces like DARTS via Nasbench-301 (Siems et al., 2020) and search spaces around Transformer-like architectures (Wang et al., 2020b; Tsai et al., 2020).

7 Ethics Statement

Our technique aims to make neural architecture search more efficient thus leading to reductions in harmful emissions especially where data centers are not using clean energy sources. NAS specifically and AutoML more broadly can be misutilized where consumers of this technology may think that the burden of checking for issues of bias, how the model may get utilized and unintended other harmful effects are no longer a problem or even their responsibility. Careful education and due diligence in deployment are still needed if not even more with NAS!

8 Reproducibility Statement

We have taken reproducibility and confidence in our results seriously and in Appendix A.1 we use the excellent NAS reproducibility checklist by Lindauer and Hutter (2019) to report the various steps we have followed in this regard.

References

Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019.

Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions, 2021.

Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2020.106622. URL https://www.sciencedirect.com/science/article/pii/S0950705120307516.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018. In press, available at http://automl.org/book.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1eYHoC5FX.

Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *CoRR*, abs/2008.09777, 2020. URL https://arxiv.org/abs/2008.09777.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=r1Ue8Hcxg.

Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G. Yen. A survey on evolutionary neural architecture search. *CoRR*, abs/2008.10937, 2020. URL https://arxiv.org/abs/2008.10937.

Colin White, Willie Neiswanger, and Yash Savani. BANANAS: bayesian optimization with neural architectures for neural architecture search. *CoRR*, abs/1910.11858, 2019. URL http://arxiv.org/abs/1910.11858.

Colin White, Sam Nolen, and Yash Savani. Local search is state of the art for NAS benchmarks. *CoRR*, abs/2005.02960, 2020a. URL https://arxiv.org/abs/2005.02960.

Lingxi Xie, Xin Chen, Kaifeng Bi, Longhui Wei, Yuhui Xu, Zhengsu Chen, Lanfei Wang, An Xiao, Jianlong Chang, Xiaopeng Zhang, and Qi Tian. Weight-sharing neural architecture search: A battle to shrink the optimization gap. *CoRR*, abs/2008.01475, 2020. URL https://arxiv.org/abs/2008.01475.

Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable {nas}. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=PKubaeJkw3.

Yao Shu, Wei Wang, and Shaofeng Cai. Understanding architectures learnt by cell-based neural architecture search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BJxH22EKPS.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gDNyrKDS.

Jianlong Chang, xinbang zhang, Yiwen Guo, GAOFENG MENG, SHIMING XIANG, and Chunhong Pan. Data: Differentiable architecture approximation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/74071a673307ca7459bcf75fbd024e09-Paper.pdf.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *International Conference on Learning Representations*, 2017.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *International Conference on Learning Representations*, 2017.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/pham18a.html.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 367–377. PMLR, 22–25 Jul 2020. URL http://proceedings.mlr.press/v115/li20c.html.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR, 2019.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf.

Mohamed S Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas Donald Lane. Zero-cost proxies for lightweight nas. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0cmMMy8J5q.

Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. Neural architecture search without training, 2021.

Difan Deng and Marius Lindauer. Literature on neural architecture search. https://www.automl.org/automl/literatureonneuralarchitecturesearch, 2016.

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

Ariel Keller Rorabaugh, Silvina Caíno-Lores, Michael R. Wyatt II, Travis Johnston, and Michela Taufer. Peng4nn: An accurate performance estimation engine for efficient automated neural network architecture search. *CoRR*, abs/2101.04185, 2021. URL https://arxiv.org/abs/2101.04185.

Colin White, Willie Neiswanger, Sam Nolen, and Yash Savani. A study on encodings for neural architecture search. *Advances in Neural Information Processing Systems*, 33, 2020b.

Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. Econas: Finding proxies for economical neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11396–11404, 2020.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJxyZkBKDr.

Shengcao Cao, Xiaofang Wang, and Kris Kitani. Efficient model performance estimation via feature histories. *CoRR*, abs/2103.04450, 2021. URL https://arxiv.org/abs/2103.04450.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1VZqjAcYX.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=SkgsACVKPH.

Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33, 2020.

Jack Turner, Elliot J. Crowley, Michael O'Boyle, Amos Storkey, and Gavin Gray. Blockswap: Fisher-guided block substitution for network compression on a budget. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SklkDkSFPB.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR, 2019.

Lukas Tuggener, Jürgen Schmidhuber, and Thilo Stadelmann. Is it enough to optimize CNN architectures on imagenet? *CoRR*, abs/2103.09108, 2021. URL https://arxiv.org/abs/2103.09108.

J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: replacing minimization with randomization in learning. In *Nips*, pages 1313–1320. Citeseer, 2008.

K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005. doi: 10.1109/TPAMI.2005.188.

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. doi: 10.1109/TPAMI.2021.3054824.

Wei Hu, Lechao Xiao, Ben Adlam, and Jeffrey Pennington. The surprising simplicity of the early-time learning dynamics of neural networks. *CoRR*, abs/2006.14599, 2020. URL https://arxiv.org/abs/2006.14599.

Lenaic Chizat and Francis Bach. Implicit bias of gradient descent for wide two-layer neural networks trained with the logistic loss, 2020.

Preetum Nakkiran, Gal Kaplun, Dimitris Kalimeris, Tristan Yang, Benjamin L. Edelman, Fred Zhang, and Boaz Barak. SGD on neural networks learns functions of increasing complexity. *CoRR*, abs/1905.11604, 2019. URL http://arxiv.org/abs/1905.11604.

Zeyuan Allen-Zhu and Yuanzhi Li. Backward feature correction: How deep learning performs deep learning. *CoRR*, abs/2001.04413, 2020. URL https://arxiv.org/abs/2001.04413.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/5a4be1fa34e62bb8a6ec6b91d2462f5a-Paper.pdf.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55:5, 2014.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the CIFAR datasets. *CoRR*, abs/1707.08819, 2017. URL http://arxiv.org/abs/1707.08819.

C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904. ISSN 00029556. URL http://www.jstor.org/stable/1412159.

Binxin Ru, Clare Lyle, Lisa Schut, Miroslav Fil, Mark van der Wilk, and Yarin Gal. Speedy performance estimation for neural architecture search, 2021.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. HAT: Hardware-aware transformers for efficient natural language processing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online, July 2020b. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.686. URL https://www.aclweb.org/anthology/2020.acl-main.686.

Henry Tsai, Jayden Ooi, Chun-Sung Ferng, Hyung Won Chung, and Jason Riesa. Finding fast transformers: One-shot neural architecture search by component composition. *CoRR*, abs/2008.06808, 2020. URL https://arxiv.org/abs/2008.06808.

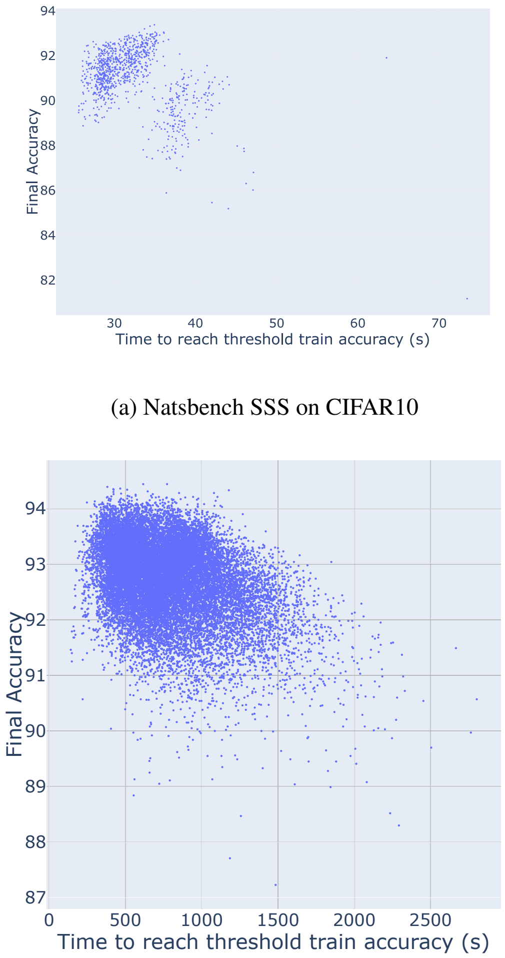
Marius Lindauer and Frank Hutter. Best practices for scientific research on neural architecture search. *CoRR*, abs/1909.02453, 2019. URL http://arxiv.org/abs/1909.02453.

A APPENDIX

A.1 REPRODUCIBILITY AND BEST PRACTICES CHECKLIST IN NAS

We use the best practices by (Lindauer and Hutter, 2019) to foster reproducibility and do better empirical NAS research.

1. Best practices for releasing code.
   (a) Code for the training pipeline used to evaluate the final architectures. - Yes.
   (b) Code for the search space. - Yes.
   (c) The hyperparameters used for the final evaluation pipeline as well as random seeds. - Yes.
   (d) Code for your NAS method. - Yes.
   (e) Hyperparameters for your NAS method, as well as random seeds. - Yes.
2. Best practices for comparing NAS methods.
   (a) For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code? - Yes.
   (b) Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)? - Yes. Specifically we run all the baselines ourselves on the same hardware.
   (c) Did you run ablation studies? - Yes.
   (d) Did you use the same evaluation protocol for the methods being compared? - Yes.
   (e) Did you compare performance over time? - Yes.
   (f) Did you compare to random search? - Yes.
   (g) Did you perform multiple runs of your experiments and report seeds? - Yes.
   (h) Did you use tabular or surrogate benchmarks for in-depth evaluations? - Yes.
3. Best practices for reporting important details. - Yes
   (a) Did you report how you tuned hyperparameters, and what time and resources this required? - Yes.
   (b) Did you report the time for the entire end-to-end NAS method (rather than, e.g., only for the search phase)? - Yes.
   (c) Did you report all the details of your experimental setup? - Yes.



(a) Natsbench SSS on CIFAR10

(b) Nasbench 301

Figure 4: [Left] Time to reach threshold training accuracy (x-axis) vs. final test accuracy on 1000 uniformly sampled architectures on Natsbench SSS on CIFAR10. [Right] 21580 architectures on the DARTS search space from Nasbench301 on CIFAR10.

A.2 Additional Search Spaces

In Figure 4a we sampled 1000 architectures from the Natsbench Size Search Space (SSS) (Dong et al., 2021) on CIFAR10 and plot the time to reach training accuracy of 60% and final test accuracy each architectures goes on to attain at end of training. Similarly in Figure 4b we use the published logs of 21580 architectures published in Nasbench 301 (Siems et al., 2020) on CIFAR10 to plot the time taken to reach 60% threshold training accuracy vs. final test accuracy. In both cases we find the same trends as described in Section 3 and Figure 2 and corresponding textual description.

A.3 Detailed Random Search Experiments

Algorithm 1 shows the simple modifications made to vanilla random search to enable architecture evaluation with `FEAR` and early-reject weaker architectures which inevitably take much longer to reach threshold accuracy than stronger candidates. See Figure 5 and associated caption for an intuitive illustration of how one can early reject most architectures from the search space.

Similar to random search we modify local search (White et al., 2020a) to use `FEAR` for architecture evaluation and observe speedups in CIFAR10 and CIFAR100 as detailed in Section 4.

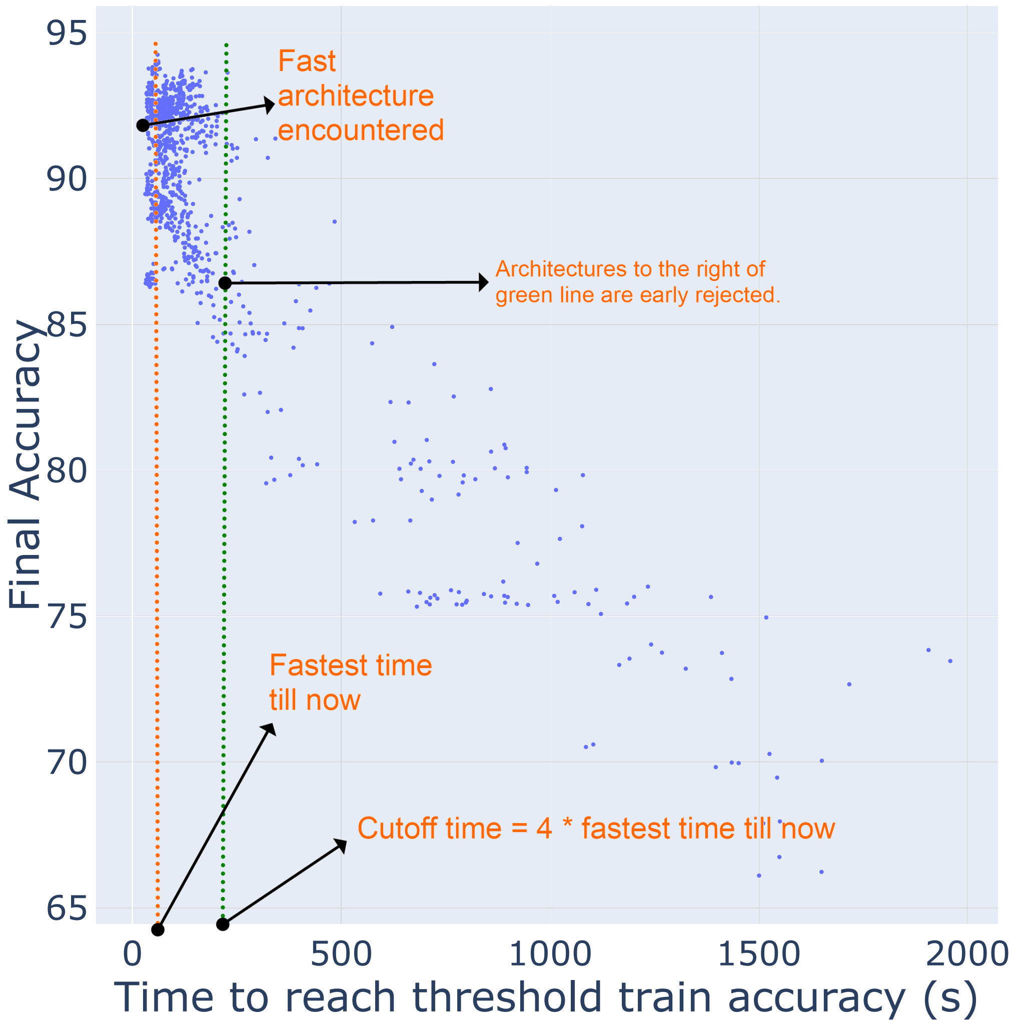


Figure 5: In this illustration on 1000 architectures from Natsbench topology search space on CIFAR10 we show how one can use FEAR in the inner loop of any discrete search algorithm to early-reject many architectures thus saving time without losing final accuracy. For simplicity we consider random search here. A simple modification to random search is made where the fastest time for any architecture to reach threshold accuracy is tracked. If any subsequent architecture improves on this time then it is updated. If any subsequent architecture during evaluation takes more than `x` × `fastest_till_now` (where `x` is usually 4.0 in our experiments as that leads to retaining most of the top ones) then evaluation is terminated and control moves on to the next randomly sampled architecture. Such architectures which are early rejected from evaluation are represented by the region to the right of the green vertical dotted line in the figure. As more and more architectures are evaluated the estimate of the `fastest_till_now` improves and so does early termination of weaker architectures. And this benefit of early stopping weaker architectures is evident in our experiments where we see that one can reduce the search time by ≈ 2.4 times on CIFAR100 and ImageNet16-120 without losing accuracy as detailed in Section 4.

Algorithm 1 Random Search with FEAR

```
Input: Architecture search space $\mathcal{A}$, ratio of fastest time to reach threshold $r$, maximum budget of architectures: $B$, threshold accuracy to train to: $t$
Output: Best architecture $a \in \mathcal{A}$
$\text{fastest\_till\_now} = \infty$
$\text{best\_train\_till\_now} = -\infty$
$\text{best\_arch} = \text{None}$
for $i = 1$ to $B$ do
    $a = \text{random\_sample}(A)$ {Uniformly random sample an architecture.}
    $\text{time\_allowed} = r \times \text{fastest\_till\_now}$ {Time allowed for architecture evaluation by FEAR .}
    $\text{result}, \text{time\_taken} = \text{FEAR}(a, t, \text{fastest\_till\_now})$ {FEAR evaluate architecture.}
    if not result then
        continue {If FEAR early-rejected then move on to next architecture.}
    else
        if $\text{result} > \text{best\_train\_till\_now}$ then
            $\text{best\_train\_till\_now} = \text{result}$ {If a better architecture was found then update.}
            $\text{best\_arch} = a$
            if $\text{time\_taken} < \text{fastest\_till\_now}$ then
                $\text{fastest\_till\_now} = \text{time\_taken}$ {Update fastest time if faster than current fastest.}
            end if
        end if
    end if
end for
return best_arch
```

A.4 Detailed Ranking Plots on Natsbench Topological Search Space

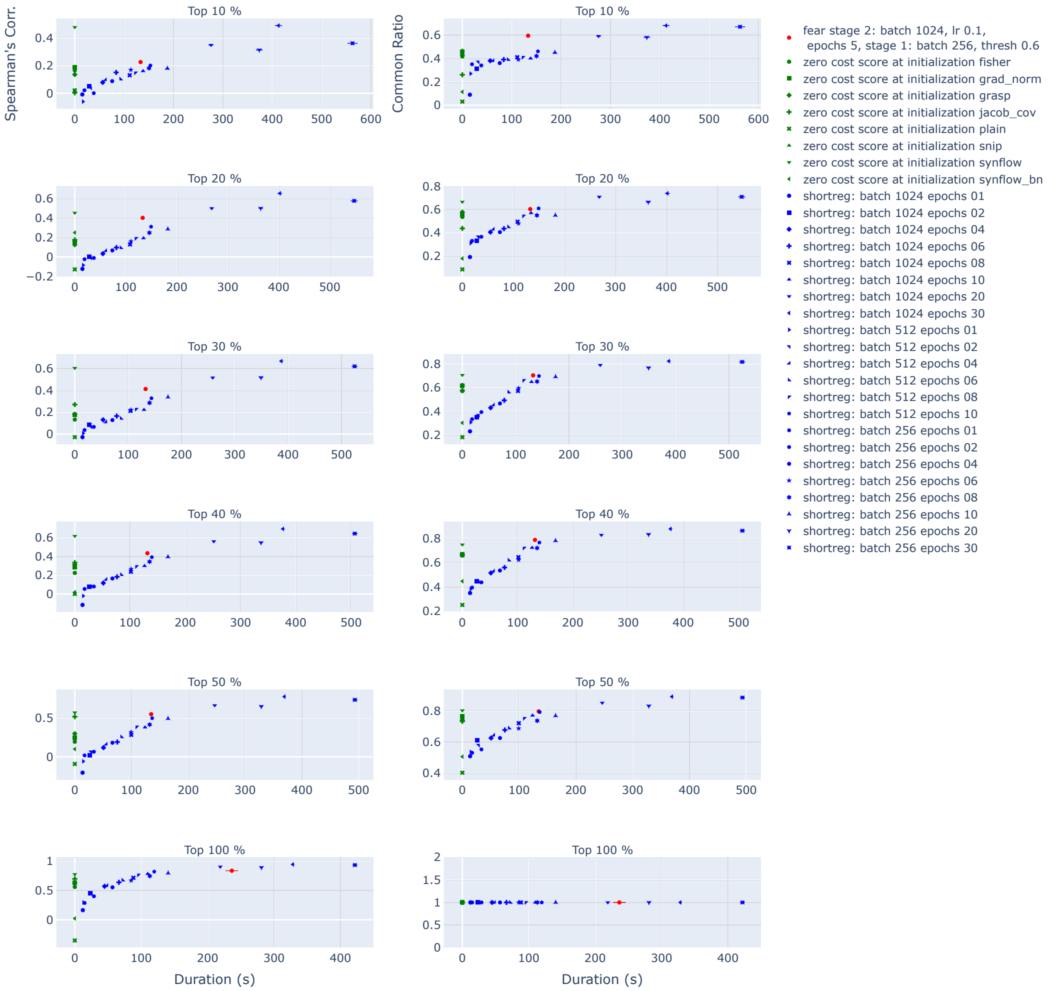


Figure 6: [Left] Average duration vs. Spearman's correlation and [Right] averge duration vs. common ratio over the top $x\%$ of the 1000 architectures sampled from Natsbench topological search space on CIFAR10. We also show the various zero-cost measures from Abdelfattah et al. (2021) in green.

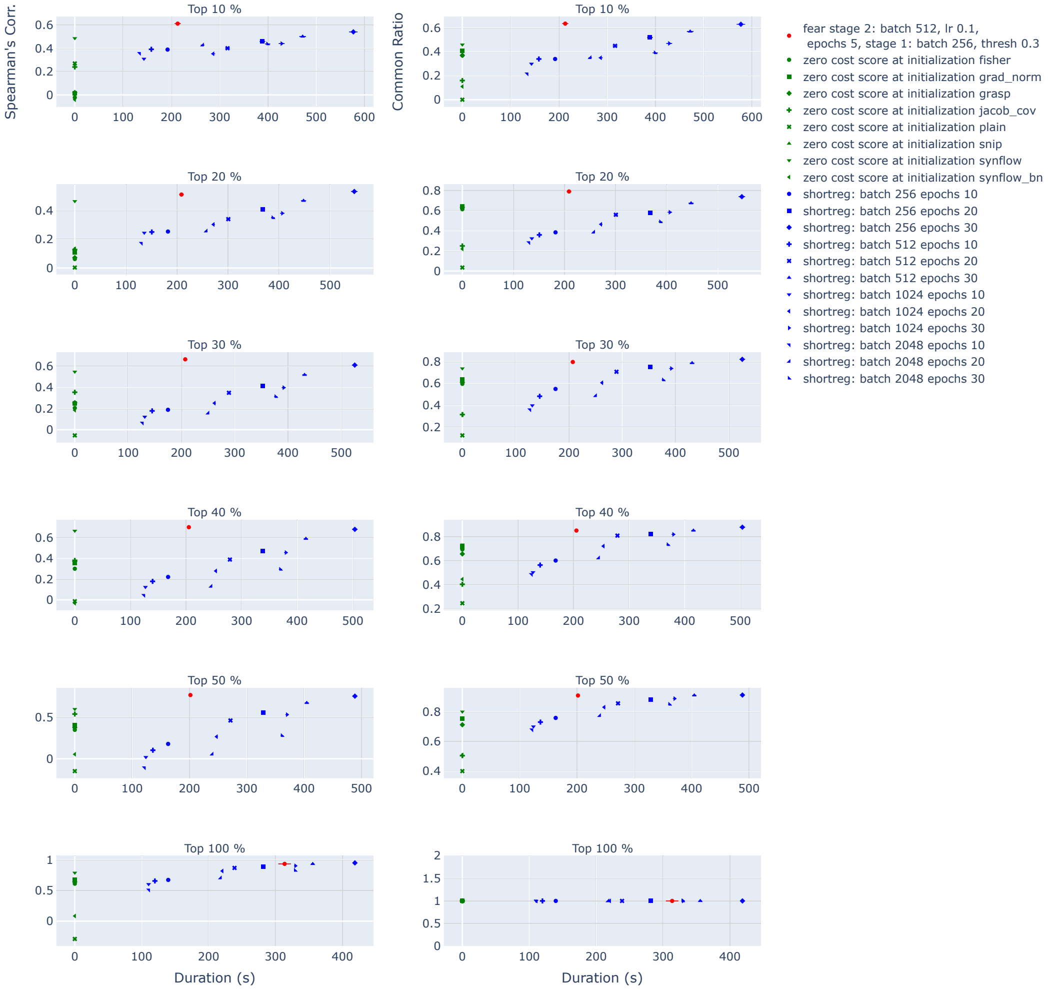


Figure 7: [Left] Average duration vs. Spearman's correlation and [Right] average duration vs. common ratio over the top $x\%$ of the 1000 architectures sampled from Natsbench topological search space on CIFAR100. We also show the various zero-cost measures from Abdelfattah et al. (2021) in green.

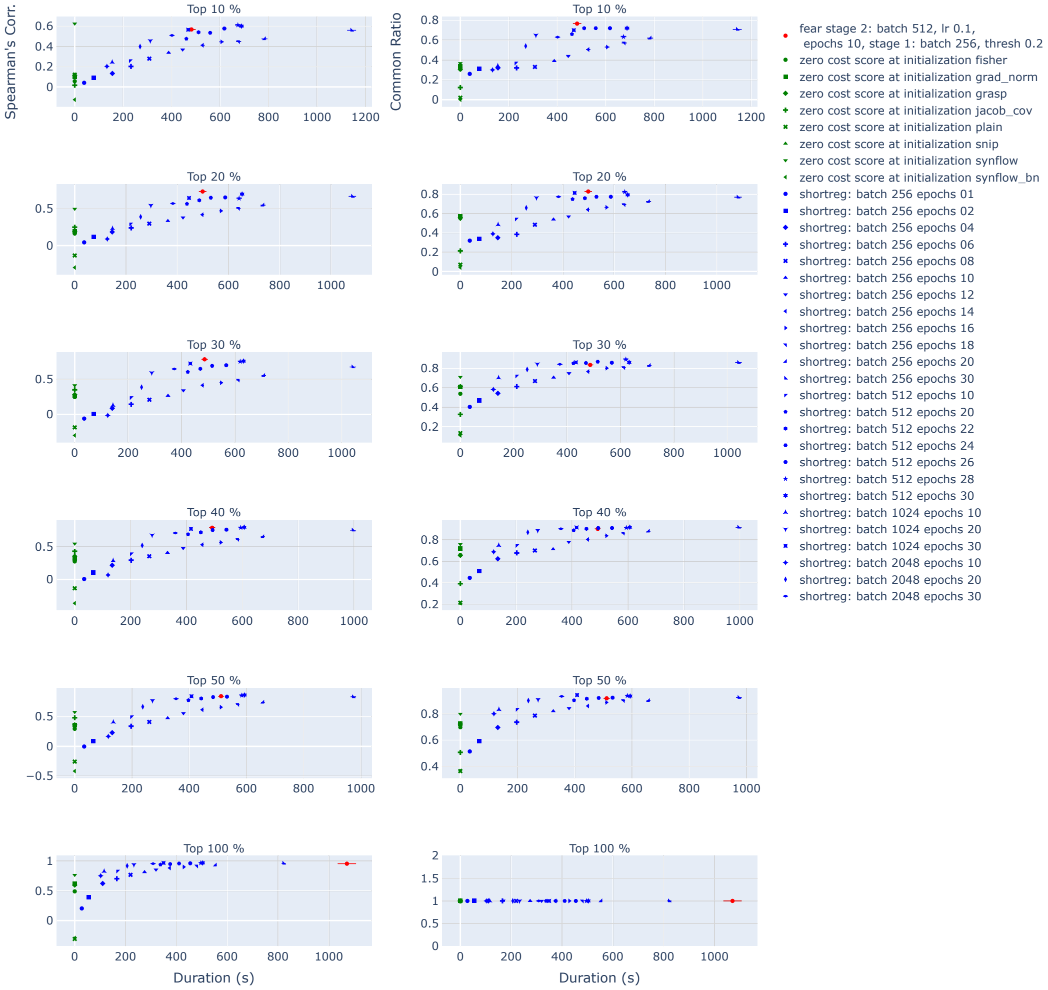


Figure 8: [Left] Average duration vs. Spearman's correlation and [Right] average duration vs. common ratio over the top $x\%$ of the 1000 architectures sampled from Natsbench topological search space on ImageNet16-120. We also show the various zero-cost measures from Abdelfattah et al. (2021) in green.

Top %	FEAR (spe, s)	Nearest Pareto (spe, s)	FEAR (common, s)	Nearest Pareto (common, s)
10	**0.22, 133.22**$^{\pm \mathbf{4.09}}$	0.16, 138.28$^{\pm 2.25}$	**0.59, 133.22**$^{\pm \mathbf{4.09}}$	0.40, 138.28$^{\pm 2.25}$
20	**0.40, 133.04**$^{\pm \mathbf{2.76}}$	0.19, 134.68$^{\pm 1.50}$	**0.60, 133.04**$^{\pm \mathbf{2.76}}$	0.57, 134.68$^{\pm 1.50}$
30	**0.41, 132.76**$^{\pm \mathbf{2.24}}$	0.28, 140.13$^{\pm 1.46}$	**0.70, 132.76**$^{\pm \mathbf{2.24}}$	0.70, 132.77$^{\pm 2.24}$
40	**0.43, 131.41**$^{\pm \mathbf{2.00}}$	0.34, 135.31$^{\pm 1.32}$	**0.78, 131.41**$^{\pm \mathbf{2.00}}$	0.72, 135.31$^{\pm 1.32}$
50	**0.55, 134.48**$^{\pm \mathbf{1.97}}$	0.50, 136.26$^{\pm 1.14}$	**0.79, 134.48**$^{\pm \mathbf{1.97}}$	0.79, 136.26$^{\pm 1.14}$
100	0.83, 236.36$^{\pm 9.39}$	**0.90, 218.97**$^{\pm \mathbf{1.44}}$	1.0, 236.36$^{\pm 9.39}$	**1.00, 218.97**$^{\pm \mathbf{1.44}}$

(a) Natsbench CIFAR 10

Top %	FEAR (spe, s)	Nearest Pareto (spe, s)	FEAR (common, s)	Nearest Pareto (common, s)
10	**0.61, 213.07**$^{\pm \mathbf{6.33}}$	0.42, 264.68$^{\pm 3.52}$	**0.63, 213.04**$^{\pm \mathbf{6.33}}$	0.35, 264.68$^{\pm 3.52}$
20	**0.51, 208.84**$^{\pm \mathbf{4.36}}$	0.25, 257.12$^{\pm 2.43}$	**0.79, 208.84**$^{\pm \mathbf{4.36}}$	0.39, 257.12$^{\pm 2.43}$
30	**0.66, 207.19**$^{\pm \mathbf{3.63}}$	0.15, 249.86$^{\pm 2.03}$	**0.79, 207.19**$^{\pm \mathbf{3.63}}$	0.49, 249.86$^{\pm 2.03}$
40	**0.70, 205.13**$^{\pm \mathbf{3.16}}$	0.27, 253.67$^{\pm 2.18}$	**0.85, 205.13**$^{\pm \mathbf{3.17}}$	0.62, 244.96$^{\pm 1.85}$
50	**0.76, 201.65**$^{\pm \mathbf{2.78}}$	0.26, 247.37$^{\pm 1.97}$	**0.90, 201.65**$^{\pm \mathbf{2.78}}$	0.77, 239.41$^{\pm 1.66}$
100	**0.93, 313.52**$^{\pm \mathbf{9.31}}$	0.90, 329.79$^{\pm 2.19}$	1.0, 313.52$^{\pm 9.31}$	**1.00, 281.30**$^{\pm \mathbf{2.42}}$

(b) Natsbench CIFAR 100

Top %	FEAR (spe, s)	Nearest Pareto (spe, s)	FEAR (common, s)	Nearest Pareto (common, s)
10	0.56, 481.75$^{\pm 17.14}$	**0.56, 468.17**$^{\pm \mathbf{6.07}}$	**0.76, 481.75**$^{\pm \mathbf{17.14}}$	0.72, 510.86$^{\pm 6.77}$
20	**0.73, 498.91**$^{\pm \mathbf{14.52}}$	0.64, 530.86$^{\pm 5.98}$	0.82, 498.91$^{\pm 14.52}$	**0.81, 445.89**$^{\pm \mathbf{5.05}}$
30	**0.78, 486.40**$^{\pm \mathbf{11.56}}$	0.69, 514.65$^{\pm 4.99}$	0.83, 486.40$^{\pm 11.56}$	**0.85, 470.47**$^{\pm \mathbf{4.58}}$
40	**0.79, 491.91**$^{\pm \mathbf{10.74}}$	0.75, 493.66$^{\pm 4.61}$	0.90, 491.91$^{\pm 10.74}$	0.91, 493.66$^{\pm 4.61}$
50	0.84, 571.37$^{\pm 10.73}$	**0.83, 532.04**$^{\pm \mathbf{4.70}}$	0.91, 511.37$^{\pm 10.73}$	0.88, 510.74$^{\pm 5.04}$
100	0.95, 1070.88$^{\pm 35.88}$	**0.96, 822.05**$^{\pm \mathbf{8.14}}$	1.0, 1070.88$^{\pm 35.88}$	**1.00, 822.05**$^{\pm \mathbf{8.14}}$

(c) Natsbench ImageNet16-120

Table 3: (Left) Spearman's correlation ('spe') comparison between FEAR and the nearest point on the pareto-frontier etched out by shortreg variants with respect to average wall-clock time (s). (Right) Ratio of overlap in bins between rankings of FEAR and nearest pareto-frontier point of shortreg . The nearest pareto-frontier points are found by inspecting figures 6, 7 and 8

A.5 Ablation Study of FEAR Hyperparameters

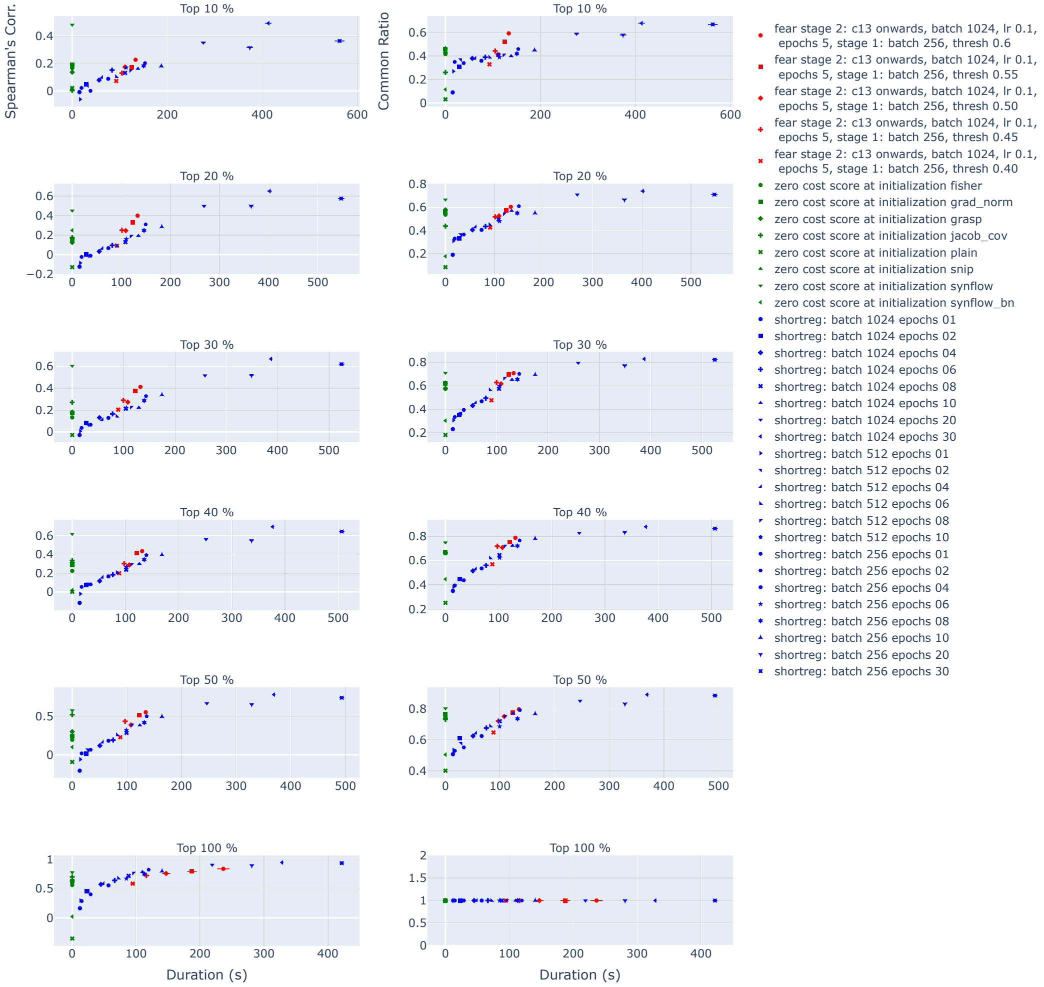


Figure 9: Natsbench CIFAR10 1000 architectures. The stage 1 training threshold was varied.

In Figures 9, 10 and 11 we varied the stage 1 training accuracy threshold to study how performance varies. The main motivation of FEAR is to remove the guesswork on number of epochs to evaluate a candidate architecture when encountering any search space and dataset pair. One accomplishes this via the training/validation accuracy obtained by a reasonable shallow method. The advantage being that one can be reasonably sure that good ranking performance will be obtained efficiently (with the added advantage of early rejection of weak architectures during search.) So, it is a bit artificial to get FEAR plots across time as by design we are giving up direct control over number of epochs in exchange for automatically decided per-architecture budget decided by the procedure itself. Nevertheless, we show these plots for the sake of getting a complete picture. A strong trend that unsurprisingly emerges is that as the stage 1 threshold is lowered the ranking ability clearly degrades. Hence it is important to get a stage 1 threshold such that the network's feature extraction abilities have stabilized enough to reliably rank them. As such these thresholds should be obtained from a reasonable shallow method.

In figures 12, 13 and 14 we changed the number of epochs of training in the second stage where only the last few layers (cell 13 onwards) continue training. We tried values of 5, 10, 15, 20, 25, 30 epochs. We see a clear trend of largely diminishing returns in ranking performance. We can safely say that 5-10 epochs is enough.

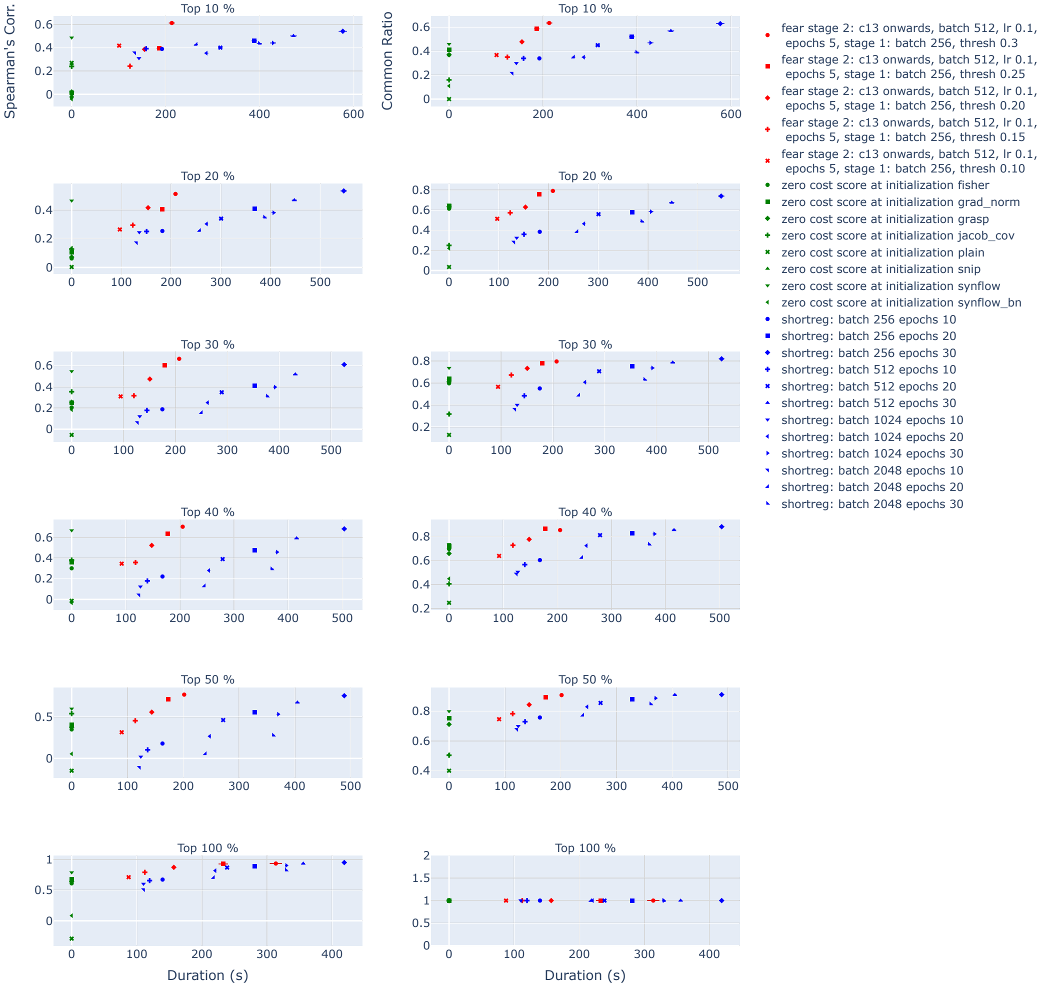


Figure 10: Natsbench CIFAR100 1000 architectures. The stage 1 training threshold was varied.

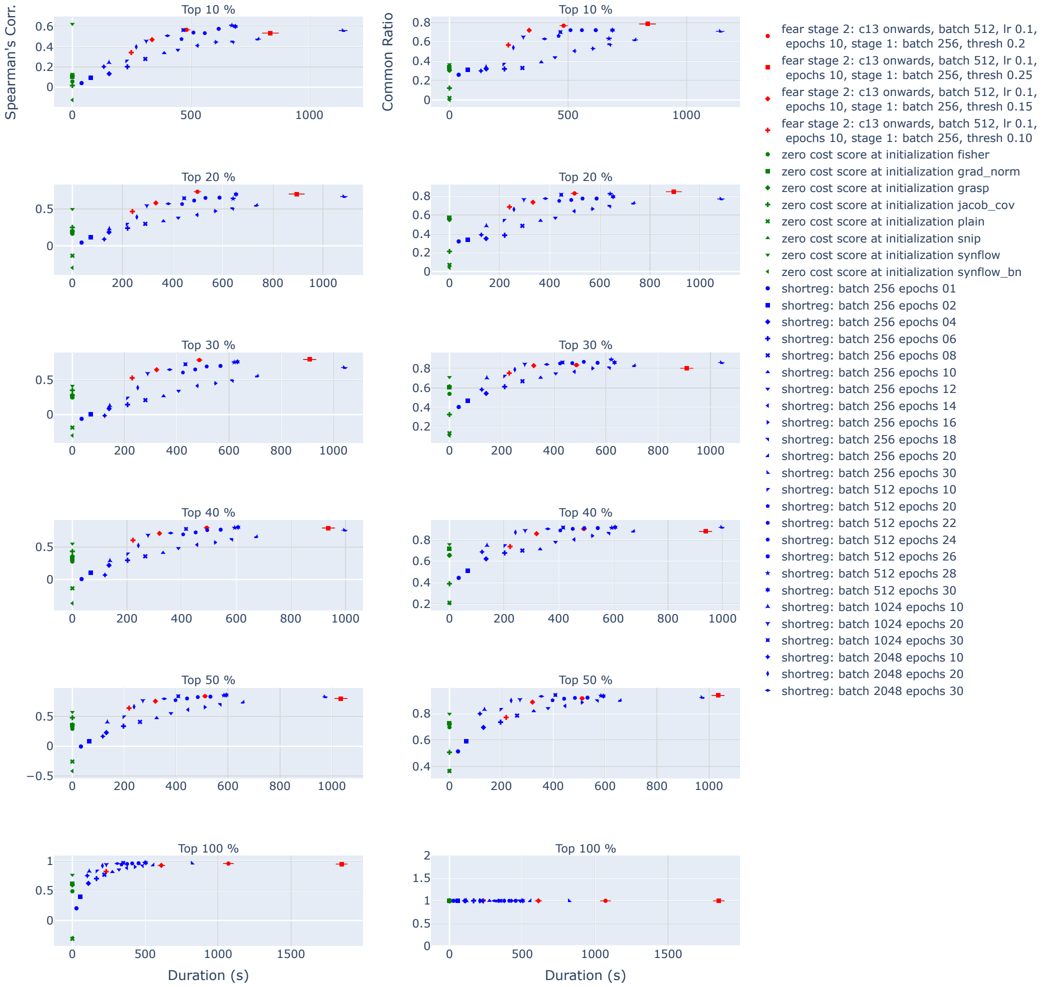


Figure 11: Natsbench ImageNet16-120 1000 architectures. The stage 1 training threshold was varied.

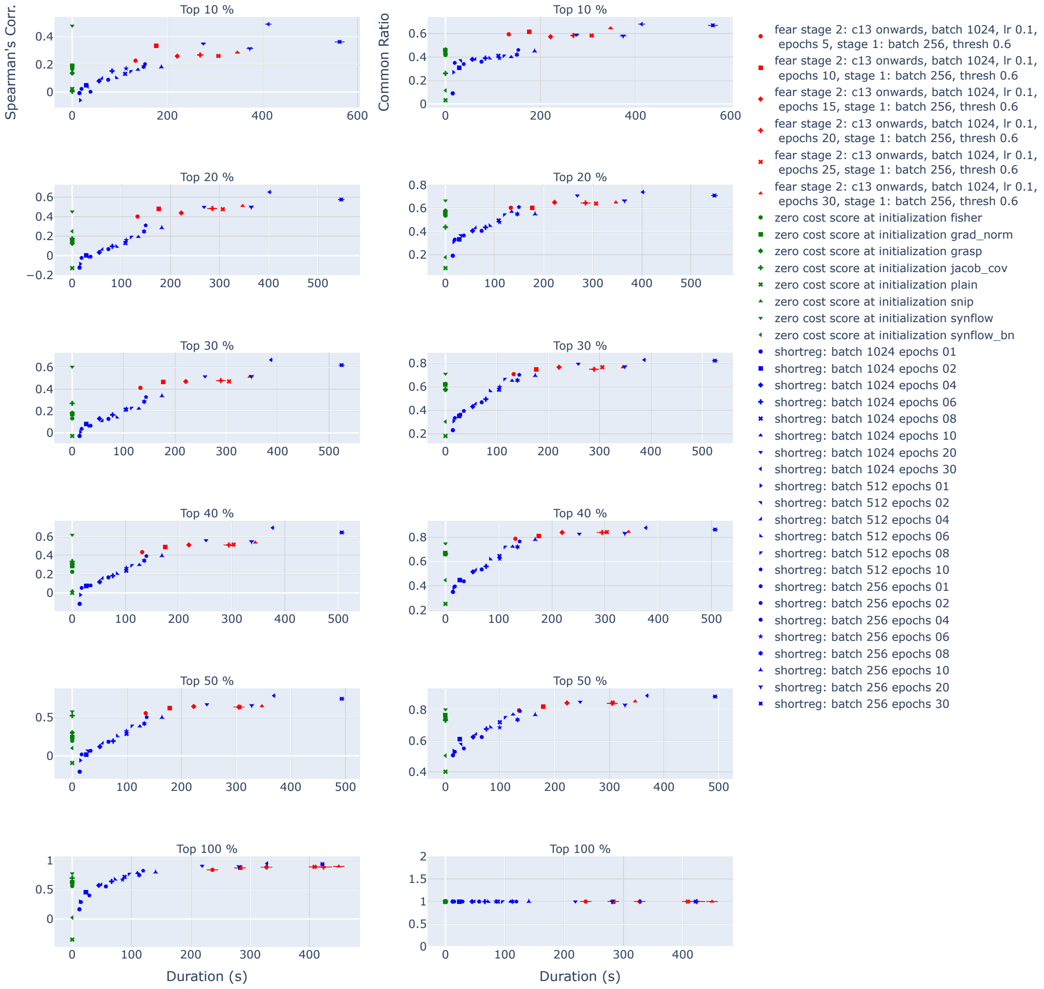


Figure 12: Natsbench CIFAR10 1000 architectures. The number of epochs of stage 2 is varied.

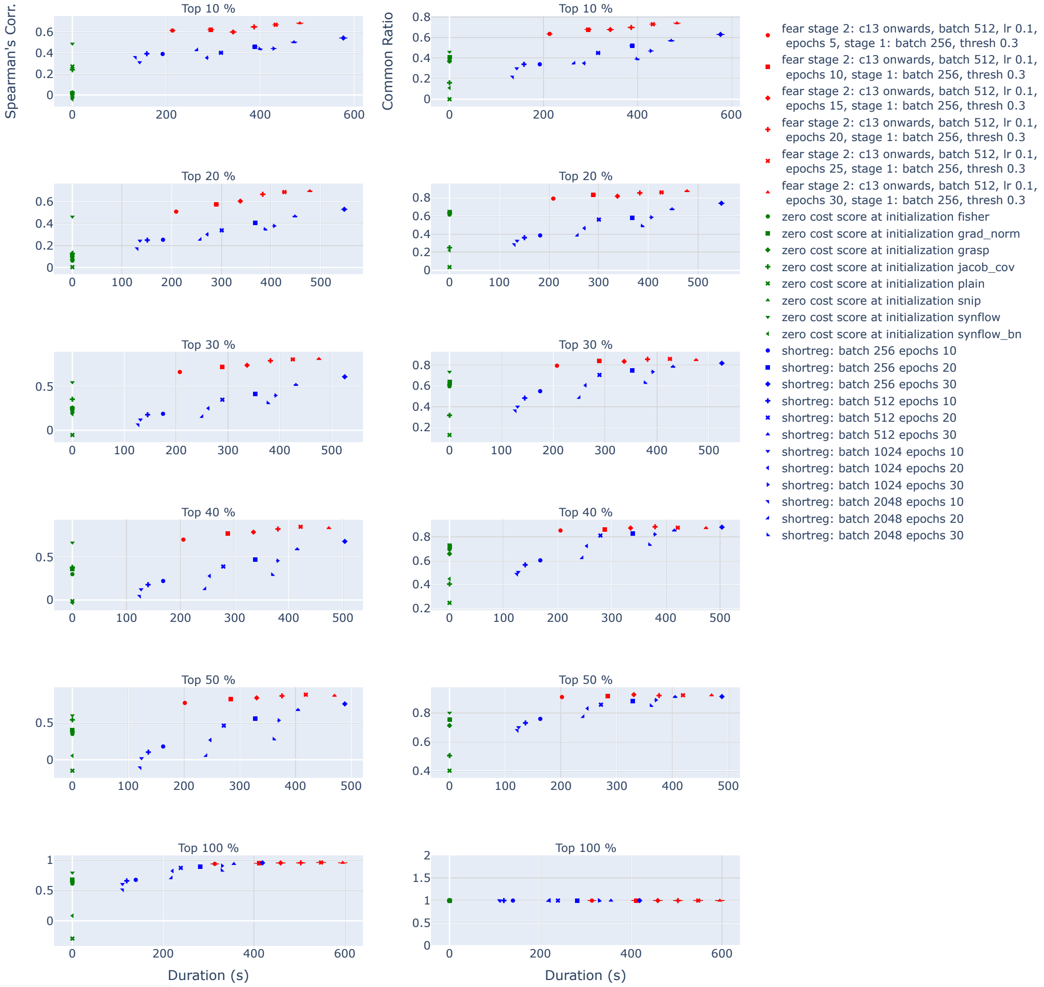


Figure 13: Natsbench CIFAR100 1000 architectures. Number of epochs of stage 2 training was varied.

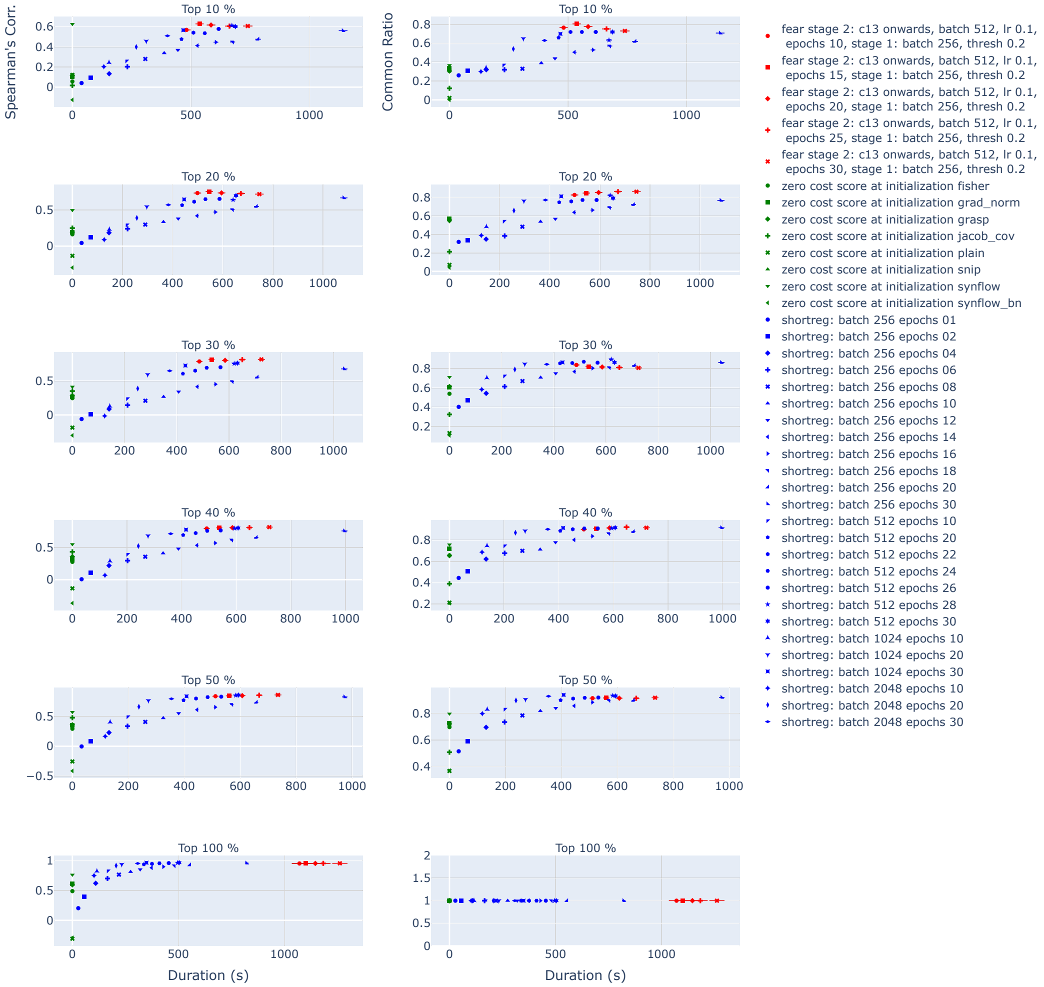


Figure 14: Natsbench ImageNet16-120 1000 architectures. Number of epochs of stage 2 training was varied.

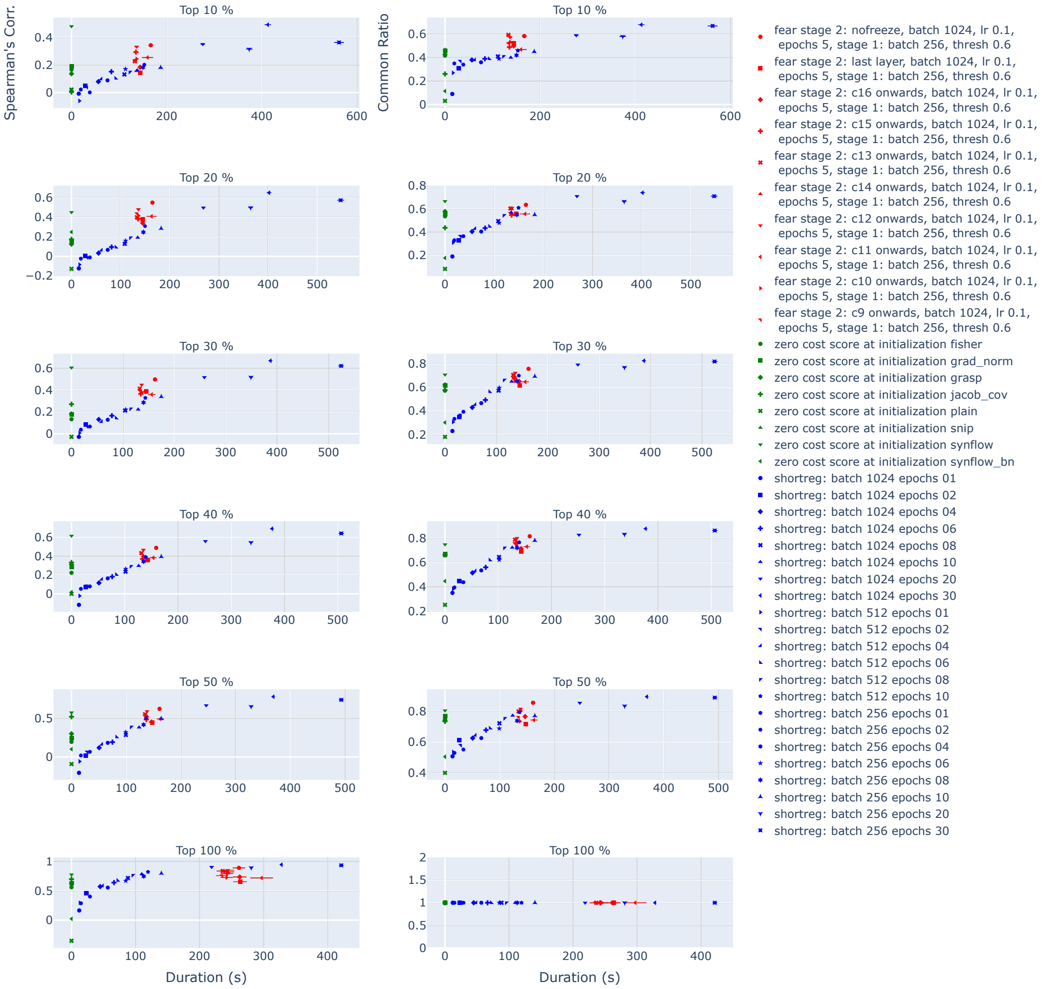


Figure 15: Natsbench CIFAR10 1000 architectures. The number of layers receiving gradients is varied.

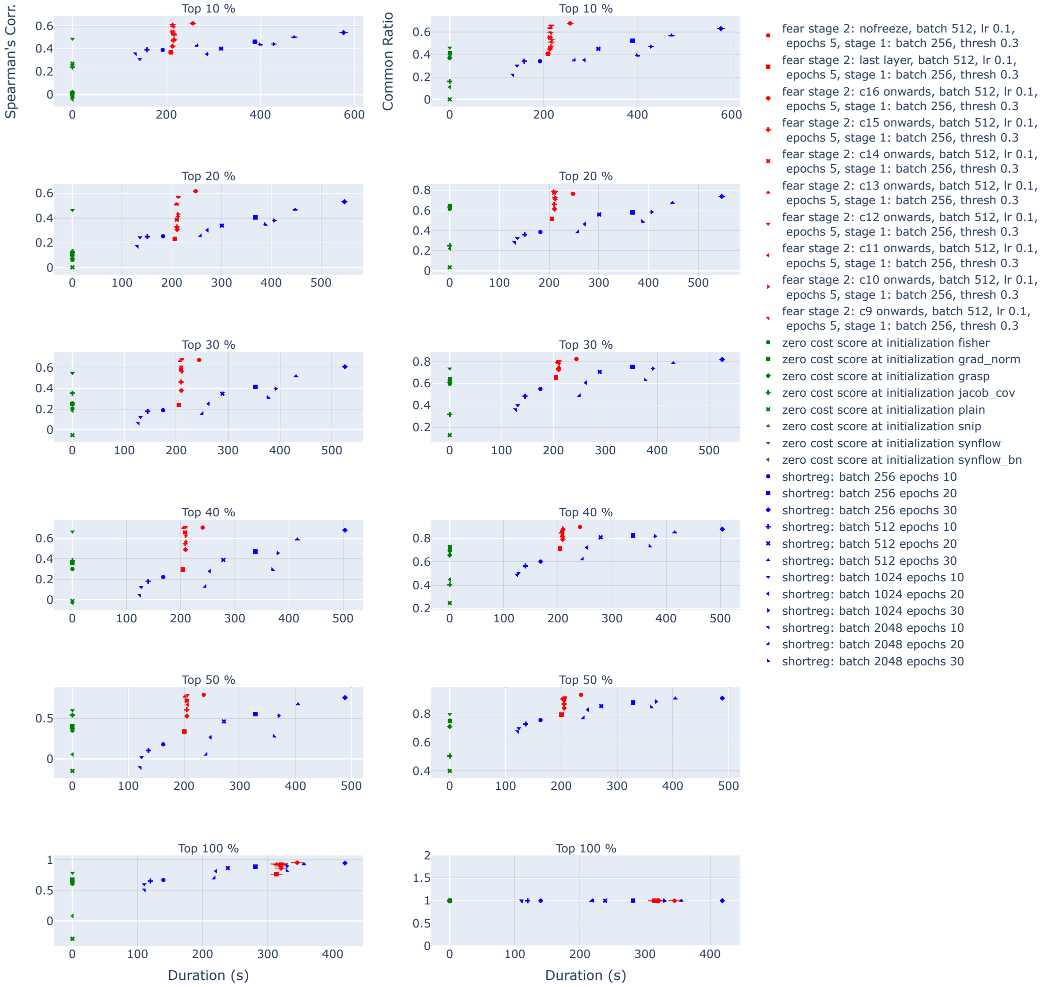


Figure 16: Natsbench CIFAR100 1000 architectures. Number of layers of stage 2 training which received gradients was varied.

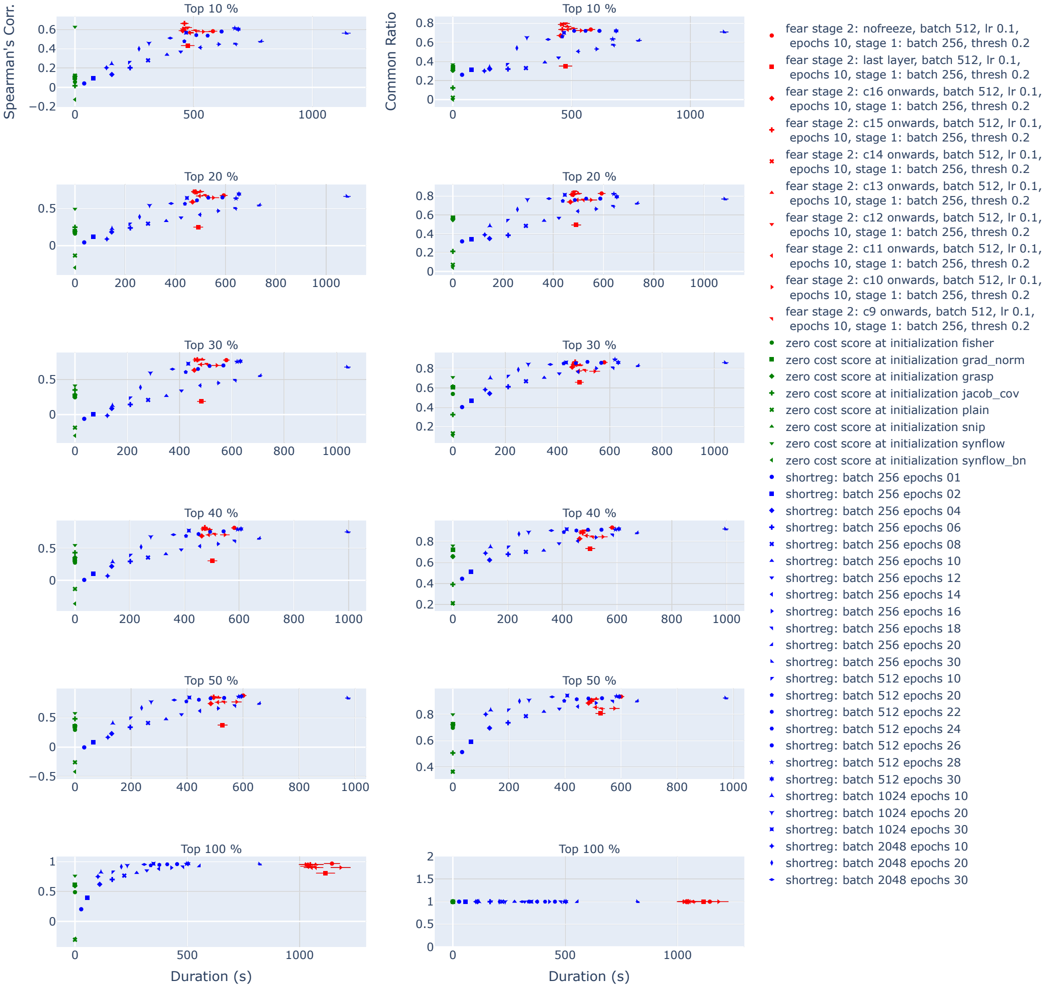


Figure 17: Natsbench ImageNet16-120 1000 architectures. Number of layers of stage 2 training which received gradients was varied.

In Figure 15, 16 and 17 varied the number of cells used for continued training in stage 2. In the legend 'c9 onwards' means cell 9 onwards we finetuned everything (i.e. cell 9 to cell 16 and the last linear classifier layer), 'c13 onwards' means cell 13 onwards and so on. 'nofreeze' means that no layers were frozen in stage 2, 'last layer' means only the last layer was trained. Note that in this search space each architecture has 16 cells. We observe the following trends:

- Performance steadily decreases as too few layers are fine-tuned in stage 2. Finetuning only the last layer or just cell 16 and last layer invariably results in worse ranking performance. (they are always at the bottom of the tall tower of red dots in the figure)
- Not freezing any layers does well in terms of ranking but because gradients must be computed for the entire network it takes a lot more time without significant gains over the closest competitor.
- We find that the sweet spot is always around cell 11 to cell 13 in terms of performance. In terms of parameters this is around 50-60% of total parameters or equivalently around 75% of the layers to be frozen and the rest fine-tuned.

A.6 Deeper Dive into Zero-Cost Measures

Method	Spearman Corr.
synflow	-0.000437
jacob_cov	-0.13
snip	-0.31
fisher	-0.42
grasp	-0.25
synflow_bn	0.18
FEAR	0.55

Table 4: Zero-Cost measures on Synthetic CIFAR10 achieve low correlation while FEAR still gets reasonable performance. This phenomenon is especially not amenable for data-agnostic measures like synflow which ignore the dataset completely.

As discussed in Section 2, Abdelfattah et al. (2021) propose using pruning-at-initialization methods like synflow , snip , grasp , fisher etc for ranking architectures without any training by summing up the per-weight saliency scores to come up with an overall architecture score. In thorough experiments, synflow emerged as a good ranking measure with a majority voting scheme with synflow , jacob_cov and snip emerging as the best overall. As discussed in Section 2, synflow's good performance is a bit perplexing since it is a data-agnostic measure. It suggests that there are inherently good and bad architectures and the particulars of the dataset should not matter. In order to investigate this we created a synthetic dataset with properties such that it would be a drop-in for CIFAR10. Specifically we created random Gaussian images of dimension [32, 32, 3] and mean 0 and variance 1. Each image was assigned a class label in $(0, 9)$ by passing each image through 10 randomly initialized neural networks and picking the id of the network which assigned the image a maximum score. Each of the networks has a simple architecture of a linear layer with dimension 3072, followed by a ReLu layer, followed by a linear layer which produces a single scalar output. A dataset of 60000 images was generated with 50000 training and the rest held-out as a test set. Each class has 6000 examples. We refer to this dataset as Synthetic CIFAR10 .

The same set of 1000 randomly sampled architectures used in above experiments were evaluated on this dataset using FEAR and the various zero-cost measures. Table 4 shows that the zero-cost measures have almost no correlation with rankings while FEAR still works reasonably. Also note that the rankings via synflow which ignore the dataset are no longer valid on Synthetic CIFAR10 . This means that architectures which performed really well on CIFAR10 don't work as well on Synthetic CIFAR10 . This is at least an existence proof of the fact that performance of an architecture is also a function of the dataset and task and is not an inherent property only of the topology of the network. This is also empirically shown recently by Tuggener et al. (2021).

In Figure 18 we evaluated zero-cost measures after each epoch of training on CIFAR10 for the 1000 randomly sampled architectures from Natsbench topological search space. We find that measures like snip and grad_norm gradually degrade in rank correlation as the network trains. jacob_cov and grasp at initialization have Spearman of 0.69 and 0.63 respectively but after even one epoch of training drastically degrade to -0.02 and -0.41. Note from figures 3, 6, 7, and 8 that ranking architectures via training error even after one or two epochs of training leads to much better ranking correlation.

Note that measures derived from snip , grasp , synflow are intended originally for the task of *pruning* architectures at initialization. So it is perhaps not surprising that the saliency scores when summed-up over individual weights to provide a global architecture score doesn't exhibit good ranking performance as the network is trained.

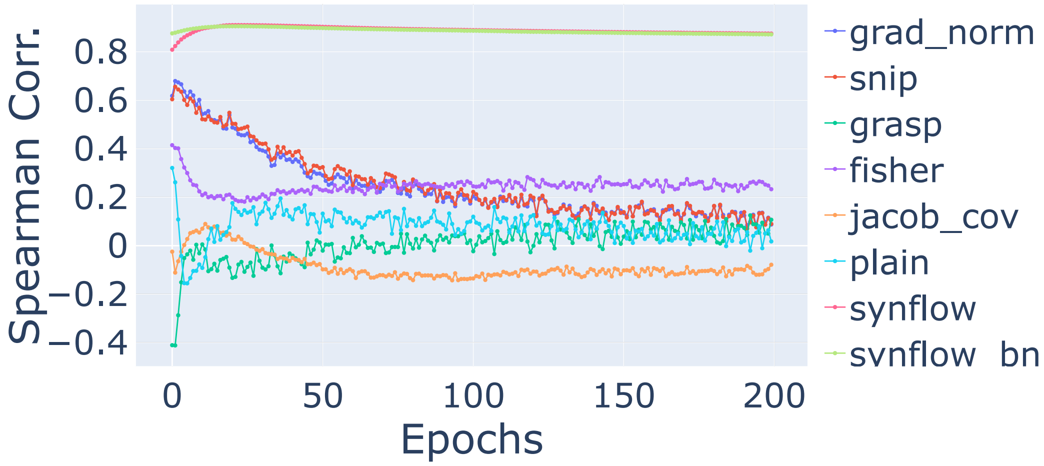


Figure 18: Evaluating the ranking performance of zero-cost measures after each epoch of training. Measures like `grasp` and `jacob_cov` demonstrate large degradation in performance after even a single epoch of training. `snip` and `grad_norm` decay gradually.