HYPERPARAMETER OPTIMIZATION IN NEURAL NETWORKS VIA STRUCTURED SPARSE RECOVERY

Minsu Cho[*]
Tandon School of Engineering
New York University
mc8065@nyu.edu

Mohammadreza Soltani
ECE Department
Duke University
mohammadreza.soltani@duke.edu

Chinmay Hegde
Tandon School of Engineering
New York University
chinmay.h@nyu.edu

ABSTRACT

In this paper, we study two important problems in the automated design of neural networks — Hyperparameter Optimization (HPO), and Neural Architecture Search (NAS) — through the lens of sparse recovery methods.

In the first part of this paper, we establish a novel connection between HPO and structured sparse recovery. In particular, we show that a special encoding of the hyperparameter space enables a natural group-sparse recovery formulation, which when coupled with HyperBand (a multi-armed bandit strategy), leads to improvement over existing hyperparameter optimization methods. Experimental results on image datasets such as CIFAR-10 confirm the benefits of our approach.

In the second part of this paper, we establish a connection between NAS and structured sparse recovery. Building upon "one-shot" approaches in NAS, we propose a novel algorithm that we call CoNAS by merging ideas from one-shot approaches with a techniques for learning low-degree sparse Boolean polynomials. We provide theoretical analysis on the number of validation error measurements. Finally, we validate our approach on several datasets and discover novel architectures hitherto unreported, achieving competitive (or better) results in both performance and search time compared to the existing NAS approaches.

1 Introduction

1.1 Motivation

Despite the success of complex deep learning (DL) models in many data-driven tasks, these models often require a substantial manual effort (involving trial-and-error) for choosing a suitable set of hyperparameters and architectures such as learning rate, regularization coefficients, dropout ratio, filter sizes, and network size. Hyper-parameter optimization (HPO) addresses the problem of searching for suitable hyperparameters that solve a given machine learning problem. Furthermore, neural architecture search (NAS) methods seek to automatically construct a suitable architecture of neural networks with competitive (or better) results over hand-designed architectures with as small computational budget as possible.

In this paper, we propose two novel methods to solve HPO and NAS problems by borrowing ideas from sparse recovery and compressive sensing (CS) [CT06, D+06]. CS has received significant attention in both signal processing and statistics over the last decade, and has influenced the development of numerous advances in nonlinear and combinatorial optimization. Compressive sensing provides an alternative to the conventional sampling paradigm by efficiently recover a sparse signal either exactly or approximately from a small number of measurements.

In the context of HPO/NAS, the main challenge is to evaluate the test/validation performance of a (combinatorially) large number of hyperparameters/architectures candidates. To overcome this, our methods leverage sparse recovery

techniques to find an approximate, yet competitive, solution through fewer number of candidate performance evaluations (measurements).

1.2   Our Contributions

Below, we describe our contributions for solving the HPO and NAS problems. The core to both of our solutions is an adaptation of techniques for learning low-degree sparse Boolean polynomial functions.

1.2.1   Hyperparameter Optimization

We propose an extension to the Harmonica algorithm ([HKY17]), a spectral approach for recovering a sparse Boolean representation of an objective function relevant to the HPO problem. While Harmonica successfully finds important categorical hyperparameters, it does not excel in finding numerical, continuous hyperparameters (such as learning rate). We propose a new group-sparse representation on continuous hyperparameter values that reduces not only the dimension of the search space, but also groups the hyperparameters; this improves both accuracy and stability of HPO. We provide visualizations of the achieved approximate minima by our proposed algorithm in hyperparameter space and demonstrates its success for classification tasks.

1.2.2   Neural Architecture Search

We propose a new NAS algorithm called CoNAS (Compressive sensing-based Neural Architecture Search), which merges ideas from sparse recovery with the so-called "one-shot" architecture search methods [BKZ$^+$18], [LT19] (Please see Section 6.1 for more details). The main idea is to introduce a new *search space* by considering a Boolean function of the possible operations as a loss function of the NAS problem. We utilize the sparse Fourier representation of the Boolean loss function as a new *search strategy* to find the (close)-optimal operations in the network. The numerical experiments show that CoNAS can discover a deep convolutional neural network with reproducible test error of $2.74 \pm 0.12\%$ for classifying of CIFAR-10 dataset. The discovered architecture outperforms the state-of-the-art methods, including DARTs [LSY18], ENAS [PGZ$^+$18], and random search with weight-sharing (RSWS) [LT19] (see Table 3).

1.2.3   Theoretical Analysis

Finally, we analyze the performance of CoNAS by giving a sufficient condition on the number of performance evaluations of sub-architectures; this provides approximate bounds on training time. This, to our knowledge, is one of the first theoretical results in the NAS literature and may be of independent interest.

1.3   Techniques

In a nutshell, our solutions to these problems are based on [HKY17] and [SK12], which have shown how to encode set functions using a sparse polynomial basis representation.

Since all HPO algorithms are computationally expensive, they should be parallelizable and scalable. Moreover, their performance should be at least as good as the random search methods [FKH18]. To achieve these goals, we use the idea of Hyperband from multi-armed bandit problems for parallelization coupled with the Harmonica algorithm by [HKY17] for scalability and high performance. In particular, we first construct a Boolean cost function by binarizing the hyperparameter space, and evaluating it with a small number of (sampled) training examples. This implies the equivalency of estimating the Fourier coefficients and finding the best hyperparameters. Since only a few number of hyperparameters can result in low cost function, estimating the Fourier coefficients boils down to a sparse recovery problem from a small number of measurements. However, our results indicates that the support of non-zero coefficients show some structure. By imposing this structure, we achieve better overall test error for a given computational budget.

Similar to HPO, NAS is also computationally intense. To address this issue, we utilize a one-shot approach [BKZ$^+$18, LT19] in which instead of evaluating several candidate architectures, a single "base" model is pre-trained. Then, a set of sub-networks is selected and evaluated on a validation set, and the best-performing sub-network is chosen to build a final architecture. We model the sub-network selection as a sparse recovery problem by considering a function f that maps sub-architectures to a measure of performance (validation loss). Assuming that f can be written as a linear combination of sparse low-degree polynomial basis functions, we can reconstruct f using a small number of sub-network evaluations; hence, reducing overall computation time. Compared to [LSY18, PGZ$^+$18], our search space allows a search over a more diverse set of candidate architectures.

The rest of this paper is organized as follows. In Section 2, we review some prior work on HPO and NAS problems. Section 3 provides some definitions and mathematical backgrounds. In Section 4 and 5, we respectively introduce our HPO algorithm and the supportive experimental results. In section 6 and 7, we present our NAS algorithm with rigorous theoretical analysis and experimental results, respectively. We conclude this paper in section 8.

2 Related Work

2.1 Prior Work in Hyperparameter Optimization (HPO)

HPO methods based on brute-force techniques such as exhaustive grid search are prohibitive for large hyperparameter spaces due to their exponential time complexity. One remedy for this problem is introduced by Bayesian Optimization (BO) techniques in which a prior distribution over the cost function is defined (typically a Gaussian process), and is updated after each "observation" (i.e., measurement of training loss) at a given set of hyperparameters [BBBK11, HHLB11, SLA12, THHLB13, EFH+13, SSZA14, IAFS17]. Subsequently, an acquisition function samples the posterior to form a new set of hyperparameters, and the process iterates. Despite the popularity of BO techniques, they often provide unstable performance, particularly in high-dimensional hyperparameter spaces. An alternative approach to BO is Random Search (RS), with efficient computational time, strong "anytime" performance with easy parallel implementation [BB12].

Multi-armed bandit (MAB) methods adapt the random search strategy to allocate the different resources for randomly chosen candidates to speed up the convergence. However, random search and the BO approaches spend full resources. Successive Halving (SH) and Hyperband are two examples of MAB methods, which ignore the hyperparameters with poor performance in the early state [JT16, LJD+17, KDVN18]. In contrast with BO techniques (which are hard to parallelize), the integration of BO and Hyperband achieve both advantages of guided selection and parallelization [WXW18, FKH18, BAPB17].

Gradient descent methods [Ben00, MDA15, LBGR15, FLF+16, FDFP17] (or more broadly, meta-learning approaches) have also been applied to solve the HPO problem, but these are only suitable to optimize continuous hyperparameters. Since this is a very vast area of current research, we do not compare our approach with these techniques.

While BO dominates model-based approaches, a recent technique called *Harmonica* utilize a *spectral* approach by applying sparse recovery techniques on a Boolean version of the objective function. This gives Harmonica the benefit of reducing the dimension of the hyperparameter space by quickly finding influential hyperparameters [HKY17].

2.2 Prior Work in Neural Architecture Search

Early NAS algorithms were using reinforced learning (RL) based controllers [ZVSL18], evolutionary algorithms [RAHL19], or sequential model-based optimization (SMBO) [LZN+18]. The performance of these methods is competitive with the manually-designed architectures such as deep ResNets [HZRS16] and DenseNets [HLVDMW17]. However, they require substantial computational resources, e.g., thousands of GPU-days. Other NAS approaches have focused on boosting search speeds by proposing novel search strategies, such as differentiable search technique [CZH19, LSY18, NNR+19, LTQ+18, XZLL18] and random search via sampling sub-networks from a one-shot super-network [BKZ+18, LT19]. In particular, DARTs [LSY18] is based on a bilevel optimization by relaxing the discrete architecture search space to a differentiable one via softmax operations. This relaxation makes it faster by orders of magnitude while achieving competitive performance compared to previous works [ZL17, ZVSL18, RAHL19, LZN+18].

Other recent methods include RL approaches via weight-sharing [PGZ+18], network transformations [CCZ+18, EMH19, JDO+17, JSH18, LSV+17, HLC+19], and random exploration [LJR+18, LT19, SYJ+19, XKGH19]. None of these methods has explored utilizing the sparse recovery techniques for NAS. The closest approach to ours but a different objective is the one proposed by [SK12] which learns a sparse graph from a small number of random cuts. While [SK12] emphasizes on *linear* measurements, CoNAS takes a different perspective by focusing on measurements that map sub-networks to performance, which are fundamentally *nonlinear*.

In [BKZ+18], the authors provided an extensive experimental analysis on one-shot architecture search based on weight-sharing and correlation between the one-shot model (super-graph) and stand-alone model (sub-graph). The authors of [LT19] proposed a simplified training procedures compare to [BKZ+18] without neither super-graph stabilizing techniques such as path dropout schedule on a direct acyclic graph (DAG) nor ghost batch normalization. We defer the details of super-graph training from [LT19] in Section 6.1.

The combination of random search via one-shot models with weight-sharing provides the best competitive baseline results reported in the one-shot NAS literature. Our CoNAS approach improves upon these reported results.

3 Preliminaries

3.1 Fourier Analysis of Boolean Functions

Throughout this paper, we denote the vectors with bold letters. Also, $[n]$ denotes the set $\{1, 2 \ldots, n\}$. A real-valued Boolean function is one that maps n-bit binary vectors (i.e., vertices of a hypercube) to real values: $f : \{-1, 1\}^n \to \mathbb{R}$. Such functions can be represented in a basis comprising real multilinear polynomials called the *Fourier* basis as follows [O'D14].

Definition 3.1. *For $S \subseteq [n]$, define the parity function $\chi_S : \{-1, 1\}^n \to \{-1, 1\}$ such that $\chi_S(\boldsymbol{\alpha}) = \prod_{i \in S} \alpha_i$. Then, the Fourier basis is the set of all 2^n parity functions $\{\chi_S\}$.*

The key fact is that the basis of parity functions forms an K-bounded orthonormal system (BOS) with $K = 1$. That is:

$$\langle \chi_S, \chi_T \rangle = \begin{cases} 1, & \text{if } S = T \\ 0, & \text{if } S \neq T \end{cases} \qquad \text{and} \tag{3.1}$$

$$\sup_{\boldsymbol{\alpha} \in \{-1,1\}^n} |\chi_S(\boldsymbol{\alpha})| \leq 1 \quad \text{for all } S \subseteq [n], \tag{3.2}$$

As it has been shown in [O'D14], any Boolean function f has a unique Fourier representation as $f(\boldsymbol{\alpha}) = \sum_{S \subseteq [n]} \hat{f}(S) \chi_S(\boldsymbol{\alpha})$, with Fourier coefficients $\hat{f}(S) = \mathbb{E}_{\boldsymbol{\alpha} \in \{-1,1\}^n}[f(\boldsymbol{\alpha}) \chi_S(\boldsymbol{\alpha})]$ where expectation is taken with respect to the uniform distribution over the vertices of the hypercube. For many objective function in machine learning, the Fourier spectrum of the function is concentrated on monomials of small degree ($\leq d$) (e.g., decision trees [HKY17]). Leveraging this property simplifies the Fourier expansion by limiting the number of basis functions. Let $\mathcal{P}_d \subseteq 2^{[n]}$ be a fixed collection of Fourier basis such that $\mathcal{P}_d := \{\chi_S \subseteq 2^{[n]} : |S| \leq d\}$. Then, $\mathcal{P}_d \subseteq 2^{[n]}$ induces a function space, consisting of all functions of order d or less, denoted by $\mathcal{H}_{\mathcal{P}_d} := \{f : \text{Supp}[\hat{f}] \subseteq \mathcal{P}_d\}$. For example, $\mathcal{P}_2$ allows us to express the function f with at most $\sum_{l=0}^{d} \binom{n}{l} \equiv \mathcal{O}(n^2)$ Fourier coefficients. Next, we define the restriction of a function f to some index set J.

Definition 3.2. *Let $f : \{-1, 1\}^n \to \mathbb{R}$, $(J, \overline{J})$ be a partition of $[n]$, and $\mathbf{z} \in \{-1, 1\}^{\overline{J}}$. The restriction of f to J using z denoted by $f_{J|\mathbf{z}} : \{-1, 1\}^J \to \mathbb{R}$ is the subfunction of f given by fixed coordinates in $\overline{J}$ to the values of $\mathbf{z}$.*

4 Hyperparameter Optimization

In this section, we present our HPO algorithm. We restrict our attention to discrete domains (we assume that continuous hyperparameters have been appropriately binned). Let $f : \{-1, 1\}^n \mapsto \mathbb{R}$ be the loss function we want to optimize. Assume there exists k different types of hyperparameters. In other words, we allocate n_i bits to the ith hyperparameter category such that $\sum_{i=1}^{k} n_i = n$. The task of HPO involves searching the global minimizer(s) of the following optimization problem as the best hyperparameters:

$$\alpha^* = \underset{\alpha \in \{-1,1\}^n}{\arg\min} \, f(\alpha). \tag{4.1}$$

We propose *Polynomial Group-Sparse Recovery within Hyperband* (PGSR-HB), a new HPO search algorithm which significantly reduces the size of hyperparameter space. We combine Hyperband, the multi-armed bandit method that balances exploration and exploitation from uniformly random sampled hyperparameter configurations, with a group sparse version of Polynomial Sparse Recovery. Algorithm 1 shows the pseudocode of PGSR-HB.

PGSR-HB adopts the decision-theoretic approach of Hyperband, but with the additional tracking the history of all loss values from different resources. Lines 7-14 in Algorithm 1, illustrates the Successive Halving (SH) subroutine of Hyperband, a early performance (e.g. validation loss) in the process of training indicates which hyperparameter configurations are worth investing further resources, and which ones are fit to discard. We defer the pseudocode of SH and Hyperband in the Appendix A.1.

Now, let R denote the (units of computational) resource to be invested in one round to observe the final performance of the model. Let η denote a scaling factor and c be the total number of rounds. Defining $s_{max} = \log_{\eta} R$, the total budget spent from SH is given by $B = (s_{max} + 1)R$. In addition, Line 6 of Algorithm 1 invokes a sub-routine (discussed below) to sample n configurations given as follows:

$$n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil \tag{4.2}$$

4

Algorithm 1 PGSR-HB
___
 1: **Inputs:** Resource R, scaling factor η, total cycle c
 2: **Initialization:** $s_{max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{max} + 1)R$, input history $H_{input} = \emptyset$, output history $H_{output} = \emptyset$
 3: **for** $round = 1 : c$ **do**
 4:    **for** $s \in \{s_{max}, s_{max} - 1, \ldots, 0\}$ **do**
 5:       $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$, $r = R\eta^{-s}$
 6:       $\mathtt{T = PGSR\_Sampling(n)}$
 7:       **for** $i \in \{0, \ldots, s\}$ **do**
 8:          $n_i = \lfloor n\eta^{-i} \rfloor$
 9:          $r_i = r\eta^i$
10:          $L = \{f(t, r_i) : t \in T\}$
11:          $H_{input,r_i} \leftarrow H_{input,r_i} \bigcup T$
12:          $H_{output,r_i} \leftarrow H_{output,r_i} \bigcup L$
13:          $T = \text{top}_k(T, L, \lfloor \frac{n_i}{\eta} \rfloor)$
14:       **end for**
15:    **end for**
16: **end for**
17: **return** Configuration with the smallest loss
___
Sub-algorithm - PGSR Sampling
___
18: **Input:** H_{input}, H_{output}, sparsity s, polynomial degree d, minimum observations T, randomness ratio ρ
19: **if** every $|H_{output,r}| < T$ **then return** random sample from original domain of f.
20: **end if**
21: Pick $H_{input,r}$ and $H_{output,r}$ with largest r: $|H_{output,r}| \geq T$.
22: Group Fourier basis based on hyperparameter structure.
23: Solve

$$\alpha^* = \arg\min_\alpha \frac{1}{2}\|\mathbf{y} - \sum_{l=1}^m \Psi^l \alpha^l\|_2^2 + \lambda \sum_{l=1}^m \sqrt{p_l}\|\alpha^l\|_2$$

24: Let $S_1, \ldots S_s$ be the indices of the largest coefficient of α. Then, $g(\alpha) = \sum_{i \in [s]} \alpha_{S_i} \chi_{S_i}(\alpha)$ and $J = \bigcup_{i=1}^s S_i$
25: With probability ρ, return random sample from original domain of f; else return random sample from reduced domain of f_{J,α^*}.
___

Also, the test loss is computed with $r = R\eta^{-s}$ epochs of training. The function $f(t, r_i)$ in Algorithm 1 (Line 10) returns the intermediate test loss of a hyperparameter configuration t with r_i training epochs. Since the test loss is a metric to measure the performance of the model, the algorithm keeps only the top $\frac{1}{\eta}$ configurations (Line 13) and repeats the process by increasing number of epochs by the factor of η until r reaches to resource R. While SH introduces a new hyperparameter s, SH aggressively explores the hyperparameter space as s close to s_{max}. SH with $s = 0$ is equivalent to random search (aggressive exploitation).[2]

4.1 PGSR Sampling

As PGSR-HB collects the outputs of the function f, PGSR-Sampling estimates Fourier coefficients of the function f using techniques from sparse recovery to reduce the hyperparameter space. We now introduce a simple mathematical expression that efficiently induces additional sparsity in the Fourier representation of f. Given k categories ($i = 1, \ldots, k$), let g and h be two functions that map binary numbers $\mathbf{x}_i$ and $\mathbf{y}_i$ with respectively α_i and β_i digits to the set of integers with cardinality 2^{α_i} and 2^{β_i}. Then we express the ith real-valued hyperparameter, hp_i with corresponding binary digits $\mathbf{x}_i$ and $\mathbf{y}_i$ in a log-linear manner as follows:

$$hp_i = 10^{g(\mathbf{x}_i)} \cdot h(\mathbf{y}_i) \tag{4.3}$$

Our experimental results show how the above nonlinear binning representation induces sparsity on function g. While PSR in Harmonica recovers the Boolean function with Lasso [Tib96], the intuitive extension (arising from the above log-linear representation) is to replace sparse recovery with Group Lasso [YL06]. This is used in Line 23 of Algorithm 1 as we group them based on the g and h based on the hyperparameter categories. Let $\mathbf{y} \in \mathbb{R}^m$ be the observation vector

___
[2]The algorithm with one cycle contains $(s_{max} + 1)$ subroutines of SH during which it tries different levels of exploration and exploitation with all possible values of s.

and $\mathbf{x} \in \{-1, 1\}^n$, and devide the hyperparameters into $k+k$ groups (corresponding to functions g and h). Let Ψ^l is the submatrix of $\Psi \in \mathbb{R}^{m \times (\sum_{i=1}^d \binom{n}{i})}$ where its columns match the lth group. Similarly, α^l is a weight vector corresponding to the submatrix Ψ^l and p_l denotes the length of vector α^l. In order to construct the submatrices as the Fourier basis with the hyperparameter structure, we assume that there exist a set of groups $G = \{g_1, \ldots, g_k, h_1, \ldots, h_k\}$ as defined above. If there are γ possible combinations of groups from G such that a d-degree Fourier basis exists ($\gamma = \sum_{i=1}^d \binom{2k}{i}$), we derive the k submatrices $\Psi^1, \ldots, \Psi^\gamma$ using Definition 3.1. Then the problem is equivalent to a convex optimization problem known as the Group Lasso:

$$\min_{\alpha} \frac{1}{2} \|y - \sum_{l=1}^m \Psi^l \alpha^l\|_2^2 + \lambda \sum_{l=1}^m \sqrt{p_l} \|\alpha^l\|_2 \tag{4.4}$$

Finally, the algorithm requires the input ρ, representing a reset probability parameter that produces random samples from the original reduced hyperparameter space. This parameter prevents the biased in different PGSR stages.

4.2 Differences between PGSR-HB and Harmonica

The standard Harmonica method samples the measurements under a uniform distribution. It then runs the search algorithm to recover the function f with PSR (the sparse recovery through l_1 penalty, or standard Lasso). Moreover, the number of randomly sampled measurements and its resources (training epochs) needs to be given before starting the search algorithm. we note that the reliability of measurements hugely depends on the number of resources used on each sampled point. While investing enormous resources in recovering Fourier coefficients guarantees the success of the Lasso recovery, this is inefficient with respect to total budget. On the other hand, collecting the measurements with small resources make PSR fail to provide the correct guidance for the outer search algorithm. Since PGSR-HB gathers all the function outputs – from cheap resources to the most expensive resources – PGSR-HB eliminates the need to set an explicit number of samples and training epochs as in Harmonica. We have also tried other penalties, and observed that the regularized regression tends to learn slower than the models without a regularization; thus, misleading the search algorithm with the worst performance. While the experimental results in [HKY17] shows promising results in finding the influential categorical hyperparameters such as presence/absence of the Batch-normalization layer, it cannot be used directly in optimizing the numerical hyperparameters such as learning rate, weight decay, and batch size. One the other hand, PGSR-HB overcomes this limitation of Harmonica with the log-linear representation in equation 4.3 and Group Lasso equation 4.4.

5 HPO Experimental Results

5.1 Robustness Test

We verify the robustness of PGSR-HB by generating a test loss surface picking two hyperparameter categories as shown in Figure 1. We calculate the test loss by training 120 epochs for classification of CIFAR-10 data set. We use the convolutional neural network architecture from the cuda-convnet-82% model that has been used in previous work ([JT16] and [LJD$^+$17]). The range of learning rate and the weight-decay on the first convolutional layer is set to be in 10^{-6} to 10^2. We use the log scale on both horizontal and vertical axis for the visualize of the loss surface.

Table 1 compares the performance of PGSR and PSR with equation 4.3, and PSR with evenly spaced hyperparameter values in log scale. The third and fourth columns in Table 1 list the reduced hyperparameter space for learning rate and l_2 regularization coefficient for first convolution layer for each algorithm. From the comparison between the reduced search space and hyperparameter loss surfaces from Figure 2 and Figure 3, we observe equation 4.3 reduce the search domain more accurately than without equation 4.3. Imposing structured sparsity on the hyperparameters by grouping them not only helps PGSR to return the correct guidance, but also it stabilize the lasso coefficient λ as shown in the test loss surfaces (Figure 2 and Figure 3) with PGSR results in Table 1.

Next, we optimize the five categories of hyperparameters including the learning rate, three convolution layers' and l_2 regularization coefficient of the fully connected layer using the architecture and dataset used in the previous section. We train the network using the stochastic gradient descent without a momentum and diminish the learning rate by a factor 0.1 every 100 epochs. We compare the test loss and accuracy of SH, Hyperband, doubled budgets Random Search with PGSR-HB algorithm. We set the resource, $R = 243$ and the discard ratio input $\eta = 3$. Training epochs is the same for all the algorithms except Random Search with times more training epochs. We run each algorithm in Table 2 for four different trials. The results verify the effectiveness of reducing the hyperparameter space through PGSR as the new algorithm returns better performance for most of the trials. Moreover, PGSR-HB finds the optimal hyperparameters with 83% test accuracy, outperforming the other algorithms from all trials.
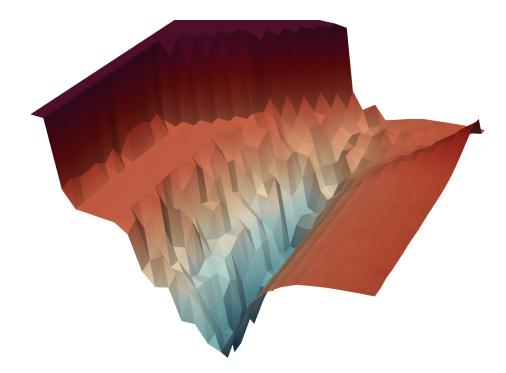
Figure 1: *Test loss surface with two hyperparameters. Learning rate vs conv1 l2 penalty.*

Table 1: Reduced Search Domain Comparison on Learning Rate and Conv1 L2

Method	λ	Learning Rate	Conv1 Penalty
PGSR	0.5	$[\mathbf{10^{-3}}, \mathbf{10^{-2}}]$	$[\mathbf{10^{-5}}, \mathbf{10^{-4}}]$
PGSR	1.0	$[\mathbf{10^{-3}}, \mathbf{10^{-2}}]$	$[\mathbf{10^{-5}}, \mathbf{10^{-4}}]$
PGSR	2.0	$[\mathbf{10^{-3}}, \mathbf{10^{-2}}]$	$[\mathbf{10^{-5}}, \mathbf{10^{-4}}]$
PSR	0.5	$[\mathbf{10^{-3}}, \mathbf{10^{-2}}]$	$[10^{-6}, 10^{2}]$
PSR	1.0	$[10^{-4}, 10^{-3}]$	$[10^{-3}, 10^{-2}]$
PSR	2.0	$[10^{0}, 10^{2}]$	$[10^{-3}, 10^{-2}]$
PSR w/o equation 4.3	0.5	$[10^{-4}, 10^{-2}]$	$[10^{-6}, 10^{-3}]$
PSR w/o equation 4.3	1.0	$[10^{-4}, 10^{-2}]$	$[10^{-6}, 10^{-3}]$
PSR w/o equation 4.3	2.0	$[10^{-4}, 10^{-2}]$	$[10^{-6}, 10^{-4}]$

Table 2: CNN Test Loss and Accuracy on CIFAR-10

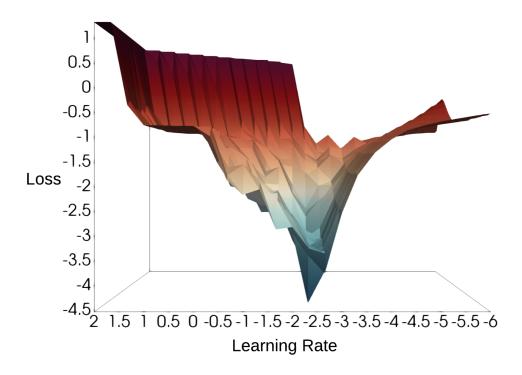| Algorithm	RS 2x	SH	HB	PGSR-HB
Loss (I)	0.7118	0.7001	0.7150	**0.6455**
Acc (I)	81.17%	79.69%	78.74%	**82.79**%
Loss (II)	0.6988	0.7179	0.6921	**0.6764**
Acc (II)	79.51%	79.30%	81.67%	**83.00**%
Loss (III)	0.6850	0.6747	0.6960	**0.6467**
Acc (III)	79.02%	79.80%	**81.47**%	80.39%
Loss (IV)	0.7293	**0.6499**	0.7215	0.6619
Acc (IV)	77.70%	80.68%	80.81%	**81.64**%

Figure 2: *The view from learning rate axis.*

6 Neural Architecture Search

In this section, we propose a a new algorithm referred to as Compressed-based Neural Architecture Search (CoNAS) for searching the CNN models. In addition, we support the performance of CoNAS with theoretical analysis. Our proposed algorithm is a novel perspective of merging the existing one-shot NAS methodology with the sparse recovery techniques. This boosts the search time to select a final candidate while outperforms existing the state-of-the-art methods.

6.1 Background on One-Shot NAS

In this section, we first briefly describe one-shot NAS techniques with respect to the search space, performance estimation strategy, and search strategy [EMH18]. Please see [BKZ$^+$18, LT19] for more discussion.

6.1.1 Search Space

Following [ZVSL18], the one-shot method searches the optimal cell as the building block (similar to [SLJ$^+$15], [HZRS16], [HLVDMW17]) to construct the final architecture. A cell is a directed acyclic graph (DAG) consisting of N nodes. A j$^{\text{th}}$ node, $n^{(j)}$ where $j \in [N]$, has directed edges (i, j) from $n^{(i)}$ where $i < j$ and $i \in [N]$ such that (i, j) transforms the node $n^{(i)}$ to $n^{(j)}$. Let $\mathcal{O}$ is the operation set (e.g. 3x3 max-pool, 3x3 average-pool, 3x3 convolution, identity in convolutional network), and there exists $|\mathcal{O}|$ direct edges between $n^{(i)}$ and $n^{(j)}$. Let $f^{(i,j)}$ maps $n^{(i)}$ to $n^{(j)}$ by summing all transformation of $n^{(i)}$ defined in $\mathcal{O}$. Each intermediate nodes (corresponding to latent representation in general) is computed by the addition of all transformation from predecessor nodes:

$$n^{(j)} = \sum_{i<j} f^{(i,j)}(n^{(i)}) \tag{6.1}$$

As each nodes are connected from previous nodes by summing all possible operations in its operation set $\mathcal{O}$, one-shot NAS literature also describe this as "weight-sharing" methods.
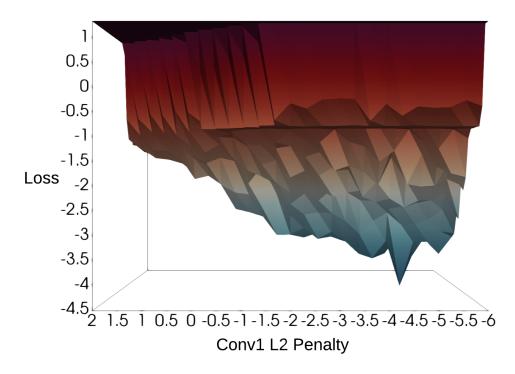
Figure 3: *The view from conv1 l2 penalty axis.*

6.1.2 Performance Estimation Strategy

The one-shot NAS models the surrogate function which takes the network architecture encoded vectors and returns the estimated performance of the sub-network. [LT19] and [BKZ$^+$18] adopts the weight-sharing paradigm following the equation 6.1. Let $\alpha \in \{-1, 1\}^n$ be the architecture encoder string where n is the total number of edges in the DAG. The one-shot surrogate model, the function $f : \{-1, 1\}^n \rightarrow \mathbb{R}$, which trained only once estimates the performance of each sub-network without training individually. While early methods including [ZVSL18] required expensive computational budget, weight-sharing paradigm significantly increased the search efficiency.

We summarize the one-shot NAS training protocol from [LT19], the training algorithm we adopted for training the super-network (surrogate model). [LT19] suggests the simple super-network training protocol as shown in Algorithm 2.

Algorithm 2 PSEUDO CODE OF ONE-SHOT MODEL TRAINING FROM [LT19]

1: **while** not converged **do**
2:      Randomly sample the architecture from DAG.
3:      Calculate the backward pass for a given minibatch.
4:      Update the weights only corresponds to the edges activated from randomly sampled architecture.
5: **end while**

6.1.3 Search Strategy

Both [BKZ$^+$18] and [LT19] explores the optimal architecture based on random search. While vanilla random search trains each candidates until its convergence (which requires exhaustive computation), one-shot model estimates the performance of candidate architecture by one forward pass evaluation. This allows to explore immense number of candidates than the vanilla random search given equivalent computational budgets.
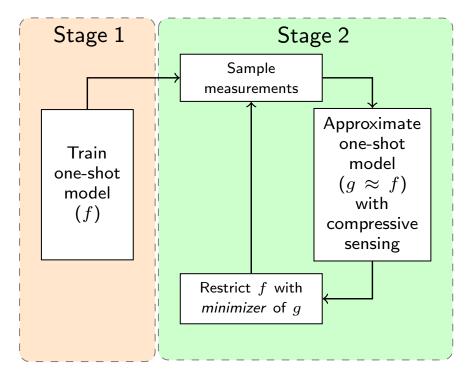
Figure 4: *Overview of CoNAS. A one-shot neural network model f is pre-trained, and an appropriate sub-graph of f is chosen by applying a sparse recovery technique. Iterative sparse recoveries allow to find the larger sub-graph from f.*
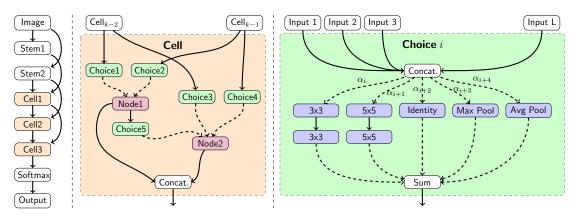

Figure 5: *Diagram inspired by [BKZ+18]. The example architecture encoder α samples the sub-architecture for $N = 5$ nodes (two intermediate nodes) with five different operations. Each component in α maps to the edges one-to-one in all Choice blocks in a cell. If a bit in α corresponds to 1, the edge activates, while -1 turns off the edge. Since the CNN search space finds both* normal cell *and* reduce cell*, the length of α is equivalent to $(2 + 3) \cdot 5 \cdot 2 = 50$.*

6.2 Proposed Algorithm

Our proposed algorithm, Compressive sensing-based Neural Architecture Search (CoNAS), infuses ideas from learning a sparse graph (Boolean Fourier analysis) into one-shot NAS. CoNAS consists of two novel components: an expanded search space, and a more effective search strategy. Figure 4 shows the overview idea of CoNAS.

Our first ingredient is an expanded search space. Following the approach of DARTS [LSY18], we define a directed acyclic graph (DAG) where all predecessor nodes are connected to every intermediate node with all possible operations. We represent any sub-graph of the DAG using a binary string α called the *architecture encoder*. Its length is the total number of edges in the DAG, and a 1 (resp. -1) in α indicates an active (resp. inactive) edge.
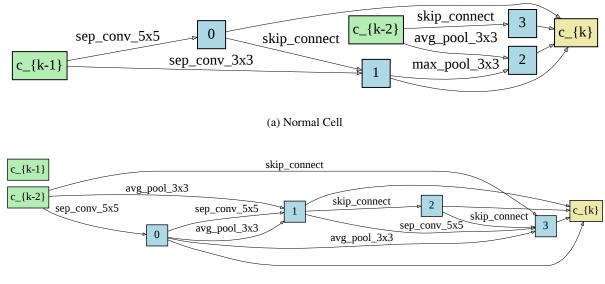
(a) Normal Cell

(b) Reduce Cell

Figure 6: Convolution Cell found from CoNAS. The reduce cell found from CoNAS have a missing connection between c_{k-1} and intermediate nodes which is a valid architecture in our search space.

Figure 5 gives an example of how the architecture encoder $\boldsymbol{\alpha}$ samples the sub-architecture of the fully-connected model in case of a convolutional neural network. The goal of CoNAS is to find the "best" encoder $\boldsymbol{\alpha}^*$, which is "close enough" to the global optimum returning the best validation accuracy by constructing the final model with $\boldsymbol{\alpha}^*$ encoded sub-graph.

Since each edge can be switched on and off independently, the proposed search space allows exploring a cell with more diverse connectivity patterns than DARTS [LSY18]. Moreover, the number of possible configurations exceeds similar previously proposed search spaces with constrained wiring rules [LT19, PGZ+18, ZVSL18, RAHL19].

We propose a compressive measuring strategy to approximate the one-shot model with a Fourier-sparse Boolean function. Let $f : \{-1, 1\}^n \rightarrow \mathbb{R}$ map the sub-graph of the one-shot pre-trained model encoded by $\boldsymbol{\alpha}$ to its validation performance. Similar to [HKY17], we collect a small number of function evaluations of f, and reconstruct the Fourier-sparse function $g \approx f$ via sparse recovery algorithms with randomly sampled measurements. Then, we solve $\arg\min_{\boldsymbol{\alpha}} g(\boldsymbol{\alpha})$ by exhaustive enumeration over all coordinates in its sub-cube $\{-1, 1\}^{\overline{J}}$ where $(J, \overline{J})$ partitions $[n]$ (Definition 3.2)[3]. If the solution of the $\arg\min_{\boldsymbol{\alpha}} g(\boldsymbol{\alpha})$ does not return enough edges to construct the cell (some intermediate nodes are disconnected), we simply connect the intermediate nodes to the previous cell output, Cell$_{k-2}$, using the Identity operation (this does not increase neither the model size nor number of multiply-add operations). Larger cells can be found from multiple iterations by restricting the approximate function g and with fixing the bit values found in the previous solution, and randomly sampling sub-graphs in the remaining edges.

We now describe CoNAS in detail, with pseudocode provided in Algorithm 3. We first train a one-shot model with standard backpropagation but only updates the weight corresponding to the randomly sampled sub-graph edges for each minibatch. Then, we randomly sample sub-graphs by generating architecture encoder strings $\boldsymbol{\alpha} \in \{-1, 1\}^n$ using a $Bernoulli(p)$ distribution for each bit of $\boldsymbol{\alpha}$ independently (We set $p = 0.5$).

In the second stage, we collect m measurements of randomly sampled sub-architecture performance denoted by $\mathbf{y} = (f(\boldsymbol{\alpha}_1), f(\boldsymbol{\alpha}_2), \ldots, f(\boldsymbol{\alpha}_m))^T$. Next, we construct the *graph-sampling matrix* $\mathbf{A} \in \{-1, 1\}^{m \times |\mathcal{P}_d|}$ with entries

$$\mathbf{A}_{l,k} = \chi_{S_k}(\boldsymbol{\alpha}_l), \qquad l \in [m], k \in [|\mathcal{P}_d|], S \subseteq [n], |S| \le d, \tag{6.2}$$

---

[3]This is similar to the idea of de-biasing in the *Hard-Thresholding (HT)* algorithm [FR17] where the support is first estimated, and then within the estimated support, the coefficients are calculated through least-squares estimation.

Algorithm 3 PSEUDOCODE OF CONAS

1: **Inputs:** Number of one-shot measurements m, stage t, sparsity s, lasso parameter, λ, Bernoulli p

---

    Stage 1 – Training the One-Shot Model

2:   **procedure** MODEL TRAINING
3:     **while** not converged **do**
4:         Randomly sample a sub-architecture encoded binary vector $\boldsymbol{\alpha}$ according to Bernoulli(p)
5:         Update $w_{\boldsymbol{\alpha}}$ by descending $\nabla_{w_{\boldsymbol{\alpha}}} \mathcal{L}_{train}(w_{\boldsymbol{\alpha}})$
6:     **end while**
7:   **end procedure**

    Stage 2 – Search Strategy

8:   **procedure** ONE-SHOT MODEL APPROXIMATION VIA COMPRESSIVE SENSING
9:     **for** $k \in \{1, \ldots, t\}$ **do**
10:       Collect $\mathbf{y} = (f(\boldsymbol{\alpha}_1), f(\boldsymbol{\alpha}_2), \ldots, f(\boldsymbol{\alpha}_m))^{\top}$.
11:       Solve

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1$$

12:       Let $x_1^*, x_2^*, \ldots, x_s^*$ be the s absolutely largest coefficients of $\mathbf{x}^*$. Construct

$$g(\boldsymbol{\alpha}) = \sum_{i=1}^{s} x_i^* \chi_i(\boldsymbol{\alpha})$$

13:       Compute minimizer $\mathbf{z} = \arg\min_{\boldsymbol{\alpha}} g(\boldsymbol{\alpha})$ and let $\overline{J}$ the set of indices of z
14:       $f = f_{J|\mathbf{z}}$
15:     **end for**
16:     Construct the cell by activating the edge where $z_i = 1$ where $i \in [n]$.
17:   **end procedure**

---

where d is the maximum degree of monomials in the Fourier expansion, and S_k is the index set corresponding to k^{th} Fourier basis. We solve the familiar Lasso problem [Tib96]:

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathbb{R}^{|\mathcal{P}_d|}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1, \tag{6.3}$$

to (approximately) recover the global optimizer $\mathbf{x}^*$, the vector contains the Fourier coefficients corresponding to $\mathcal{P}_d$. We define an approximate function $g \approx f$ with Fourier coefficients with the top-s (absolutely) largest coefficients from $\mathbf{x}^*$, and compute $\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} g(\boldsymbol{\alpha})$, resulting all the possible points in the subcube defined by the support of g (this computation is feasible if s is small). Multiple stages of sparse recovery (with successive restrictions to previously obtained optimal $\boldsymbol{\alpha}^*$) enable us to approximate additional monomial terms. Finally, we obtain a cell to construct the final architecture by activating the edges corresponding to all $i \in [n]$ such that $\alpha_i^* = 1$.

6.3 Theoretical Analysis for CoNAS.

The system of linear equations $\mathbf{y} = \mathbf{A}\mathbf{x}$ with the graph-sampling matrix $\mathbf{A} \in \{-1, 1\}^{m \times O(n^d)}$, measurements $\mathbf{y} \in \mathbb{R}^m$, and Fourier coefficient vector $\mathbf{x} \in \mathbb{R}^{O(n^d)}$ is an ill-posed problem when $m \ll O(n^d)$ for large n. However, if the graph-sampling matrix satisfies *Restricted Isometry Property (RIP)*, the sparse coefficients, $\mathbf{u}$ can be recovered:

Definition 6.1. *A matrix* $\mathbf{A} \in \mathbb{R}^{m \times \mathcal{O}(n^d)}$ *satisfies the restricted isometry property of order s with some constant δ if for every s-sparse vector* $\mathbf{u} \in \mathbb{R}^{\mathcal{O}(n^d)}$ *(i.e., only s entries are non-zero) the following holds:*

$$(1 - \delta)\|\mathbf{u}\|_2^2 \leq \|\mathbf{A}\mathbf{u}\|_2^2 \leq (1 + \delta)\|\mathbf{u}\|_2^2.$$

We defer the history of improvements on the upper bounds of the number of rows from bounded orthonormal dictionaries (matrix $\mathbf{A}$) for which $\mathbf{A}$ is guaranteed to satisfy the restricted isometry property with high probability in Appendix A.2. To the best of our knowledge, the best known result with mild dependency on δ (i.e., δ^{-2}) is due to [HR17], which we can apply for our setup. It is easy to check that the graph-sampling matrix $\mathbf{A}$ in our proposed CoNAS algorithm satisfies BOS for $K = 1$ (Eq 6.2).

Table 3: **Comparison with hand-designed networks and state-of-the-art NAS methods on CIFAR-10** *(Lower test error is better). The average test error of our experiment used five random seeds. Table entries with "-" indicates that either the field is not applicable or unknown. The methods listed in this table are trained with auxiliary towers and cutout augmentation.*

Architecture	Test Error (%)	Params (M)	Multi-Add (M)	Search GPU days
PyramidNet [YIAK18]	2.31	26	-	-
AutoAugment [CZM+19]	1.48	26	-	-
ProxylessNAS [CZH19]	2.08	5.7	-	4
NASNet-A [ZVSL18]	2.65	3.3	-	2000
AmoebaNet-B [RAHL19]	2.55 ± 0.05	2.8	-	3150
GHN+ [ZRU18]	2.84 ± 0.07	5.7	-	0.84
SNAS [XZLL18]	2.85 ± 0.02	2.8	-	1.5
ENAS [PGZ+18]	2.89	4.6	-	0.45
DARTs [LSY18]	2.76 ± 0.09	3.3	548	4
Random Search [LSY18]	3.29 ± 0.15	3.1	-	4
ASHA [LT19]	$2.85\|3.03 \pm 0.13$	2.2	-	-
RSWS [LT19]	$2.71\|2.85 \pm 0.08$	3.7	634	2.7
DARTs# [LT19]	$2.62\|2.78 \pm 0.12$	3.3	-	4
DARTs†	$2.59\|2.78 \pm 0.13$	3.4	576	4
CoNAS (t=1)	$\mathbf{2.57}\|2.74 \pm 0.12$	**2.3**	**386**	**0.4**
CoNAS (t=4)	$2.55\|2.62 \pm 0.06$	4.8	825	0.5
CoNAS (t=1, C=60)+ +AutoAugment	1.87	6.1	1019	0.4

[#] DARTS experimental results from [LT19].
[†] Used DARTS search space with five operations for direct comparisons.
[+] 'C' stands for the number of initial channels. Trained 1,000 epochs with AutoAugment.

Theorem 6.2. *Let the graph-sampling matrix $A \in \{-1,1\}^{m \times \mathcal{O}(n^d)}$ be constructed by taking m rows (random sampling points) uniformly and independently from the rows of a square matrix $\mathbf{M} \in \{-1,1\}^{\mathcal{O}(n^d) \times \mathcal{O}(n^d)}$. Then the normalized matrix $\mathbf{A}$ with $m = \mathcal{O}(\log^2(\frac{1}{\delta})\delta^{-2}s\log^2(\frac{s}{\delta})d\log(n))$ with probability at least $1 - 2^{-\Omega(d\log n \log(\frac{s}{\delta}))}$ satisfies the restricted isometry property of order s with constant δ; as a result, every s-sparse vector $\mathbf{u} \in \mathbb{R}^{\mathcal{O}(n^d)}$ can be recovered from the sample y_i's:*

$$\mathbf{y} = \mathbf{A}\mathbf{u} = \Big(\sum_{j=1}^{|\mathcal{O}(n^d)|} u_j \mathbf{A}_{i,j} \Big)_{i=1}^{m}, \tag{6.4}$$

by LASSO (equation 6.3).

Proof. First, we note that the graph-sampling matrix A is a BOS matrix with $K = 1$; hence, directly invoking Theorem 4.5 of [HR16] to our setting, we can see that matrix A satisfies RIP. Now according to Theorem 1.1 of [Can08], letting $\delta < \sqrt{2} - 1$, the l_1 minimization or LASSO will recover exactly the s sparse vector u. For instance, in our experiments, we have selected $m = 1000$ which is consistent with our parameters, $d = 2, s = 10, n = 140$. $\square$

Here, it is worthwhile to mention two points: first, the above upper bound on the number of rows of the graph-sampling matrix A is the tightest bound (according to our knowledge) for the BOS matrices to satisfy RIP. There exist series of results establishing the RIP for BOS matrices during the last 15 year. We have reviewed these results in the Appendix A.2. Second, instead of LASSO, one can use any sparse recovery method (such as IHT [BD09]) in our algorithm. In essence, Theorem 6.2 provides a successful guarantee for recovering the optimal sub-network of a given size given a sufficient number of performance measurements.

Table 4: ***Image Classification Test Error of CoNAS on Multiple Datasets***. *We compare the performance of CoNAS on different datasets with existing NAS results. The experiment details for CoNAS is described in Appendix A.3.*

Arch	CIFAR100 (%)	SVHN (%)	F-MNIST (%)	Params (M)	Search (GPU)
[XZLL18]	16.5	1.98	3.73	2.8	1.5
[LZN$^+$18]	15.9	1.83	3.72	3.2	150
[ZVSL18]	15.8	1.96	3.71	3.3	1800
[LSY18]	15.8	1.85	3.68	3.4	1
[RAHL19]	15.9	1.93	3.8	3.2	3150
[NNR$^+$19]	15.6	1.81	3.73	2.5	0.2
CoNAS	15.9	1.44	4.11	2.3	0.4

Experimental result of [XZLL18], [LZN$^+$18], [ZVSL18], [LSY18], [RAHL19], [NNR$^+$19] are taken from [NNR$^+$19].

7   NAS Experimental Results

We experiment on image classification NAS problems: CNN search on CIFAR-10, CIFAR-100, Fashion MNIST and SVHN. We describe the training details for CIFAR-10 in Sections 7.1. Our evaluation setup for training the final architecture (CIFAR-10) is the same as that reported in DARTS and RSWS.

7.1   Convolutional Neural Network

Architecture Search. We create a one-shot architecture similar to RSWS with a cell containing $N = 7$ nodes with two nodes as input and one node as output; our wiring rules between nodes are different and as in Figure 5. We used five operations: 3×3 and 5×5 separable convolutions, 3×3 max pooling, 3×3 average pooling, and Identity. On CIFAR-10, we equally divide the $50,000$-sample training set to training and validation sets to train one-shot super-network, following [LT19] and [LSY18]. We train a one-shot model by sampling the random sub-graph under Bernoulli(0.5) sampling with eight layers and 16 initial channels for 100 training epochs. All other hyperparameters used in training the one-shot model are the same as in [LT19].

We run CoNAS in two different settings to find small and large size CNN cells. Specifically, we use the sparsity parameters $s = 10$, Fourier basis degree $d = 2$, and Lasso coefficient $\lambda = 1$ (We include experiments with varying lasso coefficients in Subsection 7.3). As a result, we found the *normal cell* and *reduce cell* with one sparse recovery stage as shown in Appendix 6 (the larger CNN cells were found with multiple sparse recovery stages). Repeating four stages ($t = 4$) of sparse recovery with restriction in definition (3.2) returns an architecture encoder α^* with numerous operation edges in the cells (Please see Figure 7 for the found architecture). Now, we evaluate the model found by CoNAS as follows:

Architecture Evaluation. We re-train the final architecture with the learned cell and with the same hyperparameter configurations in DARTS to make the direct comparisons. We use NVIDIA TITAN X, GTX 1080, and Tesla V100 for final architecture training process. We use TITAN X for searching the architecture for our experiment to conduct the fair comparison on search time. CONAS search time in Table 3 includes both training an one-shot model and gathering measurements. CoNAS cells from four sparse recovery stages (t=4) cannot use the same minibatch size (i.e., 96) used in DARTs and RSWS, due to the hardware constraint; instead, we re-train the final model with minibatch size 56 with TITAN X.

CoNAS architecture with one sparse recovery (t=1) outperforms DARTs and RSWS (stronger than vanilla random search) in test errors with smaller parameters, multiply-addition operations, and search time. In addition, CoNAS with four recovery stages (t=4) performs better than CoNAS (t=1) on both lower test error average and deviation; however, it requires larger parameters and multiply-add operations compared to DARTs, RSWS, and CoNAS (t=1). We also train CoNAS (t=1) with increasing the number of channels from 36 to 60 and training epochs from 600 to 1,000 together with a recent data augmentation technique called AutoAugment [CZM$^+$19], which breaks through 2% test error barrier on CIFAR-10.

Table 5: Lasso Parameter Stability Experiment.

Criteria	$\lambda = 0.5$	$\lambda = 2.0$	$\lambda = 5.0$	$\lambda = 10.0$	Random
Hamming Dist.	0	0	8	12	29
Test Error (%)	3.74	3.74	3.51	3.62	4.43
Param (M)	2.3	2.3	2.6	2.6	2.7
MADD (M)	386	386	455	449	444

7.2 Transfer to other datasets

We test the cell found from CIFAR-10 to evaluate the transferability to different datasets: CIFAR-100, SVHN, and Fashion-MNIST in Table 4. As we can see, CoNAS achieves the competitive results with the smallest architecture size (equivalent to number of parameter) compared to the other algorithms.

7.3 Stability on Lasso Parameters

We check our algorithm's stability on lasso parameter by observing the solution given exact same measurements. Denote $\alpha^*_{\lambda=l}$ as the architecture encoded output from CoNAS given $\lambda = l$. We compare the hamming distance and the test error between $\alpha^*_{\lambda=1}$ and other λ values ($\lambda = 0.5, 2, 5, 10$). The average support of the solution from one sparse recovery is 15 out of the 140 length. The average hamming distance between two randomly generated binary strings with $\text{supp}(\alpha^*) = 15$ from $100,000$ samples was 27.58 ± 1.82. Our experiment shows a stable performance under various lasso parameters with small hamming distances regards to various λ. Also we measure the average test error with 150 training epochs on different λ values as shown in Table 5. For the baseline comparison, we compare CoNAS solutions with the randomly chosen architecture with 15 operations (edges).

7.4 Discussion

Noticeably, CoNAS achieves improved results on CIFAR-10 in both test error and search cost when compared to the previous state-of-the-art algorithms: DARTs, RSWS, and ENAS. In addition, not only CoNAS finds the cell with smallest parameter size and multiply-add operations than the other NAS approaches, but also it obtains a better test error with 2.57%. Many previous NAS papers have focused on the search strategy, while adopted the same search space to [ZVSL18] and [LSY18]. Our experimental results highlight the importance of both seeking new performance strategies and the search space.

8 Conclusions

In this paper, we considered the problems of hyperparameter optimization and neural architecture search through the lens of compressive-sensing. As our primary contribution, we first extended Harmonica algorithm by introducing the new log-linear representation for numerical hyperparameters by posing group sparsity in hyperparameters space. We support our algorithms by providing some experiments for the classification task and by visualizing the reduction of the hyperparameters space. We also tackled neural architecture search problem and proposed CoNAS, which expresses the surrogate function of the one-shot super-network via Boolean loss function. We supported our NAS algorithm with a theoretical analysis and extensive experimental results on convolutional networks. Several interesting future works remain, including applying the boolean function scheme to the other neural network architectures such as recurrent neural network, generative adversarial network, and transformers. Moreover, extending the boolean function idea to the neural network compression scheme (which attempts to prune the weights) by finding the appropriate binary mask which Hadamard product to weights while achieving minor (or no) degrade of performance will be interesting direction of future study.

A Appendix

A.1 Algorithms for SH and Hyperband

In this section, we include the pseudo algorithm including Successive Halving (Algorithm 4) and Hyperband (Algorithm 5).

Algorithm 4 Successive Halving (SH) from [JT16]

1: **Input:** Resource R, scaling factor η
2: **Initialization:** $s_{max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{max} + 1)R$
3: $n = R, r = 1$
4: T = sample_configuration(n)
5: **for** $i \in \{0, \ldots, s_{max}\}$ **do**
6:     $n_i = \lfloor n\eta^{-i} \rfloor$
7:     $r_i = r\eta^i$
8:     $L = \{f(t, r_i) : t \in T\}$
9:     $T = \text{top}_k(T, L, \lfloor \frac{n_i}{\eta} \rfloor)$
10: **end for**
11: **return** Configuration with the smallest loss

Algorithm 5 HYPERBAND FROM [LJD$^+$17]

1: **Inputs:** Resource R, scaling factor η
2: **Initialization:** $s_{max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{max} + 1)R$
3: **for** $s \in \{s_{max}, s_{max} - 1, \ldots, 0\}$ **do**
4:     $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$, $r = R\eta^{-s}$
5:     T = sample_configuration(n)
6:     **for** $i \in \{0, \ldots, s\}$ **do**
7:         $n_i = \lfloor n\eta^{-i} \rfloor$
8:         $r_i = r\eta^i$
9:         $L = \{f(t, r_i) : t \in T\}$
10:         $T = \text{top}_k(T, L, \lfloor \frac{n_i}{\eta} \rfloor)$
11:     **end for**
12: **end for**
13: **return** Configuration with the smallest loss

A.2 Prior Works on Recovery Conditions on Compressive Sensing

There has been significant research during the last decade in proving upper bounds on the number of rows of bounded orthonormal dictionaries (matrix $\mathbf{A}$) for which $\mathbf{A}$ is guaranteed to satisfy the restricted isometry property with high probability. One of the first BOS results was established by [CT06], where the authors proved an upper bound scales as $\mathcal{O}(sd^6 \log^6 n)$ for a subsampled Fourier matrix. While this result is seminal, it is only optimal up to some *polylog* factors. In fact, the authors in chapter 12 of [FR17] have shown a necessary condition (lower bound) on the number of rows of BOS which scales as $\mathcal{O}(sd \log n)$. In an attempt to achieve to this lower bound, the result in [CT06] was further improved by [RV08] to $\mathcal{O}(sd \log^2 s \log(sd \log n) \log n)$. Motivated by this result, [CGV13] has even reduced the gap further by proving an upper bound on the number of rows as $\mathcal{O}(sd \log^3 s \log n)$. The best known available upper bound on the number of rows appears to be $\mathcal{O}(sd^2 \log s \log^2 n)$; however with worse dependency on the constant δ, i.e., δ^{-4} (please see [Bou14]). To the best of our knowledge, the best known result with mild dependency on δ (i.e., δ^{-2}) is due to [HR17], and is given by $\mathcal{O}(sd \log^2 s \log n)$. We have used this result for proving Theorem 6.2.

A.3 Training Details on other Datasets

A.3.1 CIFAR-100

This dataset is extended version of CIFAR-10 with 100 classes containing 600 images each. Similar to CIFAR-10, CIFAR100 consists of 60,000 color images which splits into 50,000 training images and 10,000 test images. Following [LSY18], we train the architecture with 20 stacked cells equivalent to CIFAR-10 setting. We train the architecture for 600 epochs with cosine annealing learning rate where the initial value is 0.025. We use a batch size 96, SGD optimizer with nestrov-momentum of 0.9, and auxiliary tower with weights 0.4. For the regularization technique, we include path dropout with probability 0.2, cutout regularizer with length 16, and AutoAugment [CZM$^+$19] for CIFAR-100. Except AutoAugment, the training setup is identical to DARTs for CIFAR-10.
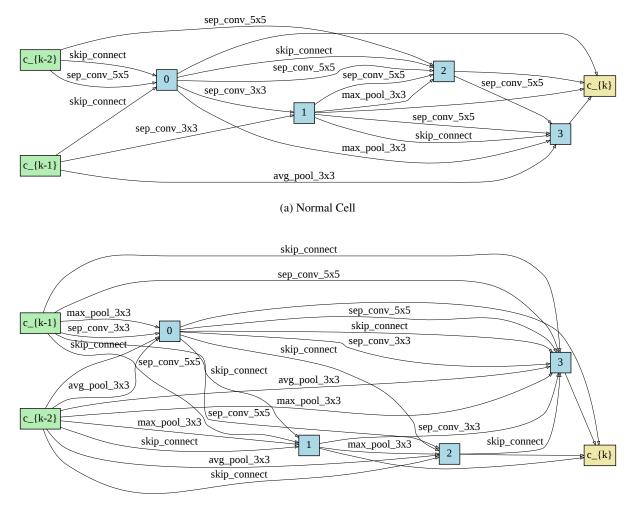
A.3.2 Street View House Numbers (SVHN)

SVHN is a digit recognition dataset of house numbers obtained from Google Street View images. SVHN consists of 73,257 train digit images, 26,032 test digit images, and additional 531,131 images. We used both train and extra (total 604,388) images for the training the architecture. Due to the large dataset, we train the architecture for 160 epochs (equivalent to) and other hyperparameter setup is equivalent to CIFAR-100.
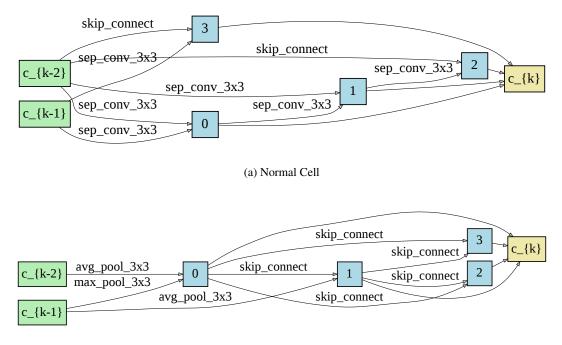
A.3.3 Fashion-MNIST

Fashion-MNIST consists of 60,000 grayscale training images and 10,000 test images with size 28×28, classified in 10 classes of objects. Training hyperparameter setup of the final architecture is equivalent to CIFAR-10 without AutoAugment [CZM$^+$19].

We list the network architectures found from our experiments which were not included in the main section.

(a) Normal Cell

(b) Reduce Cell

Figure 7: Convolution Cell found from CoNAS (t=4)

(a) Normal Cell

(b) Reduce Cell

Figure 8: Convolutional Cell found from DARTs with the original setting in [LSY18].

References

[BAPB17]   Hadrien Bertrand, Roberto Ardon, Matthieu Perrot, and Isabelle Bloch, *Hyperparameter optimization of deep neural networks: Combining hyperband with bayesian model selection.*

[BB12]   James Bergstra and Yoshua Bengio, *Random search for hyper-parameter optimization*, Journal of Machine Learning Research **13** (2012), no. Feb, 281–305.

[BBBK11]   James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl, *Algorithms for hyper-parameter optimization*, Advances in neural information processing systems, 2011, pp. 2546–2554.

[BD09]   Thomas Blumensath and Mike E Davies, *Iterative hard thresholding for compressed sensing*, Applied and computational harmonic analysis (2009).

[Ben00]   Yoshua Bengio, *Gradient-based optimization of hyperparameters*, Neural computation **12** (2000), no. 8, 1889–1900.

[BKZ+18]   Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le, *Understanding and simplifying one-shot architecture search*, Proc. Int. Conf. Machine Learning (2018).

[Bou14]   Jean Bourgain, *An improved estimate in the restricted isometry problem*, Geometric Aspects of Functional Analysis: Israel Seminar (GAFA) (2014), 65.

[Can08]   Emmanuel J Candes, *The restricted isometry property and its implications for compressed sensing*, Comptes rendus mathematique **346** (2008), no. 9-10, 589–592.

[CCZ+18]   Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang, *Efficient architecture search by network transformation*, Proc. Assoc. Adv. Art. Intell. (AAAI) (2018).

[CGV13]   Mahdi Cheraghchi, Venkatesan Guruswami, and Ameya Velingker, *Restricted isometry of fourier matrices and list decodability of random linear codes*, SIAM Journal on Computing (2013).

[CH19]   Minsu Cho and Chinmay Hegde, *Reducing the search space for hyperparameter optimization using group sparsity*, Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing (ICASSP) (2019).

[CSH19]   Minsu Cho, Mohammadreza Soltani, and Chinmay Hegde, *One-shot neural architecture search via compressive sensing*, arXiv preprint arXiv:1906.02869 (2019).

[CT06]   Emmanuel J Candes and Terence Tao, *Near-optimal signal recovery from random projections: Universal encoding strategies?*, IEEE Trans. Inform. Theory (2006).

[CZH19]   Han Cai, Ligeng Zhu, and Song Han, *Proxylessnas: Direct neural architecture search on target task and hardware*, Proc. Int. Conf. Learning Representations (2019).

[CZM$^+$19]   Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le, *Autoaugment: Learning augmentation policies from data*, IEEE Conf. Comp. Vision and Pattern Recog (2019).

[D$^+$06]   David L Donoho et al., *Compressed sensing*, IEEE Trans. Inform. Theory (2006).

[EFH$^+$13]   Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown, *Towards an empirical foundation for assessing bayesian optimization of hyperparameters*, NIPS workshop on Bayesian Optimization in Theory and Practice, vol. 10, 2013, p. 3.

[EMH18]   Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter, *Neural architecture search: A survey*, arXiv preprint arXiv:1808.05377 (2018).

[EMH19]   _____, *Efficient multi-objective neural architecture search via lamarckian evolution*, Proc. Int. Conf. Learning Representations (2019).

[FDFP17]   Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil, *Forward and reverse gradient-based hyperparameter optimization*, arXiv preprint arXiv:1703.01785 (2017).

[FKH18]   Stefan Falkner, Aaron Klein, and Frank Hutter, *Bohb: Robust and efficient hyperparameter optimization at scale*, arXiv preprint arXiv:1807.01774 (2018).

[FLF$^+$16]   Jie Fu, Hongyin Luo, Jiashi Feng, Kian Hsiang Low, and Tat-Seng Chua, *Drmad: Distilling reverse-mode automatic differentiation for optimizing hyperparameters of deep neural networks*, arXiv preprint arXiv:1601.00917 (2016).

[FR17]   Simon Foucart and Holger Rauhut, *A mathematical introduction to compressive sensing*, Bull. Am. Math **54** (2017), 151–165.

[HHLB11]   Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown, *Sequential model-based optimization for general algorithm configuration*, International Conference on Learning and Intelligent Optimization, Springer, 2011, pp. 507–523.

[HKY17]   Elad Hazan, Adam Klivans, and Yang Yuan, *Hyperparameter optimization: a spectral approach*, arXiv preprint arXiv:1706.00764 (2017).

[HLC$^+$19]   Hanzhang Hu, John Langford, Rich Caruana, Saurajit Mukherjee, Eric Horvitz, and Debadeepta Dey, *Efficient forward architecture search*, Adv. Neural Inf. Proc. Sys. (NeurIPS) (2019).

[HLVDMW17]   Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger, *Densely connected convolutional networks*, IEEE Conf. Comp. Vision and Pattern Recog (2017).

[HR16]   Ishay Haviv and Oded Regev, *The list-decoding size of fourier-sparse boolean functions*, ACM Transactions on Computation Theory (TOCT) **8** (2016), no. 3, 10.

[HR17]   _____, *The restricted isometry property of subsampled fourier matrices*, Geometric Aspects of Functional Analysis (2017), 163–179.

[HZRS16]   Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, IEEE Conf. Comp. Vision and Pattern Recog (2016).

[IAFS17]   Ilija Ilievski, Taimoor Akhtar, Jiashi Feng, and Christine Annette Shoemaker, *Efficient hyperparameter optimization for deep learning algorithms using deterministic rbf surrogates.*, AAAI, 2017, pp. 822–829.

[JDO$^+$17]   Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al., *Population based training of neural networks*, arXiv preprint arXiv:1711.09846 (2017).

[JSH18]   Haifeng Jin, Qingquan Song, and Xia Hu, *Efficient neural architecture search with network morphism*, arXiv preprint arXiv:1806.10282 (2018).

[JT16]   Kevin Jamieson and Ameet Talwalkar, *Non-stochastic best arm identification and hyperparameter optimization*, Artificial Intelligence and Statistics, 2016, pp. 240–248.

[KDVN18]   Manoj Kumar, George E Dahl, Vijay Vasudevan, and Mohammad Norouzi, *Parallel architecture and hyperparameter search via successive halving and classification*, arXiv preprint arXiv:1805.10255 (2018).

[LBGR15]   Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko, *Scalable gradient-based tuning of continuous regularization hyperparameters*, arXiv preprint arXiv:1511.06727 (2015).

[LJD+17]   Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar, *Hyperband: A novel bandit-based approach to hyperparameter optimization*, The Journal of Machine Learning Research **18** (2017), no. 1, 6765–6816.

[LJR+18]   Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar, *Massively parallel hyperparameter tuning*, arXiv preprint arXiv:1810.05934 (2018).

[LSV+17]   Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu, *Hierarchical representations for efficient architecture search*, arXiv preprint arXiv:1711.00436 (2017).

[LSY18]   Hanxiao Liu, Karen Simonyan, and Yiming Yang, *Darts: Differentiable architecture search*, Proc. Int. Conf. Machine Learning (2018).

[LT19]   Liam Li and Ameet Talwalkar, *Random search and reproducibility for neural architecture search*, arXiv preprint arXiv:1902.07638 (2019).

[LTQ+18]   Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu, *Neural architecture optimization*, Adv. Neural Inf. Proc. Sys. (NeurIPS) (2018).

[LZN+18]   Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy, *Progressive neural architecture search*, Euro. Conf. Comp. Vision (2018).

[MDA15]   Dougal Maclaurin, David Duvenaud, and Ryan Adams, *Gradient-based hyperparameter optimization through reversible learning*, International Conference on Machine Learning, 2015, pp. 2113–2122.

[NNR+19]   Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik-Manor, *Asap: Architecture search, anneal and prune*, arXiv preprint arXiv:1904.04123 (2019).

[O'D14]   Ryan O'Donnell, *Analysis of boolean functions*, Cambridge University Press, 2014.

[PGZ+18]   Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean, *Efficient neural architecture search via parameter sharing*, Proc. Int. Conf. Machine Learning (2018).

[RAHL19]   Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le, *Regularized evolution for image classifier architecture search*, Proc. Assoc. Adv. Art. Intell. (AAAI) (2019).

[RV08]   Mark Rudelson and Roman Vershynin, *On sparse reconstruction from fourier and gaussian measurements*, Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences (2008).

[SK12]   Peter Stobbe and Andreas Krause, *Learning fourier sparse set functions*, Proc. Int. Conf. Art. Intell. Stat. (AISTATS) (2012).

[SLA12]   Jasper Snoek, Hugo Larochelle, and Ryan P Adams, *Practical bayesian optimization of machine learning algorithms*, Advances in neural information processing systems, 2012, pp. 2951–2959.

[SLJ+15]   Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, *Going deeper with convolutions*, IEEE Conf. Comp. Vision and Pattern Recog (2015).

[SSZA14]   Jasper Snoek, Kevin Swersky, Rich Zemel, and Ryan Adams, *Input warping for bayesian optimization of non-stationary functions*, Proc. Int. Conf. Machine Learning, 2014, pp. 1674–1682.

[SYJ+19]   Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann, *Evaluating the search phase of neural architecture search*, arXiv preprint arXiv:1902.08142 (2019).

[THHLB13]   Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown, *Auto-weka: Combined selection and hyperparameter optimization of classification algorithms*, Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2013, pp. 847–855.

[Tib96]   Robert Tibshirani, *Regression shrinkage and selection via the lasso*, Journal of the Royal Statistical Society. Series B (Methodological) (1996), 267–288.

[WXW18]   Jiazhuo Wang, Jason Xu, and Xuejun Wang, *Combination of hyperband and bayesian optimization for hyperparameter optimization in deep learning*, arXiv preprint arXiv:1801.01596 (2018).

[XKGH19]   Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He, *Exploring randomly wired neural networks for image recognition*, arXiv preprint arXiv:1904.01569 (2019).

[XZLL18]   Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin, *Snas: stochastic neural architecture search*, arXiv preprint arXiv:1812.09926 (2018).

[YIAK18]   Yoshihiro Yamada, Masakazu Iwamura, Takuya Akiba, and Koichi Kise, *Shakedrop regularization for deep residual learning*, Proc. Int. Conf. Learning Representations Workshop (2018).

[YL06]   Ming Yuan and Yi Lin, *Model selection and estimation in regression with grouped variables*, Journal of the Royal Statistical Society: Series B (Statistical Methodology) **68** (2006), no. 1, 49–67.

[ZL17]   Barret Zoph and Quoc V Le, *Neural architecture search with reinforcement learning*, Proc. Int. Conf. Learning Representations (2017).

[ZRU18]   Chris Zhang, Mengye Ren, and Raquel Urtasun, *Graph hypernetworks for neural architecture search*, Proc. Int. Conf. Learning Representations (2018).

[ZVSL18]   Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le, *Learning transferable architectures for scalable image recognition*, IEEE Conf. Comp. Vision and Pattern Recog (2018).