Towards Regression-Free Neural Networks for Diverse Compute Platforms

Rahul Duggal* Hao Zhou Shuo Yang
Jun Fang Yuanjun Xiong Wei Xia

AWS/Amazon AI
rduggal7@gatech.edu {zhouho, shuoy, junfa, yuanjx, wxia}@amazon.com

With the shift towards on-device deep learning, ensuring a consistent behavior of an AI service across diverse compute platforms becomes tremendously important. Our work tackles the emergent problem of reducing predictive inconsistencies arising as negative flips: test samples that are correctly predicted by a less accurate model, but incorrectly by a more accurate one. We introduce **REG**ression constrained **N**eural **A**rchitecture **S**earch (REG-NAS) to design a family of highly accurate models that engender fewer negative flips. REG-NAS consists of two components: (1) A novel architecture constraint that enables a larger model to contain all the weights of the smaller one thus maximizing weight sharing. This idea stems from our observation that larger weight sharing among networks leads to similar sample-wise predictions and results in fewer negative flips; (2) A novel search reward that incorporates both Top-1 accuracy and negative flips in the architecture search metric. We demonstrate that REG-NAS can successfully find desirable architectures with few negative flips in three popular architecture search spaces. Compared to the existing state-of-the-art approach [35], REG-NAS enables $33-48\%$ relative reduction of negative flips.

1 Introduction

Consider a manufacturer that uses deep learning to detect defective products on their assembly line. To improve throughput, they use a small, low-latency, on-device model to quickly detect potentially defective products which later undergo a secondary examination by a large, and more accurate model deployed on the cloud. If there are many cases where the on-device model correctly identifies a defective product which, on second inspection the on-cloud model incorrectly believes to be qualified, the defective product may end up entering the market and damaging the reputation of the manufacturer (See Fig. 1a).

Such samples, which one model predicts correctly while the other predicts incorrectly, are called negative flips. The fraction of negative flips over the dataset size is called the negative flip rate (NFR) [35] and is used to measure regression between two models[1]. In this work, we aim to design a family of models for diverse compute budgets (*e.g.* deployed on edge, server, and cloud) that maximize Top-1 accuracy and minimizes the pairwise negative flip rate (See Fig. 1b).

* Currently at Georgia Tech. Work conducted during an internship with AWS AI.
[1] Please note this definition of regression is different from the traditional one which pertains to predicting the outcome of continuous variables.

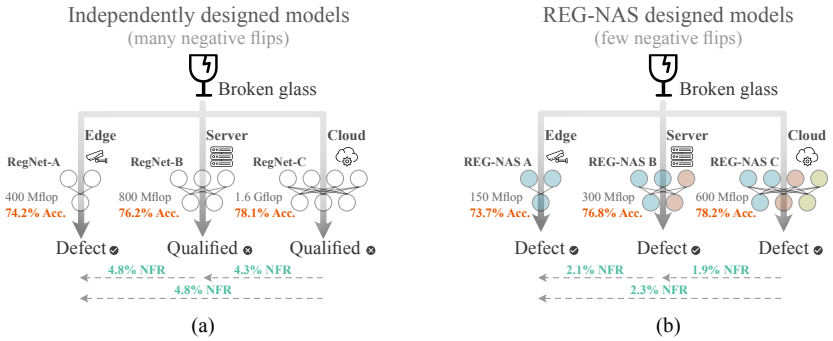

Fig. 1: Regression arises when samples that are correctly predicted by a less accurate model are *negatively flipped* or incorrectly predicted by a more accurate one. (a) Three models from the popular RegNet [23] family trained on ImageNet suffer from $4.3-4.8\%$ pairwise negative flip rate (NFR); (b) Models designed via REG-NAS achieve $1.9-2.3\%$ NFR with similar Top-1 accuracy.

Positive congruent training [35] is the first work to tackle negative flips. They propose the Focal Distillation (FD) loss, which enforces congruence with the reference model by giving more weights to samples that were correctly classified by the reference model. While [35] aims at reducing flips along the *temporal* axis, *i.e.* between different model versions trained on growing datasets, we tackle regression along the *spatial* axis, *i.e.* between models deployed on diverse compute platforms. The key difference being that the temporal setting constrains the model design to necessarily be *sequential* since the past model is fixed while the current model is designed and the future model is unknown. In contrast, the spatial setting enjoys larger flexibility wherein all models can be *jointly* adapted towards minimizing negative flips.

To quantify the difference between sequential and joint design strategies, Fig. 2a presents the case of designing a small (reference) and a large (target) model. Independently training a MobileNet-v3 reference and an EfficientNet-B1 target model using cross entropy loss achieves 73.6% and 77.08% Top-1 accuracies with more than 4.25% negative flip rate. Training the target model *sequentially* against the fixed reference model via focal distillation loss [35] reduces the NFR to 3.25% with a 0.8% drop in Top-1 accuracy. However, *jointly* designing the reference and target models using REG-NAS significantly reduces the NFR to 2.16% with a marginal 0.25% drop in the Top-1 accuracy.

REG-NAS can jointly design models by sampling sub-networks from a common super-network. We hypothesize this implicitly reduces negative flips due to large weight sharing among the sampled sub-networks (see appendix A for a detailed study). This naturally motivates the question: Could we further reduce negative flipping by maximizing weight sharing? To this end, we propose the first component of REG-NAS—a novel *architecture constraint* that maximizes weight-sharing by enabling the larger sub-networks to contain all the weights of smaller ones. Empirically, in Sec. 3.2 we show that architectures satisfying this constraint exhibit much lower negative flips while still achieving high accuracy.

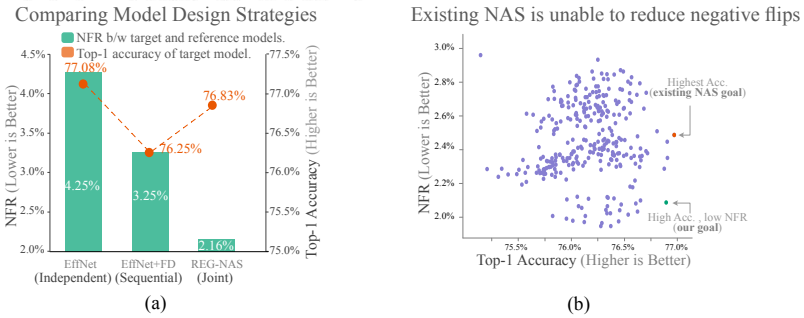

Fig. 2: (a) Independently training a MobileNet-V3 and EfficientNet-B1 as reference and target models leads to 73.6%, 77.08% Top-1 accuracies with 4.25% negative flip rate (NFR). Sequentially training the target against the reference model with the Focal Distillation loss [35] reduces NFR to the detriment of target model's accuracy. Jointly designing the reference and the target models via REG-NAS leads to lowest NFR with negligible drop in accuracy. (b) Plotting the Top-1 accuracy and NFR (measured against a common reference model) for 300 randomly sampled architectures. Observe that the NFR and Top-1 accuracy are uncorrelated which means existing NAS that maximizes Top-1 accuracy may fail to find models with the lowest negative flip rate.

While weight sharing implicitly helps reduce negative flips, it is not sufficient. Fig. 2b shows that among architectures that maximally share weights by satisfying the architecture constraint, there is a significant variance in NFR and Top-1 accuracy. Importantly, the Top-1 accuracy and NFR are *uncorrelated*. This means that existing architecture search methods that solely optimize for higher Top-1 accuracy may fail to find models with low NFR. To search for highly accurate architectures that minimize regression, in Sec. 3.3 we propose the second component of REG-NAS—a novel *search reward* that includes both Top-1 accuracy and NFR as the architecture search metric.

Together, our proposed architecture constraint and novel search reward enable REG-NAS to reduce negative flips by $33 - 48\%$ over the existing state-of-the-art FD loss [35]. We also show that REG-NAS can find low NFR subnetworks from popular architecture search spaces, demonstrating its strong generalization ability. Additionally, the models searched via REG-NAS are transitive under the NFR metric, enabling the search for N (≥ 2) models across diverse compute budgets. To summarize, our contributions are:

1. We extend the search for regression-free models from the temporal to the spatial setting. This support scenarios that simultaneously deploy a family of models across diverse compute budgets.
2. We propose two novel ideas to minimize negative flips while achieving high Top-1 accuracy: an architecture constraint that maximizes weight sharing thereby inducing similar sample-wise behavior; and a novel architecture search reward to explicitly include NFR in the architecture search metric.
3. We comprehensively evaluate our search method on ImageNet and demonstrate its generalization on three popular architecture search spaces.

2 Related Work and Background

Minimizing regression between different models has previously been explored in the temporal setting where different model versions arise during model updates [26, 27]. Generally, this task closely relates to research in continual learning [16, 5, 21], incremental learning [3, 33, 1] and model compatibility [26, 35, 13]. However, the main difference lies in how regression is measured. Traditionally, regression or forgetting was measured via the error rate [20, 17, 4], however, Yan *et.al.* [35] observe that two models having the similar error rates may still suffer from a high negative flip rate. Moreover, the methods that proposed to alleviate catastrophic forgetting are insufficient at reducing NFR. Different from these works, our work tackles NFR in the spatial setting wherein different models arise due to a diversity in deployment targets (*e.g.* edge, server, cloud). The spatial setting is fundamentally different from the temporal one since it enjoys the additional flexibility of jointly designing all architectures that can benefit from weight sharing that ultimately leads to similar sample-wise behavior and lower NFR. This key distinction calls for new methods that can leverage the flexibility of joint architecture design.

Neural architecture search (NAS) is a popular tool for automated design of neural architectures. Earlier NAS methods based on reinforcement learning [38, 18, 29], evolutionary algorithms [34, 25, 28, 24], and Bayesian optimization [15] find highly accurate networks but are notoriously computational heavy due to the independent evaluation of candidate networks. Recently One-Shot NAS approaches [22, 19, 36] successfully adopt weight sharing to alleviate the compute burden. The main innovation in this area is a super-network [14, 2, 37, 31, 30, 7, 32] that can encode and jointly train all candidate sub-networks in an architecture space. Typically, One-Shot NAS algorithms have the following two phases:

[P1] Super-network pre-training: The goal of which is to jointly optimize all sub-networks through a weight sharing super-network. The optimization process typically samples and minimizes the loss on many individual sub-networks during training. Recent methods innovate on the sampling strategy [2, 37, 31, 7, 32] or the loss formulation [30].

[P2] Sub-network searching: The goal of this phase is to find the best sub-network from the well-trained super-network under some compute measure such as flops or latency. This search is typically implemented via an evolutionary algorithm [2, 31, 30] with Top-1 accuracy as the reward function.

Our work leverages a super-network for an entirely new benefit– for reducing negative flips between networks optimized for diverse compute platforms. This is yet an unexplored problem that we believe will become increasingly important in the era of on-device AI powered by hardware optimized neural networks.

3 Regression Constrained Neural Architecture Search

Our REG-NAS algorithm relates to the super-network searching phase (**P2**) and generally works with any super-network obtained via the recent pre-training

strategies (**P1**) [2, 31, 30]. We modify the search optimization of **P2** along two directions: We propose a novel architecture constraint that maximizes weight sharing among sub-networks which is discussed in Sec. 3.2. We propose a novel search reward that incorporates both Top-1 accuracy and the NFR metric as described in Sec. 3.3. Finally, we integrate these two ideas into our REG-NAS algorithm as discussed in Sec. 3.4.

3.1 Measuring regression

We use the negative flip rate (NFR) [35] as the metric to measure the regression between two models. Given a dataset $\mathcal{D}$ that contains image and label pairs (x_i, y_i), $i \in \{1, ..., N\}$, the output of two deep CNNs ϕ_1, ϕ_2 on an input image x_i are denoted by y_i^1 and y_i^2. The NFR between ϕ_1, ϕ_2 is defined as:

$$\text{NFR}(\phi_1, \phi_2; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(y_i^1 = y_i, y_i^2 \neq y_i), \tag{1}$$

where $\mathbb{1}$ is an indicator function. The NFR measures the fraction of samples that are predicted correctly by ϕ_1 and incorrectly by ϕ_2. Note that NFR is asymmetric, so $\text{NFR}(\phi_1, \phi_2) \neq \text{NFR}(\phi_2, \phi_1)$. We select ϕ_1 as the model with a lower Top-1 accuracy and ϕ_2 as the one with the higher Top-1 accuracy. As a result, $\text{NFR}(\phi_1, \phi_2) \in [0, 1]$.

3.2 Architecture constraint to reduce negative flips

We hypothesize that sub-networks sampled from a super-network share a lot of weights, which induces similar sample-wise behavior and results in less negative flips. A natural extension of this hypothesis prompts the question: can we further reduce negative flips by maximizing weight-sharing between sub-networks? In what follows, we describe the architecture constraint that answers this question.

Consider a reference model with architecture a_r that inherits weights $W^*_{a_r}$ from a super-network. We aim to search for a target architecture a_t with weights $W^*_{a_t}$ such that the weight sharing between a_r and a_t is maximized, *i.e.*:

$$\begin{aligned} a_t \ &= \arg\max_{a \in \mathcal{A}} \mathcal{N}(W^*_a \cap W^*_{a_r}), \\ \text{s.t.} \ \ &C(a) < \tau, \end{aligned} \tag{2}$$

where $\mathcal{N}$ counts the number of weights, $\cap$ represents the intersection of weights between two architectures, C represents a performance constraint such as flops or latency and $\mathcal{A}$ represents the search space defined by the super-network. Eq. 2 specifies the architecture constraint and we refer to the process of solving it as **constrained architecture search (CAS)**.

Observe that $N(W^*_a \cap W^*_{a_r})$ is upper bounded by $N(W^*_{a_r})$, with the supremum occurring when a contains all the weights of a_r. This means Eq. 2 partitions the architecture space $\mathcal{A}$ to a space $\mathcal{A}_t$ of constrained architectures:

$$\mathcal{A}_t = \{a : W^*_{a_r} \subseteq W^*_a, \forall a \in \mathcal{A}\}, \tag{3}$$

and its complementary space $\mathcal{A}_c = \mathcal{A} - \mathcal{A}_t$ that includes target architectures that do not contain *all* the weights of the reference model. To visualize the two architecture spaces, we plot NFR *v.s.* Top-1 accuracy for 300 randomly sampled architectures from $\mathcal{A}_t$ and $\mathcal{A}_c$ in Fig. 4a. Observe that the architectures in space $\mathcal{A}_t$ generally have lower NFR and higher Top-1 accuracy compared to those in $\mathcal{A}_c$. As a result, it is easier to find architectures with lower NFR and higher Top-1 accuracy in space $\mathcal{A}_t$ than in the complete space $\mathcal{A}$ as demonstrated empirically in Sec 4.3.

Note that, not all super-networks allow for partitioning the search space according to Eq. 3. For example DARTS [19, 6] implements different operators (*e.g.* 3×3 and 5×5 conv) using a completely non-overlapping set of weights. In such search spaces, enforcing the architecture constraint trivially leads to the same architecture for the reference and target model. On the other hand, recent works [2, 31, 30] implement different operators using an overlapping set of weights *e.g.* the innermost 3×3 weights of the 5×5 kernel implements a 3×3 kernel. To enable the architecture constraint for such works, we propose the CAS recipe which specifies three rules for searching a candidate architecture a_t against a reference model a_r:

1. Add new layers within blocks of a_r (Fig. 3a).
2. Add additional channels to filters in a_r (Fig. 3b).
3. Extend kernel size for filters in a_r (Fig. 3c).

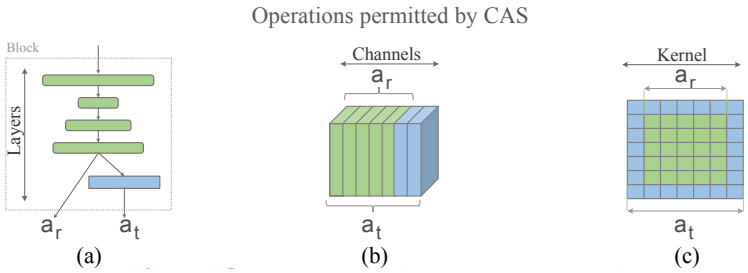

Fig. 3: Constrained Architecture Search (CAS) recipe specifies three operations for searching a target model a_t given a reference model a_r: (a) Adding new layers within an existing block of a_r; (b) Adding new channels to existing filters in a_r; (c) Extending the kernel size beyond the existing one in a_r.

3.3 Search reward to reduce NFR

While the architecture constraint implicitly helps reduce NFR through weight sharing, it is not sufficient. To illustrate this, we plot the Top-1 accuracy and the NFR of 300 randomly sampled sub-networks from $\mathcal{A}_t$ against a fixed reference model in Fig. 2b. Observe that the model with highest Top-1 accuracy has a NFR around 2.5% which is higher than many of the sampled architectures. This means that optimizing solely for the Top-1 accuracy as in existing NAS methods

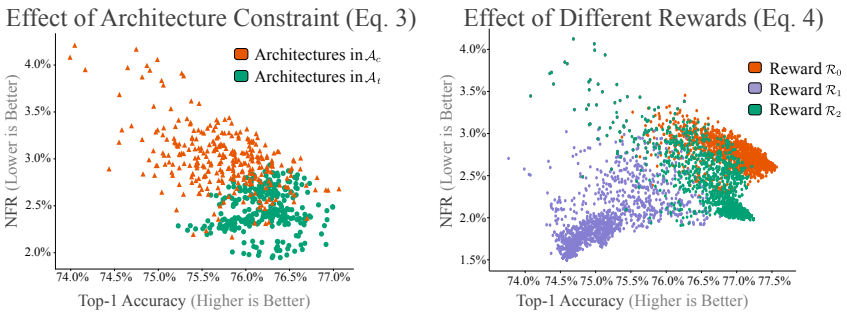

Fig. 4: Through random sampling architectures from the MobileNet-V3 search space $\mathcal{A}$ of OFA [2], we show: (a) The architecture constraint partitions $\mathcal{A}$ into A_t and A_c wherein the architectures in A_t tend to have lower NFR; (b) An evolutionary search fitted with rewards $(\mathcal{R}_0, \mathcal{R}_1, \mathcal{R}_2)$ explores different regions of the architecture space. Specifically, $\mathcal{R}_0$ leads to architectures with high Top-1 and NFR, $\mathcal{R}_1$ leads to architectures with low Top-1 and NFR while $\mathcal{R}_2$ achieves the best of both–high Top-1 accuracy and low NFR.

will not lead to architectures with small NFR. Consequently, to optimize for architectures with low NFR it is essential to include NFR in the search reward.

The observations of Fig. 2b naturally motivate the inclusion of NFR in the search reward $\mathcal{R}$ as below:

$$\begin{aligned}\mathcal{R}\left(W^*_{a_t}; W^*_{a_r},\ \mathcal{D}^{val}\right) = \lambda_1 * \text{Top-1}(W^*_{a_t};\ \mathcal{D}^{val}) \\ - \lambda_2 * \text{NFR}(W^*_{a_t}, W^*_{a_r}; \mathcal{D}^{val}),\end{aligned} \tag{4}$$

where $W^*_{a_r}, W^*_{a_t}$ represents the weights of the reference and target models while λ_1, λ_2 balance the trade off between Top-1 accuracy and NFR respectively. To illustrate the extreme effect of the two terms, we consider three different rewards obtained by setting the following values of λ_1 and λ_2.

$\mathcal{R}_0$ $(\lambda_1 = 1, \lambda_2 = 0)$: reward used by existing NAS methods that solely optimizes for high Top-1 accuracy.
$\mathcal{R}_1$ $(\lambda_1 = 0, \lambda_2 = 1)$: reward that solely optimizes for low NFR.
$\mathcal{R}_2$ $(\lambda_1 = 1, \lambda_2 = 1)$: reward that optimizes for high accuracy and low NFR.

We find that the rewards $\mathcal{R}_0$, $\mathcal{R}_1$ and $\mathcal{R}_2$ can guide evolutionary search to different regions of the architecture search space. Fig. 4b illustrates this by plotting the Top-1 accuracy and NFR of all architectures sampled during an evolutionary search fitted with the three rewards. The plot clearly shows that $\mathcal{R}_2$ guides the search to architectures with high Top-1 and low NFR. We consider $\mathcal{R}_2$ as the default choice for REG-NAS since it equally balances the Top-1 accuracy and NFR. Please refer to appendix B for a complete ablation on λ_1, λ_2.

3.4 REG-NAS: Regression constrained neural architecture search

To search for sub-networks that achieve the highest Top-1 accuracy with the minimum negative flips, we integrate the novel architecture constraint of Eq. 3 with the novel reward of Eq. 5 to propose REG-NAS, which solves the following optimization.

$$a_t = \underset{a \in \mathcal{A}}{\arg\max} \;\; \mathcal{R}\left(W_a^*; W_{a_r}^*,\; \mathcal{D}^{val}\right), \quad \text{s.t.} \;\; W_{a_r}^* \subseteq W_a^*, \text{and} \;\; C(a) < \tau \tag{5}$$

We solve the above optimization using a standard evolutionary based NAS algorithm with two modifications: the random sampling and mutate operations can only perform the operations specified by the CAS in Sec. 3.2; and the search reward uses the reward formulation of Sec. 3.3 with $\lambda_1 = 1, \lambda_2 = 1$.

4 Experiments

We begin by comparing REG-NAS against the state-of-the-art method [35] designed to reduce NFR. To understand the effectiveness of REG-NAS, we dissect the performance gains due to the novel architecture constraint and search reward. We then extend the proposed method to search for an entire family of architectures. Finally, we test the generalization of REG-NAS across: different flop budgets, super-networks, and latency constraints on diverse platforms.

4.1 Dataset, metrics and implementation details

We implement our methods using Pytorch on a system containing 8 V100 GPUs. Other details are as follows:

Dataset. We present all results on ImageNet [8]. We carve out a sub-training set of 20,000 images from the original training set to evaluate the rewards and update batchnorm statistics for each sub-network [2]. The final results are presented on the original ImageNet validation set.

Evolutionary Search. To search for a sub-network, we run the evolutionary search with the following hyper-parameters: 20 generations, 100 population size, 0.1 mutate probability, 0.5 mutation ratio and 25% Top-K architectures moving into the next generation. Overall, our search evaluates 1,525 architectures in each super-network. Following the setting of the original work, for MobileNet-V3 and ResNet-50 search spaces of OFA [2], the rewards are computed on the sampled sub-training set, while for FBNet-V3 super-network of AttentiveNAS [31] and AlphaNet [30], the search is performed on the ImageNet validation set as suggested by [31, 30].

Notation. We refer to different models following the consistent notation AA-BB-CC. The first initial "AA" refer to the super-network, the middle initial "BB" refers to the search reward and the final initial "CC" refers to the performance constraint such as flops or latency. We use CAS to represent constrained architecture search discussed in Sec. 3.2.

4.2 Regression Constrained Search

Given a small reference model, we aim to find a larger target model that, under a fixed compute budget, maximizes the Top-1 accuracy while minimizing NFR. Table 1 illustrates this task with comparative results from the MobileNet-V3 and ResNet-50 search spaces of OFA [2].

	Model	Flops	Top-1 ↑(%)	NFR ↓ (%)
ref	MB-$\mathcal{R}_0$-150 [2]	149	73.70	-
baseline	EffNet	385	77.08	4.25
PCT	EffNet+FD [35]	385	76.25	3.25
wt. share	MB-$\mathcal{R}_0$-300 [2]	298	77.11	2.51
REG-NAS	MB-($\mathcal{R}_2$+CAS)-300	294	76.83	2.16

(a) MobileNet-V3 search space of OFA

	Model	Flops	Top-1 ↑(%)	NFR ↓ (%)
ref	RN-$\mathcal{R}_0$-2000 [2]	1973	78.25	-
baseline	RN101	7598	79.21	4.83
PCT	RN101+FD [35]	7598	79.90	3.06
wt. share	RN-$\mathcal{R}_0$-3000 [2]	2941	78.78	2.04
REG-NAS	RN-($\mathcal{R}_2$+CAS)-3000	2915	78.80	1.57

(b) ResNet search space of OFA

Table 1: Comparing different model design strategies. We present results for MobileNet-V3 (MB) and ResNet-50 (RN) search spaces of OFA [2]. EffNet and RN101 represents EfficientNet-B0 and ResNet-101 trained with cross entropy loss, while EffNet+FD and RN101+FD are trained with state-of-the-art focal distillation loss [35]. Results of weight sharing and our proposed method are averaged from three runs with different random seeds.

We first search for the best reference model under 150 mega flops constraint (MB-$\mathcal{R}_0$-150) using the $\mathcal{R}_0$ reward which maximizes Top-1 accuracy. Then, we establish a baseline by training an off-the-shelf EfficientNet-B0 model[2] using vanilla cross entropy. This model improves Top-1 accuracy to 77%, but suffers from 4.25% negative flips. A stronger baseline can be achieved by training the EfficientNet using Focal Distillation [35] which is the current state-of-the-art method to reduce negative flips. It reduces the NFR to 3.25% with a 0.8% drop in Top-1 accuracy. Another strong weight sharing baseline can be obtained via searching a sub-network from the same super-network as the reference model using $\mathcal{R}_0$ reward. This leads to an NFR of 2.51% with 77.11% Top-1 accuracy. Compared to all of these, the networks obtained via REG-NAS *i.e.* (MB-($\mathcal{R}_2$+CAS)-300) achieves the lowest NFR of 2.16% with less than 0.3% drop in Top-1 accuracy compared with MB-$\mathcal{R}_0$-300. This constitutes a 50% and 33% relative reduction of NFR with respect to the baseline and existing state-of-the-art method with less than 1% change in accuracy (see appendix C). The strong performance of REG-NAS clearly demonstrates the benefits of the search reward and architecture constraint. Similar results can be observed for the ResNet-50 search space of OFA [2] in Table 1b.

[2] We choose EfficientNet-B0 since it has similar Top-1 accuracy with sub-networks searched from MobileNet-V3 in OFA [2]

	Model	Flops	Top-1 ↑ (%)	NFR ↓ (%)
ref	MB-$\mathcal{R}_0$-150	149	73.70	-
paragon Top-1	MB-$\mathcal{R}_0$-300	298	77.11	2.51
paragon NFR	MB-$\mathcal{R}_1$-300	279	76.20	2.07
	MB-$\mathcal{R}_2$-300	292	76.90	2.25
REG-NAS	MB-($\mathcal{R}_2$+CAS)-300	294	76.83	2.16

(a) MobileNet-V3 search space of OFA

	Model	Flops	Top-1 ↑(%)	NFR ↓ (%)
ref	RN-$\mathcal{R}_0$-2000	1973	78.25	-
paragon Top-1	RN-$\mathcal{R}_0$-3000	2941	78.78	2.04
paragon NFR	RN-$\mathcal{R}_1$-3000	2537	78.57	1.28
	RN-$\mathcal{R}_2$-3000	2972	78.87	1.76
REG-NAS	RN-($\mathcal{R}_2$+CAS)-3000	2915	78.80	1.57

(b) ResNet search space of OFA

Table 2: Comparing different search rewards using the MobileNet-V3 (MB) and ResNet-50 (RN) search spaces of OFA [2]. Results (except ref) are averaged from three runs with different seeds.

4.3 Dissecting the performance of REG-NAS

In this subsection, we study the contribution of different components of REG-NAS—the search reward and the constrained architecture search, in reducing NFR. Table 2 presents the performance of architectures searched via $\mathcal{R}_0$, $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_2$+CAS. We observe that since $\mathcal{R}_0$ only includes Top-1 accuracy in its formulation, the optimal architecture searched via it achieves the best accuracy, which we treat as the paragon for Top-1 accuracy. However, this same architecture also suffers from the highest NFR. On the other hand, $\mathcal{R}_1$, which only includes the NFR in its formulation, leads to a model with the lowest NFR, and also lowest Top-1 accuracy. We treat this model as the paragon for NFR.

Effect of reward $\mathcal{R}_2$. By including both the Top-1 accuracy and NFR in its formulation, reward $\mathcal{R}_2$ leads to models that lie closer to the paragon of both metrics. The best results, however, are achieved by including the architecture constraint which further brings the models closer to the respective paragons. In the case of MobileNet-V3, architecture searched by $\mathcal{R}_2$+CAS is within 0.09% of the paragon of NFR and within 0.28% of the paragon of Top-1 accuracy, which demonstrates the effectiveness of the proposed method.

Effect of architecture constraint. Besides reducing NFR, we find that CAS also help search for an architecture with low NFR much faster compared with $\mathcal{R}_2$ as shown in Fig. 5a. This is because CAS partitions the overall architecture space into a subspace containing well performing architectures that achieve low NFR and high Top-1 accuracy as described in Sec. 3.2.

Effect of model size. It is natural to ask if a smaller gap in Top-1 accuracy would imply a smaller NFR? The answer is provided in Fig. 5b which plots the accuracy and NFR for 9 target models (from 175 to 600 Mflops) searched against a common 150 Mflop reference model with 73.7% Top-1 accuracy. Observe that with $\mathcal{R}_0$, even for the smallest model with similar accuracy as the reference, the NFR is consistently high ($\approx$ 2.5%). This suggests that accuracy gap between reference and target has little impact on the NFR. Additionally, we see that the search with $\mathcal{R}_2 + CAS$ outperforms $\mathcal{R}_0$ across all model sizes.

4.4 Supercharging REG-NAS with PCT

We study if fine-tuning with FD loss proposed in [35] can help further reduce the NFR for architectures searched via REG-NAS. Table 3 compares two loss

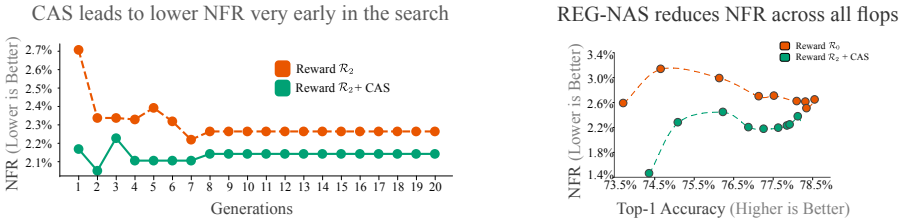

Fig. 5: (a) NFR of the optimal architecture in each generation of the evolutionary search. Observe that CAS leads to lower NFR early and throughout the search. (b) Top-1 accuracy and NFR for 9 models (175-500 mflops) searched against a common reference (150 mflops). Observe that a smaller accuracy gap doesn't imply a smaller NFR. Also $\mathcal{R}_2 + CAS$ outperforms $\mathcal{R}_0$ for all sizes.

functions: vanilla cross entropy (CE) and FD[3] for fine-tuning the target sub-networks obtained via REG-NAS.

We observe that fine tuning with CE generally results in a higher NFR. This is expected because CE optimizes the weights for higher Top-1 accuracy, which may push the weights of the target model away from those of the reference. In contrast, FD loss promotes similar sample-wise behavior by penalizing negative flips and leads to a lower NFR. This study demonstrates that FD loss can work in tandem with REG-NAS to further reduce NFR. However, even without fine tuning, the architectures searched by REG-NAS already enjoy a very low NFR, even outperforming the MB-$\mathcal{R}_0$-300 architecture with fine tuning. Considering the large time cost of fine tuning, we avoid it in the following discussion.

Super-networks	Model	Fine tune	Top-1 ↑(%)	NFR ↓ (%)
MobileNet-V3	MB-$\mathcal{R}_0$-150	-	73.70	-
	MB-$\mathcal{R}_0$-300	-	77.11	2.51
		CE	77.37	2.53
		FD [35]	76.94	2.20
	MB-($\mathcal{R}_2$+CAS)-300	-	76.83	2.16
		CE	77.05	2.37
		FD [35]	76.70	2.12
ResNet	RN-$\mathcal{R}_0$-2000	-	78.25	-
	RN-$\mathcal{R}_0$-3000	-	78.78	2.04
		CE	78.68	2.47
		FD [35]	78.37	1.33
	RN-($\mathcal{R}_2$+CAS)-3000	-	78.79	1.57
		CE	78.63	2.16
		FD [35]	78.47	0.96

Table 3: Comparing fine-tuning strategies post inheriting the weights from a super-network. "-" denotes no fine tuning, CE denotes cross entropy, and FD denotes focal distillation[35]. Results averaged across three random seeds.

[3] We use FD-KD from [35] since it outperforms FD-LM in our setting.

4.5 Additional properties of REG-NAS

In this section, we study two key properties of REG-NAS: *transitivity*, as it enables the search for models at diverse flop budgets and the *search direction*.

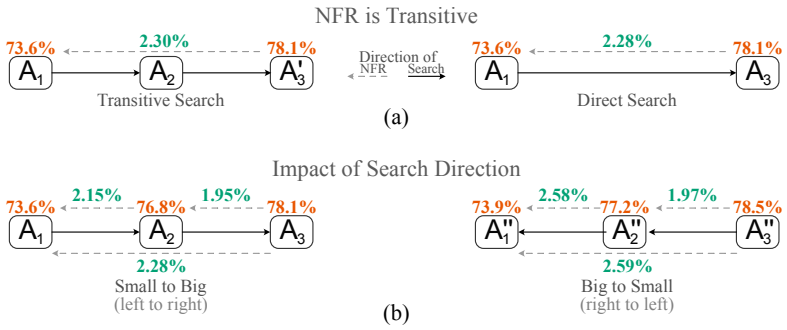

Fig. 6: Using the MobileNet-V3 search space of OFA [2] we illustrate two key properties of NFR (a) **transitivity**: (left) A 600 Mflops model A'_3 searched against a 300 Mflops model A_2 that was previously searched against a 150 Mflops one A_1, automatically achieves a similar NFR as directly searching A_3 against A_1 (right). (b) **Effect of search direction**: small to big search (left) leads to lower NFR compared to the opposite direction (right).

Transitivity of NFR. Transitivity is measured through the following experiment. Given a reference model A_1, we first search for a model A_2 having low NFR using REG-NAS. Then we study whether the model A_3 searched using A_2 as reference model can lead to a low NFR between A_3 and A_1. Fig. 6a illustrates this property using the MobileNet-V3 search space of OFA [2]. We can search for a 600 Mflops model in two ways: (1) Transitively search a 600 Mflops model A'_3 against a 300 Mflops A_2 that was previously searched the 150 Mflops reference A_1. (2) Directly search a 600 Mflops model A_3 against the 150 Mflops reference A_1; Observe that the two models A'_3, A_3 so obtained achieve a similar NFR with respect to A_1, demonstrating that NFR among architectures searched via REG-NAS is transitive. Based on transitivity property, when searching for a series of architectures, the i^{th} one can be searched against the $i-1^{th}$ one thereby ensuring a low NFR between the i^{th} and $i-2^{th}$ ones.

Search direction. Until now we have focused on a setting in which given a *small* reference model, we search for a *larger* target model. We call this the small $\rightarrow$ large setting. Fig. 6b compares the opposite setting *i.e.* large $\rightarrow$ small. We observe that the small $\rightarrow$ large setting benefits in reducing NFR for all subnets whereas the opposite direction leads to higher Top-1 accuracy at the cost of higher NFR. Since we aim to minimize NFR, we follow the small $\rightarrow$ large setting for all experiments.

4.6 Generalization Performance of REG-NAS

In this subsection, we investigate the performance of REG-NAS for searching a family of models; from different super-networks and search spaces; with latency constraints on different platforms.

REG-NAS for searching a family of models By leveraging transitive search as discussed in Sec. 4.5, we can use REG-NAS to search for a family of models suited for different devices (*e.g.* phone, laptop, cloud). We demonstrate this by searching for three different models A1, A2, A3 with different flop budgets in Table 4 where we present the NFR between each pair of models. Compared to vanilla NAS that optimizes for Top-1 accuracy using $\mathcal{R}_0$ reward, REG-NAS can consistently reduce the NFR metric. In some scenarios, the benefit is quite large *e.g.* between models A2, A3 searched in the MobileNet-V3 and ResNet-50 based super-networks of OFA [2]. Please see appendix D for results on four models.

Supernet	Method	Results
MB-V3	OFA [2]	A_1 (73.7%) ←2.51%– A_2 (77.1%) ←2.41%– A_3 (78.5%); 2.63%
	REG-NAS	A_1 (73.7%) ←2.16%– A_2 (76.8%) ←1.95%– A_3 (78.2%); 2.30%
RN-50	OFA [2]	A_1 (78.2%) ←2.04%– A_2 (78.8%) ←1.53%– A_3 (79.2%); 1.90%
	REG-NAS	A_1 (78.2%) ←1.57%– A_2 (78.8%) ←0.83%– A_3 (79.0%); 1.74%
FBNet-V3	Att-NAS [31]	A_1 (76.5%) ←2.39%– A_2 (79.7%) ←2.03%– A_3 (80.9%); 2.31%
	REG-NAS	A_1 (76.5%) ←2.26%– A_2 (79.5%) ←1.84%– A_3 (80.7%); 2.30%
FBNet-V3	α-Net [30]	A_1 (76.9%) ←2.10%– A_2 (80.0%) ←1.84%– A_3 (80.9%); 2.07%
	REG-NAS	A_1 (76.9%) ←1.97%– A_2 (79.9%) ←1.32%– A_3 (80.7%); 1.96%

Table 4: Testing generalization of REG-NAS for searching multiple models with diverse super-networks. Model size increases from A_1 to A_3. NFR is indicated in green and Top-1 accuracy in orange. REG-NAS successfully reduces NFR in all scenarios.

REG-NAS with different supernetworks One-shot NAS is an actively evolving research area, with many recent innovations in super-network pre-training. To demonstrate that REG-NAS can generalize to different super-networks, we apply REG-NAS on two state-of-the-art FBNet-V3 based super-networks: AttentiveNAS [31] that uses attentive sampling to improve sub-network accuracy

and AlphaNet [30] that uses alpha divergence to improve knowledge transfer from larger to small sub-networks. Table 4 shows that REG-NAS consistently reduces the NFR for all pairs of models, with particularly large reduction between models A_2 and A_3 in the case of AlphaNet.

Target Device	Model	Latency (ms)	Top-1 ↑(%)	NFR (%) ↓(%)
Note10	MB-$\mathcal{R}_0$-15	14.83	73.34	-
1080ti	MB-$\mathcal{R}_0$-30	29.40	77.30	2.72
	MB-($\mathcal{R}_2$+CAS)-30	29.88	77.07	2.22

Table 5: Testing generalization of REG-NAS under latency constraints. We use a model with 15ms latency on a Samsung Note 10 as the reference to search a target model with 30ms latency on an Nvidia 1080ti GPU. REG-NAS successfully reduces NFR.

REG-NAS with latency constraints We test whether REG-NAS can find low NFR sub-networks using latency (instead of flops) as the compute metric. For this experiment, we search for two models: a reference model with 15ms latency deployed on a Samsung Note 10 phone and a target model with 30 ms latency running on an Nvidia 1080ti GPU. We measure latency via latency lookup tables built on each device following OFA [2]. The results in Table 5 show that REG-NAS can indeed find a target model with considerably low NFR without a large drop in Top-1 accuracy while using latency as the compute metric.

5 Discussion

In this paper, we tackle the problem of regression (measure via NFR) between models deployed on diverse compute platforms. We observe that architectures sampled from the same One-Shot NAS super-network naturally share a lot of weights and lead to a low NFR. Inspired by this finding, we propose a novel architecture constraint that maximizes weight sharing and a novel search reward function which further reduces the NFR. There are two limitations of our work: First, the impact of different supernet pre-training strategies on the NFR needs to be investigated. Our preliminary experiments show that the same super-network (FBNet-V3) obtained via different pre-training strategies (AlphaNet [30] and AttentiveNAS [31]) lead to different NFRs, indicating a measurable impact of super-network pre-training on NFR. Second, REG-NAS samples sub-networks from the same super-network which itself may not span the entire range of compute budgets of different platforms *e.g.* tiny embedded platforms to large data-centers. A future direction may investigate sampling from many different super-networks. Additionally, designing regression-free models via filter pruning [11, 12] and early-exiting [9, 10] are interesting future directions. Overall, we believe this paper only scratches the surface for an emerging problem—designing regression-free neural networks for diverse compute platforms—that will become more prominent in the era of on-device deep learning.

References

1. Belouadah, E., Popescu, A.: Il2m: Class incremental learning with dual memory. In: ICCV (2019)
2. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once for all: Train one network and specialize it for efficient deployment. In: ICLR (2020)
3. Castro, F.M., Marín-Jiménez, M.J., Guil, N., Schmid, C., Alahari, K.: End-to-end incremental learning. In: ECCV (2018)
4. Chaudhry, A., Dokania, P.K., Ajanthan, T., Torr, P.H.: Riemannian walk for incremental learning: Understanding forgetting and intransigence. In: ECCV (2018)
5. Chen, Z., Liu, B.: Lifelong machine learning. Synthesis Lectures on Artificial Intelligence and Machine Learning **12**(3), 1–207 (2018)
6. Chu, X., Zhou, T., Zhang, B., Li, J.: Fair darts: Eliminating unfair advantages in differentiable architecture search. In: ECCV (2020)
7. Dai, X., Wan, A., Zhang, P., Wu, B., He, Z., Wei, Z., Chen, K., Tian, Y., Yu, M., Vajda, P., et al.: Fbnetv3: Joint architecture-recipe search using predictor pretraining. In: CVPR (2021)
8. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR (2009)
9. Duggal, R., Freitas, S., Dhamnani, S., Chau, D.H., Sun, J.: Elf: An early-exiting framework for long-tailed classification. arXiv preprint arXiv:2006.11979 (2020)
10. Duggal, R., Freitas, S., Dhamnani, S., Chau, D.H., Sun, J.: Har: Hardness aware reweighting for imbalanced datasets. In: BigData (2021)
11. Duggal, R., Freitas, S., Xiao, C., Chau, D.H., Sun, J.: Rest: Robust and efficient neural networks for sleep monitoring in the wild. In: Proceedings of The Web Conference 2020 (2020)
12. Duggal, R., Xiao, C., Vuduc, R., Chau, D.H., Sun, J.: Cup: Cluster pruning for compressing deep neural networks. In: BigData (2021)
13. Duggal, R., Zhou, H., Yang, S., Xiong, Y., Xia, W., Tu, Z., Soatto, S.: Compatibility-aware heterogeneous visual search. In: CVPR (2021)
14. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling. In: ECCV (2020)
15. Kandasamy, K., Neiswanger, W., Schneider, J., Poczos, B., Xing, E.: Neural architecture search with bayesian optimisation and optimal transport. arXiv preprint arXiv:1802.07191 (2018)
16. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al.: Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences **114**(13), 3521–3526 (2017)
17. Li, Z., Hoiem, D.: Learning without forgetting. IEEE transactions on PAMI **40**(12), 2935–2947 (2017)
18. Liu, H., Simonyan, K., Vinyals, O., Fernando, C., Kavukcuoglu, K.: Hierarchical representations for efficient architecture search. arXiv preprint arXiv:1711.00436 (2017)
19. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
20. Lopez-Paz, D., Ranzato, M.: Gradient episodic memory for continual learning. In: NeurIPS (2017)
21. Parisi, G.I., Kemker, R., Part, J.L., Kanan, C., Wermter, S.: Continual lifelong learning with neural networks: A review. Neural Networks **113**, 54–71 (2019)

22. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: ICML (2018)
23. Radosavovic, I., Kosaraju, R.P., Girshick, R., He, K., Dollár, P.: Designing network design spaces. In: CVPR (2020)
24. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: AAAI (2019)
25. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: ICML (2017)
26. Shen, Y., Xiong, Y., Xia, W., Soatto, S.: Towards backward-compatible representation learning. In: CVPR (2020)
27. Srivastava, M., Nushi, B., Kamar, E., Shah, S., Horvitz, E.: An empirical analysis of backward compatibility in machine learning systems. In: ACM SIGKDD (2020)
28. Suganuma, M., Ozay, M., Okatani, T.: Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. In: ICML (2018)
29. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: CVPR (2019)
30. Wang, D., Gong, C., Li, M., Liu, Q., Chandra, V.: Alphanet: Improved training of supernet with alpha-divergence. In: ICML (2021)
31. Wang, D., Li, M., Gong, C., Chandra, V.: Attentivenas: Improving neural architecture search via attentive sampling. In: CVPR (2021)
32. Wu, B., Li, C., Zhang, H., Dai, X., Zhang, P., Yu, M., Wang, J., Lin, Y., Vajda, P.: Fbnetv5: Neural architecture search for multiple tasks in one run. arXiv preprint arXiv:2111.10007 (2021)
33. Wu, Y., Chen, Y., Wang, L., Ye, Y., Liu, Z., Guo, Y., Fu, Y.: Large scale incremental learning. In: CVPR (2019)
34. Xie, L., Yuille, A.: Genetic cnn. In: ICCV (2017)
35. Yan, S., Xiong, Y., Kundu, K., Yang, S., Deng, S., Wang, M., Xia, W., Soatto, S.: Positive-congruent training: Towards regression-free model updates. In: CVPR (2021)
36. Yu, J., Huang, T.: Autoslim: Towards one-shot architecture search for channel numbers. arXiv preprint arXiv:1903.11728 (2019)
37. Yu, J., Jin, P., Liu, H., Bender, G., Kindermans, P.J., Tan, M., Huang, T., Song, X., Pang, R., Le, Q.: Bignas: Scaling up neural architecture search with big single-stage models. In: ECCV (2020)
38. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)