CONet: Channel Optimization for Convolutional Neural Networks

Mahdi S. Hosseini[1]*, Jia Shu Zhang[2]*, Zhe Liu[2]*, Andre Fu[2]*, Jingxuan Su,
Mathieu Tuli and Konstantinos N. Plataniotis [7]
[1]The Department of Electrical and Computer Engineering, University of New Brunswick
[2]The Edward S. Rogers Sr. Department of Electrical & Computer Engineering, University of Toronto
https://github.com/mahdihosseini/CONet

Abstract

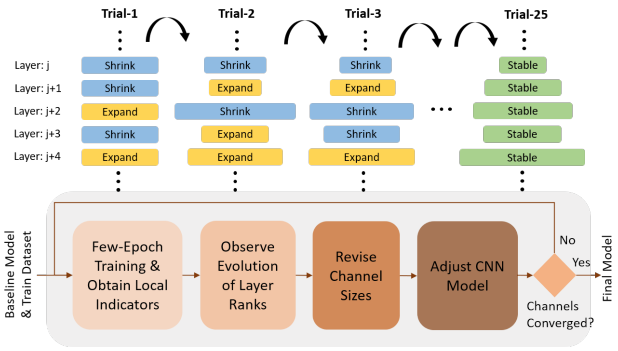
Neural Architecture Search (NAS) has shifted network design from using human intuition to leveraging search algorithms guided by evaluation metrics. We study channel size optimization in convolutional neural networks (CNN) and identify the role it plays in model accuracy and complexity. Current channel size selection methods are generally limited by discrete sample spaces while suffering from manual iteration and simple heuristics. To solve this, we introduce an efficient dynamic scaling algorithm – CONet – that automatically optimizes channel sizes across network layers for a given CNN. Two metrics – "Rank" and "Rank Average Slope" – are introduced to identify the information accumulated in training. The algorithm dynamically scales channel sizes up or down over a fixed searching phase. We conduct experiments on CIFAR10/100 and ImageNet datasets and show that CONet can find efficient and accurate architectures searched in ResNet, DARTS, and DARTS+ spaces that outperform their baseline models.

1. Introduction

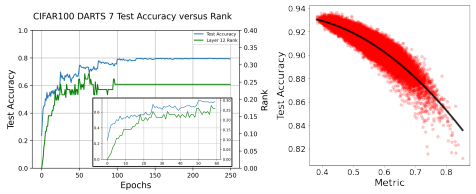
In the midst of Neural Architecture Search (NAS) algorithms [50, 51, 34, 39, 23, 46, 37, 38, 2], the generated CNN are poorly optimized due to the lack of automated scaling of channel sizes (aka the number of filters in a convolution layer). Given a dataset and a model architecture, the selection of proper channel sizes has a direct impact on the network's success. The over-parameterization of a network can (a) further complicate the training process; and (b) generate heavy models for implementation, while the under-parameterization of a network yields low performance.

Few studies show the importance of channel scaling in architecture search. Searching techniques in [36, 32, 2, 38] use a semi-automated pipeline to generate sub-optimal block architectures to fit training data. Their use of heuristics and associated hyper-parameters to search for channel size blocks them from full automation. In this paper we question whether the generated architectures are optimized to obtain a better trade-off between performance, accuracy and model complexity.

We address the above limitations by introducing a fully automated channel optimization algorithm – *"CONet"* – to search for optimal channel sizes in a network architecture. See Fig. 1(a) for the overall workflow. The proposed method depends auxiliary indicators that locally probe in-



(a) CONet Overview



(b) DARTS7 Layer 12 (c) Test Acc vs. Rank

Figure 1: (a) CONet Overview. The algorithm begins with a *small* baseline model and trains a few epochs using the train dataset. Every channel dynamically shrinks/expands based on the rank evolution. This is repeated until all channel sizes have stabilized. (b) We randomly inspect the 12th layer of DARTS7 and over 250 epochs with a zoomed version to 60 epochs in bottom right; (c) Test accuracy vs. Rank on CIFAR10 for the NATS-Benchmark [10]. Probing each layer in aggregate allows us to correlate rank with accuracy, allowing optimization of rank as a proxy for test-accuracy.

*Equal contribution

termediate layers of cell architectures and measure the utility of learned knowledge during training. Our metrics employ low-rank factorization of tensor weights in CNN and compute; (a) *rank* to determine the well-posedness of the convolution mapping space; (b) *condition* to compute the numerical stability of layer mapping with respect to minor input perturbations; and (c) *rank-slope* to compute the dimension growth of convolution weights. The rank-slope is measured after a few training epochs to evaluate how well a convolution layer is learning. We translate this into a decisive action of shrinking or expanding the channel size for the next evolving architecture. CONet is able to dynamically scale baseline models to achieve higher top-1 test accuracy with minimal parameter count increase, and in some cases, a reduction in parameter count.

The summary of our contributions is listed as follows:

1) Metric Development. We develop new metric tools that can probe intermediate layers of CNN architectures as a strong response function for channel size selection and measure how well the layers are training.

2) Independent Channel Allocation. We design an automated algorithm to find independent connections in CNN and show how individual convolution layers can be adjusted independent of its neighbouring blocks.

3) CONet. We propose channel optimization method: CONet and show how the algorithm dynamically scales channel sizes to convergence.

4) Experiments. Comprehensive experiments are conducted across three architecture spaces (i.e. ResNet [14], DARTS [23] and DARTS+ [21]) and three datasets (CIFAR10, CIFAR100, and ImageNet) and show how the original architectures are dynamically scaled to achieve superior performances compared to their baselines.

1.1. Related Works

Hand-Crafted Networks. Hand-crafted networks are heuristically defined based on expert domain knowledge. Examples include VGG [31], ResNet [14], Inception [33], ResNeXt [41], and MobileNet [30]. The rule-of-thumb is to increase the channel size by increasing the network depth layers for better image class representation.

Neural Architecture Search (NAS). Automatic NAS was first introduced in [50] to generate efficient (child) networks that achieve the performance of hand-crafted models. NAS is mainly guided by recurrent neural networks (RNN) and trained with Reinforcement Learning (RL) – known as NAS-RL. Examples are transfer learning method from NASNet [51], model latency incorporation from MnasNet [34], Q-Learning methods used in [1, 48], and child node weight-sharing used in [27, 45, 39]. All of these methods sample the channel size per layer from a discrete sample set.

Differentiable NAS (DNAS). A drawback of previous methods is that they search through a discrete space (which is limiting) and tend to take several hundreds to thousands of GPU hours to search. Differentiable Architecture Search (DARTS) [23] was recently introduced to relax this search space by defining auxiliary variables over cell operations and optimizing the architecture design through a bilevel optimization problem. Variants of DARTS have been proposed including PC-DARTS [43] to remove redundant searches, Robust-DARTS [46] to stabilize the search space, Fair-DARTS [6] to relax the exclusive competition between skip-connections, SNAS [42] that formulates a joint distribution parameter within a DARTS cell, UNAS [37] that unifies DARTS with RL, Progressive-DARTS [5] that progressively searches the network depth, and SGAS [19] that divides the search procedure and applies pruning. These methods start with a heuristic selection of fixed channel sizes and scale up after each reduction cell. None-DARTS DNAS method also exist, including FBNetV2 [38] and Single-Path NAS [32], however they suffer from similar drawbacks as DARTS, although Single-Path NAS does improve in search time. Overall, the major drawback of DNAS is the requirement of expert domain knowledge for tuning hyper-parameters for training.

Other methods. Pruning methods exist, including TAS [11], DF [20], OFA [2], AtomNAS [24], and ASAP [26]. One major drawback of the above pruning methods is that the initial large model requires extensive computational resources for training. Network search based on evolutionary algorithms also exist, include binary encoding used in GeNet [40], mutation algorithms for large-space searching used in Large Scale Evolution [29], mutation algorithms applied to NASNet used in AmoebaNet-A [28], and RL applied to evolutionary algorithms used in FPNAS [8]. In each of these methods, channel sizes follow manual selection heuristics.

2. Local Indicators for CNN

Central to our approach are three metrics developed to probe intermediate layers of CNN and evaluate the well-posedness of the trained convolutional weights.

2.1. Low-Rank Factorization

Consider a four-way array of convolution weight $\mathbf{W} \in \mathbb{R}^{N_1 \times N_2 \times N_3 \times N_4}$, where N_1 and N_2 are the kernel height and width, and N_3 and N_4 are the input and output channel sizes, respectively. In convolution, the input features $\mathbf{F}^I \in \mathbb{R}^{H \times W \times N_3}$ are mapped into an output feature map $\mathbf{F}^O_{:,:,i_4} = \sum_{i_3=1}^{N_3} \mathbf{F}^I_{:,:,i_3} * \mathbf{W}_{:,:,i_3,i_4}$, where $\mathbf{F}^O \in \mathbb{R}^{H \times W \times N_4}$. The convolution mapping acts as an encoder, and therefore we can prob the encoder to understand the structure of weights for convolution mapping. This is useful to understand the evolution of its structure over iterative training. To achieve this, we unfold the tensor into a two-dimensional matrix along a given dimension d

$$\mathbf{W}\texttt{[Tensor-4D]} \xrightarrow{\text{unfold}} W_d\texttt{[Matrix-2D]}.$$

For instance, the unfolded matrix along the output dimension i.e. $d = 4$ is given by $W_4 \in \mathbb{R}^{N_4 \times N_1 N_2 N_3}$. The weights in CNN are initialized (e.g. random Gaussian) and trained over some epochs, generating meaningful structures for encoding. However, the random perturbation inhibits proper analysis of the weights during training. We decouple the randomness to better understand the trained structure. Similar to [18], we obtain the low-rank structure

$$W_d \xrightarrow{\text{factorize}} \widehat{W}_d + E,$$

where, $\widehat{W}_d$ is the low-rank matrix structure containing limited non-zero singular values i.e. $\widehat{W}_d = U\Sigma V^T$, where $\Sigma = \text{diag}\{\sigma_1, \sigma_2, \dots \sigma_{N'_d}\}$ and $N'_d = \text{rank}\,\widehat{W}_d$. Here $N'_d < N_d$ due to the low-rank property. We then employ Variational Baysian Matrix Factorization (VBMF) method from [25] which factorizes a matrix as a re-weighted SVD structure. The method is computationally fast and can be easily applied to several layers of arbitrary size. Figure 2 demonstrates the low-rank factorization on a convolution layer of ResNet34 during two different training phases i.e. first epoch and last epoch. Notice how the low-rank structure (aka useful information) is evolved while the noise perturbation is faded at the end; leading to a stabilized layer.

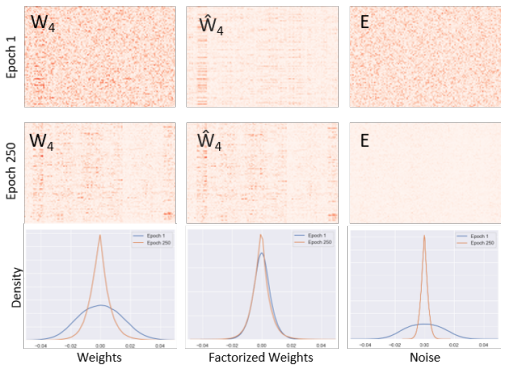


Figure 2: Low-rank factorization on a ResNet layer during first and last epoch training.

Low-Rank Measure. We define the first probing metric, the *rank*, by computing the relative ratio of the number of non-zero low-rank singular values to the given channel size

$$R(\widehat{W}_d) = N'_d / N_d. \tag{1}$$

The rank in Eq. 1 is normalized such that $R(\widehat{W}_d) \in [0, 1]$ and it measures the relative ratio of the channel capacity for convolutional encoding. Note that the unfolding can be done in either input or output dimensions i.e. $d \in \{3, 4\}$ and therefore each convolution layer yields separate input/output rank measurements i.e. $R(\widehat{W}_3)$ and $R(\widehat{W}_4)$.

Low-Rank Condition. We adopt a second probing metric, the *low-rank condition*, to compute the relative ratio of maximum to minimum low-rank singular values

$$\kappa(\widehat{\boldsymbol{W}}_d) = \sigma_1(\widehat{\boldsymbol{W}}_d) / \sigma_{N'_d}(\widehat{\boldsymbol{W}}_d). \tag{2}$$

The notion of matrix condition in Eq. 2 is adopted from matrix analysis [15] and measures the sensitivity of convolution mapping to minor input perturbations. If the condition is high for a particular layer, then the input perturbations will be propagated to the proceeding layers leading to a poor auto-encoder.

2.2. Low-Rank Evolution

While both Rank (Eq. 1) and Condition (Eq. 2) are useful metrics to observe a layer's learned knowledge compared to other layers, they are less sensitive to the selection of channel size. We observe the evolution of the metrics over training, yielding meaningful insights into how channel size selection impacts the convergence of the metric. Shown in Fig. 3, we train the network and observe the rank averages for different channel sizes of the same layer index (i.e. we vary channel size). Notice how the rank slope is inversely related to the increase in channel size.

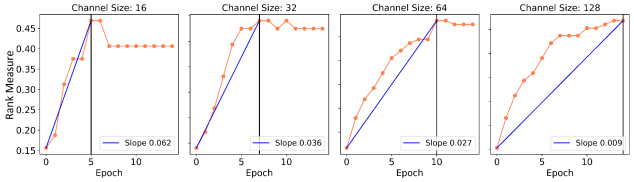


Figure 3: Evolution of rank slopes on different initial channel sizes: increasing the channel size, decreases the slope.

We define the *rank-slope* by computing the relative ratio of the rank gain to the number of epochs taken to plateau

$$S = \frac{\overline{R}(\widehat{W}_d^{t_2}) - \overline{R}(\widehat{W}_d^{t_1})}{t_2 - t_1}, \tag{3}$$

where, $\overline{R} = [R_3 + R_4]/2$ is the average rank, t_1 the starting epoch, and t_2 is the rank plateau epoch. The slope, S in Eq. 3 encodes the complexity of the channel size selection: (a) how much rank (knowledge) is gained within a layer; and (b) the amount of time to reach the maximum rank capacity –indicating a target for channel size selection.

3. Network Connectors

While we can measure the well-posedness of each layer, we are often unable to individually alter channel sizes. This is because changing the channel size of a layer affects the depth of its feature map, which impacts subsequent layers. E.g given a pair of conv-layers in series, the output

channel size of the first layer must match the input channel size of the second layer. We represent a network as a directed acyclic graph (DAG) to identify these constraints. The nodes are layer *depth* (aka *channels*). The edges are operators on the feature maps between the tail & head.

3.1. Connection Types

In Fig. 4 (a), we categorize the edges into two types: *conv* and *non-conv*. *Conv* edges represents convolution operations and *non-conv* edges represents other operations e.g skip-connection and max-pooling. There are 2 distinctions between the these edge types. (1), only conv edges can connect nodes with different depths. Non-conv edges must have the same depth for the input and output nodes. (2), we measure rank from conv edges to adjust the channel sizes of a network.

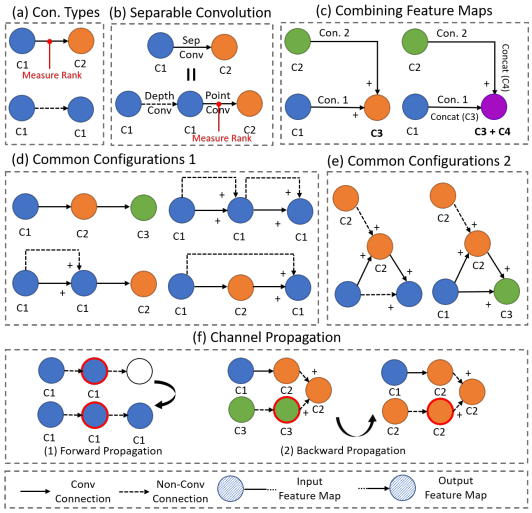


Figure 4: Overview of local connectors in a typical CNN architectures to identify independent channels for probing: (a) Elementary connections; (b) Rank measure on separable convolution; (c) Summation and Concatenation of feature maps; (d) Common configurations within residual networks; (e) Common configuration within inception/cell-based networks; (f) Propagation of Channel Constraints.

We further define an edge is *inbound* to a node if it is connected at the head of the edge. The edge is *outbound* to a node if it is connected at the tail of the edge. In most cases, the depth of the node determines the input channel size for all outbound edges and the output channel size for all inbound edges. The exception is the case for concatenated nodes, which will be described later.

Separable Convolutions. A separable convolution is comprised of a depthwise layer and pointwise layer in series. The depthwise layer contains a filter for each channel of the input feature map. Each filter has a depth of 1 and convolves only with a single channel. The output from each filter is concatenated and passed to the pointwise layer. The pointwise layer contains 1x1 kernels that combine the concatenated output along the depth dimension.

As show in in Fig. 4 (b), we treat the depthwise layer as a non-conv edge. This is because the input and output channel sizes for a depthwise layer must be the same since each filter only process a single channel from the input to produce a single channel in the output. Furthermore, we found that the rank of depthwise layers behaves unpredictably because each filter only has a depth of 1. We use the ranks of the pointwise layer to adjust the entire separable convolution.

Summation and Concatenation. When multiple edges are inbound to the same node, the output from each edge must be combined. The two ways of combining feature maps are *summation* and *concatenation*. Fig. 4 (c) demonstrates the effect on channel sizes for each method. For summation, the output channel size of each edge must be the same to avoid dimension mismatch. For concatenation, the output channel size of each edge can be different. The final depth of the concatenated node will be the total output channel size across all inbound edges.

Example 1. The configurations shown in Fig. 4 (d) are commonly found in network architectures that uses repeating skip-connections such as ResNet [14] and DenseNets [16]. The configurations show that skip-connections can propagate channel size constraints across the network.

Example 2. The configurations shown in Fig. 4 (e) are commonly found in inception [14] and cell-based [28] [22] network architectures. One simple way of identifying if the depth of a node is independent is by checking if all inbound edges are conv connections.

Algorithm 1 Unique Channel Size Assignment

Input: DAG Network
Output: Unique channels

Initialize *node* as unassigned ∀ nodes
Store nodes in *queue* in partial order // Sec. 3.2
for *node* **in** *queue* **do**
  if *node* = unassigned **then**
    if *summation* **then**
      channel ← unique size
    else if *concatenation* **then**
      channel ← sum of *inbound edges*
  for *edge* **in** *outbound edges* **do**
    if *edge* ≠ *conv* **then**
      if *connected node* = unassigned **then**
        Forward Propagate channel //Fig. 4 (f.1)
      else
        Back Propagate channel //Fig. 4 (f.2)

3.2. Channel Size Constraints

Using the DAG network representation outlined in Sec. 3, we can systematically identify channel size constraints for any CNN. We revisit each node in the network sequen-

tially and propagate any channel size constraints caused by non-conv connections throughout the network. If a node has no inbound non-conv connections, then that means it has no channel size constraints with respect to the previous nodes. The overall methodological flow is presented in Alg. 1. The steps of the flow is also described below.

Initializing the Network. We initialize all network channels as *unassigned*. We then queue the nodes of the network in partial order. For any node in the queue, its outbound edges can only connect to nodes later in the queue.

Assigning Channel Size for a Node: If the channel of a node is unassigned when we pull the node from the queue, we can assign new unique channel sizes to the network. Having an unassigned node implies that all inbound edges are conv connections. If the inbound edges are summed at the node, then we can set a new output channel size shared by all inbound edges. If the inbound edges are concatenated at the node, then each inbound edge can have an independent output channel size.

Resolving Channel Size Conflicts: We propagate the channel size of the current node to subsequent nodes connected by outbound non-conv edges. If we find that the connected node is already assigned, then we must back-propagate that channel assignment through all non-conv edges. See Fig. 4 (f.2) for an illustration of this scenario.

4. CONet: A Dynamic Optimizing Algorithm

4.1. Important Variables

R	List of input and output ranks for each layer.
κ	List of input and output Mapping condition.
$\overline{R}$	Rank averages.
$\overline{\kappa}$	Mapping condition averages.
S	Rank average slopes.
δ	*Rank average slopes threshold.*
μ	*Mapping condition threshold.*
C_ℓ^o	Old channel sizes, per layer (ℓ).
C_ℓ^n	New channel sizes, per layer (ℓ).
ω_ℓ	Last operation list, per layer (ℓ).
ϕ	Scale factors of channel sizes.
γ	*Stopping condition for scaling factor.*

Table 1: Important Variables referenced in Alg. 2. A full list of all variables and their annotations can be found in the supplementary materials. Italicized variables are the 3 key hyper-parameters outlining this algorithm.

We label some key variables used for Alg. 2 in Table 1 and highlight the following: (1) δ - threshold rank average slopes, a new hyper-parameter we are introducing, allowing the network to autonomously search with 1 key hyper-parameter, Sec. 4.2. (2) S - rank average slope (Eq. 3) - allowing probing of how well R is increasing (i.e. how well the model is learning). Finally, (3) ϕ - Channel size factor scale, allows the network to continuously alter the channel sizes until convergence.

4.2. Channel Shrinkage/Expansion

Calculation of rank average slopes: We calculate the rank average slopes $S(\ell)$ (Eq. 3) for every convolutional layer, ℓ, as described in Section 2.2. This slope represents the rate at which R grows as the layer learns.

Criteria for shrinking or expanding channel sizes: Using rank average slopes $S(\ell)$ we define a target threshold slope, δ. If $S > \delta$ we expand the channel size as it indicates R is growing quickly and as such would benefit from a larger channel size, we then assign the action $\omega_\ell = +1$ for expansion. If $S < \delta$ we shrink the channel size as it indicates R is not increasing rapidly so additional channel sizes mean an increase in parameters that do not contribute to model performance. We then assign the action $\omega_\ell = -1$ for shrinking. If $S = \delta$ or a no-operation decision was taken we assign $\omega_\ell = 0$.

Calculation of new channel size: After a decision to shrink or expand the channel size has been determined, we change the scale the channel size by a factor ϕ_ℓ. Initially we set this scale $\phi_\ell \leftarrow \lfloor 20\% \rfloor$. The new channel size C_ℓ^n are then computed as a factor scale of the old channel size C_ℓ^o.

$$C_\ell^n = C_\ell^o(1 + \omega_\ell \cdot \phi_\ell) \tag{4}$$

At the end of every trial a new channel size for every layer is calculated and implemented. If the channel size begins to oscillate between C_ℓ^n and C_ℓ^o it indicates the optimal channel size is between the two channel sizes and so we half the factor scale continuously until channel size converges and a no-operation decision is taken.

4.3. CONet Algorithm

Intuition & Overview: CONet algorithm[1] shown in Alg. 2, begins with a small network using standard channel sizes (e.g 32) for all layers in the network. We begin a trial, training on the training-dataset, for a predefined number of epochs while recording the average rank per epoch, see Fig. 3. After the trial ends, we stop training, and analyze the rank average slope, S comparing the slope for that layer to our hyper-parameter δ. As per Sec. 4.2 we then take a shrink or expand decision and calculate a new channel size per Eq. 4. We continuously execute trials until a predefined number of trials is reached where the channel sizes have converged. By only analyzing S growth, our method never sees the test dataset, thereby optimizing solely on information gained from training data.

[1] A PyTorch package is submitted in the supplementary materials.

Algorithm 2 Channel Size Optimizaion

Input: Network, hyperparameters $\delta, \gamma, \mu, \phi$
Output: Optimized Network

for trial **in** trials **do**
  Train current model for a few epochs
  Compute average rank ($\overline{R}$) per layer // Eq. 1
  Compute average MC ($\overline{\kappa}$) per layer // Eq. 2
  From $\overline{R}$ get rank-slope (S) per layer // Eq. 3
  for layer **in** network **do**
    Set action to Expand
    if S[layer] $< \delta$ or $\overline{\kappa}$[layer] $> \mu$ **then**
      Set action to Shrink
    if Last action $\neq$ current action **then**
      if $\phi < \gamma$ **then**
        Stop, channel sizes have converged
      $\phi \leftarrow \phi$ /2
    Calculate New Channels // Eq. 4
  Instantiate model with new channels

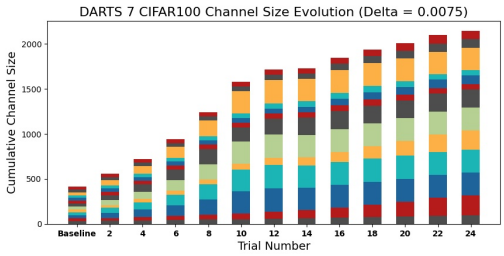


Figure 5: Channel size evolution example of DARTS7 using CONet algorithm on CIFAR100. Channel sizes from only the last cell are shown for better visualization. Specific channel sizes may expand or shrink but the network as a whole stabilizes by the last trial.

Delta threshold tuning: We can vary the model parameter count and accuracy trade-off through the δ_S hyperparameter. Once an initial δ is chosen, test values around it and observe the performance change. The δ values we tested are listed under Sec. 5.1.

Epochs & Trials: The number of epochs per trial is given from number of epochs to rank convergence. For DARTS and DARTS+, we recommend 20 epochs per trial is optimal for convergence. The number of trials is given from number of trials to channel size convergence. For DARTS and DARTS+ 25 trials is optimal for convergence. Results using these hyper-parameters have been described in Sec. 5.1.

Algorithm channel size updates: The algorithm collects the metrics from the first trial, then computes S for each layer (Sec. 4.2). The decision to shrink or expand the channels are given in Sec.4.2. The ability to expand and shrink the channel sizes dynamically is a key advantage over other methods that only scale up [35] with no consideration of the accuracy-parameter trade-off. The algorithm then returns the new channel sizes, C_ℓ^n.

5. Experiments

Experiment Setup. We complete experiments on ResNet, DARTS7 (7 cells), DARTS14 (14 cells), DARTS+7 and DARTS+14. Each model is applied to CIFAR10, CIFAR100 & ImageNet [17, 9]. All hyper-parameter choices and further implementation details can be found in Supplementary material.

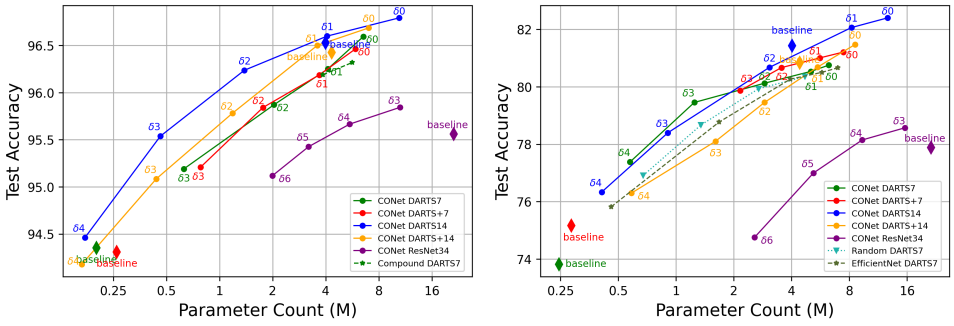
Channel Searching. For the chosen search spaces, we perform a channel search for each model and respective (training set) datasets, with the following thresholds: $\delta = \{\delta_0, \delta_1, \delta_2, \delta_3\}$, and for DARTS14 and DARTS+14, $\delta = \{\delta_0, \delta_1, \delta_2, \delta_3, \delta_4\}$. Each channel search runs for 25 trials each with 20 epochs per trial.

Model evaluation. We evaluated both, the CONet optimized model and unaltered baseline model on CIFAR10/100. For the ImageNet dataset, we use the DARTS14 model searched based on CIFAR100 with $\delta = \{\delta_2, \delta_3\}$ and the baseline model of DARTS14 from the DARTS paper[22]. The baseline models were run on 2 Nvidia RTX 2080Tis while the other SOTA results were taken from their respective papers.

5.1. Results

We implement CONet within the DARTS, DARTS+ and ResNet search spaces. We compare our method to the Efficient Net scaling algorithm [35] and show that the ability to dynamically scale up and down the channel sizes has merit over simple scaling factors as it doesn't consider knowledge growth, redundant channel sizes & the parameter-accuracy trade-off. There are two phases per experiment, (1) the channel searching phase, searched on training data, and (2) the dataset evaluation phase, evaluated on test data. We conduct searching on CIFAR10 and CIFAR100 and show architecture transferability for ImageNet training.

Parameter-Accuracy Trade-off. As per Sec.2 the rank slope indicates efficacy of layer learning. We can then assign a threshold, δ to inform our choice of rank-slope. In choosing δ, CONet considers the information gain compared to the number of additional channels, implicitly balancing accuracy gain and model size. A small δ yields larger parameter counts, and larger δ yields smaller parameter (param) counts. As in Table 3, DARTS14 searched on CIFAR100 with δ_2 reduces the param-count 3.1M while using δ_0 increases the param-count to 12.7M. In particular, ResNet CIFAR10 with δ_2 maintains a similar accuracy, while reducing param-counts from 21.28M to 5.45M, representing a **4x reduction**. This demonstrates CONet's ability to remove redundant channel sizes without losing performance. Furthermore, DARTS7 achieves 6.94% accuracy gain with δ_6 compared to the baseline, indicating the ability of CONet to further improve the performance of the architecture through dynamic channel sizes adjustment.



(a) CIFAR10 (b) CIFAR100

Figure 6: Testing accuracy vs. parameter count for CONet experiment applied on (a) CIFAR10 and (b) CIFAR100. Note that the baselines results are plotted as a diamond scatter plot, and sequential deltas for the CONet are plotted in line graphs with the annotated delta thresholds. Also, the models produced by compound scaling are plotted in dashed line graphs.

Ablation Study of Hyperparameters (HP): We highlight that the δ is the main HP for searching CONet. The rest of HPs fall in to three categories: (1) Convergence (i.e epochs-per-trial, total trials, & ϕ), decreasing these HPs will produce an "early" result; (2) Limits: bound ranges for parameter counts; and (3) Placeholder values (γ, μ) maintain CONet's robustness. Table 2 studies how changing epochs-per-trial and ϕ, leads to different model sizes. Changing these values only affects the search range for δ in turn controlling model size.

Table 2: Ablative study on Epoch-per-trial and ϕ.

Epoch/Trial	Param (M)	Acc (%)
10	4.78	80.65
15	2.17	79.01
20 (Default)	1.24	79.45

Factor Scale (ϕ)	Param (M)	Acc (%)
0.1	1.13	78.16
0.2 (Default)	1.24	79.45
0.3	1.78	79.42
0.4	2.51	79.52

Comparison to Compound Scaling: EfficientNet's compound scaling [35] heuristically increases the channel sizes relative to the network depth. This poses challenges where it can unnecessarily increase the number of channels per layer without consideration of (1) the information that has been learned and (2) the amount of redundant channel sizes. The CONet Alg. 2 is able to inspect weights to observe how well the model is learning and dynamically increase *or* decrease the number of channels (if necessary), allowing the network to evolve itself to the optimal number of channels. As shown in Fig.6, CONet remains superior compared to compound scaling on CIFAR/DARTS7 related experiments.

Architecture Transferability: The DARTS7 and DARTS14 architectures searched on CIFAR10/100 are evaluated on ImageNet to test architecture transferability, seen in Table 4. We compare the performance of our searched models to their baseline as well as other SOTA methods with a similar search space to DARTS. Comparing against the baseline, we see a 21.2% increase in performance for DARTS-7 δ_1 and 2.9% increase in performance for DARTS-14 δ_1. This demonstrates that our larger model is also capable of achieving a higher performance than our smaller models on ImageNet, reflecting the performance gain as shown on CIFAR10/100.

Generalization: In addition to DAG-based relaxed search spaces like DARTS and traditional CNNs like ResNet, the methodology of CONet can be applied to any type of CNN. We have demonstrated the ability of CONet to balance performance and parameters and shown the clear merits of our method to dynamically *scale-up* or *scale-down* an architecture as compared to heuristic channel size assignments and EfficientNet compound scaling. This method can be applied co-currently to other architecture search that trains a network during search. The methodology we propose provides an indicator to probe the useful information learned by a CNN layer, which can then be used to facilitate channel size selection during architecture search.

Rank correlation to test-accuracy: The Rank metric closely follows test accuracy while *only being exposed to train data*. In Fig 1(b) the layer rank follows test accuracy for 250 epochs. Additionally, our CONet Alg. 2 operates in the growth phase of training, seen in Fig 1(b) bottom right, where the rank measure is directly proportional to the test accuracy. The NATS benchmark [10] is composed of 15, 625 neural architectures, in Fig. 1(c) we inspect the rank of each at the 90th epoch & show it is highly correlated to test-accuracy with a Spearman correlation coefficient of 0.913. Therefore, rank as a novel metric allows channel optimizations from only train data.

6. Conclusion

In this work, we studied an important problem in NAS: channel size optimization given a specific model and task. To this end, we proposed CONet; a dynamic scaling algorithm that uses the *Rank* measurements from low-rank fac-

Table 3: Comparison of performance of **CONet** on ResNet34, DARTS7, DARTS+7, DARTS14 and DARTS+14 searched on CIFAR10/100 with different thresholds. Note $\delta_0 = 0.0025$, $\delta_1 = 0.005$, $\delta_2 = 0.0075$, $\delta_3 = 0.01$, $\delta_4 = 0.015$, $\delta_5 = 0.02$, $\delta_6 = 0.025$. We also included performance of DARTS7 with compound scaling [35] for $\phi_1 = \sqrt{2}$, $\phi_3 = \sqrt{2}^3$, $\phi_5 = \sqrt{2}^5$, $\phi_7 = \sqrt{2}^7$, and $\phi_9 = \sqrt{2}^9$. For compound scaling, each channel is scaled by ϕ.

Architecture	CIFAR10 Top-1 (%)	CIFAR10 Params (M)	CIFAR10 Search Cost (GPU-days)	CIFAR10 GPUs	CIFAR100 Top-1 (%)	CIFAR100 Params (M)	CIFAR100 Search Cost (GPU-days)	CIFAR100 GPUs
ResNet34(baseline)[14]	$95.56_{\pm 0.11}$	21.28	Manual	–	$77.88_{\pm 0.43}$	21.28	Manual	–
ResNet34(δ_6)	$95.12_{\pm 0.08}$ (−0.44)	2.05	0.3	1	$74.69_{\pm 0.02}$ (−3.19)	2.56	0.3	1
ResNet34(δ_5)	$95.34_{\pm 0.06}$ (−0.22)	3.20	0.3	1	$76.92_{\pm 0.22}$ (−0.96)	5.20	0.3	1
ResNet34(δ_4)	$95.59_{\pm 0.13}$ (+0.03)	5.47	0.4	1	$78.07_{\pm 0.15}$ (+0.19)	9.33	0.4	1
ResNet34(δ_3)	**$95.74_{\pm 0.06}$** (+0.23)	10.55	0.5	1	**$78.43_{\pm 0.40}$** (+0.55)	15.60	0.5	1
DARTS7(baseline)[22]	$94.30_{\pm 0.18}$	0.20	4	1	$73.81_{\pm 0.31}$	0.25	4	1
DARTS7(δ_4)	–	–	–	–	$77.38_{\pm 0.47}$ (+3.57)	0.58	0.3	1
DARTS7(δ_3)	$95.18_{\pm 0.17}$ (+0.88)	0.63	0.3	1	$79.45_{\pm 0.33}$ (+5.64)	1.24	0.3	1
DARTS7(δ_2)	$95.87_{\pm 0.12}$ (+1.57)	2.04	0.3	1	$80.12_{\pm 0.2}$ (+6.31)	2.89	0.3	1
DARTS7(δ_1)	$96.25_{\pm 0.13}$ (+1.95)	4.11	0.4	1	$80.53_{\pm 0.08}$ (+6.72)	5.04	0.4	1
DARTS7(δ_0)	$96.69_{\pm 0.12}$ (+2.39)	6.54	0.5	1	$80.75_{\pm 0.31}$ (+6.94)	6.25	0.5	1
DARTS7(ϕ_1)	–	–	Manual	–	$75.81_{\pm 0.30}$ (+2.00)	0.46	Manual	–
DARTS7(ϕ_3)	–	–	Manual	–	$78.77_{\pm 0.14}$ (+4.96)	1.67	Manual	–
DARTS7(ϕ_5)	$96.16_{\pm 0.094}$ (+1.89)	3.85	Manual	–	$80.27_{\pm 0.25}$ (+6.46)	3.94	Manual	–
DARTS7(ϕ_7)	$96.32_{\pm 0.16}$ (+2.02)	5.66	Manual	–	$80.50_{\pm 0.25}$ (+6.85)	5.76	Manual	–
DARTS7(ϕ_9)	–	–	Manual	–	$80.66_{\pm 0.21}$ (+6.94)	6.96	Manual	–
DARTS+7(baseline)[21]	$94.31_{\pm 0.07}$	0.26	0.2	1	$75.17_{\pm 0.31}$	0.28	0.2	1
DARTS+7(δ_3)	$95.21_{\pm 0.11}$ (+0.9)	0.81	0.3	1	$79.87_{\pm 0.27}$ (+4.7)	2.16	0.3	1
DARTS+7(δ_2)	$95.84_{\pm 0.15}$ (+1.53)	1.38	0.3	1	$80.67_{\pm 0.34}$ (+5.5)	3.54	0.3	1
DARTS+7(δ_1)	$96.19_{\pm 0.12}$ (+1.88)	4.2	0.4	1	$81.01_{\pm 0.13}$ (+5.84)	5.61	0.4	1
DARTS+7(δ_0)	$96.46_{\pm 0.11}$ (+2.15)	7.05	0.5	1	$81.21_{\pm 0.21}$ (+6.04)	7.44	0.5	1
DARTS14(baseline)[22]	$96.53_{\pm 0.12}$	3.93	4	1	$81.43_{\pm 0.17}$	4.00	4	1
DARTS14(δ_3)	$94.46_{\pm 0.12}$ (−2.07)	0.17	0.8	1	$76.33_{\pm 0.33}$ (−5.1)	0.41	0.8	1
DARTS14(δ_3)	$95.54_{\pm 0.1}$ (−0.99)	0.46	0.8	1	$78.4_{\pm 0.31}$ (−3.03)	0.91	0.8	1
DARTS14(δ_2)	$96.24_{\pm 0.14}$ (−0.29)	1.38	0.8	1	$80.67_{\pm 0.41}$ (−0.76)	3.06	0.8	1
DARTS14(δ_1)	$96.6_{\pm 0.08}$ (+0.07)	4.07	0.9	1	$82.06_{\pm 0.33}$ (+0.63)	8.2	0.9	1
DARTS14(δ_0)	$96.79_{\pm 0.09}$ (+0.26)	10.42	1	1	$82.4_{\pm 0.57}$ (+0.97)	12.65	1	1
DARTS+14(baseline)[21]	$96.43_{\pm 0.09}$	3.35	0.2	1	$80.84_{\pm 0.89}$	3.40	0.2	1
DARTS+14(δ_4)	$94.18_{\pm 0.17}$ (-2.25)	0.17	0.8	1	$76.3_{\pm 0.49}$ (−4.89)	0.59	0.8	1
DARTS+14(δ_3)	$95.08_{\pm 0.08}$ (-1.34)	0.44	0.8	1	$78.1_{\pm 0.14}$ (−2.73)	1.60	0.8	1
DARTS+14(δ_2)	$95.78_{\pm 0.09}$ (-0.64)	1.19	0.8	1	$79.46_{\pm 0.33}$ (−1.37)	2.89	0.8	1
DARTS+14(δ_1)	$96.5_{\pm 0.19}$ (0.07)	3.59	0.9	1	$80.68_{\pm 0.25}$ (−0.15)	5.46	0.9	1
DARTS+14(δ_0)	$96.69_{\pm 0.16}$ (0.26)	7.01	1	1	$81.47_{\pm 0.26}$ (+0.64)	8.57	1	1

Table 4: Comparison with DARTS-related architectures on ImageNet ordered by novelty. DARTS7 and DART14 are DARTS(2nd) 7 cells and 14 cells respectively. Note $\delta_0 = 0.0025$, $\delta_1 = 0.005$, $\delta_2 = 0.0075$, $\delta_3 = 0.01$, $\delta_4 = 0.015$, $\delta_5 = 0.02$, $\delta_6 = 0.025$.

Architecture	ImageNet Results Top-1 (%)	Top-5 (%)	Params (M)	Architecture Search Cost (GPU-days)	Searched On Dataset	GPUs	ImageNet Training Setup Training Epochs	Batch Size	Optimizer	LR-Schedular
NASNet-A[51]	74.0	91.6	5.3	3-4	CIFAR10	450	250	128	SGD	StepLR: Gamma=0.97,Step-size=1
AmoebaNet-C[28]	75.7	92.4	6.4	7	CIFAR10	250	250	128	SGD	StepLR: Gamma=0.97,Step-size=2
DARTS-7[22]	52.9	76.9	0.5	4	CIFAR10	1	200	128	SGD	StepLR: Gamma=0.5,Step-size=25
DARTS-14[22]	73.3	91.3	4.7	4	CIFAR10	1	250	128	SGD	StepLR: Gamma=0.97,Step-size=1
GHN[47]	73.0	91.3	6.1	0.84	CIFAR10	1	250	128	SGD	StepLR: Gamma=0.97,Step-size=1
ProxylessNAS[3]	74.6	92.2	5.8	8.3	ImageNet	–	300	256	Adam	–
SNAS[42]	72.7	90.8	4.3	1.5	CIFAR10	1	250	128	SGD	StepLR: Gamma=0.97,Step-size=1
BayesNAS[49]	73.5	91.1	3.9	0.2	CIFAR10	1	250	128	SGD	StepLR: Gamma=0.97,Step-size=1
P-DARTS[5]	75.6	92.6	4.9	0.3	CIFAR10	8	250	1024	SGD	OneCycleLR: Linear Annealing
GDAS[12]	72.5	90.9	4.4	0.17	CIFAR10	1	250	128	SGD	StepLR: Gamma=0.97,Step-size=1
PC-DARTS[43]	75.8	92.7	5.3	3.83	ImageNet	8	250	1024	SGD	OneCycleLR: Linear Annealing
DARTS+[21]	76.3	92.8	5.1	0.2	CIFAR100	1	800	2048	SGD	OneCycleLR: Cosine Annealing
NASP[44]	73.7	91.4	9.5	0.1	CIFAR10	1	250	128	SGD	StepLR: Gamma=0.97,Step-size=1
SGAS[19]	75.9	92.7	5.4	0.25	CIFAR10	1	250	1024	SGD	OneCycleLR: Linear Annealing
MiLeNAS[13]	75.3	92.4	5.3	0.3	CIFAR10	1	250	128	SGD	StepLR: Gamma=0.97,Step-size=1
SDARTS[4]	75.3	92.2	3.3	1.3	CIFAR10	1	250	1024	SGD	OneCycleLR: Linear Annealing
FairDARTS[7]	75.6	92.6	4.3	3	ImageNet	1	250	1024	SGD	OneCycleLR: Linear Annealing
DARTS7(δ_3)	67.9	88.0	1.8	0.3	CIFAR100	1	200	128	SGD	StepLR: Gamma=0.5,Step-size=25
DARTS7(δ_2)	72.0	90.6	3.4	0.4	CIFAR100	1	200	128	SGD	StepLR: Gamma=0.5,Step-size=25
DARTS7(δ_1)	74.1	91.7	5.6	0.5	CIFAR100	1	200	128	SGD	StepLR: Gamma=0.5,Step-size=25
DARTS-14(δ_2)	67.2	87.6	1.8	0.8	CIFAR10	1	200	128	SGD	StepLR: Gamma=0.97,Step-size=1
DARTS-14(δ_1)	74.0	91.8	4.8	0.8	CIFAR10	1	200	128	SGD	StepLR: Gamma=0.97,Step-size=1
DARTS14(δ_1)	**76.6**	**93.2**	9.0	0.9	CIFAR100	1	200	128	SGD	StepLR: Gamma=0.5,Step-size=25

torized tensor weights to efficiently scale channel sizes layers to achieve higher performance. We demonstrated that layer *Rank* takes a few training cycles to stabilize and we can use the average *Rank* of multiple layers to evaluate their performance as a whole. By representing the model as *conv* and *non-conv* connectors, we can automatically adjust the channel sizes of any given CNN with any dataset. Furthermore, we have shown that the adjustments CONet makes is *better* than uniformly scaling channel sizes with compound scaling method. In the future, we plan on extending CONet to kernel size optimization and to develop deeper integration with other popular NAS techniques so that they may run concurrently during architecture search.

References

[1] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. 2016. 2

[2] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. 1, 2

[3] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 8

[4] Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. In *International Conference on Machine Learning*, pages 1554–1565. PMLR, 2020. 8

[5] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1294–1303, 2019. 2, 8

[6] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair darts: Eliminating unfair advantages in differentiable architecture search. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 465–480. Springer International Publishing, 2020. 2

[7] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair darts: Eliminating unfair advantages in differentiable architecture search. In *European Conference on Computer Vision*, pages 465–480. Springer, 2020. 8

[8] Jiequan Cui, Pengguang Chen, Ruiyu Li, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Fast and practical neural architecture search. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6509–6518, 2019. 2

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 6

[10] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. NATS-Bench: Benchmarking nas algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021. doi:10.1109/TPAMI.2021.3054824. 1, 7

[11] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*, pages 760–771, 2019. 2

[12] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1761–1770, 2019. 8

[13] Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11993–12002, 2020. 8

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 2, 4, 8

[15] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012. 3

[16] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 4

[17] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6

[18] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V Oseledets, and Victor S Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *ICLR*, 2015. 3

[19] Guohao Li, Guocheng Qian, Itzel C Delgadillo, Matthias Muller, Ali Thabet, and Bernard Ghanem. Sgas: Sequential greedy architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1620–1630, 2020. 2, 8

[20] Xin Li, Yiming Zhou, Zheng Pan, and Jiashi Feng. Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*, pages 9145–9153, 2019. 2

[21] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping, 2020. 2, 8

[22] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 4, 6, 8

[23] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search, 2019. 1, 2

[24] Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. Atomnas: Fine-grained end-to-end neural architecture search. In *International Conference on Learning Representations*, 2020. 2

[25] Shinichi Nakajima, Masashi Sugiyama, S Derin Babacan, and Ryota Tomioka. Global analytic solution of fully-observed variational bayesian matrix factorization. *Journal of Machine Learning Research*, 14(Jan):1–37, 2013. 3

[26] Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. Asap: Architecture search, anneal and prune. In *International Conference on Artificial Intelligence and Statistics*, pages 493–503. PMLR, 2020. 2

[27] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 2

[28] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019. 2, 4, 8

[29] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911, 2017. 2

[30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 2

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 2

[32] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 481–497. Springer, 2019. 1, 2

[33] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 2

[34] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 1, 2

[35] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019. 6, 7, 8

[36] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020. 1

[37] Arash Vahdat, Arun Mallya, Ming-Yu Liu, and Jan Kautz. Unas: Differentiable architecture search meets reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11266–11275, 2020. 1, 2

[38] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12965–12974, 2020. 1, 2

[39] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Neural architecture search using deep neural networks and monte carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9983–9991, 2020. 1, 2

[40] Lingxi Xie and Alan Yuille. Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1379–1388, 2017. 2

[41] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 2

[42] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018. 2, 8

[43] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. *arXiv preprint arXiv:1907.05737*, 2019. 2, 8

[44] Quanming Yao, Ju Xu, Wei-Wei Tu, and Zhanxing Zhu. Efficient neural architecture search via proximal iterations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6664–6671, 2020. 8

[45] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 702–717. Springer International Publishing, 2020. 2

[46] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020. 1, 2

[47] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018. 8

[48] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2423–2432, 2018. 2

[49] Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. In *International Conference on Machine Learning*, pages 7603–7613. PMLR, 2019. 8

[50] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. 1, 2

[51] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 1, 2, 8