ONE-SHOT NEURAL ARCHITECTURE SEARCH VIA COMPRESSIVE SENSING

Minsu Cho[*]
ECE Department
Iowa State University
Ames, IA 50010
chomd90@iastate.edu

Mohammadreza Soltani
ECE Department
Duke University
Durham, NC, 27708
mohammadreza.soltani@duke.edu

Chinmay Hegde
ECE Department
Iowa State University
Ames, IA, 50010
chinmay@iastate.edu

ABSTRACT

Neural architecture search (NAS), or automated design of neural network models, remains a very challenging meta-learning problem. Several recent works (called "one-shot" approaches) have focused on dramatically reducing NAS running time by leveraging proxy models that still provide architectures with competitive performance. In our work, we propose a new meta-learning algorithm that we call CoNAS, or Compressive sensing-based Neural Architecture Search. Our approach merges ideas from one-shot approaches with iterative techniques for learning low-degree sparse Boolean polynomial functions. We validate our approach on several standard test datasets, discover novel architectures hitherto unreported, and achieve competitive (or better) results in both performance and search time compared to existing NAS approaches. Further, we support our algorithm with a theoretical analysis, providing upper bounds on the number of measurements needed to perform reliable meta-learning; to our knowledge, these analysis tools are novel to the NAS literature and may be of independent interest.

1 Introduction

Motivation. Choosing a suitable neural network architecture for complex prediction tasks such as image classification and language modeling often requires a substantial effort of trial-and-error. Therefore, there has been a growing interest to *automatically learn* (or meta-learn) the architecture of neural networks that can achieve competitive (or better) results over hand-designed architectures. The sub-field of neural architecture search (NAS) addresses the problem of designing competitive architectures with as small a computational budget as possible.

Our Contributions. Numerous approaches for neural architecture search already exist in the literature, each with their own pros and cons: these include black-box optimization based on reinforcement learning (RL) [1], evolutionary search [2], and Bayesian optimization [3, 4]. Though the algorithmic details vary, most of these NAS methods face the common challenge of evaluating the test/validation performance of a (combinatorially) large number of candidate architecture evaluations. In a departure from traditional methods, we approach the NAS problem via the lens of *compressive sensing*. The field of compressive sensing (or sparse recovery), introduced by the seminal works of [5, 6], has received significant attention in both ML theory and applications over the last decade, and has influenced the development of numerous advances in nonlinear and combinatorial optimization.

We leverage these advances for the NAS problem. We develop a new NAS method called CoNAS (Compressive sensing-based Neural Architecture Search), which merges ideas from sparse recovery with so-called "one-shot" architecture search methods [7], described in greater detail below. CoNAS consists of two new innovations: (i) a new *search space* that permits exploration of a large(r) number of diverse candidate architectures, and (ii) a new *search strategy* that borrows ideas from recovery of Boolean functions from their (sparse) Fourier expansions.

Our experiments show that CoNAS is able to design a deep convolutional neural network with test error $2.62 \pm 0.06\%$ on CIFAR-10 classification, outperforming existing state-of-the-art methods such as DARTs [8], ENAS [9], and

[*]Corresponding author. Website: http://dice.ece.iastate.edu/ .

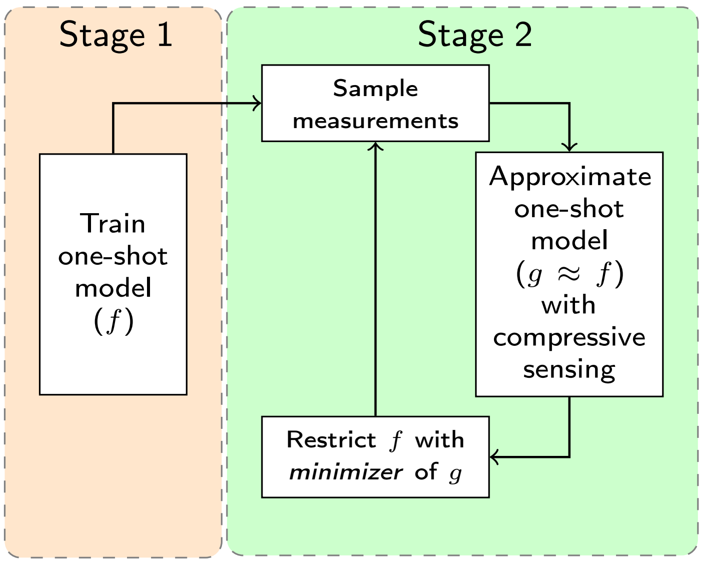

Stage 1
Stage 2
Sample measurements
Train one-shot model (f)
Approximate one-shot model ($g \approx f$) with compressive sensing
Restrict f with *minimizer* of g

Figure 1: Overview of CoNAS. A one-shot neural network model f is pre-trained, and an appropriate sub-graph of f is chosen by iteratively applying sparse recovery techniques.

random search with weight-sharing (RSWS) [10]. Moreover, CoNAS performance is competitive to NASNet [11] and AmoebaNet [2], despite requiring less than one GPU-day of computation. Our experiments on designing recurrent neural networks for language modeling are somewhat short of the state-of-the-art [12], but we find that CoNAS still finds competitive results with lesser search time than previous NAS approaches. Our results are exactly reproducible (having been trained with fixed pseudorandom seeds), and an implementation of CoNAS will be made publicly available post-peer review.

Finally, while our original motivation was to devise a empirically useful NAS method, a nice benefit is that CoNAS can be *theoretically analyzed*, since existing theoretical results for Fourier-sparse Boolean functions can be ported over in order to provide upper bounds for the number of one-shot evaluations required. This, to our knowledge, is one of the first results of their kind in the NAS literature and may be of independent interest. We defer this to the supplementary material.

Techniques. The intuition behind compressive sensing is that if a signal (or function) can be represented via a sparse basis expansion, then it can be recovered (either exactly or approximately) from a small number of randomized measurements. CoNAS leverages this intuition in the context of one-shot architecture search [7]. In one-shot NAS, instead of evaluating several candidate architectures, a single "base" neural network model is pre-trained; a class of sub-networks is identified (called the search space) and the performance of each sub-network is evaluated on a validation set; and the best-performing sub-network is finally selected and fine-tuned.

We model the sub-network selection as a sparse recovery problem. More concretely, consider a function f that maps sub-architectures to a measure of performance (validation loss). We assume that f can be written as a sparse, low-degree polynomial in the (discrete) Fourier basis[2]. If the sparsity assumption is satisfied, then we claim the function f can be reconstructed using a very small number of measurements (evaluations), thus reducing overall compute time. A key challenge lies in defining a suitable search space; we propose one that is considerably larger than the one used in DARTS or ENAS, allowing us to (putatively) search over a more diverse set of candidate architectures.

Prior work. Due to tight space limits, our review of prior work will be unfortunately brief; we refer to the recent paper by [10] for a thorough treatment. Early NAS approaches used RL-based controllers [11]) or evolutionary algorithms [2], and showed competitive performance with manually-designed architectures such as deep ResNets [14] and DenseNets [15]. However, these approaches required substantial computational resources, running into thousands of GPU-days. Subsequent NAS works have focus on boosting search speeds by proposing novel search strategies, such as sequential model-based optimization (SMBO) techniques [16], Bayesian optimization [3, 4]) and Monte Carlo tree search (MCTS) [17]. Other recent NAS approaches include weight-sharing [9], hypernetworks [18], one-shot models [7], network transformations [19, 20, 21, 22, 23], gradient descent [24, 8, 25, 26], and random exploration [27, 10, 28, 29].

[2]Intuitively, this means that f is well-approximated by a decision tree; for a formal proof, see [13].

2 Background

One-shot NAS We briefly describe one-shot neural architecture search techniques [7, 10]; a lengthier description is available in Appendix A.1. Following the recent NAS survey paper [30], one-shot NAS approaches have three main components: a search space, a search strategy, and a performance estimation strategy.

Search Space. The goal of one-shot NAS is to find the best performing *cell*, a fundamental component from which more complex architectures are constructed via stacking. Following [8], a cell is a directed acyclic graph (DAG) where a node corresponds to the latent representation, and a directed edge transforms predecessor nodes using a given *operation*; common operations used in CNNs include 3×3 and 5×5 separable convolutions, 3×3 max pooling, 3×3 average pooling, and Identity. Each cell has two input nodes and one output node, and intermediate nodes can only be connected by predecessor nodes including input nodes. Intermediate nodes are wired to two predecessor nodes in CNNs and one predecessor node in RNNs.

Search Strategy and Performance Estimation Strategy. Having defined a search space, one-shot NAS approaches employ four steps: (i) train a single "one-shot" base model that is capable of predicting the performance of sub-architectures; (ii) *randomly sample* sub-architectures of a trained one-shot model and measure performance over a hold-out validation set of samples; (iii) select the candidate (cell) with best validation performance. (iv) retrain a deeper final architecture using the best cell. Using the one-shot model as the proxy measurements of the candidate architecture corresponds to the performance estimation strategy (first and second step). The search strategy of one-shot architecture search on both [7] and [10] is equivalent to random search.

Fourier analysis of Boolean functions. We follow the treatment given in [31]. A real-valued Boolean function is one that maps n-bit binary vectors (i.e., nodes of the 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, defined as follows:

Definition 2.1 (Fourier Basis [31]). *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 defined as 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 $K = 1$, therefore satisfying two properties:

$$\langle \chi_S, \chi_T \rangle = \begin{cases} 1, & \text{if } S = T \\ 0, & \text{if } S \neq T \end{cases} \qquad \text{and} \qquad \sup_{\boldsymbol{\alpha} \in \{-1,1\}^n} |\chi_S(\boldsymbol{\alpha})| \leq 1 \quad \text{for all } S \in [n] \tag{2.1}$$

Due to orthonormality, any Boolean function f has a unique Fourier representation, given by $f(\boldsymbol{\alpha}) = \sum_{S \subseteq [n]} \hat{f}(S) \chi_S(\boldsymbol{\alpha})$, 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 nodes of the hypercube.

A modeling assumption is that the Fourier spectrum of the function is concentrated on monomials of small degree ($\leq d$). This corresponds to the case where f is a decision tree [13], and allows us to simplify the Fourier expansion by limiting its support. 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 $\mathcal{H}_{\mathcal{P}_d} := \{f : Supp[\hat{f}] \subseteq \mathcal{P}_d\}$. For example, $\mathcal{P}_2$ allows to express the function f with at most $\sum_{l=0}^{d} \binom{n}{2} \equiv \mathcal{O}(n^2)$ Fourier coefficients.

Lastly, if we have prior knowledge of some fixed set of bits indexed by set $\overline{J}$, we often use an operation called *restriction*.

Definition 2.2 (Restriction [31]). *Let $f : \{-1, 1\}^n \to \mathbb{R}$, $(J, \overline{J})$ be a partition of $[n]$, and $z \in \{-1, 1\}^{\overline{J}}$. The restriction of f to J using z denoted by $f_{J|z} : \{-1, 1\}^J \to \mathbb{R}$ is the subfunction of f given by fixing the coordinates in $\overline{J}$ to the bit values z.*

3 Proposed Algorithm: CoNAS

Overview. Our proposed algorithm, Compressive sensing-based Neural architecture Search (CoNAS), infuses ideas from Boolean Fourier analysis into one-shot NAS. CoNAS consists of two novel components: an expanded search space, and a more effective search strategy.

Search Space. Following the approach of DARTS [8], 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

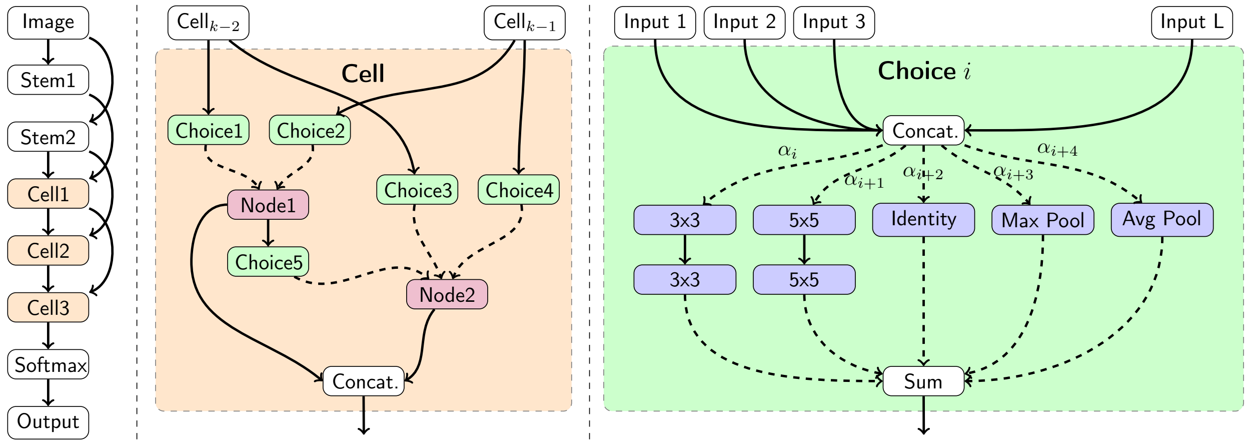


Figure 2: Diagram adapted from [7]. The architecture encoder $\boldsymbol{\alpha}$ samples the sub-architecture for $N = 5$ nodes (two intermediate nodes) with five different operations. Each component in $\boldsymbol{\alpha}$ maps to the edges one-to-one in all *Choice* blocks in a cell. If a bit in $\boldsymbol{\alpha}$ 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 $\boldsymbol{\alpha}$ is equivalent to $(2 + 3) \cdot 5 \cdot 2 = 50$.

using a binary string $\boldsymbol{\alpha}$ called the *architecture encoder*. Its length is the total number of edges in the DAG, and a 1 (resp. -1) in $\boldsymbol{\alpha}$ indicates an active (resp. inactive) edge.

Figure 2 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 [8]. Moreover, the number of possible configurations exceeds similar previously proposed search spaces with constrained wiring rules [10, 9, 2, 11]. For example, suppose the DAG for a convolutional neural network model has $N = 7$ nodes with five operations. The total number of edges is equivalent to $E = (2 + 3 + 4 + 5) \cdot 5 \cdot 2 = 140$; if $1/4$ of the edges are activated, then the number of possible combinations is equal to $\binom{140}{35} \approx 1.2 \times 10^{33}$, while DARTs with five operations only supports $((5 + 1)^{14})^2 \approx 6.1 \times 10^{21}$ configurations.

Search Strategy. We propose a compressive measuring strategy to approximate the one-shot model with a Fourier-sparse Boolean function. Let $f : \{-1, 1\}^n \to \mathbb{R}$ map the sub-graph of the one-shot pre-trained model encoded by $\boldsymbol{\alpha}$ to its validation performance. Similar to [13], we collect a small number of function evaluations of f, and reconstruct the Fourier-sparse function $g \approx f$ via compressive sensing 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 $[n]$. 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 perform sparse recovery multiple times, following [13]. In each iteration, we restrict the approximate function g by by fixing the bit values found in the previous solution, and randomly sample sub-graphs in the remaining edges.

Full Algorithm. We now describe CoNAS in detail, with pseudocode shown in Algorithm 1. We first train a one-shot model with standard backpropagation but only updates the weight corresponding to 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 $\boldsymbol{\alpha}$.

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

$$\mathbf{A}_{l,k} = \chi_S(\boldsymbol{\alpha}_l), \qquad l \in [m], k \in [|\mathcal{P}_d|], S \subseteq [n], |S| \leq d \tag{3.1}$$

where d is the max degree of monomials in the Fourier expansion. We solve the familiar Lasso problem [32]:

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathbb{R}^{|\mathcal{P}_d|}}{\arg\min} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \lambda \|\mathbf{x}\|_1 \tag{3.2}$$

to (approximately) recover the global optimizer $\mathbf{x}^*$, the vector containing 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

Algorithm 1 CoNAS

```
1: Inputs: Number of one-shot measurements $m$, stage $t$, sparsity $s$, lasso parameter $\lambda$
   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 $\alpha$
5:         Update weights $w_{\boldsymbol{\alpha}}$ 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 $k \in \{1, \dots, t\}$ do
10:        Collect $\mathbf{y} = (f(\boldsymbol{\alpha}_1), f(\boldsymbol{\alpha}_2), \dots, f(\boldsymbol{\alpha}_m))^\top$.
11:        Solve
               $\mathbf{x}^* = \arg\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1$
12:        Let $x_1^*, x_2^*, \dots, x_s^*$ 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
```

from $\mathbf{x}^*$. We compute the minimizer $\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} g(\boldsymbol{\alpha})$ calculating all possible points in the subcube defined by the support of g (the minimizer computation is feasible if s is small). Multiple stages of sparse recovery (with successive restrictions to previously obtained optimal $\boldsymbol{\alpha}^*$) enable to approximate additional monomial terms. Finally, we obtain the cell to construct the final architecture by activating the edges corresponding to all $i \in [n]$ such that $\alpha_i^* = 1$. We include theoretical support for CoNAS in Appendix A.2.

4 Experiments and Results

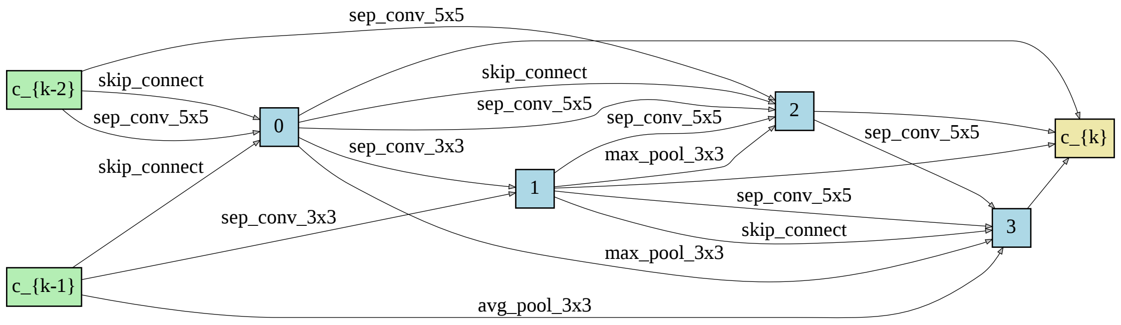
We experimented on two different NAS problems: a CNN search for CIFAR-10, and an RNN search for Penn Treebank (PTB). We describe the training details for CIFAR-10 and PTB in Sections 4.1 and 4.2 respectively. Our experimental setup for building the final architecture is the same as that reported in DARTS and RSWS.

4.1 CIFAR-10

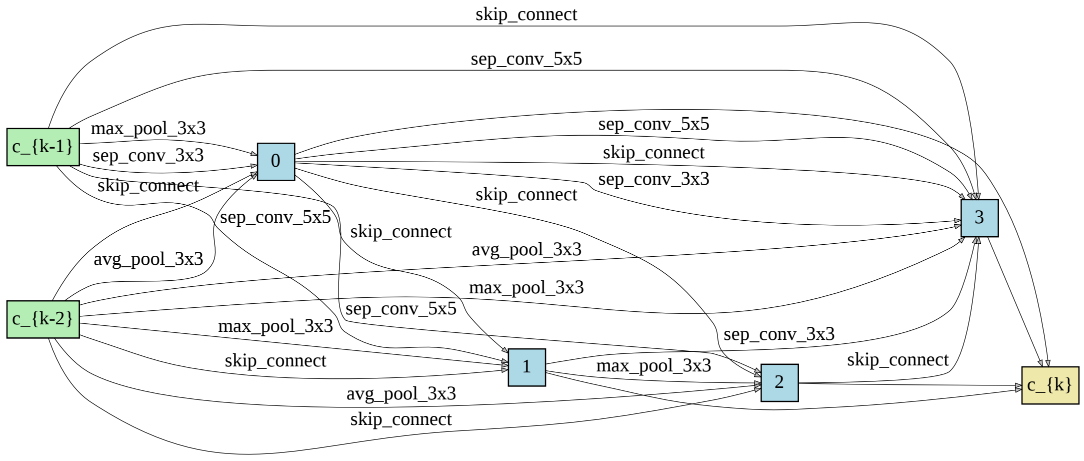
Architecture Search. We create a one-shot architecture, similar to RSWS, with a cell containing $N = 7$ nodes with two nodes as input and one node as output; our wiring rules between nodes are different and as in Section 3. We used five operations: 3×3 and 5×5 separable convolutions, 3×3 max pooling, 3×3 average pooling, and Identity. On CIFAR-10, we divide the 50,000 training set to 25,000 training set and 25,000 validation set as the equivalent experiment setup in [10] and [8]. We trained a one-shot model with eight layers and 16 initial channels for 150 epochs. All other hyperparameters used in training the one-shot model are the same as in RSWS.

We run CoNAS with multiple sparse recovery stages. Specifically, following [13], we used the parameters $s = 10$, Fourier basis degree $d = 2$, and Lasso coefficient $\lambda = 1$ for each stage. Repeating three to four stages ($t = 3, 4$) of sparse recovery with restriction returned an architecture encoder $\boldsymbol{\alpha^*}$ with sufficient number of edges needed to construct both *normal cell* and *reduce cell* (e.g., see Figure 3). The final architecture found from CoNAS used $t = 4$.

Architecture Evaluation. We re-trained the final architecture with the learned cell with the equivalent hyper-parameter configurations in DARTS and RSWS to make the direct comparisons. We used NVIDIA Titan X for



(a) Normal Cell



(b) Reduce Cell

Figure 3: Convolution Cell found from CoNAS

the training process. Since DARTS and RSWS used somewhat different GPUs (GTX1080-Ti and Tesla P100 respectively), we included not only the experimental results that have been published, but also our results of DARTs and RSWS on a Titan X to compare the search time with equivalent GPU environment in Table 1. Due to hardware memory constraints, the learned cell from CoNAS could not use the same minibatch size (96) used in DARTs and RSWS; instead, we re-trained the final model with minibatch size (56) [3].

Effect of Increasing Connections in the Cell. We provide a further supplementary experiment of training results of randomly wired cells which have a similar number of edges to the cell found by CoNAS. We randomly sampled the architecture code $\boldsymbol{\alpha}$ with $Binomial(140, 1/4)$ for each digit being 1 that the expected number of total edges in *normal cell* and *reduce cell* is 35. We randomly selected three architectures as shown in Appendix A.4.4 and trained with the equivalent environment to the section 4.1. Table 2 shows that choosing a larger number of randomly chosen edges is not sufficient to improve the model's performance.

4.2 Penn Treebank

Architecture Search. Similar to the setup of DARTS and RSWS, CoNAS explores the cell with the following operations: Tanh, ReLU, Sigmoid, and Identity. We augment the RNN cell with a variation of highway connections suggested by Pham et al [9]. Layers of depth l in Recurrent Highway Network [12] utilize a nonlinear transformation from its hidden state h_l as follows:

$$h_l = (1 - c_l) \otimes \texttt{activation}(h_{l-1} \cdot W^{(h)}_{l,l-1}) + h_{l-1} \otimes c_l$$

[3]In our experience, a larger minibatch size enabled training the model quicker than minibatches of smaller size.

Table 1: **CIFAR-10 Benchmark: Comparison with hand-designed networks and state-of-the-art NAS methods** (Lower test error is better). The results are grouped as follows: manually designed networks, published NAS algorithms, and our experimental results. 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. The search cost from our experiment is based on NVIDIA Titan X GPU.

Architecture	**Test Error** Best	Average	**Params** (M)	**Search Cost** GPU days	**Search Method**
Shake-Shake [33]	-	2.56 ± 0.07	26.2	-	manual
PyramidNet [34]	2.31	-	26	-	manual
ProxylessNAS⁺ [24]	2.08	-	5.7	4	gradient
NASNet-A [11]	-	2.65	3.3	2000	RL
AmoebaNet-B [2]	-	2.55 ± 0.05	2.8	3150	evolution
GHN⁺ [35]	-	2.84 ± 0.07	5.7	0.84	hypernetwork
SNAS [26]	-	2.85 ± 0.02	2.8	1.5	gradient
ENAS [9]	2.89	-	4.6	0.45	RL
Random search [8]	-	3.29 ± 0.15	3.1	4	random
DARTs (first order) [8]	-	3.00 ± 0.14	3.3	1.5	gradient
DARTs (second order) [8]	-	2.76 ± 0.09	3.3	4	gradient
ASHA [10]	2.85	3.03 ± 0.13	2.2	-	random
RSWS [10]	2.71	2.85 ± 0.08	4.3	2.7	random
DARTs# (second order) [10]	2.62	2.78 ± 0.12	3.3	4	gradient
DARTs† (second order)	2.59	2.78 ± 0.13	3.4	4	gradient
RSWS*	2.47	2.59 ± 0.11	4.7	0.7	random
CoNAS	2.55	2.62 ± 0.06	4.8	0.7	CS

# DARTS experimental results from [10].
† Used DARTS search space with five operations for direct comparisons.
* Used the search space defined in Section 3. Used five operations for direct comparison.
⁺ The search space is not comparable to CoNAS.

Table 2: **Randomly Wired Model Performance on CIFAR-10**. (Lower test error is better) Trained with auxiliary towers and cutout augmentation.

Method	Number of Edges	Size (M)	Test Error
Randomly Wired Model (1)	31	5.2	3.45%
Randomly Wired Model (2)	35	4.5	2.89%
Randomly Wired Model (3)	32	3.1	3.52%

where "$\otimes$" denotes element-wise multiplication. Since we use an expanded search space, we allow for mulitple operations; in such cases, we replace $\texttt{activation}(h_{l-1} \cdot W^{(h)}_{l,l-1})$ with its sum-pooled version, $\frac{1}{n}\sum_i^n \texttt{activation}^{\texttt{i}}(h_{l-1} \cdot W^{(h)}_{l,l-1})$

Pre-training the above RNN using weight-sharing showed unstable results since the sub-graphs could have some internal nodes with no connections, leading to exploding gradients; increasing the p-parameter in the Bernoulli sampling to enforce connectivity significantly slowed down computations. Hence, we added the additional heuristic of randomly activating an edge to connect the intermediate node if the node does not have any input edge according to its architecture encoder α.

After obtaining the one-shot model, we randomly sample the measurements of the sub-graph without the above heuristic used in the training stage. While running CoNAS, two stages of sparse recovery with $s = 10$ and $s = 5$ respectively found the enough number of edges for the RNN cell. If the final resulting cells has intermediate nodes with a disconnected input, we added ReLU operations from the previous intermediate node. The visualization of the RNN cell found by CoNAS is shown in Appendix A.4.1.

Architecture Evaluation. The PTB results for recurrent architectures are presented in Table 3. We trained the final RNN model with the learned cell with the equivalent hyperparameters in DARTs and RSWS, except with minibatch

Table 3: **PTB Benchmark: Comparision of the state-of-the-art NAS methods and hand-designed networks** (Lower perplexity is better). The results are grouped in following orders: manually designed networks, published NAS algorithms, and our experimental results. The average test error of our experiment used five random seeds. Table entries with "-" indicates either the field is not applicable or unknown. The search cost from our experiment is based on a NVIDIA Titan X GPUs.

Architecture	**Test Perplexity** Valid	Test	**Params** (M)	**Search Cost** GPU days	**Search Method**
Variational RHN [12]	67.9	65.4	23	-	manual
LSTM + DropConnect [36]	60.0	57.3	24	-	manual
LSTM + Mos [37]	56.5	54.4	22	-	manual
NAS [1]	-	64.0	25	1e4	RL
ENAS† [9]	-	56.3	24	0.5	RL
Random search† [8]	61.8	59.4	23	2	random
DARTS (1st order)† [8]	60.2	57.6	23	0.5	gradient
DARTS (2nd order)† [8]	58.1	55.7	23	1	gradient
ASHA* [10]	58.6	56.4	23	-	random
RSWS* [10]	57.8	55.5	23	0.25	random
RSWS$^{\#,+}$	60.6	57.9	23	0.25	random
CoNAS$^+$	59.1	56.8	23	0.25	CS

$^\#$ We used the RSWS code with adjusting the search time equivalent to CoNAS.
† Used NVIDIA GTX 1080Ti GPU for training/searching.
* Used Tesla P100 GPU for training/searching.
$^+$ Used NVIDIA Titan X GPU for training/searching.

size = 128 (due to hardware constraints). We also included the experimental results with RSWS methods allocating the equivalent search time with our methods to make the direct comparison with CoNAS. Since the published NAS literature in Table 3 uses different GPU hardware (e.g. DARTs and ENAS: NVIDIA GTX 1080Ti, RSWS: Tesla P100), a one-to-one comparison of the search cost value listed in Table 3 is not applicable. For example, both CoNAS and RSWS required 0.25 GPU-days, but the latter used a more powerful GPU. We will provide a more precise one-to-one comparison if this manuscript passes peer review.

4.3 Discussion

Noticeably, CoNAS achieved 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. We see that RSWS with an equivalent search space to CoNAS obtained a better architecture (with best test error 2.47%); however, the standard deviation of test errors over different pseudorandom was greater than CoNAS. Many previous NAS papers have focused on the search strategy while adopted the same search space tol [11] and [8]. Our experimental results highlight the importance of considering not only seeking new performance strategies, but also the search space.

On PTB, CoNAS found a better RNN architecture than RSWS using an equivalent search cost. In the overall comparison, DARTs (second order) and RSWS outperformed both valid and test perplexity of CoNAS. We could not include true one-to-one comparisons with other algorithms since they used different search spaces.

Acknowledgements

The authors would like to thank Amitangshu Mukherjee, Soumik Sarkar, and Alberto Speranzon for helpful discussions. This work was supported in part by NSF grants CCF-1566281, CAREER CCF-1750920, a GPU gift grant from the NVIDIA Corporation, and a faculty fellowship from the Black and Veatch Foundation.

References

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

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

[3] Shengcao Cao, Xiaofang Wang, and Kris M Kitani. Learnable embedding space for efficient neural architecture compression. *Proc. Int. Conf. Learning Representations*, 2019.

[4] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, 2018.

[5] Emmanuel Candes, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Inform. Theory*, 2006.

[6] David L Donoho et al. Compressed sensing. *IEEE Trans. Inform. Theory*, 2006.

[7] 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.

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

[9] 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.

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

[11] 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.

[12] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. *Proc. Int. Conf. Machine Learning*, 2017.

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

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

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

[16] 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.

[17] Renato Negrinho and Geoff Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.

[18] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *Proc. Int. Conf. Learning Representations*, 2018.

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

[20] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *Proc. Int. Conf. Learning Representations*, 2019.

[21] 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.

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

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

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

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

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

[27] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning. *Adv. Neural Inf. Proc. Sys. Workshop (NeurIPS Workshop)*, 2018.

[28] 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.

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

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

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

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

[33] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

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

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

[36] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *Proc. Int. Conf. Learning Representations*, 2018.

[37] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: A high-rank rnn language model. *Proc. Int. Conf. Learning Representations*, 2018.

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

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

[40] 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.

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

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

[43] Ishay Haviv and Oded Regev. The restricted isometry property of subsampled fourier matrices. *Geometric Aspects of Functional Analysis*, pages 163–179, 2017.

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

[45] Joel A Tropp and Anna C Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Trans. Inform. Theory*, 2007.

A Appendix

A.1 Detailed Background of One-Shot Neural Architecture Search

In this section, we explain the one-shot neural architecture search in more details regarding of three directions introduced by [30].

Search Space. First, we start with the search space, defining the principles of constructing neural architecture. As discussed before, simplifying the search space using human prior knowledge can help search algorithms to find an optimal candidate faster. However, it may limit the algorithm to find a novel architecture beyond our knowledge due to human bias.

Search Strategy Next, we have a search strategy which is considered as a methodology to find the best neural architecture. In the last two years, different methodologies for search space such as reinforcement learning, evolutionary algorithms, SMBO, Bayesian optimization, bilevel optimization, and randomness are introduced in NAS literature. For instance, one-shot architecture uses random search as a search strategy.

Performance Estimation Strategy. The last direction aims to reduce the computational cost efficiently by estimating the validation performance. The conventional performance estimating methods require the massive computational resources as each model need to be trained and evaluated separately. In particular, assume that our goal is to construct a CNN by using a proxy architecture through randomly selected cell samples by concatenating eight layers with 16 initial channels (This is the same proxy model setup in DARTs). Then, we measure the validation loss from 50 epochs trained model where each epoch takes five minutes of training. As a result, the total sampling time to collect 4,000 randomly sampled of the architecture performance (which is the same number of measurements as our experiment using CoNAS) equals to $5 \cdot 50 \cdot 4000 \cdot \frac{1}{60} \cdot \frac{1}{24} \approx 694$ days.

A.2 Theoretical Support for CoNAS

A Boolean function f defined over a binary n-bit input may expresses using 2^n polynomial basis functions (which is intractable). As discussed in [13], instead, we can focus on all the polynomial basis with degree at most d, that is, we have $|\mathcal{P}_d| \equiv \mathcal{O}(n^d)$ basis (parity) functions. On the other hand, the system of linear equations $\mathbf{y} = \mathbf{A}\mathbf{u}$ 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 $\mathbf{u} \in \mathbb{R}^{O(n^d)}$ is an ill-posed problem when $m \ll O(n^d)$ for large n. While the underdetermined system of linear equations is an ill-posed, it has been shown that if the graph-sampling matrix satisfies *Restricted Isometry Property (RIP)*, the sparse coefficients, $\mathbf{u}$ can be recovered.

We recall the definition of RIP:

Definition A.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:*

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

Now, we assume that the Fourier support of f is s-sparse. In addition, in our case, the columns of graph-sampling matrix $\mathbf{A}$ (the atoms of dictionary $\mathbf{A}$) include the parity function. Hence, the columns of matrix A form a bounded orthonormal system (BOS), i.e., $\sup_{\boldsymbol{\alpha} \in \{-1,1\}^n} |\chi_S(\boldsymbol{\alpha})| \leq 1$ for all $S \in [n]$.

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 [38], 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 [39] 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 [38] was further improved by [40] to $\mathcal{O}(sd \log^2 s \log(sd \log n) \log n)$. Motivated by this result, [41] 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 [42]). To the best of our knowledge, the best known result with mild dependency on δ (δ^{-2}) is due to [43], which we can invoke for our setup. In our case, the graph-sampling matrix $\mathbf{A}$ in our proposed CoNAS algorithm satisfies BOS for $K = 1$ (equation 3.1).

Theorem A.2. *Let 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 normalized matrix $\mathbf{A}$ (multiplied by $\sqrt{\frac{\mathcal{O}(n^d)}{m}}$) 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 $\mathbf{u} \in \mathbb{R}^{\mathcal{O}(n^d)}$ is recovered from the samples*

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

by LASSO (equation 3.2). It is worthwhile to mention that we can use any other sparse recovery method such as IHT [44], or OMP [45] in our algorithm.

Therefore, Theorem A.2 leads to successfully guarantee approximation of the function using the Boolean Fourier basis given the sufficient number of measurements.

A.3 Effect of Multiple Stages Sampling

Now we talk briefly about the advantage of using multiple stages in CoNAS. As mentioned in the algorithm pseudocode, after each stage of recovery, the algorithm uses multiple stages of recovery process to find the final architecture. This is similar to the idea of de-biasing step in *Hard-Thresholding (HT)* algorithm [39] introduced in the sparse recovery literature where first the support is estimated and then within the estimated support the sparse coefficients are calculated through a least-square computation. However in our setup, we have to solve a non-linear optimization problem to get better estimation of the sparse Fourier coefficients. For instance, in Table 4, we have illustrated an experiment which shows that the sparse recovery process with multiple stages can discover the important edges; as a result to find a more accurate architecture. In this experiment, we have sampled 1000 measurements and computed the mean and the standard deviation of validation loss/perplexity for each stage. As we can see, the larger number of stages, the smaller loss/perplexity.

Table 4: Changes in Measurements for Each Stage.

Architecture Type	Stage 1	Stage 2	Stage 3	Stage 4
CNN (Valid loss)	8.21 ± 26.76	3.01 ± 8.48	2.48 ± 3.55	2.37 ± 1.13
RNN (Valid perplexity)	340.65 ± 239.84	207.30 ± 56.74	-	-

A.4 Architecture Found from Our Experiment

A.4.1 Compressive sensing-based Neural Architecture Search (CoNAS) for RNN

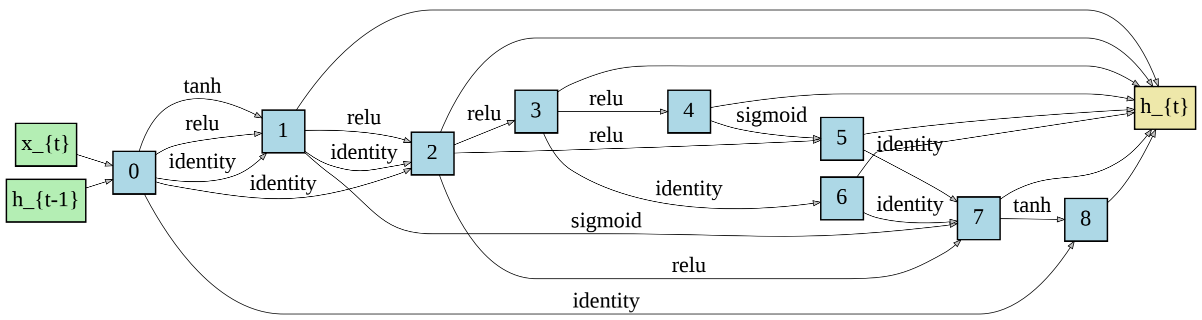
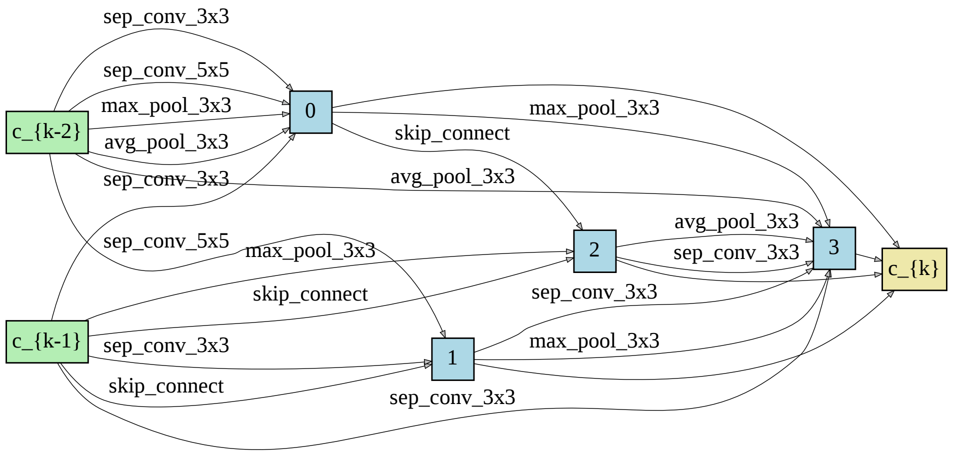
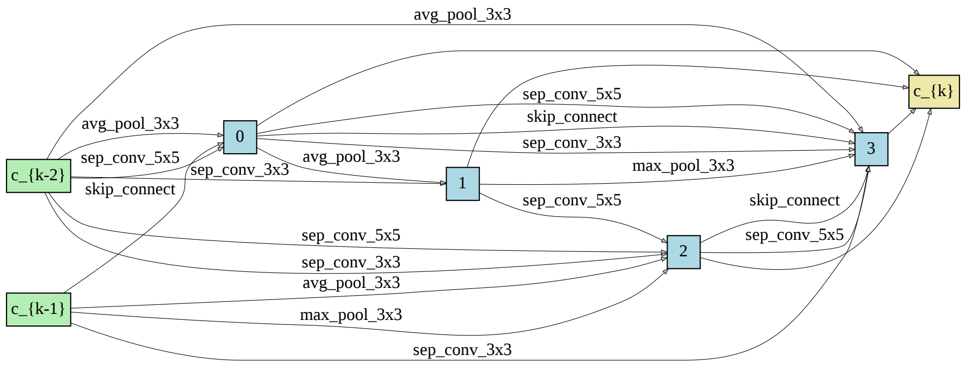


Figure 4: Recurrent Cell found from CoNAS

A.4.2 Random Search with Weight-Sharing (RSWS) for CNN



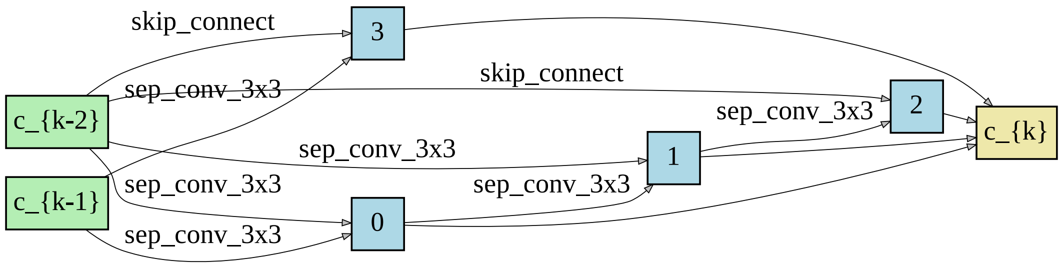
(a) Normal Cell



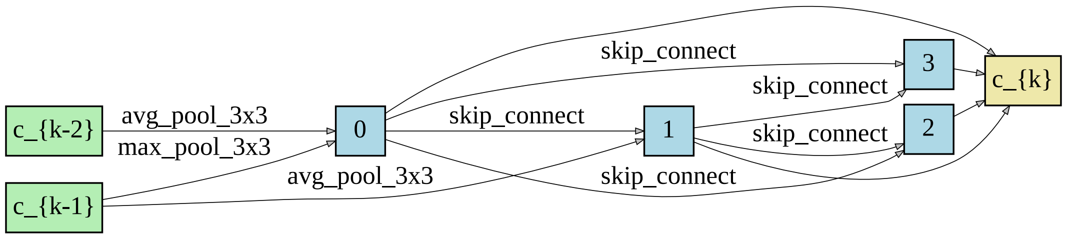
(b) Reduce Cell

Figure 5: Convolutional cell found from RSWS with search space in Section 3

A.4.3 Differentiable Neural Architecture Search (DARTs) for CNN



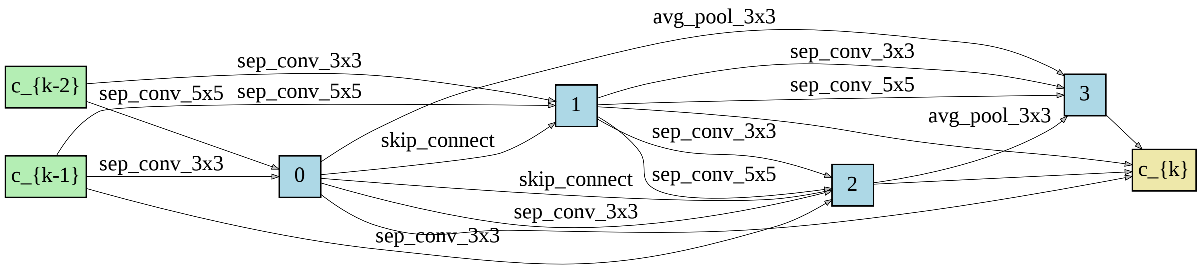
(a) Normal Cell



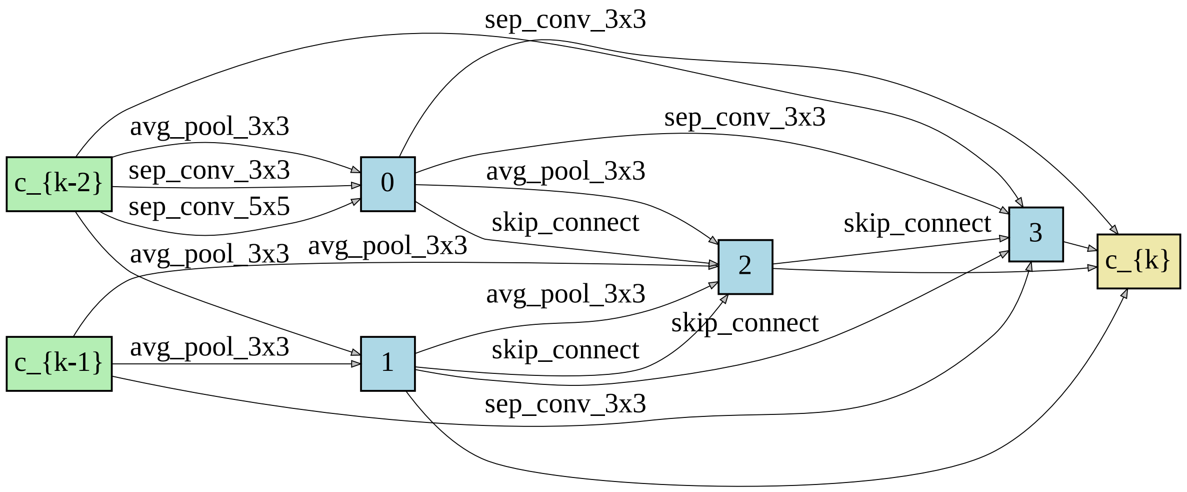
(b) Reduce Cell

Figure 6: Convolutional Cell found from DARTs with the original setting in [8].

A.4.4 Randomly Wired CNN Architectures

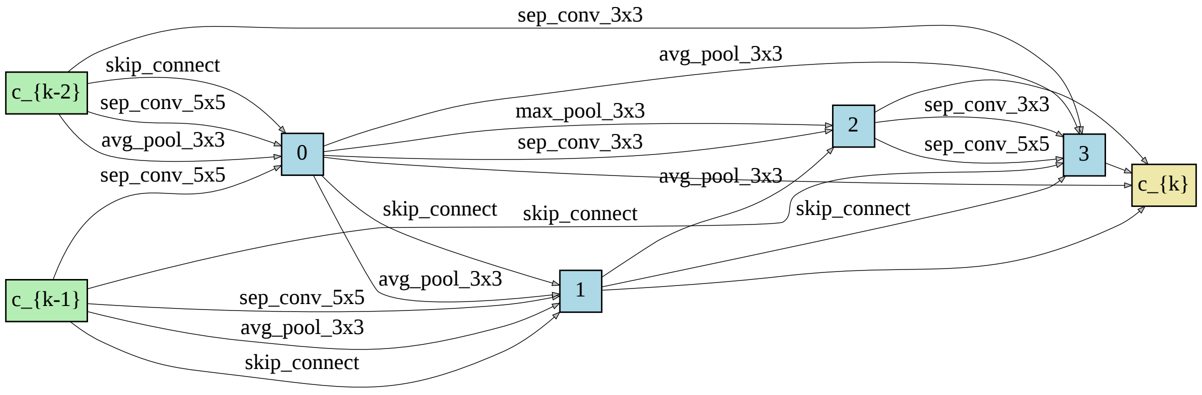


(a) Normal Cell

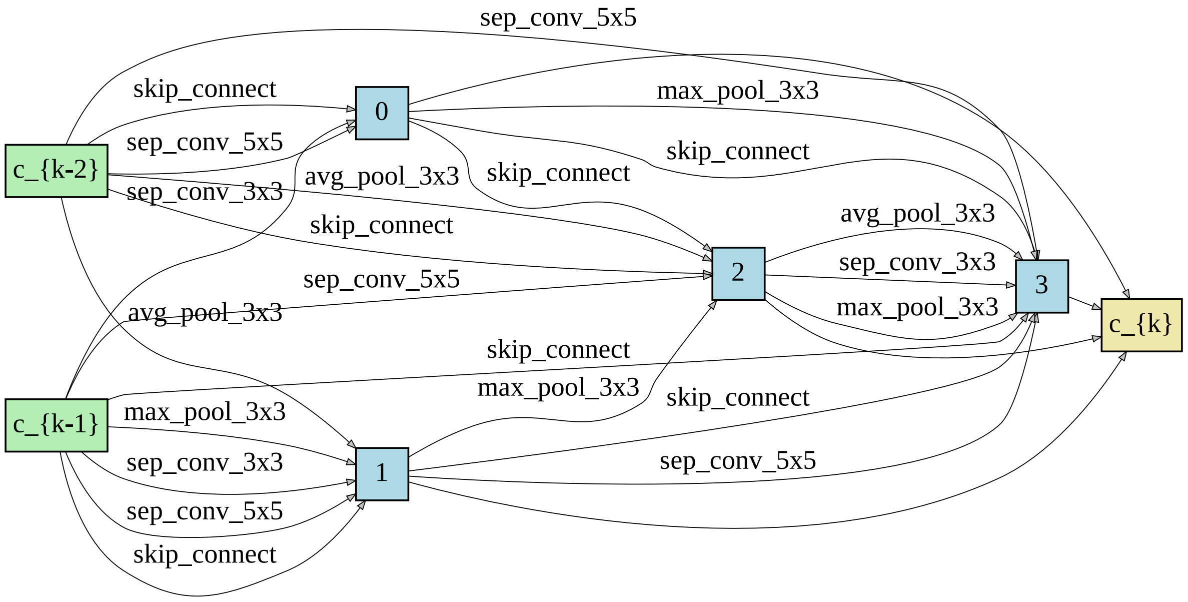


(b) Reduce Cell

Figure 7: Randomly wired cell corresponding to the first row result to Table 2.

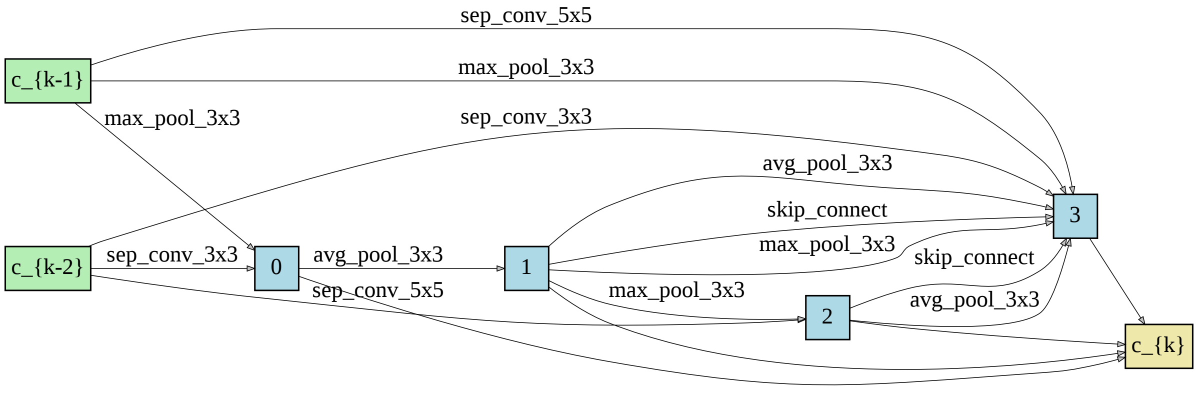


(a) Normal Cell

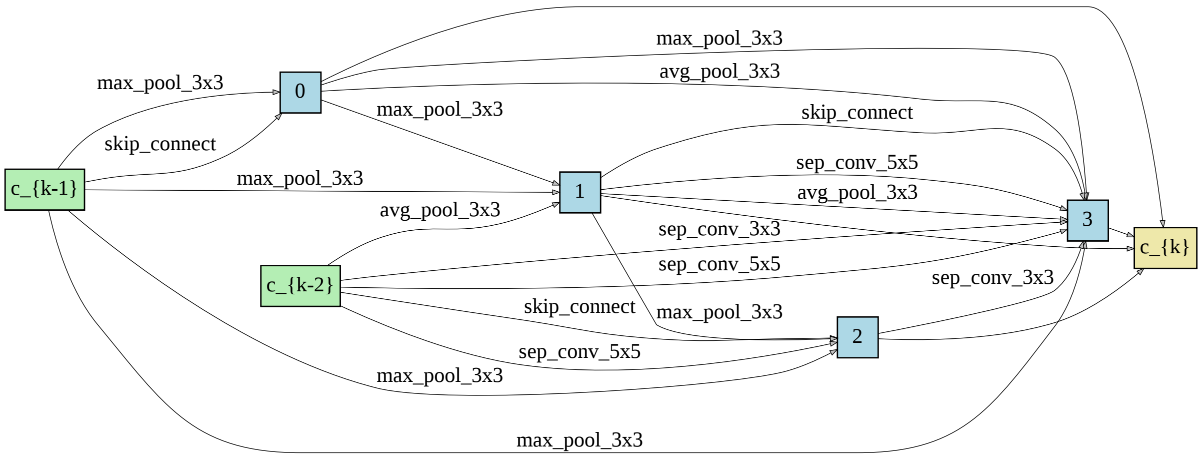


(b) Reduce Cell

Figure 8: Randomly wired cell corresponding to the second row result to Table 2.



(a) Normal Cell



(b) Reduce Cell

Figure 9: Randomly wired cell corresponding to the third row result to Table 2.

A.4.5 Random Search with Weight-Sharing (RSWS) for RNN

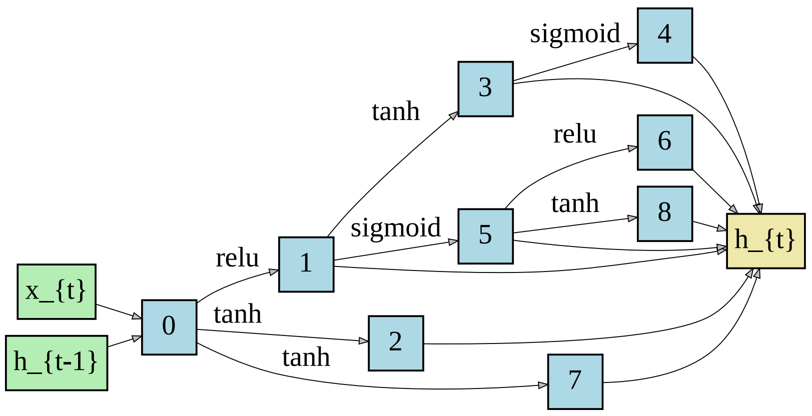


Figure 10: Recurrent cell found from RSWS allocating equal amount of search time to CoNAS