Resource Constrained Neural Network Architecture Search:
Will a Submodularity Assumption Help?

Yunyang Xiong   Ronak Mehta   Vikas Singh
University of Wisconsin Madison

yxiong43@wisc.edu   ronakrm@cs.wisc.edu   vsingh@biostat.wisc.edu

Abstract

The design of neural network architectures is frequently either based on human expertise using trial/error and empirical feedback or tackled via large scale reinforcement learning strategies performed over distinct discrete architecture choices. In the latter case, the optimization is often non-differentiable and also not very amenable to derivative-free optimization methods. Most methods in use today require sizable computational resources. And if we want networks that additionally satisfy resource constraints, the above challenges are exacerbated because the search must now balance accuracy with certain budget constraints on resources. We formulate this problem as the optimization of a set function – we find that the empirical behavior of this set function often (but not always) satisfies marginal gain and monotonicity principles – properties central to the idea of submodularity. Based on this observation, we adapt algorithms within discrete optimization to obtain heuristic schemes for neural network architecture search, where we have resource constraints on the architecture. This simple scheme when applied on CIFAR-100 and ImageNet, identifies resource-constrained architectures with quantifiably better performance than current state-of-the-art models designed for mobile devices. Specifically, we find high-performing architectures with fewer parameters and computations by a search method that is much faster.

1. Introduction

The design of state-of-the-art neural network architectures for a given learning task typically involves extensive effort: human expertise as well as significant compute power. It is well accepted that the trial-and-error process is tedious, often requiring iterative adjustment of models based on empirical feedback – until one discovers the "best" structure, often based on overall accuracy. In other cases it may also be a function of the model's memory footprint or speed at test time. This architecture search process becomes more challenging when we seek *resource-constrained* networks to eventually deploy on small form factor devices: accuracy and resource-efficiency need to be carefully balanced. Further, each type of mobile device has its own hardware idiosyncrasies and may require different architectures for the best accuracy-efficiency trade-off. Motivated by these considerations, researchers are devoting effort into the development of algorithms that automate the process of architecture search and design. Many of the models that have been identified via a judicious use of such architecture search schemes (often together with human expertise) currently provide excellent performance in classification [3, 31, 44] and object detection [31].

The superior performance of the architectures identified via the above process notwithstanding, it is well known that those search algorithms are time consuming and compute-intensive. For reference, even for a smaller dataset such as CIFAR-10, [44] requires 3150 GPU days for a reinforcement learning (RL) model. A number of approaches have been proposed to speed up architecture search algorithms. Some of the strategies include adding a specific structure to the search space to reduce search time [25, 36], sharing weights across various architectures [5, 30], and imposing weight or performance prediction constraints for each distinct architecture [3, 4, 26]. These ideas all help in various specific cases, but the inherent issue of a large search space and the associated difficulties of scalability still exists.

Notice that one reason why many search methods based on RL, evolutionary schemes, MCTS [28], SMBO [25] or Bayesian optimization [18] are compute intensive in general is because architecture search is often set up as a black-box optimization problem over a large discrete domain, thus leading to a large number of architecture evaluations during search. Further, many architecture search methods [8, 25, 44] do not directly take into account certain resource bounds (e.g., # of FLOPs) although the search space can be pre-processed to filter out those regions of the search space. As a result, few methods have been widely used for identifying deployment-ready architectures for mobile/embedded devices. When the resource of interest is memory, one

choice is to first train a network and then squeeze or compress it for a target deployment device [2, 9, 12, 37, 40].

Key idea. Here, we take a slightly different line of attack for this problem which is based loosely on ideas that were widely used in computer vision in the early/mid 2000s [21]. First, we move away from black box optimization for architecture search, similar to strategies adopted in other recent works [14, 18, 25, 28, 33]. Instead, we view the architecture as being composed of primitive basic blocks. The "skeleton" or connectivity between the blocks is assumed fixed whereas the actual functionality provided by each block is a decision variable – this is precisely what our search will be performed on. The goal then is to identify the assignment of blocks so that the overall architecture satisfies two simple properties: **(a)** it satisfies the user provided resource bounds and **(b)** is accurate for the user's task of interest.

While we will make the statement more formal shortly, it is easy to see that the above problem can be easily viewed as a set function. Each empty block can be assigned to a specific type of functional module. Once all blocks have been assigned to some functional module, we have a "proposal" network architecture whose accuracy can be evaluated — either by training to completion or stopping early [24, 41]. A different assignment simply yields a different architecture with a different performance profile. In this way, the set function (where accuracy can be thought of as function evaluation) can be queried/sampled. We find that empirically, when we evaluate the behavior of this set function, it often exhibits nice *marginal gain* or *diminishing returns properties*. Further, the performance (accuracy) typically *improves* or stays nearly the same when *adding* functional modules to a currently empty block (akin to adding an element to a set). These properties are central to **submodularity**, a key concept in many classical methods in computer vision. Mathematically, of course, our set function is **not** submodular. However, our empirical study suggests that the set function generally behaves well. Therefore, similar to heuristic application of convex optimization techniques to nonconvex problems, we utilize submodular optimization algorithms for architecture search algorithm design.

Main contributions and results. We adapt a simple greedy algorithm with performance guarantees for submodular optimization – in this case, we obtain a heuristic that is used to optimize architecture search with respect to its validation set performance. This design choice actually enables achieving very favorable performance relative to state-of-the-art approaches using **orders of magnitude** less computation resources, while concurrently being competitive with other recent efficient search methods, such as ENAS, ProxylessNAS, DARTS and FBNet [6, 27, 30, 38]. This greedy search is also far simpler to implement than many existing search methods: no controllers [3, 30, 43, 44], hypernetworks [4], or performance predictors [25] are required.

It can be easily extended to a number of different resource bounded architecture search applications.

Our **contributions** are: **(1)** We formulate Resource Constrained Architecture Search (RCAS) as a set function optimization and design a heuristic based on ideas known to be effective for constrained submodular optimization. **(2)** We describe schemes by which the algorithm can easily satisfy constraints imposed due to specific deployment platforms (# of FLOPs, power/energy). **(3)** On the results side, we achieve remarkable architecture search efficiency. On CIFAR-100, our algorithm takes only **two days on 2 GPUs** to obtain an architecture with similar complexity as MobileNetV2. On ImageNet, our algorithm runs in 8 days on 2 GPUs and identifies an architecture similar in complexity and performance to MobileNetV2. **(4)** We show that the architectures learned by RCAS on CIFAR-100 can be directly transferred to ImageNet with good performance.

2. Preliminaries

2.1. Architecture Search

We aim to include computational constraints in the design of mobile Convolutional Neural Networks (MCNNs). Consider a finite computational budget available for a specific prediction task, B. The problem of designing efficient MCNNs can be viewed as seeking the most accurate CNN model that fits within said budget B:

$$\max_{cnn} \quad f(cnn) \quad \text{subject to} \quad Cost(cnn) \le B \quad (1)$$

where f denotes a score function, typically the validation accuracy on a held out set of samples $X_{valid} := (\mathbf{x}_i, y_i)_{i=1}^m$, i.e., $f(cnn) = \frac{1}{m} \sum_{i=1}^m \mathbb{1}_{[cnn_w(\mathbf{x}_i)=y_i]}$. The parameters w are learned with a training set X_{train}, often with a surrogate cross-entropy loss and stochastic gradient descent.

Resource constraints. Model cost is typically measured [10] in two ways, analogous to algorithmic space and time complexities: first by the number of parameters, and second by the number of computations, multiply-adds (MAdds) or floating point operations per second (FLOPs). With this in mind, we can more concretely define the budget constraint. For a budget B, assume there is a corresponding maximum number of MAdds B_m and number of parameters B_p:

$$\max_{cnn} \quad f(cnn) \quad (2)$$
$$\text{s.t.} \quad MAdds(cnn) \le B_m, \quad Param(cnn) \le B_p$$

where $MAdds(cnn)$ denotes the number of multiply-adds and $Param(cnn)$ denotes the number of parameters of the model cnn. Typical hardware constraints are given in these formats, either through physical memory specifications or processor speeds and cache limits.

Modern CNNs are built by constructing a sequence of various types of basic blocks. In network design, one may

have a variety of options in types of basic blocks, block order, and number of blocks. Examples include ResNet blocks, 3x3 convolutional layers, batchnorm layers, etc. [11]. Given a large set of various basic blocks, we would like to find a subset $\mathcal{S}$ that leads to a well performing, low cost CNN. Blocks may have different associated costs, both in MAdds and in number of parameters.

2.2. Submodular Optimization

The search over the set of blocks that maximizes accuracy and remains within budget is NP-hard, even when the cost of each block with respect to number of parameters and computations is equal among all elements. However, using ideas from submodular optimization we can derive heuristics for architecture search.

Definition 1 *A function $F : 2^{\mathcal{V}} \to \mathbb{R}$, where $\mathcal{V}$ is a finite set and $2^{\mathcal{V}}$ denotes the power set of $\mathcal{V}$, is **submodular** if for every $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$ and $v \in \mathcal{V} \setminus \mathcal{B}$ it holds that*

$$F(\mathcal{A} \cup \{v\}) - F(\mathcal{A}) \geq F(\mathcal{B} \cup \{v\}) - F(\mathcal{B}) \quad (3)$$

Intuitively, submodular functions have a natural *diminishing returns* property. Adding additional elements to an already large set is not as valuable as adding elements when the set is small. A subclass of submodular functions are **monotone**, where for any $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}, F(\mathcal{A}) \leq F(\mathcal{B})$. Submodular functions enjoy a number of other properties, including being closed under nonnegative linear combinations.

Typical submodular optimization $\max_{\mathcal{S} \subseteq \mathcal{V}} F(\mathcal{S})$ involves finding a subset $\mathcal{S} \subseteq \mathcal{V}$ given some constraints on the chosen set: cardinality constraints $|\mathcal{S}| \leq r$ being the simplest. Formal optimization in these cases is NP-hard for general forms of submodular functions $F(\mathcal{S})$, and requires complex and problem-specific algorithms to find good solutions [19]. However, it has been shown that the *greedy algorithm* can obtain good results in practice.

Starting with the empty set, the algorithm iteratively adds an element v_k to the set $\mathcal{S}_{k-1}$ with the update:

$$v_k = \operatorname{argmax}_{v \in \mathcal{V} \setminus \mathcal{S}_{k-1}} F(\mathcal{S}_{k-1} \cup \{v\}) - F(\mathcal{S}_{k-1}), \quad (4)$$

where $F(\mathcal{S}_{k-1} \cup \{v\}) - F(\mathcal{S}_{k-1})$ is the marginal improvement in F of adding v to the previous set $\mathcal{S}_{k-1}$. Results in [22] show that for a nonnegative monotone submodular function, the greedy algorithm can find a set $\mathcal{S}_r$ such that $F(\mathcal{S}_r) \geq (1 - 1/e) \max_{|\mathcal{S}| \leq r} F(\mathcal{S})$. We use this result to derive heuristics for finding good architectures.

3. Submodular Neural Architecture Search

Assume N block "positions" need to be filled for building an efficient CNN, and each block *position* has L types that can be chosen from. Denote a block of type $l \in \mathcal{L} =$ $\{1, \dots, L\}$ at position $n \in \mathcal{N} = \{1, \dots, N\}$ as l_n, $\mathcal{V}$ denote a finite set with N elements (blocks) and F a set function defined over the power set of $\mathcal{V}$, $F : L^{\mathcal{V}} \to \mathbb{R}$.

Given a set of blocks, $\mathcal{S} \in L^{\mathcal{V}}$ (e.g., $\mathcal{S} = \{3_2, 1_1\}$), we build the model CNN cnn (e.g., the first block with type 1 (1_1) and the second with type 3 (3_2)), and take the validation accuracy of the CNN as the value of $F(\mathcal{S})$, $F(\mathcal{S}) = f(cnn)$. Then accuracy is exactly our map from the set of blocks to reals, and for each $\mathcal{S} \in L^{\mathcal{V}}$, a CNN is built with the selected blocks based on $\mathcal{S}$. Our total search space size would be L^N.

For each set of blocks $\mathcal{S}$, the associated cost $c(\mathcal{S})$ cannot exceed the specified budget B. Using the above notions of accuracy and cost, our goal is to solve the problem,

$$\max_{\mathcal{S} \subseteq \mathcal{V}} \quad F(\mathcal{S}) \quad \text{subject to} \quad c(\mathcal{S}) \leq B \quad (5)$$

The accuracy objective $F(\mathcal{S})$ has an important property, given we can find the best possible parameter setting (global optimum) during SGD training. F is *monotone*, i.e., $F(\mathcal{A}) \leq F(\mathcal{B})$ for any $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$. Intuitively, adding blocks (making the network larger) can only improve accuracy in general. For each $\mathcal{S}$, we can obtain its corresponding number of parameters $Param(\mathcal{S})$ or number of multiply-adds $MAdds(\mathcal{S})$. In practice, training by SGD may not reach the global optimum: in this case adding blocks may *not* improve accuracy. However, our own empirical results and those in existing literature suggest that this nondecreasing behavior is typically true, i.e., in ResNet [11].

Denote each cost-accuracy pair at global optimality as $(c_i, f_i), i = 1, \dots, L^N$, and add three virtual points, $(0, 0)$, $(c_{L^N}, 0)$, $(c_{L^N}, \max\{f_1, \dots, f_{L^N}\})$. This set can be seen as a *convex hull*, where for each cost we assign its associated positive value on the convex hull. If $F(\mathcal{S})$ can always reach its convex hull point with respect to $c(\mathcal{S})$, the accuracy objective satisfies both nonnegativity and nondecreasing monotonicity. This is exactly the diminishing returns property associated with submodularity: adding a block to a small set of selected blocks $\mathcal{A}$ improves accuracy at least as much as if adding it to a larger selected block $\mathcal{B} \supseteq \mathcal{A}$. If we let the accuracy of the CNN be 0 when no blocks are selected, $F(\emptyset) = 0$, then we immediately have that,

Lemma 1 *For any selected blocks $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$ and blocks $v \in \mathcal{V} \setminus \mathcal{B}$, it holds that*

$$F(\mathcal{A}) \geq 0 \quad (6)$$
$$F(\mathcal{A} \cup \{v\}) - F(\mathcal{A}) \geq 0 \quad (7)$$
$$F(\mathcal{A} \cup \{v\}) - F(\mathcal{A}) \geq F(\mathcal{B} \cup \{v\}) - F(\mathcal{B}) \quad (8)$$

where F reaches its convex hull point w.r.t. the cost.

Thus the neural architecture search problem can be solved as the problem of maximizing a nonnegative nondecreasing function, subject to parameter and computational budget constraints.

The simple greedy algorithm described in Section 2.2 (4) assumes equal costs for all blocks. Naturally it can perform arbitrarily badly in the case where $c(\mathcal{S}) = \sum_{v \in \mathcal{S}} c(v)$, by iteratively adding blocks until the budget is exhausted. A block containing a very large number of parameters or expensive MAdds with accuracy f_o will be preferred over a cheaper block offering accuracy $f_o - \epsilon$. To deal with these *knapsack constraints*, the marginal gain update in (4) can be modified to the *marginal gain ratio*,

$$v_k = \mathrm{argmax}_{v \in \mathcal{V} \backslash \mathcal{S}_{k-1}} \frac{F(\mathcal{S}_{k-1} \cup \{v\}) - F(\mathcal{S}_{k-1})}{c(v)} \quad (9)$$

The modified greedy algorithm with marginal gain ratio rule attempts to maximize the cost/benefit ratio, and stops when the budget is exhausted. However, even with this modification, the greedy algorithm can still perform arbitrarily poorly with respect to global optima. For example, consider the parameter costs of picking between two blocks v_1 and v_2, $\mathrm{Param}(v_1) = \epsilon$, $\mathrm{Param}(v_2) = p$. If we compute the accuracy of adding the blocks as $F(v_1) = 3\epsilon$, $F(v_2) = 2p$, then the cost/benefit ratios are $\frac{F(v_1) - F(\emptyset)}{Param(v_1)} = 3$ and $\frac{F(v_2) - F(\emptyset)}{Param(v_2)} = 2$. The modified greedy algorithm will pick block v_1. If v_1 is picked and added to current set, and we do not have enough budget to next add v_2, we only achieve accuracy 3ϵ. However, the optimal solution is to pick v_2 given any budget less than $p + \epsilon$.

Fortunately, the greedy algorithm can be further adapted. We compute $\tilde{\mathcal{S}}_{APR}$ using the accuracy parameter ratio (APR) with rule (9) and cost $Params(v)$, use the accuracy MAdds ratio (AMR) with the same and cost $MAdds(v)$ to get $\tilde{\mathcal{S}}_{AMR}$, *and* take the *uniform cost* (UC) greedy algorithm with rule Eq. (4) to get $\tilde{\mathcal{S}}_{UC}$. The new modified Cost-Effective Greedy (CEG) algorithm returns the model which achieves maximum accuracy. With these rules, CEG can still achieve a constant ratio approximation.

Theorem 1 *If F is a nondecreasing set function satisfying diminishing return property and $F(\emptyset) = 0$, then the CEG algorithm achieves a constant ratio $\frac{1}{2}(1 - \frac{1}{e})$ of the optima:*

$$\max\{F(\tilde{\mathcal{S}}_{UC}), F(\tilde{\mathcal{S}}_{APR}), F(\tilde{\mathcal{S}}_{AMR})\} \quad (10)$$

$$\geq \frac{1}{2}(1 - \frac{1}{e}) \max_{\substack{MAdds(\mathcal{S}) \leq B_m \\ Param(\mathcal{S}) \leq B_p}} F(\mathcal{S})$$

The proof is in the supplement. If we consider the time-cost of the accuracy function evaluation as $O(T)$ (T is the time to train the network by SGD), then the running time of CEG is $O(|\mathcal{V}| \Phi T)$, where $|\mathcal{V}| = LN$ is the total number of blocks, $\Phi = \max_{1 \leq k \leq LN}\{\frac{B_p}{Param(e_k)}, \frac{B_m}{MAdds(e_k)}\}$. The CEG algorithm is at most $O(T|\mathcal{V}|^2)$. While this is a sizable improvement over our initial combinatorial approach, simple technical/empirical observations allow us to scale and speed up the CEG algorithm by early stopping during training and with lazy function evaluation.

Algorithm 1 Cost-Effective Greedy CNN Search (CEG)

function CEG($\mathcal{V}$, F, B, $c(\cdot)$)
   $\mathcal{S} \leftarrow \mathrm{arg\,max}_{v \in \mathcal{V}} \frac{F(v)}{c(v)}$
   while c($\mathcal{S}$) $\leq B$ **do**
      $v^{\star} = \mathrm{arg\,max}_{v \in \mathcal{V} \backslash \mathcal{S}} \frac{F(S \cup \{v\}) - F(S)}{c(v)}$
      $S \leftarrow S \cup \{v^{\star}\}$
   end while
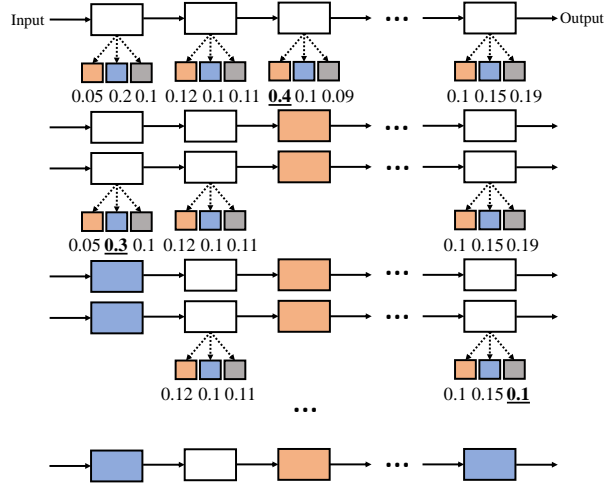   return $\mathcal{S}$
end function

Figure 1: An overview of our Lazy Cost-Effective Greedy Search. Colors correspond to different basic block types, and empty boxes represent positions in the network to be considered for filling block. Numbers below blocks indicate marginal benefit of the block filled at that position. Step 1 searches over all blocks. Step 2 selects the block with highest marginal benefit, 1_3, for filling in position 3. Step 3 updates the next highest marginal benefit with 1_3 added. Step 4 picks the highest marginal benefit block, 2_1, for filling in position 1. Step 5 updates the next highest marginal benefit with $\{1_3, 2_1\}$ added, 3_N. However, the marginal benefit of 3_N is not the highest, so we did not pick this block and the search continues. The final architecture is obtained once the budget is exhausted (unfilled blocks are replaced with identity operations).

3.1. Early Stop Training

The running time is linear with respect to function evaluation time. This time includes the full training time to learn network weights by SGD in order to achieve test set accuracy close to the global optimum. This is expensive, and so even the "low" number of $O(|\mathcal{V}|^2)$ number of CNNs to search is prohibitive in practical situations in which one may not have access to large GPU clusters or months of training time. In this case, to reduce the time to learn the weights of CNNs for accuracy function evaluation, we reduce the number of epochs and train the network with early stopping. Our experiments indicate this leads to learned architectures similar to those when trained to optimality.

3.2. Lazy Function Evaluation

With early stopping and caching, while we can quickly evaluate the accuracy function $F(\mathcal{S})$ when adding any block, we still need to make a large number of function evaluations in order to run CEG. The running time is at least linear in the total number of blocks, and in the worst case quadratic if the number of selected blocks can be as high as total number of blocks LN. If we select K blocks for building our CNN among $|\mathcal{V}|$ blocks, $O(K|\mathcal{V}|)$ evaluations are needed. The submodularity property can be exploited further to perform far fewer function evaluations.

The marginal benefit of adding any one block can be written as, for all $v \in \mathcal{V} \setminus \mathcal{S}$, $\Delta(v|\mathcal{S}) = F(\mathcal{S} \cup \{v\}) - F(\mathcal{S})$. The key idea of exploiting submodularity is that, as the set of our selected blocks grows, the marginal benefit can never increase, which means that if $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$, it holds that $\Delta(v|\mathcal{A}) \geq \Delta(v|\mathcal{B})$. Therefore we do not need to update $\Delta(v|\mathcal{S})$ for every network after adding a new block v' and we can perform *lazy evaluation*. This Lazy Cost Effective Greedy algorithm (LCEG), Alg. 2, can be described as follows: 1. Apply rule Eq. 4 to search all LN blocks and keep an ordered list of all marginal benefits with decreasing order by priority queue. 2. Select the top element of the priority queue as the first selected block at the first iteration. 3. Reevaluate $\Delta(v|\mathcal{S})$ for the top element v in the priority queue. 4. If adding block v, $\Delta(v|\mathcal{S})$ is larger than the top element in priority queue, so pick block v. Otherwise, insert it with the updated $\Delta(v|\mathcal{S})$ back in the queue. 5. Repeat steps 2-4 until the budget is exhausted. An overview of LCEG can be seen in Figure 1.

In many cases, the reevaluation of $\Delta(v|\mathcal{S})$ will result in a new but not much smaller value, and the top element will often stay at the top. In this way, we can often find the next block to add without having to recompute all blocks. This lazy evaluation thus leads to far fewer evaluations of F and means we need to train much fewer networks when trying to add one block. The final algorithm includes taking advantage of our LCEG procedure. Resource Constrained Architecture Search, RCAS is defined in Algorithm 3.

4. Experiments

It is expensive to search directly for CNN models on ImageNet and it can take several days to find a network architecture (even without resource constraints). Previous works [31, 44] suggest that we can perform our architecture search experiments on a smaller proxy task and then transfer the top-performing architecture discovered during search to the target task. However, [36] shows that it is not-trivial to find a good proxy task under constraints. Experiments on CIFAR-10 [23] and the Standford Dogs Dataset [20] demonstrate these datasets are not good proxy tasks for ImageNet when a budget constraint is taken into ac-

---

Algorithm 2 Lazy Cost-Effective Greedy Search

---

function LAZY-CEG($\mathcal{V}$, F, B_p, B_m, $c(\cdot)$)
   $\mathcal{S} \leftarrow \emptyset$
   $PriorityQueue\ Q \leftarrow PriorityQueue()$
   for all $v \in \mathcal{V}$ **do**   ▷ First iteration
      if Param(v) $\leq B_p$ AND MAdds(v) $\leq B_m$ **then**
         Q.push($\{v, \frac{F(v)}{c(v)}\}$)
      end if
   end for
   $\mathcal{S} \leftarrow \mathcal{S} \cup \{Q.pop()\}$
   while $\exists v \in Q$:Param($\mathcal{S} \cup \{v\}$) $\leq B_p$ AND MAdds($\mathcal{S} \cup \{v\}$) $\leq B_m$ **do**   ▷ Lazy update
      $v^\star \leftarrow Q.pop()$
      if $v^\star \in \mathcal{V} \setminus \mathcal{S}$ **then**
        if $\frac{F(S \cup \{v^\star\}) - F(S)}{c(v^\star)} \geq \frac{F(S \cup \{Q.top()\}) - F(S)}{c(v)}$ **then**
          $S \leftarrow S \cup \{v^\star\}$
        else
          Q.push($\{v^\star, \frac{F(S \cup \{v^\star\}) - F(S)}{c(v^\star)}\}$)
        end if
      end if
   end while
   return $\mathcal{S}$
end function

Algorithm 3 Resource Constrained Architecture Search (RCAS)

---

function **RCAS**($\mathcal{V}$, F, B_p, B_m)
   $\tilde{\mathcal{S}}_{UC} \leftarrow$ LAZY-CEG($\mathcal{V}$, F, B_p, B_m, $const(\cdot)$)
   $\tilde{\mathcal{S}}_{APR} \leftarrow$ LAZY-CEG($\mathcal{V}$, F, B_p, B_m, Param($\cdot$))
   $\tilde{\mathcal{S}}_{AMR} \leftarrow$ LAZY-CEG($\mathcal{V}$, F, B_p, B_m, MAdds($\cdot$))
   return $\arg\max\{\tilde{\mathcal{S}}_{UC}, \tilde{\mathcal{S}}_{APR}, \tilde{\mathcal{S}}_{AMR}\}$
end function

---

count. RCAS shines in this problem setting, allowing us to perform our architecture search on a much larger dataset, CIFAR-100. Indeed, we also can directly perform our architecture search on the ImageNet training set, to directly evaluate and compare the architectures learned. In these large scale cases, we train for fewer steps on CIFAR-100 and ImageNet.

Our experiments on CIFAR-100 and ImageNet have two steps: architecture search and architecture evaluation. In the first step, we search for block architectures using RCAS and pick the best blocks based on their validation performance. In the second step, the picked blocks are used to build CNN models, which we train from scratch and evaluate the performance on the test set. Finally, we extend the network architecture learned from CIFAR-100 and evaluate the performance on ImageNet, comparing with the architecture learned through RCAS applied directly to ImageNet.

Layer	Input	Operator	Output
Group-wise expansion layer	$H \times W \times C_1$	1x1 gconv2d group= g_e, ReLU6	$H \times W \times (C_1 \times t)$
Depthwise layer	$H \times W \times (C_1 \times t)$	3x3 dwise stride = s, ReLU6	$H/s \times W/s \times (C_1 \times t)$
Group-wise projection layer	$H/s \times W/s \times (C_1 \times t)$	linear 1x1 gconv2d group = g_p	$H/s \times W/s \times C_2$

Table 1: Parameter and performance efficient depth-wise based basic blocks used in Resource Constrained Architecture Search. The structure of basic blocks derive from depth-wise based MobileNetV2 blocks, changing the expansion factor t and using group convolutions. Basic blocks transform from C_1 to C_2 channels with expansion factor t, expansion group g_e and projection group g_p with stride s.

4.1. Architecture Search

As our main purpose is to look for low cost mobile neural networks, the following basic blocks (using depthwise convolution extensively) are included for architecture search, varying from MobileNetV2 blocks by using different expansion ratios and group convolutions for expansion and projection (see Table 1). Each type of block is shown in Figure 2 and consists of different types of layers. We have $L = 6$ different basic blocks to pick from and $N = 36$ number of positions can be filled for building networks under parameter and MAdds constraints for our low cost architecture search. An overview of picking basic blocks to fill positions can be seen in Figure 1. During architecture search, only one basic block can be picked to fill a position, otherwise the procedure will not insert any block. The input
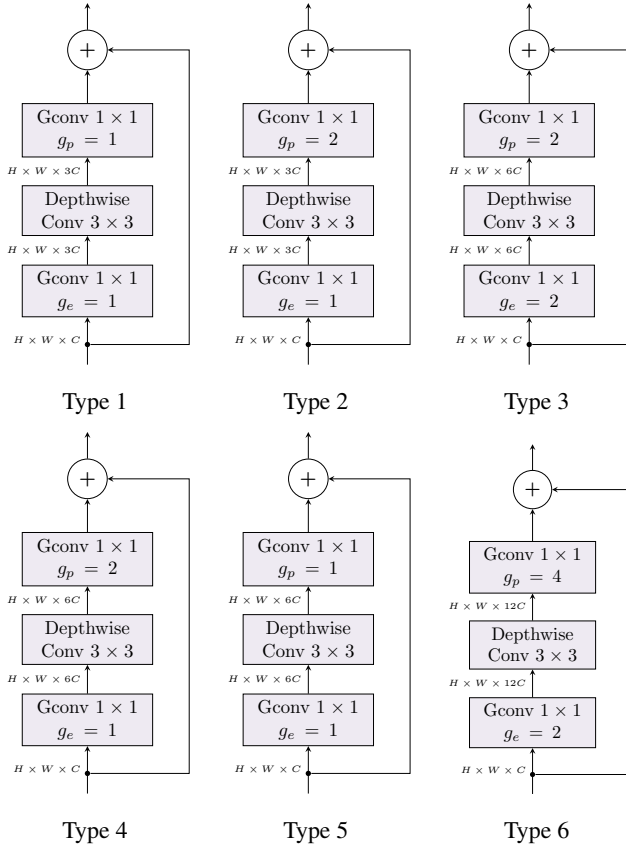
for the kth picked block is the output of the $(k-1)^{th}$ picked block, naturally stacking and forming the network.

Standard practice in architecture search [31] suggests a separate validation set be used to measure accuracy: we randomly select 10 images per class from the training set as the fixed validation set. During architecture search by RCAS, we train each possible model (adding one basic block with different options at every position) on 10 epochs of the proxy training set using an aggressive learning rate schedule, evaluating the model through the accuracy set function $F(\mathcal{S})$ on the fixed validation set. We use stocastic gradient descent (SGD) to train each possible model for training with *nesterov* momentum set to 0.9. We employ a multi-step learning rate schedule with initial learning rate 0.1 and multiplicative decay rate $g = 0.1$ at epochs $4, 7$ and 9 for fast learning. We set the regularization parameter for weight decay to $4.0e^{-5}$, following InceptionNet [34].

4.2. Architecture Evaluation

Applying our RCAS, we obtain a selected architecture under the given parameter and MAdds budget. To evaluate the selected architecture, we train it from scratch and evaluate the computational efficiency and accuracy on the test set. Given mobile budget constraints, we compare our selected architecture with mobile baselines, MobileNetV2 and ShuffleNet [13, 32]. We take the parameter number and MAdds as the computational efficiency and report the latency and model size on a typical mobile platform (iPhone 5s). We evaluate the performance of our selected architecture on both the CIFAR-100 dataset and ImageNet dataset. Following prior work [34], we use the validation dataset as a proxy for test set ImageNet classification accuracy. Our RCAS algorithm and subsequent Resource Constrained CNN (RC-Net) are implemented using PyTorch [29]. We use built-in
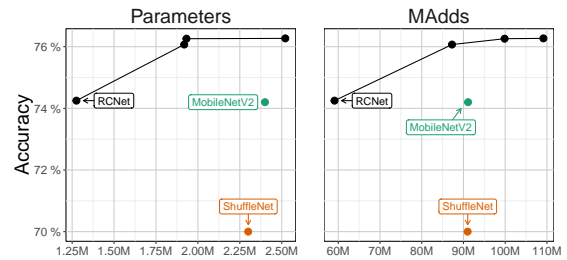

Figure 2: The 6 types depthwise-based basic blocks used in RCAS.

Figure 3: Architectures along the search path over constraints are stored. RCAS uses ($\sim 45\%$) fewer parameters and ($\sim 35\%$) fewer MAdds to achieve similar accuracy.

Architecture	Top-1 Accuracy	Top-5 Accuracy	Parameters	MAdds	Search Method	Cost (GPU Days)
MobileNetv2	74.2	93.3	2.4M	91.1M	manual	-
ShuffleNet(1.5)	70.0	90.8	2.3M	91.0M	manual	-
NASNet-A	70.0	86.0	3.61M	132.0M	RL	1800
DARTS (searched on **CIFAR-10**)	75.9	93.8	3.40M	198.0M	gradient-based	4
RCNet (searched on **CIFAR-100**)	**76.1**	**94.0**	**1.92M**	**87.3M**	RCAS	**2**

Table 2: Comparison with state-of-the-art image classifiers on CIFAR-100. Our searched model performs significantly better than other manual methods. Given MobileNetV2 parameter and MAdds constraints, our model still outperforms DARTS with $\sim 44\%$ fewer parameters and $\sim 50\%$ fewer MAdds. Additionally, both RCAS and RCNet run on CIFAR-100 much faster than DARTS.
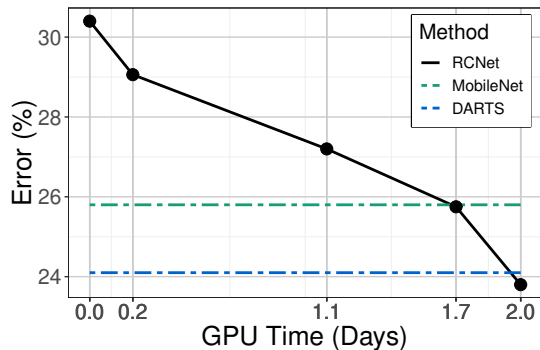

Figure 4: Search progress of RCAS on CIFAR-100. We keep track of the most recent architecture over time, with MobileNetV2 and the final DARTS architecture as reference.

1×1 convolution and group convolution implementations. The methods are easy to reproduce in other deep learning frameworks such as Caffe [17] and TensorFlow [1], using built-in layers as long as 1×1 standard convolutions and group convolutions are available. For CIFAR-100, we use similar parameter settings as during search, with the exception of a maximum number of epochs of 200 and a learning rate schedule updating at epochs 60, 120, and 180. For ImageNet, we use an initial learning rate 0.01, and decay at epochs 200 and 300 with maximum training epoch 400. We use the same default data augmentation module as in ResNet for fair comparisons. Random cropping and horizontal flipping are used for training images, and images are resized or cropped to 224×224 pixels for ImageNet and 32×32 pixels for CIFAR-100. At test time, the trained model is evaluated on center crops.

4.3. CIFAR-100

The CIFAR-100 dataset [23] consists of $50,000$ training RGB images and $10,000$ test RGB images with 100 classes. The image size is 32×32. We take the state-of-the-art mobile network architecture MobileNetV2 as our baseline. All the hyperparameters and preprocessing are set to be the same in order to make a fair comparison. The 32×32 images are converted to 40×40 with zero-padding by 4 pixels and then randomly cropped to 32×32. Horizontal flipping and RGB mean value substraction are applied as well.

We evaluate the top-1 and top-5 accuracy and compare MAdds and the number of parameters for benchmarking.

The performance comparison between baseline models and our RCNet is shown in Table 2. RCNet achieves significant improvements over MobileNetV2 and ShuffleNet with fewer computations and fewer parameters. Our RCNet achieves similar accuracy with MobileNetV2 with $\sim 45\%$ parameter reduction and $\sim 35\%$ computation reduction (see Figure 3). With $\sim 20\%$ fewer parameters, RCNet achieves a 1.9% accuracy improvement. Search progress on CIFAR-100 can be seen in Figure 4.

Does submodular property with early stopping hold? On CIFAR-100, we find that our procedure displays monotonicity and diminishing returns over accuracy averaged over 500 networks built on random block additions (out of our 6 types) with early stopping: 56.96%, 57.98%, 58.81%, 59.47%, 59.94%, 60.35%, 60.70%, 60.72%. It may be the case that with early stopping we may not be identifying the absolute best block at a given step in the algorithm, but as demonstrated, empirically we find that the final architecture identified is competitive with state-of-the-art.

Does block diversity help? To show the gains from block diversity, we use our method to search *only* with block type 5. On CIFAR-100, we obtain a top-1 accuracy of 75.8%, worse than the original searched model with 76.1%, under the same constraints. This points to the importance of block diversity in resource constrained CNN models.

Does our search procedure help? To show the gains from our search procedure, we use the searched solution and replace the last several blocks with random blocks. With ~ 1.8M parameters and ~ 73.0M MAdds, the random-block solution only yields 74.9% top-1 accuracy on CIFAR-100, as opposed to the original searched solution, 76.1%. Randomly adding one more block only gives 75.1% top-1 accuracy with ~ 2.9M parameters and ~ 88M MAdds.

4.4. ImageNet

There are $1.28M$ training images and $50K$ validation images from $1,000$ classes in the ImageNet dataset [7]. Following the procedure for CIFAR-100, we learn the RCNet architecture on the training set and report top-1 and top-5 validation accuracy with the corresponding parameters and MAdds of the model. The details of our learned RCNet architecture can be seen in the supplement. We compare our models with other low cost models (e.g. $\sim 3.4M$ parameters and $\sim 300M$ MAdds) in Table 3. RCNet achieves con-

Architecture	Top-1 Accuracy	Top-5 Accuracy	Parameters	MAdds	Search Method	Cost (GPU Days)
InceptionV1 [35]	69.8	89.9	6.6M	1448M	manual	-
MobileNetV1 [15]	70.6	88.2	4.2M	575M	manual	-
ShuffleNet(1.5) [42]	71.5	-	3.4M	292M	manual	-
CondenseNet(G=C=4) [16]	71.0	90.0	2.9M	274M	manual	-
MobileNetV2 [32]	72.0	91.0	3.4M	300M	manual	-
ANTNet [39]	73.2	91.2	3.7M	322M	manual	-
NASNet-A [44]	74.0	91.6	5.3M	564M	RL	1800
AmoebaNet-A [31]	74.5	92	5.1M	555M	RL	1800
MNasNet-92 (searched on **ImageNet**) [36]	**74.8**	**92.1**	4.4M	388M	RL	-
Proxyless-R [6]	74.6	92.2	-	-	RL	9
PNASNet [25]	74.2	91.9	5.1M	588M	SMBO	~255
DPP-Net-Panaca (searched on **CIFAR-10**) [8]	74.0	91.8	4.8M	523M	SMBO	$8^{\dagger}$
DARTS (searched on **CIFAR-10**) [27]	73.1	91.0	4.9M	595M	gradient-based	$4^{\dagger}$
FBNet-C [38]	74.9	-	5.5M	375M	gradient-based	9
RCNet (searched on **ImageNet**)	72.2	91.0	3.4M	294M	RCAS	**8**
RCNet-B (searched on **ImageNet**)	**74.7**	**92.0**	4.7M	471M	RCAS	**9**

Table 3: Performance Results on ImageNet Classification. Given $3.4M$ parameters and $300M$ MAdds constraints, RCAS finds a model searching on ImageNet using 8 GPU days, much faster than other automated methods and RCNet performs better than "manual" methods with similar complexity. With $5M$ parameters and $500M$ MAdds constraints, RCNet-B achieves comparable accuracy to MNasNet-92 with fewer computation resources. RCNet-B outperforms DPP-Net-Panaca by 0.7% and DARTS by 1.6% with similar computation resources (The methods marked by † are searched on CIFAR-10, while our method is searched on ImageNet directly).

sistent improvement over MobileNetV2 by 0.2% Top-1 accuracy and ShuffleNet (1.5%) by 0.7%. Compared with the most resource-efficient model, CondenseNet ($G = C = 4$), our RCNet performs better with 1.2% accuracy gain.

Using the model found with CIFAR-100, we retrain the same model with ImageNet. Performance is comparable to MobileNetV2 with similar complexity, indicating that our procedure can effectively transfer to new and challenging datasets. Here, the adapted RCNet obtains favorable results compared with state-of-the-art RL search methods, with three orders of magnitude fewer computational resources. More details can be found in the supplement. The final model constructed by RCAS includes 18 basic blocks with 6 types in the following sequence:

$$[5, 1, 4, 5, 1, 5, 2, 4, 1, 6, 4, 6, 5, 3, 3, 6, 3, 6]$$

Remarks. There are a few interesting observations to be made here. First, given the limited parameter and MAdds budget, RCAS picks very few blocks with higher cost. Additionally, picking too many high dimensional blocks decreases the performance of the model compared to selecting fewer more low dimensional blocks. Additional details regarding the search path is in the supplement. Second, with a specified maximum cost set to approximately the size of MobileNetV2, we identify a similar number of blocks. However, the blocks identified are *diverse*. Common mobile architectures consist of replications of the same type of block, e.g., MobileNetV2. This may suggest that block di-

versity is a valuable component of designing resource constrained mobile neural networks.

4.5. Inference Time

We test the actual inference speed on an iOS-based phone, iPhone 5s (1.3 GHz dual-core Apple A7 processor and 1GB RAM), and compare with baseline model MobileNetV2. To run the models, we convert our trained model to a *CoreML* model and deploy it using Apple's machine learning platform. We report the inference time of our models in Table 4 (average over 10 runs). As expected, RCNet and MobileNetV2 have similar inference times.

5. Conclusion

Mobile architecture search is becoming an important topic in computer vision, where algorithms are increasingly being integrated and deployed on heterogenous small devices. Borrowing ideas from submodularity, we propose algorithms for resource constrained architecture search. With resource constraints defined by model size and complexity, we show that we can efficiently search for neural network architectures that perform quite well. On CIFAR-100 and ImageNet, we identify mobile architectures that match or outperform existing methods, but with far fewer parameters and computations. Our algorithms are easy to implement and can be directly extended to identify efficient network architectures in other resource-constrained applications. Code/supplement is available at https://github.com/yyxiongzju/RCNet.

Model	MAdds	CoreML Model Size	Inference Time
MobileNetV2	300 M	14.7 MB	197.2 ms
RCNet	294 M	14.6 MB	183.5 ms

Table 4: Inference time running on an actual device, iPhone 5s. As expected, our searched model, RCNet, use similar average inference time as MobileNetV2 per image.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. N2n learning: Network to network compression via policy gradient reinforcement learning. *arXiv preprint arXiv:1709.06030*, 2017.

[3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[4] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

[5] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. AAAI, 2018.

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

[7] 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.

[8] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 517–531, 2018.

[9] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks.

[10] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[13] Michael G Hluchyj and Mark J Karol. Shuffle net: An application of generalized perfect shuffles to multihop lightwave networks. *Journal of Lightwave Technology*, 9(10):1386–1397, 1991.

[14] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. *arXiv preprint arXiv:1905.02244*, 2019.

[15] 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.

[16] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[17] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[18] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. Neural architecture search with bayesian optimisation and optimal transport. *arXiv preprint arXiv:1802.07191*, 2018.

[19] Yoshinobu Kawahara, Kiyohito Nagano, Koji Tsuda, and Jeff A Bilmes. Submodularity cuts and applications. In *Advances in Neural Information Processing Systems*, pages 916–924, 2009.

[20] Aditya Khosla, Nepali Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. Novel dataset for fine-grained image categorization: Stanford dogs.

[21] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? In *Proceedings of the 7th European Conference on Computer Vision*, ECCV '02, pages 65–81, 2002.

[22] Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *AAAI*, volume 7, pages 1650–1654, 2007.

[23] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[24] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[25] 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, 2018.

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

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

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

[29] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. Pytorch. *Computer software. Vers. 0.3*, 1, 2017.

[30] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[31] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

[32] 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.

[33] Richard Shin*, Charles Packer*, and Dawn Song. Differentiable neural network architecture search, 2018.

[34] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[35] 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.

[36] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

[37] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019.

[38] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[39] Yunyang Xiong, Hyunwoo J Kim, and Varsha Hedau. Antnets: Mobile convolutional neural networks for resource efficient image classification. *arXiv preprint arXiv:1904.03775*, 2019.

[40] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. *Energy*, 41:46.

[41] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[42] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[43] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[44] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2(6), 2017.