ASAP: Architecture Search, Anneal and Prune

Asaf Noy
Alibaba Group

Niv Nayman
Alibaba Group

Tal Ridnik
Alibaba Group

Nadav Zamir
Alibaba Group

Sivan Doveh
Tel-Aviv University

Itamar Friedman
Alibaba Group

Raja Giryes
Tel-Aviv University

Lihi Zelnik-Manor
Alibaba Group

Abstract

Automatic methods for Neural Architecture Search (NAS) have been shown to produce state-of-the-art network models. Yet, their main drawback is the computational complexity of the search process. As some primal methods optimized over a discrete search space, thousands of days of GPU were required for convergence. A recent approach is based on constructing a differentiable search space that enables gradient-based optimization, which reduces the search time to a few days. While successful, it still includes some noncontinuous steps, e.g., the pruning of many weak connections at once. In this paper, we propose a differentiable search space that allows the annealing of architecture weights, while gradually pruning inferior operations. In this way, the search converges to a single output network in a continuous manner. Experiments on several vision datasets demonstrate the effectiveness of our method with respect to the search cost and accuracy of the achieved model. Specifically, with 0.2 GPU search days we achieve an error rate of 1.68% on CIFAR-10.

1 Introduction

Over the last few years, deep neural networks highly succeed in computer vision tasks, mainly because of their automatic feature engineering. This success has led to large human efforts invested in finding good network architectures. A recent alternative approach is to replace the manual design with an automated Network Architecture Search (NAS) (Elsken et al., 2018). NAS methods have succeeded in finding more complex architectures that pushed forward the state-of-the-art in both image and sequential data tasks.

One of the main drawbacks of some NAS techniques is their large computational cost. For example, the search for NASNet (Zoph et al., 2018) and AmoebaNet (Real et al., 2018), which achieve state-of-the-art results in classification (Huang et al., 2018), have required 1800 and 3150 GPU days respectively. On a single GPU, this corresponds to years of training time. More recent search methods, such as ENAS (Pham et al., 2018) and DARTS (Liu et al., 2018b), reduced the search time to a few GPU days, while not compromising much on accuracy. While this is a major advancement, there is still a need to speed up the process to make the automatic search affordable and applicable to more problems.

In this paper we propose an approach that further reduces the search time to a few hours, rather than days or weeks. The key idea that enables this speed-up is relaxing the discrete search space to be continuous, differentiable, and annealable. For continuity and differentiability, we follow the approach in DARTS (Liu et al., 2018b), which shows that a continuous and differentiable search space allows for gradient-based optimization, resulting in orders of magnitude fewer computations in comparison with black-box search, e.g., of (Zoph and Le, 2017; Real et al., 2018). We adopt this line of thinking, but in addition we construct the search space to be annealable, emphasizing strong connections during the search in a continuous manner. We back this selection theoretically and demonstrate that an annealable search space implies a more continuous optimization, and hence both faster convergence and higher accuracy.

The annealable search space we define is a key factor to both reducing the network search time and obtaining high accuracy. It allows gradual pruning of weak weights, which reduces the number of computed connections throughout the search, thus, gaining the computational speed-up. In addition, it allows the network weights to adjust to the expected final architecture, and choose the components that are most valuable, thus, improving the classification accuracy of the generated architecture.

Proceedings of the 23[rd]International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).

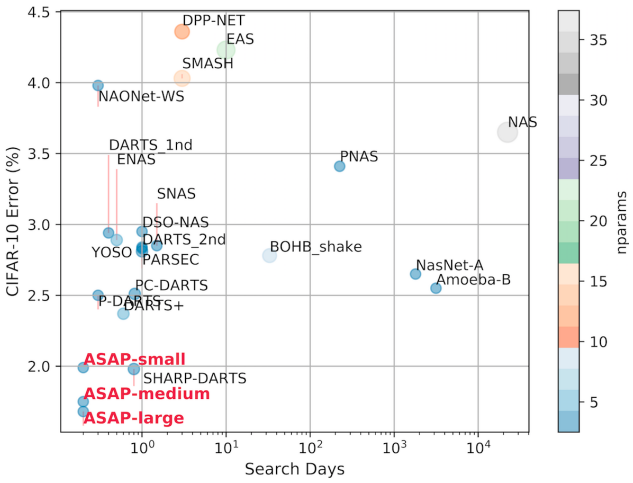

Figure 1: Comparison of CIFAR-10 error vs search days of top NAS methods. size and color are correlated with the memory footprint.

We validate the search efficiency and the performance of the generated architecture via experiments on multiple datasets. The experiments show that networks built by our algorithm achieve either higher or on par accuracy to those designed by other NAS methods, even-though our search requires only a few hours of GPU rather than days or weeks. See for example results on CIFAR-10 in Fig. 1. Specifically, searching for 4.8 GPU-hours on CIFAR-10 produces a model that achieves a mean test error of 1.68%.

2 Related Work

Architecture search techniques. A reinforcement learning based approach has been proposed by Zoph et al. for neural architecture search (Zoph and Le, 2017). They use a recurrent network as a controller to generate the model description of a child neural network designated for a given task. The resulted architecture (NASnet) improved over the existing hand-crafted network models at its time. An alternative search technique has been proposed by (Real et al., 2017, 2018), where an evolutionary (genetic) algorithm has been used to find a neural architecture tailored for a given task. The evolved neural network (Real et al., 2018), AmoebaNet, improved further the performance over NASnet. Although these works achieved state-of-the-art results on various classification tasks, their main disadvantage is the amount of computational resources they demanded.

To overcome this problem, new efficient architecture search methods have been developed, showing that it is possible, with a minor loss of accuracy, to find neural architectures in just few days using only a single GPU. Two notable approaches in this direction are the differentiable architecture search (DARTS) (Liu et al., 2018b) and the efficient NAS (ENAS) (Pham et al., 2018). ENAS is similar to NASnet (Zoph et al., 2018), yet its child models share weights of their shared operations from the large computational graph. During the search phase child models performance are estimated as a guiding signal to find more promising children. The weight sharing significantly reduces the search time since the child models performance are estimated with a minimal fine-tune training.

In DARTS, the entire computational graph, which includes repeating computation cells, is learned all together. At the end of the search phase, a pruning is applied on connections and their associated operations within the cells, keeping those having the highest magnitude of their related architecture weights multipliers (which represents the connections' strength). These architecture weights are learned in a continuous way during the search phase. While DARTS achieves good accuracy, our hypothesis is that the harsh pruning which occurs only once at the end of the search phase is sub-optimal and that a gradual pruning of connections can improve both the search efficiency and accuracy.

In YOSO (Zhang et al., 2018), they apply sparsity regularization over architecture weights and in the final stage they remove useless connections and isolated operations.

While both of these works (Xie et al., 2019; Zhang et al., 2018) propose an alternative to the DARTS connections selection rule, none of them managed to surpass its accuracy on CIFAR-10. In our solution we improve both the accuracy and efficiency of the DARTS searching mechanism.

Neural networks pruning strategies. Many pruning techniques exist for neural networks, since the networks may contain meaningful amount of operations which provides low gain (Han et al., 2015).

The methods for weight pruning can be divided into two main approaches: pruning a network post-training and performing the pruning during the training phase. In *post-training pruning* the weight elimination is performed only after the network is trained to learn the importance of the connections in it (LeCun et al., 1990; Hassibi and Stork., 1993; Han et al., 2015). One selection criterion removes the weights based on their magnitude. The main disadvantage of post-training methods is twofold: (i) long training time, which requires a train-prune-retrain schedule; (ii) pruning of weights in a single shot might lose some dependence between some of them that become more apparent if they are removed gradually.

Pruning-during-training as demonstrated in (Zhu and Gupta, 2017; Alvarez and Salzmann, 2017) helps to reduce the overall training time and get a sparser network. In (Zhu and Gupta, 2017), a method for gradually pruning the network has been proposed, where the number of pruned weights at each iteration depends on the final desired sparsity and a given pruning frequency. Another work suggested a compression aware method that encourages the

weight matrices to be of low-rank, which helps in pruning them in a second stage (Alvarez and Salzmann, 2017).

In the context of neural architecture search, XNAS (Nayman et al., 2019) suggested a 'Wipeout' mechanism for dynamic pruning of network connections with relatively poor performance, leading to faster and improved convergence.

3 Method

Our goal is to design an efficient algorithm for architecture search. We start by defining a differentiable search space that allows gradient-based optimization. A key characteristic of the search space we define is allowing for annealing of architecture weights.

Annealing of the architecture weights and pruning weak ones is a key for converging to a single architecture. However, too fast annealing schedule or too strict pruning policy could end with convergence to inferior architectures.
We provide a theory for selecting the critical combination of annealing schedule and pruning policy, which guarantees that the pruning will not affect the quality of the final cell, as only inferior operations will be pruned along the search. Selecting a combination according to the theory will provide the advantages of annealing and pruning, converging to a better architecture, faster.

This theory provides us with some insights as to the importance of using an annealing schedule. Yet, as it requires a relatively slow schedule for the guarantees in it, we suggest afterwards another gradual pruning schedule that empirically leads to faster convergence in the network search.

3.1 ASAP: Architecture Search, Anneal and Prune

Our approach can be viewed as a generalization of DARTS (Liu et al., 2018b) into an annealable search space. The key idea behind DARTS is the definition of a continuous search space that facilitates the definition of a differentiable training objective. DARTS continuous relaxation scheme leads to an efficient optimization and a fast convergence in comparison to previous solutions.

The architecture search in DARTS focus on finding a repeating structure in the network, which is called cell. They follow the observation that modern neural networks consist of one or few computational blocks (e.g. res-block) that are stacked together to form the final structure of the network. While a network might have hundreds of layers, the structure within each of them is repetitive. Thus, it is enough to learn one or two structures, which is denoted as cell, to design the whole network.

A cell is represented as a directed acyclic graph, consisting of an ordered sequence of nodes. Every node $\boldsymbol{x}^{(i)}$ is a feature map and each directed connection (i,j) is associated with some operation $o^{(i,j)} \in \mathcal{O}$ that transforms $\boldsymbol{x}^{(j)}$ and connects it to $\boldsymbol{x}^{(i)}$. Intermediate nodes are computed based on their predecessors,

$$\boldsymbol{x}^{(i)} = \sum_{j<i} o^{(i,j)} \left(\boldsymbol{x}^{(j)} \right) \tag{1}$$

In particular, in DARTS, they search for two types of cells: normal and reduction. In normal cells the operations preserve the spatial dimensions, while in reduction cells the operations adjacent to the input nodes are of stride two, so the spatial dimensions at the output are halved.

The goal of the search phase is to select the operations $o^{(i,j)}$ which yield an overall architecture with the best performance. There are seven candidate operations: 3x3 and 5x5 separable and dilated separable convolutions, 3x3 max-pooling, 3x3 average-pooling and an identity.

This scheme does not encourage convergence towards a reasonably sized architecture, and produces an over-parametrized network. Therefore, hard pruning is applied over the network connections in order to derive the final *child network*. As shown in (Xie et al., 2019), this introduces a *relaxation bias* that could result in a significant drop in accuracy between the un-pruned network and the final child network.

To overcome this limitation we define a search space that allows gradual pruning via annealing. Our search converges gradually to the final child network without requiring a hard thresholding at the end of the search.

We construct an architecture by stacking normal and reduction cells, similarly to DARTS and ENAS. Also here, we connect nodes using the a mixed operation $\bar{o}^{(i,j)}$ edge. Yet, now we allow architecture weights annealing in it via a temperature parameter T:

$$\bar{o}^{(i,j)}(\boldsymbol{x}; T) \quad = \quad \sum_{o \in \mathcal{O}} \Phi_o(\boldsymbol{\alpha}^{(i,j)}; T) \cdot o(\boldsymbol{x}) \tag{2}$$

where $\boldsymbol{\alpha}_o^{(i,j)}$ is the architecture weight associated with operation $o \in \mathcal{O}$ at edge (i,j), and Φ_o forms a probability distribution. The function Φ_o should be designed such that it guides the optimization to select a single operation out of the mixture in a finite time. Initially Φ_o should be a uniform distribution, allowing consideration of all the operations. As the iterations continue the temperature is reduced and Φ_o should converge into a degenerated distribution that selects a single operation. Mathematically, this implies that Φ_o should be uniform for $T \to \infty$, and sparse for $T \to 0$. Thus, we select the following probability distribution,

$$\Phi_o(\boldsymbol{\alpha}^{(i,j)}; T) = \frac{\exp\left\{\frac{\boldsymbol{\alpha}_o^{(i,j)}}{T}\right\}}{\sum_{o' \in \mathcal{O}} \exp\left\{\frac{\boldsymbol{\alpha}_{o'}^{(i,j)}}{T}\right\}} \tag{3}$$

This definition is closely related to the Gibbs-Boltzmann distribution, where the weights $\boldsymbol{\alpha}_o^{(i,j)}$ correspond to negative energies, and operations $o^{(i,j)}$ to mixed system states (Van Laarhoven and Aarts, 1987).

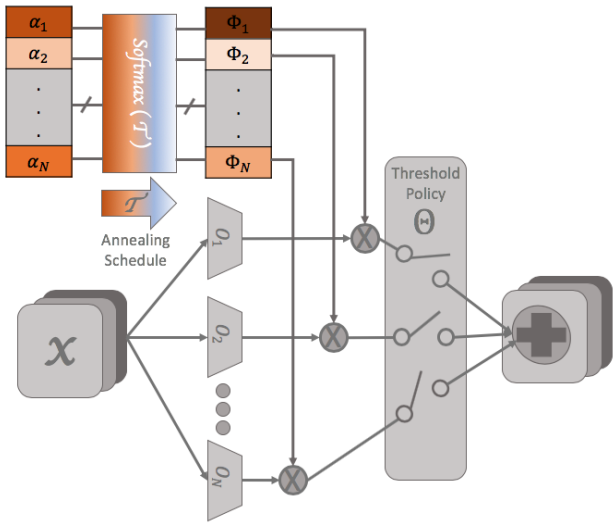

Figure 2: An illustration of our gradual annealing and pruning algorithm for architecture search depicted for a cell in the architecture. An annealing schedule is applied over architecture weights (top-left), which facilitates fast and continuous pruning of connections (right).

The architecture weights are updated via gradient descent,

$$\begin{aligned} \alpha_k &\leftarrow \alpha_k - \eta \nabla_{\alpha_k} \mathcal{L}_{val} \\ \nabla_{\alpha_k} \mathcal{L}_{val} &= \Phi_{o_k}(\boldsymbol{\alpha}; T) \left[\nabla_{\bar{o}} \mathcal{L}_{val} \cdot (o_k - \bar{o}) \right] \end{aligned} \quad (4)$$

Note, that DARTS forms a special case of our approach when setting a fixed $T = 1$. A key challenge here is how to select T. Hereafter in Section 3.2, we provide a theory for selecting it followed by a heuristic annealing schedule strategy to further speed up the convergence of our search. In the experiments we show its effectiveness.

We summarize our algorithm for Architecture Search, Anneal, and Prune (ASAP), outlined in Algorithm 1 and visualized in Figure 2. ASAP anneals and prunes the connections within the cell in a continuous manner, leading to a short search time and an improved cell structure. It starts with a few "warm-up" iterations for reaching well-balanced weights. Then it optimizes over the network weights while pruning weak connections, as described next.

At initialization, network weights are randomly set. This could result in a discrepancy between parameterized operations (e.g. convolution layers) and non-parameterized operations (e.g. pooling layers). Applying pruning at this stage with imbalanced weights could lead to wrong premature decisions. Therefore, we start with performing a number of gradient-descent based optimization steps over the network-weights only, without pruning, i.e. *grace-cycles*. We denote the number of these cycles by τ.

Once this warm-up ends, we perform a gradient-descent based optimization for the architecture-weights, in an alternating manner (with the updates of $\boldsymbol{\alpha}$), after every network-weights optimization. Throughout this iterative process, the temperature T decays according to a predefined annealing schedule. Operations are pruned if their corresponding architecture-weights go below a pruning threshold θ, which is updated along the iterations. The process ends once we meet a stopping criterion. In our current implementation we used a simple criterion that stops when we reach a single operation per mixed operation. Other criteria could be easily adopted as well.

3.2 Annealing Schedule and Thresholding Policy

We now turn to describe the annealing schedule that determines the temperature T through time and the threshold policy governing the updates of θ. This choice is dominated by a trade-off. On the one hand, pruning-during-training simplifies the optimization objective and reduces the complexity, assisting the network to converge and reducing overfitting. In addition, it accelerates the search process, since pruned operations are removed from the optimization. This encourages the selection of fast temperature decay and a harsh thresholding. On the other hand, premature pruning of operations during the search could lead to a sub-optimal child network. This suggests we should choose a slow temperature decay and a soft thresholding.

To choose schedule and policy that balance this trade-off, we suggest Theorem 2 that provides a functional form for (T_t, θ_t). It views the pruning performed as a selection problem, i.e., pruning all the inferior operations, such that the ones with the highest expected architecture weight remains, in a setup where only the empirical value is measured. The theorem guarantees under some assumptions with a high probability, that Algorithm 1 prunes inferior operations out of the mixed-operation along the run (lines 7-8) and outputs the best operation (line 14)). More formally, we follow Definition 1 of [10] and prove that Algorithm 1 is a (ϵ, δ)-Probably Approximately Correct (PAC) algorithm.

Definition 1 ((ϵ, δ)-PAC) *An algorithm that outputs an ϵ-optimal operation with probability of at least $1 - \delta$ is (ϵ, δ)-PAC.*

In our case, we are able to prove that with high probability Algorithm 1 outputs the best operation, that is $\epsilon = 0$.

Theorem 2 *Assuming $\nabla_{\alpha_{t,i}} \mathcal{L}_{val}(\boldsymbol{\omega}, \boldsymbol{\alpha}; T_t)$ is independent through time and bounded by $\mathbf{L}$ in absolute value for all t and i, then Algorithm 1 with a threshold policy,*

$$\theta_t = \nu_t e^{-t} \quad (5)$$

$$\nu_t \in \Upsilon = \left\{ \nu_t \mid \nu_t \geq 0; \lim_{t \to \infty} \frac{\log(\nu_t)}{t} = 0 \right\} \quad (6)$$

and an annealing schedule,

$$T(t) = \eta \mathbf{L} \rho_t \sqrt{\frac{8}{t} \log\left(\frac{\pi^2 N t^2}{3\delta}\right)} \tag{7}$$

$$\rho_t = \frac{t}{t + \log\left(\frac{1}{N\nu_t}\right)} \tag{8}$$

is $(0, \delta)$-PAC.

In other words, the theorem shows that with a proper annealing and thresholding the algorithm has a high likelihood to prune only sub-optimal operations. A proof sketch appears below in Section 3.3. The full proof is deferred to the supplementary material.

While these theoretical results are strong, the critical schedule they suggest is rather slow, because PAC bounds tend to be overly pessimistic in practice. Therefore, for practical usage we suggest a more aggressive schedule. It is based on observations explored in Simulated Annealing, which shares a similar tradeoff between the continuity of the optimization process and the time required for deriving the solution (Busetti, 2003). Among the many alternatives we choose the exponential schedule:

$$T(t) = T_0 \beta^t, \tag{9}$$

which has been shown to be effective when the optimization steps are expensive, e.g. (Ingber, 1989; Nourani and Andresen, 1998; Kirkpatrick et al., 1983). This schedule starts with a relatively high temperature T_0 and decays fast. As for the pruning threshold, we chose the simplest solution of a fixed value $\Theta \equiv \theta_0$. We demonstrate the effectiveness of our choices via experiments in Section 4.

Algorithm 1 ASAP for a single Mixed Operation

Input: Operations $o_i \in \mathcal{O} \;\; i \in \{1, .., N\}$,
Annealing schedule T_t,
Grace-temperature τ,
Threshold policy θ_t,
Init: $\boldsymbol{\alpha}_i \leftarrow 0, \;\; i \in \{1, .., N\}$.
while $|\mathcal{O}| > 1$ **do**
  Update $\boldsymbol{\omega}$ by descent step over $\nabla_{\boldsymbol{\omega}} \mathcal{L}_{\text{train}}(\boldsymbol{\omega}, \boldsymbol{\alpha}; T_t)$
  if $T_t < \tau$ **then**
    Update $\boldsymbol{\alpha}$ by descent step over $\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\text{val}}(\boldsymbol{\omega}, \boldsymbol{\alpha}; T_t)$
    for each $o_i \in \mathcal{O}$ such that $\Phi_{o_i}(\boldsymbol{\alpha}; T_t) < \theta_t$ **do**
      $\mathcal{O} = \mathcal{O} \setminus \{o_i\}$
    end for
  end if
  Update T_t
  Update θ_t
end while
return $\mathcal{O}$

3.3 Theoretical Analysis

The proof of the theorem relies on two main steps: (i) Reducing the algorithm to a *Successive-Elimination* method (Even-Dar et al., 2006), and (ii) Bounding the probability of deviation of $\boldsymbol{\alpha}$ from its expected value. We sketch the proof using Claim 1 and Theorem 3 below.

Claim 1 *The pruning rule (Step 7) in Algorithm 1 is equivalent to pruning $o_i \in \mathcal{O}$ at time t if,*

$$\frac{\alpha_{t,i}}{t} + \beta_t < \frac{\alpha_t^*}{t} - \beta_t, \tag{10}$$

where $\alpha_t^* = \max_i \{\alpha_{t,i}\}$ and $\beta_t = \frac{T_t}{2\rho_t}$.

Although involving the empirical values of α, the condition in Claim 1 avoids the pruning of the operation with the highest expected α. For this purpose we bound the probability for the deviation of each empirical α from its expected value by the specified margin β_t. We provide Theorem 3, which states that for any time t and operation $o_i \in \mathcal{O}$, the value of $\frac{\alpha_{t,i}}{t}$ is within β_t of its expected value $\frac{\bar{\alpha}_{t,i}}{t} = \frac{1}{t}\sum_{s=1}^{t} \mathbb{E}\left[g_{s,i}\right]$, where,

$$g_{t,i} = -\eta \nabla_{\alpha_{t,i}} \mathcal{L}_{val}(w, \alpha; T_t) \in [-\eta \mathbf{L}, \eta \mathbf{L}]. \tag{11}$$

Theorem 3 *For any time t and operation $\{o_i\}_{i=1}^N \in \mathcal{O}$, we have,*

$$\mathbb{P}\left\{\frac{1}{t}\left|\alpha_{t,i} - \bar{\alpha}_{t,i}\right| \leq \beta_t\right\} \geq 1 - \frac{\delta}{N}. \tag{12}$$

Requiring this to hold for all the N operations, we get that the probability of pruning the best operation is below $1 - \delta$. Choosing an annealing schedule and threshold policy such that β_t goes to zero as t increases, guarantees that eventually all operations but the best one are pruned. This leads to the desired result. Full proofs appear in the supplementary material.

4 Experiments

To show the effectiveness of ASAP we test it on common benchmarks and compare to the state-of-the-art. We experimented on the popular benchmarks, CIFAR-10 and ImageNet, as well as on five other classification datasets, for providing a more extensive evaluation. In addition, we explore alternative pruning methods and compare them to ASAP.

4.1 Architecture Search on CIFAR-10

We search on CIFAR-10 for convolutional cells in a small parent network. Then we build a larger network by stacking the learned cells, train it on CIFAR-10 and compare the results against common NAS methods.

We create the parent network by stacking 8 cells with architecture weights sharing. Each cell contains 4 ordered nodes, each of which is connected via mixed operations to all previous nodes in the cell and also to the two previous

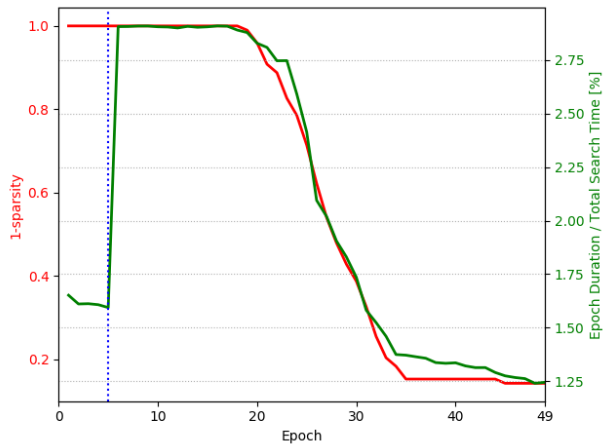

Figure 3: Relative epoch duration during search and model $1-$sparsity vs epoch number. The dotted line represents grace cycles.

cells' outputs. The output of a cell is a concatenation of the outputs of all the nodes within the cell.

The search phase lasts until each mixed operation is fully pruned, or until we reach 50 epochs. We use the first-order approximation (Liu et al., 2018b), relating to $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ as independent thus can be optimized separately. The train set is divided into two parts of equal sizes: one is used for training the operations' weights $\boldsymbol{\omega}$ and the other for training the architecture weights $\boldsymbol{\alpha}$.

For the gradual annealing, we use Equation 9 with an initial temperature of $T_0 = 1.3$ and an epoch decay factor of $\beta = 0.95$, thus $T \approx 0.1$ at the end of the search phase. Considering we have N possible operations to choose from in a mixed operation, we set our pruning threshold to $\theta_t \equiv \frac{0.4}{N}$. Other hyper-parameters follow (Liu et al., 2018b).

As the search progresses, continuous pruning reduces the network size. With a batch size of 96, one epoch takes 5.8 minutes in average on a single GPU[1], summing up to 4.8 hours in total for a single search.

Figure 3 illustrates an example of the epoch duration and the network sparsity (1 minus the relative part of the pruned connections) during a search. The smooth removal of connections is evident. This translates to a significant decrease in an epoch duration along the search. Note that the first five epochs are grace-cycles, where only the network weighs are trained as no operations are pruned. Figures 4 and 5 show our learned normal and reduction cells, respectively.

[1]Experiments were performed using a NVIDIA GTX 1080Ti GPU.

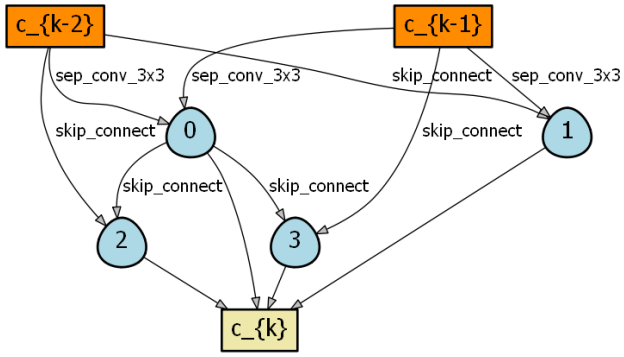

Figure 4: ASAP learned normal cell on CIFAR-10.

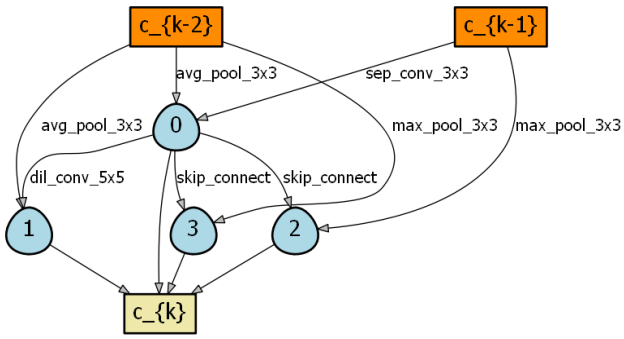

Figure 5: ASAP learned reduction cell on CIFAR-10.

4.2 CIFAR-10 Evaluation Results

We built the evaluation network by stacking 20 cells, 18 normal cells and 2 reduction cells. We place the reduction cells after $1/3$ and $2/3$ of the network, where after each reduction we double the amount of channels in the network. We trained the network for 1500 epochs using a batch size of 128 and SGD optimizer with nesterov-momentum. Our learning rate regime was composed of 5 cycles of power cosine annealing learning rate (Hundt et al., 2019), with amplitude decay factor of 0.5 per cycle. For regularization we used cutout (DeVries and Taylor, 2017), auxiliary towers (Szegedy et al., 2015), scheduled drop-path (Larsson et al., 2016), label smoothing, AutoAugment (Cubuk et al., 2018) and weight decay. To understand the effect of the network size on the final accuracy, we chose to test 3 architecture configurations with 36, 44 and 50 initial network channels, which we named respectively ASAP-Small, ASAP-Medium and ASAP-Large.

Table 1 shows the performance of our learned models compared to other state-of-the-art NAS methods. Figure 1 provides an even more comprehensive comparison in a graphical manner.

Table 1 and Figure 1 show that our ASAP based network

Architecture	Test error(%)	Param (M)	Search cost↓
AmoebaNet-A (Real et al., 2018)	3.34 ± 0.06	3.2	3150
AmoebaNet-B (Real et al., 2018)	2.55 ± 0.05	2.8	3150
NASNet-A (Zoph et al., 2018)	2.65	3.3	1800
PNAS (Liu et al., 2018a)	3.41	3.2	150
SNAS (Xie et al., 2019)	2.85 ± 0.02	2.8	1.5
DSO-NAS (Zhang et al., 2018)	2.95 ± 0.12	3	1
PARSEC (Casale et al., 2019)	2.81 ± 0.03	3.7	1
DARTS(2nd) (Liu et al., 2018b)	2.76 ± 0.06	3.4	1
PC-DARTS (Laube, 2019)	2.51 ± 0.09	4.0	0.82
DARTS+ (Liang et al., 2019)	2.37 ± 0.13	4.3	0.6
ENAS (Pham et al., 2018)	2.89	4.6	0.5
DARTS(1nd) (Liu et al., 2018b)	2.94	2.9	0.4
P-DARTS (Chen et al., 2019)	2.50	3.4	0.3
DARTS(1nd) (Liu et al., 2018b)	2.94	2.9	0.4
NAONet-WS (Luo et al., 2018)	3.53	**2.5**	0.3
ASAP-Small	1.99	**2.5**	**0.2**
ASAP-Medium	1.75	3.7	**0.2**
ASAP-Large	**1.68**	6.0	**0.2**

Table 1: Classification errors of ASAP compared to state-of-the-art NAS methods on CIFAR-10. The Search cost is measured in GPU days.

outperforms previous state-of-the-art NAS methods, both in terms of the classification error and the search time.

4.3 Transferability Evaluation

Using the cell found by ASAP search on CIFAR-10, we preformed transferability tests on 6 popular classification benchmarks: ImageNet, CINIC10, Freiburg, CIFAR-100, SVHN and Fashion-MNIST (FMNIST).

ImageNet For testing ASAP cell transferability performance on larger datasets, we experimented on the popular ImageNet dataset (Deng et al., 2009). The network was composed of 14 stacked cells, with two initial stem cells. We used 50 initial channels, so the total number of network FLOPs is below 600[M], similar to other ImageNet architectures with small computation regime(Liu et al., 2018b). We trained the network for 250 epochs using a nesterov-momentum optimizer. The results are presented in Table 2. It can be seen that the cell found by ASAP transfers well to ImageNet - accuracy second only to (Real et al., 2018), with significant faster search time.

Other classification datasets We further extend our transferability testing by training our ASAP cell and other top published NAS cells on 5 additional benchmarks: CINIC10, Freiburg, CIFAR-100, SVHN and Fashion-MNIST. Datasets details appear in 6.2.2. For a fair comparison, we trained all of the cells using the publicly available DARTS training code [2], with exactly the same network configuration and hyperparameters, except from the cell itself. Results are presented in Table 2.

Table 2 shows that our ASAP cell transfers well to other computer vision datasets, surpassing all other NAS cells in four out of five datasets tested, while having the lowest search cost. In addition, note that on two datasets, CINIC-10 and FREIBURG, our ASAP network accuracy is better than previously known state-of-the-art.

[2] https://github.com/quark0/darts

4.4 Other Pruning Methods

In addition to comparisons to other SotA algorithms, we also evaluate alternative pruning-during-training techniques. We select two well known pruning approaches and adapt those to the neural architecture search framework. The first approach is magnitude based. It naively cuts connections with the smallest weights, as is the practice in some network pruning strategies, e.g. (Han et al., 2015; Zhu and Gupta, 2017) (as well as in (Liu et al., 2018b), yet with a harsh single pruning-after-training step). The second approach (Lis, 2018) prunes connections with the lowest accumulated absolute value of the gradients of the loss with respect to $\boldsymbol{\alpha}$. Our evaluation is based on the number of operations to prune in each step, according to the formula suggested in (Zhu and Gupta, 2017):

$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^p \tag{13}$$

where $p \in \{1, 3\}$, $t \in \{t_0, t_0 + \Delta t, ..., t_0 + n\Delta t\}$, s_f is the final desired sparsity (the fraction of pruned weights), s_i is the initial sparsity, which is 0 in our case, t_0 is the iteration we start the pruning at, and $E_f = t_0 + n\Delta t$ is the total number of iterations used for the search. For $p = 1$ the pruning rate is constant, while for $p = 3$ it decreases. The evaluation was performed on CIFAR-10 with $t_0 = 20$ and E_f in the range from 50 to 70. Table 3 presents the results. We can see from Table 3 that both pruning methods achieved lower accuracy than ASAP on CIFAR-10, with a larger memory footprint.

4.5 Search Process Analysis

In this section we wish to provide further insights as to why ASAP leads to a higher accuracy with a lower search time. To do that we explore two properties along the iterative learning process: the validation accuracy and the cell entropy. We compare ASAP to two other efficient methods: DARTS (Liu et al., 2018b), and the reinforcement-learning based ENAS (Pham et al., 2018). We further compare to the pruning alternative based on accumulated gradients described in Section 4.4, as it achieved better results on CIFAR-10 them magnitude pruning, as shown in Table 3. We compare with both $p \in \{1, 3\}$.

Figure 6 presents the validation accuracy along the architecture search iterations. ENAS achieves low and noisy accuracy along the search, inferior to all differentiable-space based methods. DARTS accuracy climbs nicely across iterations, but then significantly drops at the end of the search due to the relaxation bias following the harsh prune. The pruning-during-training methods suffer from perturbations

Architecture	CINIC-10 Error(%)	FREIBURG Error(%)	CIFAR-100 Error(%)	SVHN Error(%)	FMNIST Error(%)	ImageNet Error(%)	Search cost ↓
Known SotA	8.6	21.1	8.7	1.02	3.65	15.7	-
AmoebaNet-A	7.18	11.8	15.9	1.93	3.8	**24.3**	3150
NASNet	6.93	13.4	15.8	1.96	3.71	26.0	1800
PNAS	7.03	12.3	15.9	1.83	3.72	25.8	150
SNAS	7.13	14.7	16.5	1.98	3.73	27.3	1.5
DARTS-Rev1	7.05	11.4	15.8	1.94	3.74	26.9	1
DARTS-Rev2	6.88	10.8	15.7	1.85	**3.68**	26.7	1
ASAP	**6.83**	**10.7**	**15.6**	**1.81**	3.73	24.4	**0.2**

Table 2: Transferability classification error of ASAP, compared to top NAS cells, on several vision classification datasets. Error refers to top-1 test error. Search cost is measured in GPU days. Known state of the art results: CINIC-10 (Darlow et al., 2018) FREIBURG (Jund et al., 2016), CIFAR-100 (Huang et al., 2018), SVHN (Cubuk et al., 2018), FMNIST (Zhong et al., 2017), and ImageNet (Huang et al., 2018).

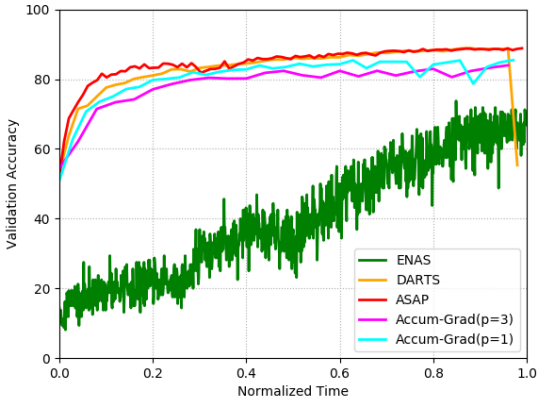

Figure 6: Search validation accuracy for ENAS, DARTS, ASAP and Accum-Grad over time (scaled to [0, 1]).

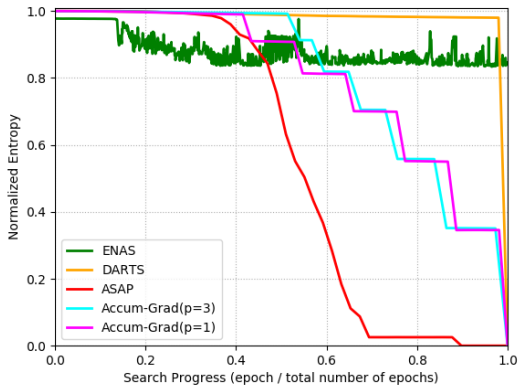

Figure 7: The total normalized entropy during search of normal and reduction cells for ENAS, DARTS, ASAP and Accum-Grad vs the time scaled to [0, 1].

Architecture type	Test error Top-1(%)	Params (M)
DARTS(2st)	2.76	3.4
Magnitude prune	2.9	4.4
Accum-Grad prune	2.76	4.3
ASAP-Small	1.99	2.5

Table 3: Comparison of pruning methods on CIFAR-10.

in accuracy following pruning, however, ASAP's perturbations are frequent and smaller, as its architecture-weights which get pruned are insignificant. Its validation accuracy quickly recovers due to the continuous annealing of weights. Therefore, With reduced network complexity, it achieves the highest accuracy at the end. Figure 7 illustrates the average cell entropy over time, averaged across all the mixed operations within the cell, normalized by $\ln(|\mathcal{O}|)$. While the entropy of DARTS and ENAS remain high along the entire search, the entropy of the pruning methods decrease to zero as those do not suffer from a relaxation bias. The high entropy of DARTS when final prune is done leads to a quite arbitrary selection of a child model, as top architecture weights are of comparable value. It can be seen that the entropy of ASAP decreases gradually, hence allowing efficient optimization in the differentiable-annealable space. This provides further motivation to the advantages of ASAP.

5 Conclusion

In this paper we presented ASAP, an efficient state-of-the-art neural architecture search framework. The key contribution of ASAP is the annealing and gradual pruning of connections during the search process. This approach avoids hard pruning of connections that is used by other methods. The gradual pruning decreases the architecture search time, while improving the final accuracy due to smoother removal of connections during the search phase.

On CIFAR-10, our ASAP cell outperforms other published cells, both in terms of search cost and in terms of test error. We also demonstrate the effectiveness of ASAP cell by showing good transferability quality compared to other top NAS cells on multiple computer vision datasets.

References

Jose M. Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Franco Busetti. Simulated annealing overview. *World Wide Web URL www. geocities. com/francorbusetti/saweb. pdf*, 2003.

Francesco Paolo Casale, Jonathan Gordon, and Nicolo Fusi. Probabilistic neural architecture search. *arXiv preprint arXiv:1902.05116*, 2019.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *arXiv preprint arXiv:1904.12760*, 2019.

Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501v2*, 2018.

Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

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

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

Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of machine learning research*, 7(Jun):1079–1105, 2006.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems (NIPS)*, 1993.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*, 2018.

Andrew Hundt, Varun Jain, and Gregory D Hager. sharpdarts: Faster and more accurate differentiable architecture search. *arXiv preprint arXiv:1903.09900*, 2019.

Lester Ingber. Very fast simulated re-annealing. *Mathematical and computer modelling*, 12(8):967–973, 1989.

Philipp Jund, Nichola Abdo, Andreas Eitel, and Wolfram Burgard. The freiburg groceries dataset. *arXiv preprint arXiv:1611.05799*, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

Kevin Alexander Laube. Prune and replace nas. *arXiv preprint arXiv:1906.07528*, 2019.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems (NIPS)*, 1990.

Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

Maximilian Golub Guy Lemieux Mieszko Lis. Dropback: Continuous pruning during training. *arXiv preprint arXiv:1806.06949*, 2018.

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. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, pages 7827–7838, 2018.

Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik. Xnas: Neural architecture search with expert advice. In *Advances in Neural Information Processing Systems*, pages 1975–1985, 2019.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Yaghout Nourani and Bjarne Andresen. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373, 1998.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, , and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning (ICML)*, 2018.

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 (ICML)*, 2017.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *International Conference on Machine Learning - ICML AutoML Workshop*, 2018.

Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596*, 2014.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: Stochastic neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

Xinbang Zhang, Zehao Huang, and Naiyan Wang. You only search once: Single shot neural architecture search via direct sparse optimization. *arxiv 1811.01567*, 2018.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896*, 2017.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. In *2017 NIPS Workshop on Machine Learning of Phones and other Consumer Devices*, 2017.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

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.