2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) | 978-1-6654-4509-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/CVPR46437.2021.01601

FBNetV3: Joint Architecture-Recipe Search using Predictor Pretraining

Xiaoliang Dai[1*], Alvin Wan[2*], Peizhao Zhang[1*], Bichen Wu[1], Zijian He[1], Zhen Wei[3], Kan Chen[1], Yuandong Tian[1], Matthew Yu[1], Peter Vajda[1], and Joseph E. Gonzalez[2]

[1]Facebook Inc., [2]UC Berkeley, [3]UNC Chapel Hill

{xiaoliangdai,stzpz,wbc,zijian,kanchen18,yuandong,mattcyu,vajdap}@fb.com

{alvinwan,jegonzal}@berkeley.edu, zhenni@cs.unc.edu

Abstract

*Neural Architecture Search (NAS) yields state-of-the-art neural networks that outperform their best manually-designed counterparts. However, previous NAS methods search for architectures under one set of training hyper-parameters (i.e., a training recipe), overlooking superior architecture-recipe combinations. To address this, we present Neural Architecture-Recipe Search (NARS) to search both (a) architectures and (b) their corresponding training recipes, simultaneously. NARS utilizes an accuracy predictor that scores architecture **and** training recipes jointly, guiding both sample selection and ranking. Furthermore, to compensate for the enlarged search space, we leverage "free" architecture statistics (e.g., FLOP count) to pretrain the predictor, significantly improving its sample efficiency and prediction reliability. After training the predictor via constrained iterative optimization, we run fast evolutionary searches in just CPU minutes to generate architecture-recipe pairs for a variety of resource constraints, called FBNetV3. FBNetV3 makes up a family of state-of-the-art compact neural networks that outperform both automatically and manually-designed competitors. For example, FBNetV3 matches both EfficientNet and ResNeSt accuracy on ImageNet with up to 2.0× and 7.1× fewer FLOPs, respectively. Furthermore, FBNetV3 yields significant performance gains for downstream object detection tasks, improving mAP despite 18% fewer FLOPs and 34% fewer parameters than EfficientNet-based equivalents.*

1. Introduction

Designing efficient computer vision models is a challenging but important problem: A myriad of applications from autonomous vehicles to augmented reality require compact models that must be highly accurate – even under constraints on power, computation, memory, and latency. The number of possible constraint and architecture combinations is combinatorially large, making manual design a near impossibility.

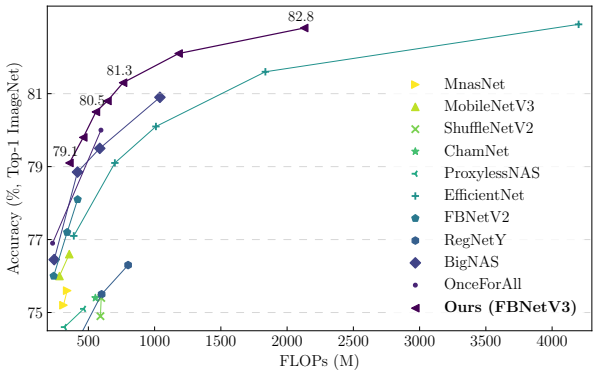

Figure 1: ImageNet accuracy vs. model FLOPs comparison of FBNetV3 with other efficient convolutional neural networks. FBNetV3 achieves 80.8% (82.8%) top-1 accuracy with 557M (2.1G) FLOPs, setting a new SOTA for accuracy-efficiency trade-offs.

In response, recent work employs neural architecture search (NAS) to design state-of-the-art efficient deep neural networks. One category of NAS is differentiable neural architecture search (DNAS). These path-finding algorithms are efficient, often completing a search in the time it takes to train one network. However, DNAS cannot search for non-architecture hyperparameters, which are crucial to the model's performance. Furthermore, supernet-based NAS methods suffer from a limited search space, as the entire supergraph must fit into memory to avoid slow convergence [5] or paging. Other methods include reinforcement learning (RL) [45], and evolutionary algorithms (ENAS) [41]. However, these methods share several drawbacks:

1. **Ignore training hyperparameters**: NAS, true to its name, searches only for architectures but not the associated training hyperparameters (i.e., "training recipe"). This ignores the fact that different training recipes may

*Equal contribution

978-1-6654-4509-2/21/$31.00 ©2021 IEEE
DOI 10.1109/CVPR46437.2021.01601

Model \ Training	Recipe-1	Recipe-2
ResNet18 (1.4x width)	**70.8%**	73.3%
ResNet18 (2x depth)	70.7%	**73.8%**

Table 1: Different training recipe could switch the ranking of architectures. ResNet18 1.4x width and 2x depth refer to ResNet18 with 1.4 width and 2.0 depth scaling factor, respectively. Training recipe details can be found in Appendix A.1.

drastically change the success or failure of an architecture, or even switch architecture rankings (Table 1).

2. **Support only one-time use**: Many conventional NAS approaches produce one model for a specific set of resource constraints. This means that deploying to a line of products, each with different resource constraints, requires rerunning NAS once for each resource setting. Alternatively, model designers may search for one model and scale it suboptimally, using manual heuristics, to fit new resource constraints.

3. **Prohibitively large search space to search:** Naïvely including training recipes in the search space is either impossible (DNAS, supernet-based NAS) or prohibitively expensive, as architecture-only accuracy predictors are already computationally expensive to train (RL, ENAS).

To overcome these challenges, we propose Neural Architecture-Recipe Search (NARS) to address the above limitations. Our insight is three-fold: (1) To support re-use of NAS results for multiple resource constraints, we train an accuracy predictor, then use the predictor to find architecture-recipe pairs for new resource constraints in just CPU minutes. (2) To avoid the pitfalls of architecture-only or recipe-only searches, this predictor scores both training recipes and architectures simultaneously. (3) To avoid prohibitive growth in predictor training time, we pretrain the predictor on proxy datasets to predict architecture statistics (*e.g.*, FLOPs, #Parameters) from architecture representations. After sequentially performing predictor pretraining, constrained iterative optimization, and predictor-based evolutionary search, NARS produces generalizable training recipes and compact models that attain state-of-the-art performance on ImageNet, outperforming all the existing manually designed or automatically searched neural networks. We summarize our contributions below:

1. **Neural Architecture-Recipe Search**: We propose a predictor that jointly scores both training recipes and architectures, the first joint search, over *both* training recipes and architectures, at scale to our knowledge.

2. **Predictor pretraining**: To enable efficient search over this larger space, we furthermore present a pretraining technique, significantly improving the accuracy predictor's sample efficiency.

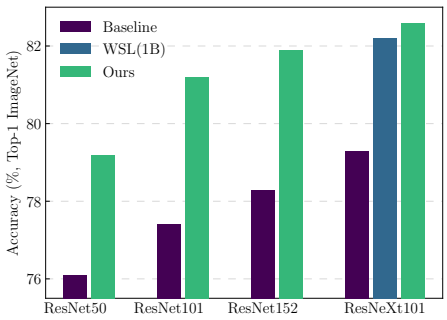

Figure 2: Accuracy improvement on existing architectures with the searched training recipe. WSL refers to the weakly supervised learning model using 1B additional images [33].

3. **Multi-use predictor**: Our predictor can be used in fast evolutionary searches to quickly generate models for a wide variety of resource budgets in just CPU minutes.

4. **State-of-the-art ImageNet accuracy** per FLOP for the searched FBNetV3 models. For example, our FBNetV3 matches EfficientNet accuracy with as low as 49.3% fewer FLOPs, as shown in Fig. 1.

5. **Generalizable training recipe**: NARS's recipe-only search achieves significant accuracy gains across various neural networks, as illustrated in Fig. 2. Our ResNeXt101-32x8d achieves 82.6% top-1 accuracy; this even outperforms its weakly-supervised counterpart trained on 1B extra images [33].

2. Related work

Work on compact neural networks began with manual design, which can be divided into architectural and non-architectural modifications.

Manual architecture design: Most early work compresses *existing* architectures. One method is pruning [12, 7, 60, 4], where either layers or channels are removed according to certain heuristics. However, pruning either considers only one architecture [13] or can only sequentially search smaller and smaller architectures [58]. This limits the search space. Other work designs *new* architectures from the ground up, using new operations that are cost-friendly. This includes convolutional variants like the depthwise convolutions in MobileNet; inverted residual blocks in MobileNetV2; activations such as hswish in MobileNetV3 [18, 42, 17]; and operations like shift [52] and shuffle [32]. Although many of these are still used in state-of-the-art neural networks, manually-designed *architectures* have been superseded by automatically-searched counterparts.

Non-architectural modifications: A number of network compression techniques include low-bit quantization [12]

to as few as two [65] or even one bit [21]. Other work downsamples input non-uniformly [53, 57, 34] to reduce computational cost. These methods can be combined with architecture improvements for roughly additive reduction in latency. Other non-architecture modifications involve hyperparameter tuning, including tuning libraries from the pre-deep-learning era [2]. Several deep-learning-specific tuning libraries are also widely used [26]. A newer category of approaches automatically searches for the optimal combination of data augmentation strategies. These methods use policy search [6], population-based training [16], Bayesian-based augmentation [47], or Bayesian optimization [22].

Automatic architecture search: NAS automates neural network design for state-of-the-art performance. Several of the most common techniques for NAS include reinforcement learning [66, 45], evolutionary algorithms [41, 40, 59], and DNAS [30, 51, 48, 11, 56]. DNAS trains quickly with few computational resources but is limited by search space size due to memory constraints. Several works seek to address this issue, by training only subsets at a time [5] or by introducing approximations [48]. However, its flexibility is still less than that of rival reinforcement learning methods and evolutionary algorithms. In turn, these prior works search for only the model architecture [29, 50, 49, 43, 4] or perform neural architecture-recipe search searches on small-scale datasets (e.g., CIFAR) [1, 62]. By contrast, our NARS jointly searches both architectures and training recipes on ImageNet. To compensate for the larger search space, we (a) introduce a predictor pretraining technique to improve the predictor's rate of convergence and (b) employ predictor-based evolutionary search to design architecture-recipe pairs in just CPU minutes, for any resource constraint setting–outperforming the predictor's highest-ranked candidate before evolutionary search significantly. We also note prior work that generates a family of models with negligible or no cost after one search [11, 59, 31].

3. Method

Our goal is to find the most accurate architecture and training recipe combination, to avoid overlooking architecture-recipe pairs as prior methods have. However, the search space is typically combinatorially large, making exhaustive evaluation an impossibility. To address this, we train an accuracy predictor that accepts architecture and training recipe representations (Sec 3.1). To do so, we employ a three-stage pipeline (Algorithm 1): (1) Pretrain the predictor using architecture statistics, significantly improving its accuracy and sample efficiency (Sec 3.2). (2) Train the predictor using constrained iterative optimization (Sec 3.3). (3) For each set of resource constraints, run predictor-based evolutionary search in just CPU minutes to produce high-accuracy architecture-recipe pairs (Sec 3.4).

Algorithm 1: Three-stage Constraint-aware Neural Architecture-Recipe Search

Input:
Ω: the designed search space;
n: size of candidate pool Λ in constrained iterative optimization;
m: the number of DNN candidates ($\mathcal{X}$) to train in each iteration;
T: the number of batches for constrained iterative optimization;
Stage 1: Pretrain Predictor
Generate a pool Λ with n samples with QMC sampling from the search space Ω;
Pretrain accuracy predictor u with architecture statistics;
Stage 2: Train Predictor (Constrained Iterative Optimization):
Initialize $\mathcal{D}_0$ as $\emptyset$;
for $t = 1, 2, ..., T$ **do**
  Find a batch of the most promising DNN candidates $\mathcal{X} \subset \Lambda$ based on predicted scores, $u(x)$;
  Evaluate all $x \in \mathcal{X}$ by training in parallel;
  if $t = 1$: Determine early stopping criteria;
  Update the dataset:
  $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(x_1, acc(x_1)), (x_2, acc(x_2)), ...\}$;
  Retrain the accuracy predictor u on $\mathcal{D}_t$;
end
Stage 3: Use Predictor (Predictor-Based Evolutionary Search)
Initialize $\mathcal{D}^\star$ with p best-performing samples in $\mathcal{D}_T$ and q randomly generated samples paired with scores predicted by u;
Initialize $s^\star$ with the best score in $\mathcal{D}^\star$; set $s_0^\star = 0$; set $\epsilon = 10^{-6}$;
while $(s^\star - s_0^\star) > \epsilon$ **do**
  for $x \in \mathcal{D}^\star$ **do**
    Generate a set of children $\mathcal{C} \subset \Omega$ subject to resource constraints, by the adaptive genetic algorithm [8];
  end
  Augment $\mathcal{D}^\star$ with $\mathcal{C}$ paired with scores predicted by u;
  Select top K candidates from the augmented set to update $\mathcal{D}^\star$;
  Update the previous best ranking score by $s_0^\star = s^\star$;
  Update the current best ranking score $s^\star$ by the best predicted score in $\mathcal{D}^\star$.
end
Result: $\mathcal{D}^\star$, i.e., all the top K best samples with their predicted scores.

3.1. Predictor

Our predictor aims to predict accuracy given representations of an architecture and a training recipe. The architecture and training recipe are encoded using one-hot categorical variables (e.g., for block types) and min-max normalized continuous values (e.g., for channel counts). See the full search space in Table 2.

The predictor architecture is a multi-layer perceptron (Fig. 3) consisting of several fully-connected layers and two heads: (1) An auxiliary "proxy" head, used for pretraining the encoder, predicts architecture statistics (e.g., FLOPs and #Parameters) from architecture representations; and (2) the accuracy head, fine-tuned in constrained iterative optimization (Sec 3.3), predicts accuracy from joint representations of the architecture and training recipe.

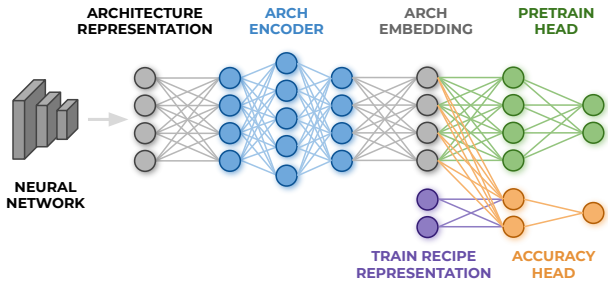

Figure 3: Pretrain to predict architecture statistics (top). Train to predict accuracy from architecture-recipe pairs (bottom)

3.2. Stage 1: Predictor pretraining

Training an accuracy predictor can be computationally expensive, as each training label is ostensibly a fully-trained architecture under a specific training recipe. To alleviate this, our insight is to first pretrain on a proxy task. The pretraining step can help the predictor to form a good internal representation of the inputs, therefore reducing the number of accuracy-architecture-recipe samples needed. This can significantly mitigate the search cost required.

To construct a proxy task for pretraining, we can use "free" source of labels for architectures: namely, architecture statistics like FLOPs and numbers of parameters. After this pretraining step, we transfer the pretrained embedding layer to initialize the accuracy predictor (Fig. 3). This leads to significant improvements in the final predictor's sample efficiency and prediction reliability. For example, to reach the same prediction mean square error (MSE), the pretrained predictor only requires 5× less samples than its counterpart without pretraining, as shown in Fig. 4(e). As a result, predictor pretraining reduces the overall search cost substantially.

3.3. Stage 2: Training predictor

In this step, we train the predictor and generate a set of high-promise candidates. As mentioned prior, our goal is to find the most accurate architecture and training recipe combination under given resource constraints. We thus formulate the architecture search as a constrained optimization problem:

$$\max_{(A,h)\in\Omega} \; acc(A,h), \; \text{s.t.} \; g_i(A) \leqslant C_i, \quad i = 1, ..., \gamma \qquad (1)$$

where A, h, and Ω refer to the neural network architecture, training recipe, and designed search space, respectively. acc maps the architecture and training recipe to accuracy. $g_i(A)$ and γ refer to the formula and count of resource constraints, such as computational cost, storage cost, and run-time latency.

Constrained iterative optimization: We first use Quasi Monte-Carlo (QMC) [37] sampling to generate a sample pool of architecture-recipe pairs from the search space. Then, we train the predictor iteratively: We (a) shrink the candidate space by selecting a subset of favorable candidates based on predicted accuracy, (b) train and evaluate the candidates using an early-stopping heuristic, and (c) fine-tune the predictor with the Huber loss. This iterative shrinking of the candidate space avoids unnecessary evaluations and improves exploration efficiency.

- **Training candidates with early-stopping**. We introduce an early stopping mechanism to cut down on the computational cost of evaluating candidates. Specifically, we (a) rank samples by both early-stopping and final accuracy after the first iteration of constrained iterative optimization, (b) compute the rank correlation, and (c) find the epoch e where correlation exceeds a particular threshold (e.g., 0.92), as shown in Fig. 5.

  For all remaining candidates, we train (A, h) only for e epochs to approximate $acc(A, h)$. This allows us to use much fewer training iterations to evaluate each queried sample.

- **Training the predictor with Huber loss**. After obtaining the pretrained architecture embedding, we first train the predictor for 50 epochs with the embedding layer frozen. Then, we train the entire model with reduced learning rate for another 50 epochs. We adopt the Huber loss to train the accuracy predictor, i.e., $\mathcal{L} = 0.5(y-\hat{y})^2$ if $|y-\hat{y}| < 1$ else $|y-\hat{y}| - 0.5$, where y and $\hat{y}$ are the prediction and ground truth label, respectively. This prevents the model from being dominated by outliers, which shows can confound the predictor [50].

3.4. Stage 3: Using predictor

The third stage of the proposed method is an iterative process based on adaptive genetic algorithms [44]. The best-performing architecture-recipe pairs from the second stage are inherited as part of the first generation candidates. In each iteration, we introduce mutations to the candidates and generate a set of children $\mathcal{C} \subset \Omega$ subject to given constraints. We evaluate the score for each child with the pretrained accuracy predictor u, and select top K highest-scoring candidates for the next generation. We compute the gain of the highest score after each iteration, and terminate the loop when the improvement saturates. Finally, the predictor-based evolutionary search produces high-accuracy neural network architectures and training recipes.

Note that with the accuracy predictor, searching for networks to fit different use scenarios only incurs negligible cost. This is because the accuracy predictor can be substantially reused under different resource constraints, while predictor-based evolutionary search takes just CPU minutes.

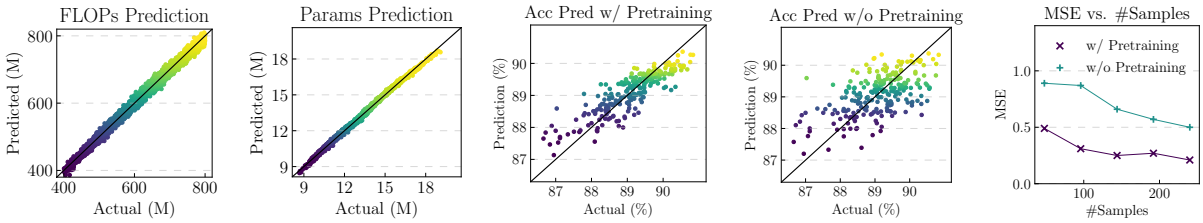

Figure 4: (a) and (b): Predictor's performance on the proxy metrics, (c) and (d): Predictor's performance on accuracy with and without pretraining, (e): Predictor's MSE vs. number of samples with and without pretraining.

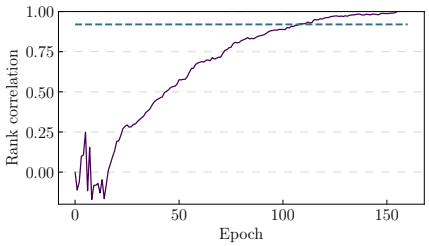

Figure 5: Rank correlation vs. epochs. Correlation threshold (cyan) is 0.92.

3.5. Predictor search space

Our search space consists of both training recipes and architecture configurations. The search space for training recipes features optimizer type, initial learning rate, weight decay, mixup ratio [63], drop out ratio, stochastic depth drop ratio [20], and whether or not to use model exponential moving average (EMA) [23]. Our architecture configuration search space is based on the inverted residual block [42] and includes input resolution, kernel size, expansion, number of channels per layer, and depth, as detailed in Table 2.

In recipe-only experiments, we only tune training recipes on a fixed architecture. However, for joint search, we search both training recipes and architectures, within the search space in Table 2. Overall, the space contains 10^{17} architecture candidates with 10^7 possible training recipes. Exploring such a vast search space for an optimal network architecture and its corresponding training recipe is non-trivial.

4. Experiments

In this section, we first validate our search method in a narrowed search space to discover the training recipe for a given network. Then, we evaluate our search method for joint search over architecture and training recipes. We use PyTorch [38], and conduct our search on the ImageNet 2012 classification dataset [9]. In the search process, we randomly sample 200 classes from the entire dataset to reduce the training time. Then, we randomly withhold 10K images from the 200-class training set as the validation set.

4.1. Recipe-only search

To establish that even modern NAS-produced architecture's performance can be further improved with better training recipe, we optimize over training recipes for a fixed architecture. We adopt FBNetV2-L3 [48] (Appendix A.2) as our base architecture, which is a DNAS searched architecture that achieves 79.1% top-1 accuracy with the original training method used in [48]. We set the sample pool size $n = 20K$, batch size $m = 48$ and iteration $T = 4$ in constrained iterative optimization. We train the sampled candidates for 150 epochs with a learning rate decay factor of 0.963 per epoch during the search, and train the final model with $3\times$ slower learning rate decay (i.e., 0.9875 per epoch). We show the distribution of samples at each round as well as the final searched result in our experiments in Fig. 6, where the first-round samples are randomly generated. The searched training recipe (Appendix A.3) improves the accuracy of our base architecture by 0.8%.

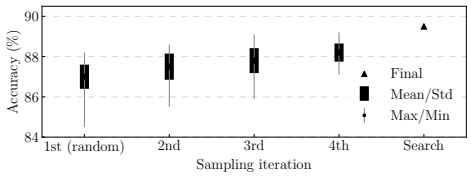

Figure 6: Illustration of the sampling and search process.

We extend the NARS-searched training recipe to other commonly-used neural networks to further validate its generality. Although the NARS-searched training recipe was tailored to FBNetV2-L3, it generalizes surprisingly well, as shown in Table 3. The NARS-searched training recipe leads to substantial accuracy gains of up to 5.7% on ImageNet. In fact, ResNet50 outperforms the baseline ResNet152 by 0.9%. ResNeXt101-32x8d even surpasses the weakly supervised learning model, which is trained with 1 billion weakly-labeled images and achieves 82.2% top-1 accuracy. Notably, it is possible to achieve even better performance by searching for specific training recipe for each neural network, which would increase the search cost.

block	k	e	c	n	s	se	act.
Conv	3	-	(16, 24, 2)	1	2	-	hswish
MBConv	[3, 5]	1	(16, 24, 2)	(1, 4)	1	N	hswish
MBConv	[3, 5]	(4, 7) / (2, 5)	(20, 32, 4)	(4, 7)	2	N	hswish
MBConv	[3, 5]	(4, 7) / (2, 5)	(24, 48, 4)	(4, 7)	2	Y	hswish
MBConv	[3, 5]	$(4, 7)^1$ / $(2, 5)^2$	(56, 84, 4)	(4, 8)	2	N	hswish
MBConv	[3, 5]	$(4, 7)^1$ / $(2, 5)^2$	(96, 144, 4)	(6, 10)	1	Y	hswish
MBConv	[3, 5]	(4, 7)	(180, 224, 4)	(5, 9)	2	Y	hswish
MBConv	[3, 5]	6	(180, 224, 4)	1	1	Y	hswish
MBPool	[3, 5]	6	1984	1	-	-	hswish
FC	-	-	1000	1	-	-	-
res	lr(10^{-3})	optim	ema	p(10^{-2})	d(10^{-1})	m(10^{-1})	wd(10^{-6})
(224, 272, 8)	(20, 30)	[RMSProp, SGD]	[true, false]	(1, 31)	(10, 31)	(0, 41)	(7, 21)

Table 2: The network architecture configuration and search space in our experiments. MBConv, MBPool, k, e, c, n, s, se, and act. refer to the inverted residual block [42], efficient last stage [17], kernel size, expansion, #Channel, #Layers, stride, squeeze-and-excitation, and activation function, respectively. res, lr, optim, ema, p, d, m, and wd refer to resolution, initial learning rate, optimizer type, EMA, dropout ratio, stochastic depth drop probability, mixup ratio, and weight decay, respectively. Expansion on the left of the slash is used in the first block in the stage, while that on the right for the rest. Tuples of three values in parentheses represent the lowest value, highest, and steps; two-value tuples imply a step of 1, and tuples in brackets represent all available choices during search. Note that lr is multiplied by 4 if the optim chooses SGD. Architecture parameters with the same superscript share the same values during the search.

Model	Top-1 Accuracy (%)		
	Original	Recipe-only	Δ
FBNetV2-L3 [48]	79.1	79.9	+0.8
AlexNet [25]	56.6	62.3	+5.7
ResNet34 [15]	73.3	76.3	+3.0
ResNet50 [15]	76.1	79.2	+3.1
ResNet101 [15]	77.4	81.2	+3.8
ResNet152 [15]	78.3	81.9	+3.6
DenseNet201 [19]	77.2	80.2	+3.0
ResNeXt101 [55]	79.3	82.6	+3.3

Table 3: Accuracy improvements with the searched training recipes on existing neural networks. Above, ResNeXt101 refers to the 32x8d variant.

4.2. Neural Architecture-Recipe Search (NARS)

Search settings Next, we perform a joint search of architecture and training recipes to discover compact neural networks. Note that based on our observations in Sec. 4.1, we shrink the search space to always use EMA. Most of the settings are the same as in the recipe-only search, while we increase the optimization iteration $T = 5$ and set the FLOPs constraint for the sample pool from 400M to 800M. We pretrain the architecture embedding layer using 80% of the sample pool which contains 20K samples, and plot the validation on the rest 20% in Fig. 4. In the predictor-based evolutionary search, we set four different FLOPs constraints: 450M, 550M, 650M, and 750M and discover four models (namely FBNetV3-B/C/D/E) with the same accuracy predictor. We further scale down and up the minimum and maximum models and generate FBNetV3-A and FBNetV3-F/G to fit more use scenarios, respectively, with compound scaling proposed in [46].

Training setup For model training, we use a two-step distillation based training process: (1) We first train the largest model (i.e., FBNetV3-G) with the searched recipe with ground truth labels. (2) Then, we train all the models (including FBNetV3-G itself) with distillation, which is a typical training technique adopted in [4][61]. Different from the in-place distillation method in [4][61], the teacher model here is the ImageNet pretrained FBNetV3-G derived from step (1). The training loss is a sum of two components: Distillation loss scaled by 0.8 and cross entropy loss scaled by 0.2. During training, we use synchronized batch normalization in distributed training with 8 nodes and 8 GPUs per node. We train the models for 400 epochs with a learning rate decay factor of 0.9875 per epoch after a 5-epoch warmup. We train the scaled models FBNetV3-A and FBNetV3-F/G with the searched training recipes for FBNetV3-B and FBNetV3-E, respectively, only increasing the stochastic depth drop ratio for FBNetV3-F/G to 0.2. More training details can be found in Appendix A.5.

Searched models We compare our searched model against other relevant NAS baselines and hand-crafted compact neural networks in Fig. 1, and list the detailed performance metrics comparison in Table 4, where we group the models by their top-1 accuracy. Among all the existing efficient models such as EfficientNet [46], MobileNetV3 [17], ResNeSt [64], and FBNetV2 [48], our searched model delivers substantial improvements on the accuracy-efficiency trade-off. For example, on low computation cost regime,

Model	Search method	Search space	Search cost (GPU/TPU hours)	FLOPs	Accuracy (%, Top-5)	Accuracy (%, Top-1)
FBNet [51]	gradient	arch	0.2K	375M	-	74.9
ProxylessNAS [5]	RL/gradient	arch	0.2K	465M	-	75.1
ChamNet [8]	predictor	arch	28K	553M	-	75.4
RegNetY [39]	pop. param.*	arch	11K	600M	-	75.5
MobileNetV3-1.25x [17]	RL/NetAdapt	arch	>91K	356M	-	76.6
EfficientNetB0 [46]	RL/scaling	arch	>91K	390M	93.3	77.3
AtomNAS [35]	gradient	arch	0.8K	363M	-	77.6
FBNetV2-L2 [48]	gradient	arch	0.6K	423M	-	78.1
FBNetV3-A	**NARS**	**arch/recipe**	10.7K	**357M**	**94.5**	**79.1**
ResNet152 [15]	manual	-	-	11G	93.8	78.3
EfficientNetB2 [46]	RL/scaling	arch	>91K	1.0G	94.9	80.3
ResNeXt101-32x8d [55]	manual	-	-	7.8G	94.5	79.3
Once-For-All [4]	gradient	-	-	595M	-	80.0
FBNetV3-C	**NARS**	**arch/recipe**	10.7K	**557M**	**95.1**	**80.5**
BigNASModel-XL [61]	gradient	arch	2.3K	1.0G	-	80.9
ResNeSt-50 [64]	manual	-	-	5.4G	-	81.1
FBNetV3-E	**NARS**	**arch/recipe**	10.7K	**762M**	**95.5**	**81.3**
EfficientNetB3 [46]	RL/scaling	arch	>91K	1.8G	95.7	81.7
ResNeSt-101 [64]	manual	-	-	10.2G	-	82.3
EfficientNetB4 [46]	RL/scaling	arch	>91K	4.2G	96.4	82.9
FBNetV3-G	**NARS**	**arch/recipe**	10.7K	**2.1G**	**96.3**	**82.8**

Table 4: Comparisons of different compact neural networks. For baselines, we cite statistics on ImageNet from the original papers. Our results are bolded. *: population parameterization. See A.6 for discussions about the training tricks and additional EfficientNet comparisons.

FBNetV3-A achieves 79.1% top-1 accuracy with only 357M FLOPs (2.5% higher accuracy than MobileNetV3-1.25x [17] with similar FLOPs). On high accuracy regime, FBNetV3-E achieves 0.2 higher accuracy with over 7× fewer FLOPs compared to ResNeSt-50 [64], while FBNetV3-G achieves the same level of accuracy as EfficientNetB4 [46] with 2× fewer FLOPs. Note that we have further improved the accuracy of FBNetV3 by using larger teacher models for distillation, as shown in Appendix A.7.

4.3. Transferability of the searched models

Classification on CIFAR-10 We further extend the searched FBNetV3 on CIFAR-10 dataset that has 60K images from 10 classes [24] to validate its transferability. Note that different from [46] that scales up the base input resolution to 224×224, we keep the original base input resolution as 32×32, and scale up the input resolutions for larger models based on the scaling ratio. We also replace the second stride-two block with a stride-one block to fit the low-resolution inputs. We don't include distillation for simplicity. We compared the performance of different models in Fig. 7. Again, our searched models significantly outperform the EfficientNet baselines.

Detection on COCO To further validate the transferability of the searched models on different tasks, we use FB-

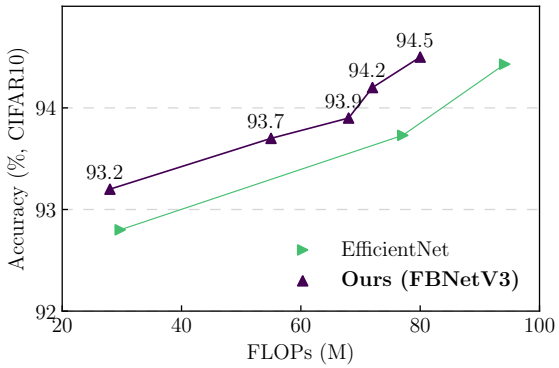

Figure 7: Accuracy vs. FLOPs comparison on the CIFAR-10 dataset.

Backbone	#Params (M)	FLOPs (G)	mAP
EfficientNetB0	8.0	3.6	30.2
FBNetV3-A	5.3	2.9	30.5
EfficeintNetB1	13.3	5.6	32.2
FBNetV3-E	10.6	5.3	33.0

Table 5: Object detection results of Faster RCNN with different backbones on COCO.

NetV3 as a replacement for the backbone feature extractor for Faster R-CNN with the conv4 (C4) backbone and compare with other models on the COCO detection dataset. We adopt most of the training settings in [54] with 3× training iterations, while use synchronized batch normalization, initialize the learning rate at 0.16, switch on EMA, reduce the non-maximum suppression (NMS) to 75, and change to learning rate schedule to Cosine after warming up. Note that we only transfer the searched architectures and use the same training protocol for all the models.

We show the detailed COCO detection results in Table 5. With similar or higher mAP, our FBNetV3 reduces the FLOPs and number of parameters by up to 18.3% and 34.1%, respectively, compared to EfficientNet backbones.

5. Ablation study and discussions

In this section, we revisit the performance improvements obtained from joint search, significance of the predictor-based evolutionary search, and the impact and generality of several training techniques.

Architecture and training recipe pairing. Our method yields different training recipes for different models. For example, we observe that smaller models tend to prefer less regularization (e.g., smaller stochastic depth drop ratio and mixup ratio). To illustrate the significance of neural architecture-recipe search, we swap the training recipes searched for FBNetV3-B and FBNetV3-E, observing a significant accuracy drop for both models, as shown in Table 6. This highlights the importance of correct architecture-recipe pairings, emphasizing the downfall of conventional NAS: Ignoring the training recipe and only searching for the network architecture fails to obtain optimal performance.

	FBNetV3-B Train recipe	FBNetV3-E Train recipe
FBNetV3-B Arch	**79.8%**	78.5%
FBNetV3-E Arch	80.8%	**81.3%**

Table 6: Accuracy comparison for the searched models with swapped training recipes.

Predictor-based evolutionary search improvements. Predictor-based evolutionary search yields substantial improvement on top of constrained iterative optimization. To demonstrate this, we compare the best-performing candidates derived from the second search stage with the final searched FBNetV3 under the same FLOPs constraints (Table 7). We observe an accuracy drop of up to 0.8% if the third stage is discarded. Thus, the third search stage, though requiring only negligible cost (i.e., several CPU minutes), is equally crucial to the final models' performance.

Model	Evolutionary Search	FLOPs	Accuracy
FBNetV3-B	Y	461M	79.8%
FBNetV3-B*	N	448M	79.0%
FBNetV3-E	Y	762M	81.3%
FBNetV3-E*	N	746M	80.7%

Table 7: Performance improvement by the predictor-based evolutionary searchsearch. *: Models derived from constrained iterative optimization.

Impact of distillation and model averaging We show the model performance on FBNetV3-G in Table 8 with different training configurations, where the baseline refers to the vanilla training without EMA or distillation. EMA brings substantially higher accuracy, especially during the middle stage of training. We hypothesize EMA intrinsically functions as a strong "ensemble" mechanism and thus improves single-model accuracy. We additionally observe distillation brings notable performance improvement. This is consistent with the observations in [4, 61]. Note since the teacher is a pretrained FBNetV3-G, FBNetV3-G is self-distilled. The combination of EMA and distillation improves the model's top-1 accuracy from 80.9% to 82.8%.

Model \ Training	Baseline	EMA	Dist*	Dist*+EMA
FBNetV3-G	80.9%	82.3%	82.2%	82.8%

Table 8: Performance improvement with EMA and distillation. *: Distillation-based training

6. Conclusion

True to their name, previous neural architecture search methods search only over architectures, using a fixed set of training hyperparameters (i.e., "training recipe"). As a result, previous methods overlook higher-accuracy architecture-recipe combinations. However, our NARS does not, being the first algorithm to jointly search over both architectures and training recipes simultaneously for a large dataset like ImageNet. Critically, NARS's predictor pretrains on "free" architecture statistics–namely, FLOPs and #Parameters–to improve the predictor's sample efficiency significantly. After training and using the predictor, the resulting FBNetV3 architecture-recipe pairs attain state-of-the-art per-FLOP accuracies on ImageNet classification. [1]

[1] **Acknowledgments**: Alvin Wan is supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE 1752814. In addition to NSF CISE Expeditions Award CCF-1730628, UC Berkeley research is supported by gifts from Alibaba, Amazon Web Services, Ant Financial, CapitalOne, Ericsson, Facebook, Futurewei, Google, Intel, Microsoft, Nvidia, Scotiabank, Splunk and VMware.

References

[1] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017. 3

[2] James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013. 3

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *ICLR*, 2019. 12

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *ICLR*, 2020. 2, 3, 6, 7, 8

[5] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *ICLR*, 2019. 1, 3, 7

[6] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, 2019. 3

[7] Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. NeST: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Trans on Computers*, 2019. 2

[8] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *CVPR*, 2019. 3, 7

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 5

[10] Piotr Dollár, Mannat Singh, and Ross Girshick. Fast and accurate model scaling. *arXiv preprint arXiv:2103.06877*, 2021. 12

[11] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *ECCV*, 2020. 3

[12] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *ICLR*, 2016. 2

[13] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015. 2

[14] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020. 12

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 6, 7

[16] Daniel Ho, Eric Liang, Ion Stoica, Pieter Abbeel, and Xi Chen. Population based augmentation: Efficient learning of augmentation policy schedules. *ICML*, 2019. 3

[17] 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. *ICCV*, 2019. 2, 6, 7

[18] 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. 2

[19] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 6

[20] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016. 5

[21] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NeurIPS*, 2016. 3

[22] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *NeurIPS*, 2018. 3

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 5

[24] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 7

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 6

[26] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *ICML AutoML Workshop*, 2018. 3

[27] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017. 12

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 12

[29] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *ECCV*, 2018. 3

[30] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ICLR*, 2019. 3

[31] Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In *European Conference on Computer Vision*, pages 35–51. Springer, 2020. 3

[32] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. *ECCV*, 2018. 2

[33] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *ECCV*, 2018. 2

[34] Dmitrii Marin, Zijian He, Peter Vajda, Priyam Chatterjee, Sam Tsai, Fei Yang, and Yuri Boykov. Efficient segmentation: Learning downsampling near semantic boundaries. In *ICCV*, 2019. 3

[35] Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. Atomnas: Fine-grained end-to-end neural architecture search. *ICLR*, 2020. 7

[36] Luke Metz, Niru Maheswaranathan, Ruoxi Sun, C Daniel Freeman, Ben Poole, and Jascha Sohl-Dickstein. Using a thousand optimization tasks to learn hyperparameter search strategies. *arXiv preprint arXiv:2002.11887*, 2020. 11

[37] Harald Niederreiter. *Random number generation and quasi-Monte Carlo methods*. SIAM, 1992. 4

[38] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NeurIPS Workshop on Autodiff*, 2017. 5

[39] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. *CVPR*, 2020. 7

[40] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019. 3

[41] 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 *JMLR*, 2017. 1, 3

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CVPR*, 2018. 2, 5, 6

[43] Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James T Kwok, and Tong Zhang. Efficient sample-based neural architecture search with learnable predictor. *arXiv*, pages arXiv–1911, 2019. 3

[44] Mandavilli Srinivas and Lalit M Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Trans. Systems, Man, and Cybernetics*, 1994. 4

[45] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. MnasNet: Platform-aware neural architecture search for mobile. *CVPR*, 2019. 1, 3

[46] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*, 2019. 6, 7

[47] Toan Tran, Trung Pham, Gustavo Carneiro, Lyle Palmer, and Ian Reid. A bayesian data augmentation approach for learning deep models. In *NeurIPS*, 2017. 3

[48] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. *CVPR*, 2020. 3, 5, 6, 7

[49] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Neural architecture search using deep neural networks and monte carlo tree search. In *AAAI*, 2020. 3

[50] Wei Wen, Hanxiao Liu, Hai Li, Yiran Chen, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. *ECCV*, 2020. 3, 4

[51] 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 *CVPR*, 2019. 3, 7

[52] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero FLOP, zero parameter alternative to spatial convolutions. *CVPR*, 2018. 2

[53] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. In *ICRA*, 2018. 3

[54] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 8, 12

[55] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 6, 7

[56] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. *ICLR*, 2019. 3

[57] Chenfeng Xu, Bichen Wu, Zining Wang, Wei Zhan, Peter Vajda, Kurt Keutzer, and Masayoshi Tomizuka. Squeezesegv3: Spatially-adaptive convolution for efficient point-cloud segmentation. *ECCV*, 2020. 3

[58] 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. In *ECCV*, 2018. 2

[59] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1829–1838, 2020. 3

[60] Hongxu Yin, Pavlo Molchanov, Zhizhong Li, Jose M Alvarez, Arun Mallya, Derek Hoiem, Niraj K Jha, and Jan Kautz. Dreaming to distill: Data-free knowledge transfer via deepinversion. *CVPR*, 2020. 2

[61] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. *ECCV*, 2020. 6, 7, 8

[62] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv preprint arXiv:1807.06906*, 2018. 3

[63] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *ICLR*, 2018. 5

[64] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Mueller, R Manmatha, et al. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020. 6, 7

[65] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *ICLR*, 2017. 3

[66] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *ICLR*, 2017. 3