Saliency-Aware Neural Architecture Search

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recently a wide variety of NAS methods have been proposed and achieved considerable success in automatically identifying highly-performing architectures of neural networks for the sake of reducing the reliance on human experts. Existing NAS methods ignore the fact that different input data elements (e.g., image pixels) have different importance (or saliency) in determining the prediction outcome. They treat all data elements as being equally important and therefore lead to suboptimal performance. To address this problem, we propose an end-to-end framework which dynamically detects saliency of input data, reweights data using saliency maps, and searches architectures on saliency-reweighted data. Our framework is based on four-level optimization, which performs four learning stages in a unified way. At the first stage, a model is trained with its architecture tentatively fixed. At the second stage, saliency maps are generated using the trained model. At the third stage, the model is retrained on saliency-reweighted data. At the fourth stage, the model is evaluated on a validation set and the architecture is updated by minimizing the validation loss. Experiments on several datasets demonstrate the effectiveness of our framework.

1 Introduction

Neural architecture search (NAS) [79, 41, 48], which aims to automatically identify highly performant neural architectures, has received much attention recently. Existing NAS methods treat all elements in an input data example (such as pixels in an image, tokens in a sentence, etc.) as being equally important, without considering the different saliency of individual elements, which leads to less-optimal performance. In machine learning applications, an input data example typically consists of many data elements. For instance, an image example consists of a grid of pixel elements and a sentence example consists of a sequence of token elements. When used to make a prediction, different data elements have different importance (or saliency). For example, when predicting which object category an image belongs to, pixels in foreground object regions are more important than those in background regions. Such saliency information is not leveraged by existing NAS methods.

In this paper, we aim to bridge this gap, by proposing a saliency-aware NAS method which automatically identifies the saliency of data elements and leverages that to search for better architectures. Our framework is formulated as a four-level optimization problem. At the first level, the model tentatively fixes its architecture and trains its first set of network weights. At the second level, the trained model generates saliency maps using an adversarial attack based method [18]. At the third level, input data is reweighted using saliency maps and the second set of model weights are trained using reweighted data. At the fourth level, the two sets of trained model weights are evaluated on a human-provided validation set and the architecture is optimized by minimizing the validation losses.

The major contributions of this paper are as follows:

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

- We propose a framework which performs end-to-end detection of saliency and leverage saliency-reweighted data to improve neural architecture search.
- We formulate our method as a multi-level optimization problem which performs model training on unweighted data, saliency map generation, model retraining on saliency-reweighted data, and architecture update in a unified manner.

2 Related works

Neural architecture search (NAS). Early NAS approaches [79, 46, 80] are mostly based on reinforcement learning (RL) which use a policy network to generate architectures and evaluate these architectures on validation sets. The validation loss is used as a reward to optimize the policy network and train it to produce high-quality architectures. Differentiable search methods [5, 41, 67] parameterize architectures as differentiable functions and perform search using efficient gradient-based methods. In these methods, the search space of architectures is composed of a large set of building blocks where the output of each block is multiplied with a smooth variable indicating how important this block is. Under such a formulation, search becomes solving a mathematical optimization problem defined on the importance variables where the objective is to find an optimal set of variables that yield the lowest validation loss. This optimization problem can be solved efficiently using gradient-based methods. Differentiable NAS research is initiated by DARTS [41] and further improved by subsequent works such as P-DARTS [11], PC-DARTS [68], etc. P-DARTS [11] grows the depth of architectures progressively in the search process. PC-DARTS [68] samples sub-architectures from a super network to reduce redundancy during search. Our proposed framework is orthogonal to existing NAS methods and can be used in combination with any differentiable NAS method to further improve these methods. Besides RL-based approaches and differentiable NAS, another paradigm of NAS methods [40, 48] are based on the evolutionary algorithm where architectures are formulated as individuals in a population. High-quality architectures produce offspring to replace low-quality architectures, where the quality is measured using fitness scores.

Saliency detection. Many methods [72, 78, 57, 45, 52, 51, 47] have been proposed for saliency detection, based on perturbing inputs [72, 78], propagating gradients [3, 56, 57], attention [37, 71, 45], model approximation [50, 2], etc. Several works [52, 51, 47] show that leveraging saliency of input data can enhance model's predictive power. Rieger et al. [51] leverage domain-specific rules or knowledge to provide "groundtruth" saliency. Such rules/knowledge are difficult to obtain in many applications. Pillai and Pirsiavash [47] encourage a prediction model to produce saliency maps that are consistent with those generated by GradCAM [54]. GradCAM is an unsupervised approach; without any supervision from humans, its generated saliency maps may not be reliable. For example, it is shown in [58] that GradCAM cannot highlight adversarial image patches that cause wrong predictions. In [52], saliency maps are either labeled by humans or auto-generated based on gradient magnitude with no human supervision, which suffers the same problems as [54, 51]. Different from existing works, our method generates saliency maps with weak supervision such as human-provided class labels. Such weak supervision is much easier to obtain than human annotations of saliency maps and can yield more reliable saliency maps than using no supervision at all.

Bi-level optimization. Many ML methods [19, 4, 20, 41, 55, 75] have been formulated as bi-level optimization (BLO) problems. In these methods, network weights are learned by solving an inner optimization problem defined on a training set while meta parameters are learned by solving an outer optimization problem defined on a validation set. BLO-based methods have been applied for neural architecture search [41], hyperparameter tuning [19], learning rate adaptation [4], data selection [55, 49, 65], meta learning [20], label correction [75], etc., where meta parameters are neural architectures, hyperparameters, importance weights of data examples, meta network weights, etc. Many optimization algorithms [14, 21, 23, 32, 42, 69] have been developed for solving BLO problems where convergence analysis is provided.

3 Methods

In this section, we propose a four-level optimization based framework to perform saliency-aware neural architecture search. For the ease of presentation, we assume the task is image classification. In the experiments, we show that our method can be applied for other tasks as well.

3.1 A four-level optimization framework

In our framework (Figure 1), a model has a learnable architecture A and two sets of learnable network weights W_1 and W_2. W_2 is a tensor that has the same dimensions as W_1. The weight values in W_2 and W_1 are different. It consists of four stages performed end-to-end. At the first stage, the model trains its network weights W_1 with the architecture A tentatively fixed. At the second stage, the trained W_1 generates saliency maps for input images: a saliency score is calculated for each pixel. At the third stage, images are reweighted using saliency maps and saliency-reweighted images are used to train model weights W_2. At the fourth stage, the trained W_2 is evaluated on a human-labeled validation set and the architecture A is updated by minimizing the validation loss.

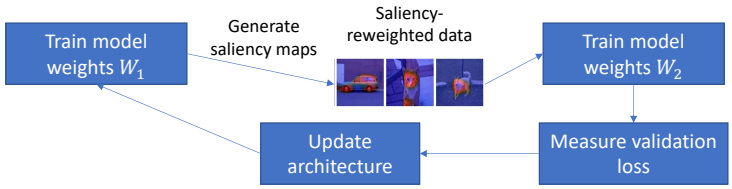

Figure 1: Overview of our framework.

Stage I. At the first stage, the model trains its first set of network weights W_1 by minimizing a loss L on training dataset $D^{(\text{tr})}$, with the architecture A tentatively fixed:

$$W_1^*(A) = \text{argmin}_{W_1}\ L(W_1, A, D^{(\text{tr})}). \tag{1}$$

To define the training loss, we need to use both the architecture parameters A and network weights W_1. However, A cannot be updated by minimizing the training loss. Otherwise, a trivial solution of A will be yielded: A can perfectly overfit the training data but will make incorrect predictions on unseen data examples. $W_1^*(A)$ denotes that the optimal weights W_1^* depends on A. This is because $L(W_1, A, D^{(\text{tr})})$ is a function of A, and W_1^* depends on $L(W_1, A, D^{(\text{tr})})$.

Stage II. At the second stage, the trained $W_1^*(A)$ is used to generate saliency maps. Specifically, given an input image x, we first use $W_1^*(A)$ and A to predict the class label (denoted by $f(x; W_1^*(A), A)$) of x. Then an adversarial attack based approach [22, 18] is leveraged to calculate saliency scores of pixels. Adversarial attack adds small random perturbations δ to pixels in x so that the prediction outcome on the perturbed image $x+\delta$ is no longer $f(x; W_1^*(A), A)$. Pixels perturbed more are more correlated with the prediction outcome $f(x; W_1^*(A), A)$ and are considered to be more salient. This process amounts to solving the following optimization problem:

$$\{\delta_i^*(W_1^*(A), A)\}_{i=1}^N = \text{argmax}_{\{\|\delta_i\|_\infty \le \varepsilon\}_{i=1}^N}\ \sum_{i=1}^N \ell(f(x_i+\delta_i; W_1^*(A), A), f(x_i; W_1^*(A), A)) \tag{2}$$

where δ_i is the perturbation added to image x_i and ε is a small norm-bound. N is the number of images. $f(x_i+\delta_i; W_1^*(A), A)$ and $f(x_i; W_1^*(A), A)$ are predictions made by $W_1^*(A)$ and A on $x_i+\delta_i$ and x_i. Assume the number of classes is K. $f(x_i+\delta_i; W_1^*(A), A)$ and $f(x_i; W_1^*(A), A)$ are K-dimensional vectors containing prediction probabilities on individual classes. $\ell(\cdot,\cdot)$ is the cross-entropy loss with $\ell(\mathbf{a}, \mathbf{b}) = -\sum_{k=1}^K b_i \log a_i$. In this optimization problem, we aim to find perturbations for each image so that the predicted outcome on the perturbed image is largely different from that on the original image. The learned optimal perturbations are considered as saliency scores of pixels: larger perturbations indicate that the corresponding pixels are more correlated with the prediction outcome and therefore are more salient. δ_i^* depends on $W_1^*(A)$ and A since δ_i^* depends on the loss in Eq.(2), and the loss is a function of $W_1^*(A)$ and A.

Stage III. At the third stage, given the saliency scores $\{\delta_i^*(W_1^*(A), A)\}_{i=1}^N$, the second set of model weights W_2 are trained. We use the saliency scores to reweight the pixels: $x \odot \delta$, where $\odot$ denotes element-wise multiplication (we compare with other reweighting mechanisms in Table 5). Pixels that are more salient are given more weights. Then W_2 is trained on these weighted pixels:

$$W_2^*(\{\delta_i^*(W_1^*(A), A)\}_{i=1}^N, A) = \text{argmin}_{W_2}\ \sum_{i=1}^N \ell(f(\delta_i^*(W_1^*(A), A) \odot x_i; W_2, A), t_i), \tag{3}$$

where $f(\delta_i^*(W_1^*(A), A) \odot x_i; W_2, A)$ is the prediction made by W_2 and A on the weighted image $\delta_i^*(W_1^*(A), A) \odot x_i$, and t_i is the class label. W^* depends on $\{\delta_i^*(W_1^*(A), A\}_{i=1}^N$ and A since W^* depends on the loss in Eq.(3), and the loss is a function of $\{\delta_i^*(W_1^*(A), A)\}_{i=1}^N$ and A.

Stage IV. At the fourth stage, $W_2^*(\{\delta_i^*(W_1^*(A),A)\}_{i=1}^N,A)$ and $W_1^*(A)$ are evaluated on a human-labeled validation set $D^{(\text{val})}$. The architecture A is updated by minimizing the validation losses:

$$\min_A L(W_2^*(\{\delta_i^*(W_1^*(A),A)\}_{i=1}^N,A),A,D^{(\text{val})}) + \gamma L(W_1^*(A),A,D^{(\text{val})}), \tag{4}$$

where γ is a tradeoff parameter.

Four-level optimization framework. We integrate the four stages into a unified four-level optimization framework and obtain the following formulation:

$$\begin{aligned}
&\min_A L(W_2^*(\{\delta_i^*(W_1^*(A),A)\}_{i=1}^N,A),A,D^{(\text{val})}) + \gamma L(W_1^*(A),A,D^{(\text{val})}) \\
&s.t.\ \ W_2^*(\{\delta_i^*(W_1^*(A),A)\}_{i=1}^N,A) = \text{argmin}_{W_2} \textstyle\sum_{i=1}^N \ell(f(\delta_i^*(W_1^*(A),A)\odot x_i;W_2,A),t_i) \\
&\qquad \{\delta_i^*(W_1^*(A),A)\}_{i=1}^N = \text{argmax}_{\{\|\delta_i\|_\infty\le\varepsilon\}_{i=1}^N} \textstyle\sum_{i=1}^N \ell(f(x_i+\delta_i;W_1^*(A),A),f(x_i;W_1^*(A),A)) \\
&\qquad W_1^*(A) = \text{argmin}_{W_1} L(W_1,A,D^{(\text{tr})}).
\end{aligned} \tag{5}$$

In this framework, there are four optimization problems, each corresponding to a learning stage. From bottom to up, the optimization problems correspond to learning stage I to IV respectively. The first three optimization problems are nested on the constraint of the fourth optimization problem. These four stages are conducted end-to-end in this unified framework. The solution $W_1^*(A)$ obtained at the first stage is used to generate explanations at the second stage. The saliency maps $\{\delta_i^*(W_1^*(A),A)\}_{i=1}^N$ obtained at the second stage are used to train W_2 at the third stage. The solutions obtained at the first and third stage are used to calculate validation losses at the fourth stage. The architecture A updated at the fourth stage changes the training loss at the first stage and consequently changes the solution $W_1^*(A)$, which subsequently changes $\{\delta_i^*(W_1^*(A),A)\}_{i=1}^N$ and $W_2^*(\{\delta_i^*(W_1^*(A),A)\}_{i=1}^N,A)$.

Optimization algorithm. To solve the problem in Eq.(5), we used a standard algorithm developed in [41], which is broadly used in many previous works and demonstrated to be effective in the literature. The convergence analysis of this algorithm has been given in many works [21, 23, 32, 42, 69]. The optimization algorithm is not the focus or contribution of our work. In this algorithm, we calculate the gradient of $L(W_1,A,D^{(\text{tr})})$ w.r.t W_1 and approximate $W_1^*(A)$ using a one-step gradient descent update of W_1. We plug the approximation W_1' of $W_1^*(A)$ into $\ell(f(x_i+\delta_i;W_1^*(A),A),f(x_i;W_1^*(A),A))$ and obtain an approximated objective denoted by O_{δ_i}. Then we approximate $\delta_i^*(W_1^*(A),A)$ using a one-step gradient ascent update of δ_i based on the gradient of O_{δ_i}. Next, we plug the approximation δ_i' of $\delta_i^*(W_1^*(A),A)$ into $\ell(f(\delta_i^*(W_1^*(A),A)\odot x_i;W_2,A),t_i)$ and get another approximated objective denoted by O_{W_2}. Then we approximate $W_2^*(\{\delta_i^*(W_1^*(A),A)\}_{i=1}^N,A)$ using a one-step gradient descent update of W_2 based on the gradient of O_{W_2}. Finally, we plug the approximation W_1' of $W_1^*(A)$ and the approximation W_2' of $W_2^*(\{\delta_i^*(W_1^*(A),A)\}_{i=1}^N,A)$ into the validation loss and get the third approximate objective O_A. A is updated by descending the gradient of O_A. These steps iterate until convergence.

4 Experiments

In this section, we present experimental results. Please refer to the supplements for detailed hyperparameter settings and additional results (e.g., on inference costs, ablation studies).

4.1 Experiments on image classification

Following [41] the architecture A is parameterized in a differentiable way. A contains a set of importance weights, each multiplied to the output of a candidate architecture block. Architecture search amounts to learning these weights using gradient methods. After learning, blocks with top weights compose an architecture. Each experiment consists of two phrases: 1) architecture search where an optimal cell is identified, and 2) architecture evaluation where multiple copies of the optimal cell are stacked into a larger network, which is retrained from scratch.

Datasets We used three datasets: CIFAR-10 [35], CIFAR-100 [36], and ImageNet [15]. Both CIFAR-10 and CIFAR-100 contain 60K images from 10 classes. For each of them, we split it into a train, validation, and test set with 25K, 25K, and 10K images respectively. ImageNet contains 1.2M training images and 50K test images from 1000 objective classes. Following [68], we randomly sample 10% of the 1.2M images to form a new training set and another 2.5% to form a validation set, then perform a search on them.

Table 1: Robustness against overfitting: test errors on CIFAR-10 (C10) and CIFAR-100 (C100).

Data	Space	DARTS [41]	DartsES [73]	EC [52]	CDEP [51]	GMPGC [47]	Ours
C10	S1	4.66±0.71	3.05±0.07	4.43±0.69	4.18±0.45	4.26±0.72	**2.88**±0.07
	S2	4.42±0.40	3.41±0.14	3.96±0.52	4.13±0.48	3.69±0.70	**3.29**±0.08
	S3	4.12±0.85	3.71±1.14	3.85±0.46	3.97±0.23	3.92±0.68	**3.54**±0.23
	S4	6.95±0.18	4.17±0.21	5.88±0.38	5.36±0.52	4.98±0.61	**3.68**±0.06
C100	S1	29.93±0.41	28.90±0.81	29.54±0.68	29.07±0.42	28.96±0.65	**27.95**±0.16
	S2	28.75±0.92	24.68±1.43	26.49±0.82	25.17±0.64	24.93±0.57	**23.45**±0.51
	S3	29.01±0.24	26.99±1.79	28.58±0.32	28.12±0.55	27.51±0.49	**25.29**±0.24
	S4	24.77±1.51	23.90±2.01	24.58±1.25	24.63±1.38	23.42±1.56	**23.49**±0.57

Experimental settings We experimented with the search spaces in DARTS [41], P-DARTS [11], and PC-DARTS [68]. The tradeoff parameter γ is set to 2. The norm bound ε of perturbations is set to 0.03. During architecture search, for CIFAR-10 and CIFAR-100, the architecture of the target is a stack of 8 cells. Each cell consists of 7 nodes. Initial channel number is 16. The search algorithm was based on SGD, with a batch size of 64, an initial learning rate of 0.025 with cosine scheduling, an epoch number of 50, a weight decay of 3e-4, and a momentum of 0.9. We update the architecture A every 5 mini-batches (iterations), update model weights W_2 and perturbations δ every 3 mini-batches, and update W_1 on every mini-batch. The rest of hyperparameters mostly follow those in DARTS, P-DARTS, and PC-DARTS. We compare with the following baselines: 1) explanation constraints (EC) [52]; 2) contextual decomposition explanation penalization (CDEP) [51]; 3) global max pooling + GradCAM (GMPGC) [47]. The mean and standard deviation of 10 random runs are reported.

Table 2: Test errors on CIFAR-100 (C100) and CIFAR-10 (C10), number of model parameters (in millions), and search cost (GPU days on a Nvidia 1080Ti). SANAS-darts2nd represents that SANAS is applied to DARTS-2nd. Similar meanings hold for other notations in such a format. * means the results are taken from DARTS$^-$ [12].

Method	Error-C100	Error-C10	Param.	Cost
*ResNet [26]	22.10	6.43	1.7	-
*DenseNet [30]	17.18	3.46	25.6	-
*PNAS [39]	19.53	3.41±0.09	3.2	150
*ENAS [46]	19.43	2.89	4.6	0.5
*AmoebaNet [48]	18.93	2.55±0.05	3.1	3150
*GDAS [17]	18.38	2.93	3.4	0.2
*R-DARTS [74]	18.01±0.26	2.95±0.21	-	1.6
*DARTS$^-$ [12]	17.51±0.25	2.59±0.08	3.3	0.4
AutoFormer [8]	17.42±0.17	2.72±0.09	3.7	2.8
Sampling [24]	17.30±0.10	2.75±0.11	3.8	2.0
*DropNAS [27]	16.95±0.41	2.58±0.14	4.4	0.7
*DrNAS [10]	-	2.54±0.03	4.0	0.4
*ISTA-NAS [70]	-	2.54±0.05	3.3	0.1
*MiLeNAS [25]	-	2.51±0.11	3.9	0.3
*GAEA [38]	-	2.50±0.06	-	0.1
*PDARTS-ADV [9]	-	2.48±0.02	3.4	1.1
*Darts2nd [41]	20.58±0.44	2.76±0.09	3.5	1.5
EC-darts2nd [52]	20.05±0.31	2.83±0.12	3.6	2.2
CDEP-darts2nd [51]	19.53±0.46	2.75±0.05	3.7	2.1
GMPGC-darts2nd [47]	19.08±0.36	2.81±0.07	3.7	2.2
Ours-darts2nd	**16.42**±0.09	**2.54**±0.05	3.7	1.8
†Pcdarts [68]	17.96±0.15	2.57±0.07	3.9	0.1
EC-pcdarts [52]	17.83±0.28	2.63±0.11	4.1	0.9
CDEP-pcdarts [51]	17.88±0.13	2.75±0.08	4.0	1.1
GMPGC-pcdarts [47]	17.73±0.09	2.64±0.05	4.0	1.0
Ours-pcdarts	**16.19**±0.04	**2.49**±0.03	3.9	0.8
†Prdarts [76]	16.48±0.06	2.37±0.03	3.4	0.2
EC-prdarts [52]	17.32±0.14	2.58±0.08	3.5	1.1
CDEP-prdarts [51]	16.86±0.07	2.54±0.05	3.4	1.3
GMPGC-prdarts [47]	16.37±0.10	2.46±0.06	3.6	1.1
Ours-prdarts	**16.01**±0.03	**2.30**±0.04	3.6	0.8
†Pdarts [11]	17.52±0.06	2.54±0.04	3.6	0.3
EC-pdarts [52]	17.25±0.11	2.68±0.07	3.8	1.3
CDEP-pdarts [51]	17.49±0.08	2.63±0.10	3.6	0.9
GMPGC-pdarts [47]	17.33±0.10	2.59±0.07	3.7	1.1
Ours-pdarts	**15.16**±0.09	**2.45**±0.03	3.6	1.1

Results and analysis on CIFAR-100 and CIFAR-10 Table 2 shows results on CIFAR-100 and CIFAR-10. Applying our framework to DARTS, P-DARTS, and PC-DARTS, test errors are greatly reduced, which shows that by end-to-end detecting and leveraging saliency of pixels, the quality of searched architectures can be improved. Another observation from Table 2 is: the performance gain of our method does not come at the cost of substantially increasing model size (number of parameters) or search cost.

Robustness against overfitting The other reason that our method works better than baselines is: by focusing on salient pixels, the architecture is more robust against noises and is less sensitive to irrelevant information. As a result, the architecture has better generalization ability and is more resilient to performance collapse. As recognized in [73, 12], performance collapse is a common issue of existing differentiable NAS methods. To empirically verify the robustness of our method against performance collapse, we experimented with four architecture search spaces proposed in [73]. These spaces are designed specifically for evaluating how effectively an NAS method can alleviate performance collapse. Table 1 shows that the errors achieved on CIFAR-100/10 test sets by our method are lower than those of baselines. This demonstrates that our method has higher efficacy in combating performance collapse.

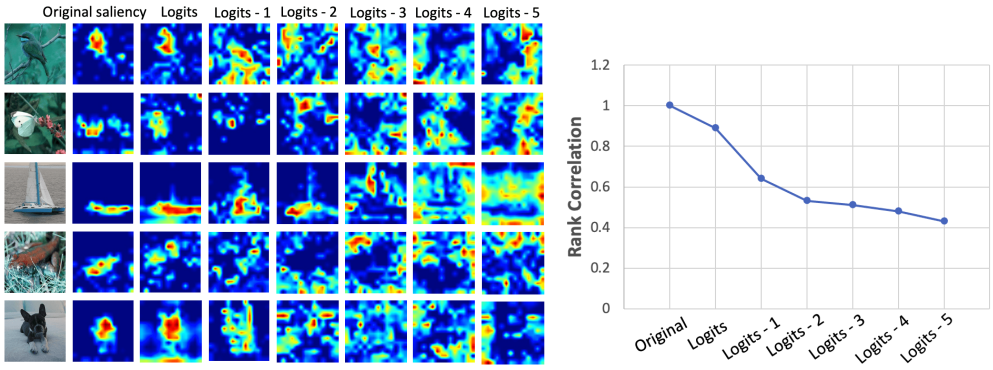

Figure 2: Sanity check of saliency maps. Logits$-n$ is the n-th layer below the logits layer.

Table 2 shows that our methods outperform EC, CDEP, and GMPGC. These methods use GradCAM, gradient magnitude, and pretrained semantic segmentation models to calculate saliency scores, which are not very reliable. In contrast, the calculation of saliency in our method is weakly supervised by the validation loss of the second model calculated on human-provided class labels, which have higher fidelity. As analyzed earlier, saliency with higher fidelity can result in higher-quality architectures.

Table 3: Top-1 and top-5 test errors on ImageNet in the mobile setting. Results marked with * are obtained from DARTS$^-$ [12] and DrNAS [10]. The rest notations are the same as those in Table 2.

Method	Top-1	Top-5
*Inception-v1 [61]	30.2	10.1
*MobileNet [28]	29.4	10.5
*ShuffleNet 2× (v2) [43]	25.1	7.6
*NASNet-A [80]	26.0	8.4
*PNAS [39]	25.8	8.1
*MnasNet-92 [62]	25.2	8.0
*AmoebaNet-C [48]	24.3	7.6
*PARSEC-CIFAR10 [6]	26.0	8.4
*GDAS-CIFAR10 [17]	26.0	8.5
*DSNAS-ImageNet [29]	25.7	8.1
*AutoFormer [8]	25.3	7.4
Sampling [24]	25.3	-
*SDARTS-ADV-CIFAR10 [9]	25.2	7.8
*PC-DARTS-CIFAR10 [68]	25.1	7.8
*ProxylessNAS-ImageNet [5]	24.9	7.5
*FairDARTS-ImageNet [13]	24.4	7.4
*PR-DARTS [76]	24.1	7.3
*DARTS$^-$-ImageNet [12]	23.8	7.0
*Darts2nd-cifar10 [41]	26.7	8.7
EC-darts2nd-cifar10 [52]	26.4	8.5
CDEP-darts2nd-cifar10 [51]	26.5	8.5
GMPGC-darts2nd-cifar10 [47]	26.3	8.2
SANAS-darts2nd-cifar10 (ours)	**24.8**	**8.3**
*Pdarts-cifar100 [11]	24.7	7.5
EC-pdarts-cifar100 [52]	24.5	7.3
CDEP-pdarts-cifar100 [51]	24.6	7.4
GMPGC-pdarts-cifar100 [47]	24.5	7.4
SANAS-pdarts-cifar100 (ours)	**23.8**	**6.6**
*Pcdarts-ImageNet [68]	24.2	7.3
EC-pcdarts-ImageNet [52]	24.0	7.2
CDEP-pcdarts-ImageNet [51]	23.9	7.3
GMPGC-pcdarts-ImageNet [47]	24.0	7.3
SANAS-pcdarts-ImageNet (ours)	**22.2**	**6.1**

Results on ImageNet In Table 3, we compare different methods on ImageNet. In experiments based on PC-DARTS, architectures are searched on a subset of ImageNet. In other experiments, architectures are searched on CIFAR-10 and CIFAR-100. Ours-darts2nd-cifar10 denotes that our method is applied to DARTS-2nd and performs search on CIFAR10. Similar meanings hold for other notations in such a format. The observations made from these results are consistent with those made from Table 2. The architectures searched using our methods are consistently better than those searched by corresponding baselines. These results again show that by end-to-end detecting and leveraging saliency can improve architecture search.

Sanity check of saliency maps We evaluate saliency maps generated by the adversarial saliency method using model parameter cascading randomization tests [1]. The model architecture is searched by SANAS on ImageNet. Figure 2(left) shows that saliency maps change considerably as more layers are randomized, on multiple ImageNet examples. Figure 2(right) shows the Spearman rank correlation (with absolute values) between original saliency maps and randomized saliency maps, on ImageNet. The rank correlation consistently decreases as more layers are randomized. These results demonstrate that saliency maps generated by the adversarial saliency method are sensitive to model parameters and pass the sanity check.

Human evaluation of saliency We randomly sampled 100 images from the test set of ImageNet and generated saliency maps for them using different methods. Then we asked three undergraduates to judge whether the saliency maps are sensible. The ratings are from 1-5 (higher is better). Table 4 summarizes the mean and standard deviation of ratings. Our methods achieve significantly higher ratings (significance test results are in the supplements), which demonstrates that our methods can generate more accurate saliency maps than the baselines. Inter-annotator Kappa score is 0.71, which shows strong agreements among annotators.

Visualization of saliency In Figure 3, we visualize the saliency maps generated for some randomly-sampled ImageNet images. These saliency maps are very sensible. Warmer color (representing higher saliency) regions correspond to objects. Colder color regions correspond to background. These results show that our method is effective in generating correct saliency maps. In contrast, the saliency maps generated by baselines are less sensible. For example, in the schipperke, hay, container ship, ice lolly images, higher saliency regions of baselines are in the background instead of on objects.

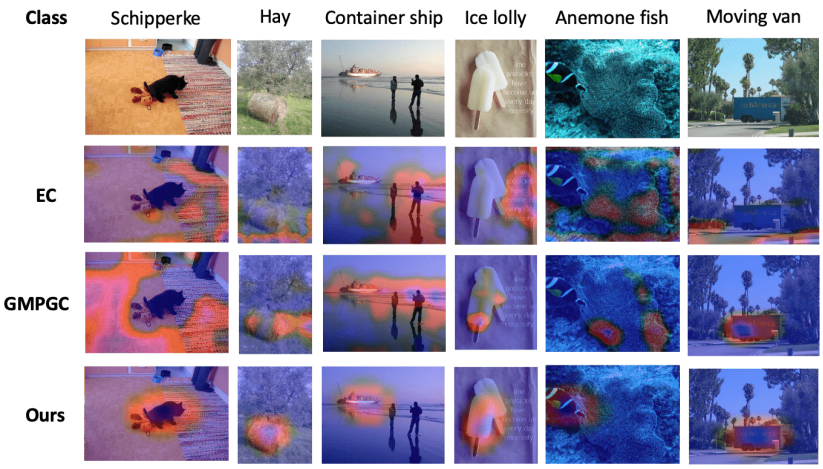

Figure 3: Visualization of saliency maps.

Ablation studies In terms of how to use saliency scores to reweight pixels, we compare the element-wise product between pixels and saliency scores in Eq.(3) with 1) element-wise addition between pixels and saliency scores; 2) element-wise product between pixels and the absolute values of saliency scores; 3) concatenate saliency map with input image and feed the concatenation into the second model W_2; 4) no reweighting. Table 5 shows results where our method is applied to DARTS2nd and P-DARTS. From this table, we make the following observations. **First**, product works better than addition. The reason is: magnitude of saliency scores (perturbations) is very small; adding them to pixels does not render a significant change of pixel values, and consequently cannot distinguish important pixels from unimportant ones. In contrast, the relative difference between saliency scores is significantly large; multiplying them to pixels can better distinguish important and unimportant pixels. **Second**, reweighting pixels using signed saliency score and absolute saliency scores does not have a significant difference. This shows that signs of saliency scores do not matter too much. **Third**, product works better than concatenation. The possible reason is: compared with concatenation, product can better differentiate important and unimportant pixels using the saliency scores. **Fourth**, reweighting pixels works better than no reweighting. This demonstrates that multiplying saliency scores to inputs is indeed helpful in identifying important pixels, which helps to improve classification performance.

Table 4: Human evaluation of saliency.

	Ratings
Darts2nd	3.30±0.24
EC-darts2nd [52]	3.42±0.16
CDEP-darts2nd [51]	3.58±0.11
GMPGC-darts2nd [47]	3.51±0.12
SANAS-darts2nd (ours)	**3.93**±0.06
Pdarts	3.26±0.14
EC-pdarts [52]	3.59±0.21
CDEP-pdarts [51]	3.46±0.19
GMPGC-pdarts [47]	3.50±0.09
SANAS-pdarts (ours)	**4.07**±0.11

In the next ablation study, we compare the adversarial attack (AA) based saliency detection method with another two saliency detection methods, including integrated gradients (IG) [60] and SmoothGrad (SG) [57] by plugging them into our framework. These studies are performed on Darts2nd and Pdarts. Table 6 shows the results, where we make two observations. First, our framework with IG and SG still outperforms vanilla Darts2nd and Pdarts. This demonstrates that our framework is a general one that generalizes beyond a single saliency detection method. Second, IG and SG perform worse than AA. A possible reason is: IG and SG restrict the definition of saliency to be gradient-based. In contrast, AA treats saliency scores as optimization variables and automatically learns them by solving an optimization problem, which is more flexible.

Table 5: Test errors of different reweighting mechanisms.

	CIFAR-100		CIFAR-10	
	Ours+darts	Ours+pdarts	Ours+darts	Ours+pdarts
Product	**16.4**±0.09	**15.2**±0.09	**2.54**±0.05	**2.45**±0.03
Addition	20.3±0.27	17.7±0.03	2.71±0.11	2.53±0.06
Absolute	16.8±0.11	15.4±0.12	2.55±0.08	2.48±0.07
Concatenate	17.9±0.11	16.4±0.06	2.74±0.07	2.61±0.08
No reweight	20.6±0.44	17.5±0.06	2.76±0.09	2.54±0.04

Table 7: Results on the GLUE benchmark. "# Param." denotes the number of model parameters. "Infer" denotes the speedup of inference time compared with $BERT_{12}$.

Method	# Param.	Infer	SST-2	MRPC	QQP	MNLI	QNLI	RTE	Average
$BERT_{12}$	109M	1x	93.5	88.9	71.2	84.6	90.5	66.4	82.5
$BERT_{12}$-T	109M	1x	93.3	88.7	71.1	84.8	90.4	66.1	82.4
$BERT_6$-PKD	67.0M	1.9x	92.0	85.0	70.7	81.5	89.0	65.5	80.6
$BERT_3$-PKD	45.7M	3.7x	87.5	80.7	68.1	76.7	84.7	58.2	76.0
$DistilBERT_4$	52.2M	3.0x	91.4	82.4	68.5	78.9	85.2	54.1	76.8
$TinyBert_4$	14.5M	9.4x	92.6	86.4	71.3	82.5	87.7	62.9	80.6
$BiLSTM_{SOFT}$	10.1M	7.6x	90.7	-	68.2	73.0	-	-	-
AdaBERT	8.3M	16.1x	91.9	85.3	70.2	81.9	86.9	64.8	80.2
EC-AdaBERT	8.7M	15.8x	91.9	85.6	70.7	81.8	86.9	65.0	80.3
CDEP-AdaBERT	8.8M	16.4x	92.5	85.9	70.6	82.2	87.4	65.2	80.6
GMPGC-AdaBERT	8.1M	15.5x	92.0	85.6	71.4	82.8	87.1	65.0	80.7
Ours-AdaBERT	8.2M	16.3x	93.4	87.0	71.8	83.7	88.5	66.6	81.8

To better understand the effectiveness of the proposed four stages performed end-to-end, we compare with the following two ablation settings, which are performed on Darts2nd and Pdarts.

- Perform the four stages separately (denoted as **Separate**) instead of end-to-end.
- Perform stages I, II, III by optimizing the weighted sum of their objective functions with weights 1, 0.5, 1, in a multi-task learning (MTL) manner (denoted as **MTL**).

Table 8 shows the results on Separate and MTL. We make two observations. First, our end-to-end method works better than Separate which conducts the four stages separately. Conducting the four stages end-to-end can enable them to mutually influence each other to achieve the overall best performance. In contrast, when conducted separately, earlier stages cannot be influenced by later stages (e.g., stage I cannot be influenced by stage IV), which leads to worse performance. Second, our method performs better than MTL. The tasks in stages I-III have an inherent order: before detecting saliency maps using a model, we first need to train this model; before training the second model on saliency-reweighted data, we need to detect the saliency maps first. MTL performs these three tasks simultaneously by minimizing a single objective, which breaks their inherent order and therefore leads to worse performance. In contrast, our method preserves this order using multi-level optimization.

Figure 4(right) shows how the test error of SANAS-darts2nd on CIFAR100 varies with γ. When $\gamma = 0$, the validation loss of W_1 is not used for architecture search and the performance is inferior (compared with $\gamma = 2$). A γ in the middle ground which properly balances the two validation losses yields the optimal performance. Using the validation loss of W_1 only is equivalent to vanilla DARTS-2nd (results are in the supplements).

Table 6: Ablation results on saliency detection methods.

	CIFAR-100	CIFAR-10
Darts [41]	20.58±0.44	2.76±0.09
IG+darts	16.92±0.08	2.62±0.06
SG+darts	17.05±0.11	2.59±0.03
AA+darts	**16.42**±0.09	**2.54**±0.05
Pdarts [11]	17.52±0.06	2.54±0.04
IG+pdarts	15.83±0.08	2.47±0.03
SG+pdarts	15.81±0.05	2.48±0.04
AA+pdarts	**15.16**±0.09	**2.45**±0.03

4.2 Experiments on text classification

In this section, we apply the proposed framework for text classification. The Gumbel softmax trick [31] is leveraged to deal with non differentiability of texts.

Dataset We conduct experiments on six datasets in the GLUE benchmark [64]: SST-2, MRPC, QQP, MNLI, QNLI and RTE. SST-2 is a sentiment classification dataset where the input text is movie review and the output label is whether the review is positive or negative. In MRPC and QQP, the input is a pair of sentences and the output is whether they are semantically equivalent. MNLI, QNLI, and RTE are textual entailment recognition datasets.

Baselines We compare with the following baselines: 1) BERT [16], 2) BERT-PKD [59], 3) DistilBERT [53], 4) TinyBERT [33], 5) $BiLSTM_{SOFT}$ [63], 6) AdaBERT [7], 7) EC-AdaBERT [52], 8) CDEP-AdaBERT [51], and 9) GMPGC-AdaBERT [47].

Hyperparameter settings Candidate operations are commonly used operations in convolutional networks, including 1D convolution, dilated convolution, pooling, identify, and zero. In dilated convolution, the kernel size includes 3, 5, and 7. Each convolution operation consists of an Relu-Conv-BatchNorm sequence. For pooling, we used average pooling and max pooling, where the kernel size is set to 3. The "SAME" padding is utilized for convolution and pooling. We optimize weight parameters using SGD. The initial learning rate is set to $2e-2$. It is annealed using a cosine

scheduler. The momentum is set to 0.9. We use Adam [34] to optimize the architecture variables. The learning rate is set to $3e-4$ and weight decay is set to $1e-3$.

Main results Table 7 shows the results. We make the following observations. First, our method works better than AdaBERT, which is an NAS method without saliency detection. This further demonstrates the effectiveness of saliency detection in improving NAS. Second, our method works better than EC, CDEP, and GMPGC. This further shows that performing saliency detection and NAS jointly is better than conducting them separately as the three baselines do. Third, compared with BERT_{12} and BERT_{12}-T, our method achieves similar performance while using much fewer parameters and being much faster during inference.

Qualitative results Table 9 shows salient words detected by different methods on a randomly sampled sentence (whose sentiment is labeled as being positive). As can be seen, our method can successfully recognize the words "entertaining" and "please" which are mostly relevant to a positive sentiment. In contrast, the two baselines fail to do that.

Table 8: Ablation results on Separate and MTL.

	CIFAR-100	CIFAR-10
Separate+darts	18.05±0.27	2.68±0.06
MTL+darts	18.26±0.12	2.70±0.05
Ours+darts	**16.42**±0.09	**2.54**±0.05
Separate+pdarts	16.49±0.07	2.51±0.03
MTL+pdarts	16.83±0.10	2.52±0.04
Ours+pdarts	**15.16**±0.09	**2.45**±0.03

5 Conclusions and discussions

In this paper, we propose to leverage the saliency information of input data to improve NAS. Our work makes the following contributions. First, our method can detect saliency and perform NAS end-to-end, based on a four-level optimization framework. The framework performs four stages in a unified way: train a preliminary model, generate saliency maps using the preliminary model, retrain the model on saliency-reweighted data, and update architecture by minimizing validation losses. Second, our framework is end-to-end differentiable, allowing using efficient gradient-based algorithms as solvers. Third, our method provides a mechanism to evaluate generated saliency maps by checking whether they are helpful for improving classification performance. We demonstrate the effectiveness of our method on several datasets.

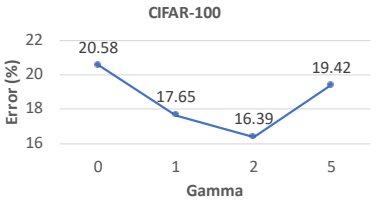

Figure 4: How errors change with γ.

One major limitation of this work is that it cannot be easily applied to non-differentiable NAS methods that are based on reinforcement learning (RL) and evolutionary algorithm (EA). The reason is that our method uses a gradient-based optimization algorithm to solve the multi-level optimization problem. For non-differentiable NAS methods, their non-differentiable objective functions do not have gradients, which therefore are not compatible with the gradient-based algorithm used by our method. Please see Appendix A.1 for discussion on how to extend our method to non-differentiable NAS methods. Another limitation of our method is its higher time cost than baselines, due to the extra computation needed for detecting saliency maps. Considering the benefits and limitations of our method, we recommend using our method in applications that strongly need high-performance architectures capable of generating sensible saliency maps but do not have strong efficiency requirements on architecture search time. For applications which have high restrictions on search cost but allow sacrificing some performance and ignoring saliency maps, other NAS methods might be better choices. Please see Appendix A.1 for a more detailed discussion.

One potential negative societal impact of our work is: in mission-critical applications such as disease diagnosis and autonomous driving, if saliency maps generated by our method are not correct, they may mislead human decision-makers. For future works, we plan to investigate these ideas: 1) formulate saliency-based network pruning [77] as a saliency-aware NAS problem and automatically search for the optimal pruning decisions based on detected saliency; 2) extend the notion of saliency from input data to blocks in neural networks, develop multi-level optimization based frameworks to detect the saliency of blocks, and perform pruning on blocks based on detected saliency.

Table 9: Top-2 salient words (marked with red color) detected by different methods.

EC	an entertaining mix of period drama and flat-out farce that should please history fans.
GMPGC	an entertaining mix of period drama and flat-out farce that should please history fans.
Ours	an entertaining mix of period drama and flat-out farce that should please history fans.

References

[1] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. *Advances in neural information processing systems*, 31, 2018.

[2] David Alvarez-Melis and Tommi S Jaakkola. Towards robust interpretability with self-explaining neural networks. *arXiv preprint arXiv:1806.07538*, 2018.

[3] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *Journal of Machine Learning Research*, 11(Jun):1803–1831, 2010.

[4] Atilim Gunes Baydin, Robert Cornish, David Martínez-Rubio, Mark Schmidt, and Frank D. Wood. Online learning rate adaptation with hypergradient descent. *CoRR*, abs/1703.04782, 2017.

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

[6] Francesco Paolo Casale, Jonathan Gordon, and Nicoló Fusi. Probabilistic neural architecture search. *CoRR*, abs/1902.05116, 2019.

[7] Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. Adabert: Task-adaptive bert compression with differentiable neural architecture search. *arXiv preprint arXiv:2001.04246*, 2020.

[8] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12270–12280, 2021.

[9] Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. *CoRR*, abs/2002.05283, 2020.

[10] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. *CoRR*, abs/2006.10355, 2020.

[11] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, 2019.

[12] Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. DARTS-: robustly stepping out of performance collapse without indicators. *CoRR*, abs/2009.01027, 2020.

[13] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair DARTS: eliminating unfair advantages in differentiable architecture search. *CoRR*, abs/1911.12126, 2019.

[14] Nicolas Couellan and Wenjuan Wang. On the convergence of stochastic bi-level gradient methods. *Optimization*, 2016.

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

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[17] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four GPU hours. In *CVPR*, 2019.

[18] Christian Etmann, Sebastian Lunz, Peter Maass, and Carola-Bibiane Schönlieb. On the connection between adversarial robustness and saliency map interpretability. *arXiv preprint arXiv:1905.04172*, 2019.

[19] Matthias Feurer, Jost Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[20] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[21] Saeed Ghadimi and Mengdi Wang. Approximation methods for bilevel programming. *arXiv preprint arXiv:1802.02246*, 2018.

[22] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[23] Riccardo Grazzi, Luca Franceschi, Massimiliano Pontil, and Saverio Salzo. On the iteration complexity of hypergradient computation. In *International Conference on Machine Learning*, pages 3748–3758. PMLR, 2020.

[24] 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. In *European Conference on Computer Vision*, pages 544–560. Springer, 2020.

[25] Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation, 2020.

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

[27] Weijun Hong, Guilin Li, Weinan Zhang, Ruiming Tang, Yunhe Wang, Zhenguo Li, and Yong Yu. Dropnas: Grouped operation dropout for differentiable architecture search. In *IJCAI*, 2020.

[28] 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. *CoRR*, abs/1704.04861, 2017.

[29] Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. DSNAS: direct neural architecture search without parameter retraining. In *CVPR*, 2020.

[30] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[31] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[32] Kaiyi Ji, Junjie Yang, and Yingbin Liang. Bilevel optimization: Convergence analysis and enhanced design. In *International Conference on Machine Learning*, pages 4882–4892. PMLR, 2021.

[33] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

[34] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[35] Alex Krizhevsky and Geoff Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40(7):1–9, 2010.

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

[37] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing neural predictions. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016.

[38] Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search, 2021.

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

[40] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018.

[41] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR*, 2019.

[42] Risheng Liu, Yaohua Liu, Shangzhi Zeng, and Jin Zhang. Towards gradient-based bilevel optimization with non-convex followers and beyond. *Advances in Neural Information Processing Systems*, 34, 2021.

[43] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: practical guidelines for efficient CNN architecture design. In *ECCV*, 2018.

[44] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[45] James Mullenbach, Sarah Wiegreffe, Jon Duke, Jimeng Sun, and Jacob Eisenstein. Explainable prediction of medical codes from clinical text. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1101–1111, 2018.

[46] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

[47] Vipin Pillai and Hamed Pirsiavash. Explainable models with consistent interpretations. *UMBC Student Collection*, 2021.

[48] 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.

[49] Zhongzheng Ren, Raymond Yeh, and Alexander Schwing. Not all unlabeled data are equal: Learning to weight data in semi-supervised learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21786–21797. Curran Associates, Inc., 2020.

[50] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

[51] Laura Rieger, Chandan Singh, William Murdoch, and Bin Yu. Interpretations are useful: penalizing explanations to align neural networks with prior knowledge. In *International Conference on Machine Learning*, pages 8116–8126. PMLR, 2020.

[52] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 2662–2670, 2017.

[53] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[54] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[55] Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. Meta-weight-net: Learning an explicit mapping for sample weighting. In *Advances in Neural Information Processing Systems*, pages 1919–1930, 2019.

[56] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[57] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smooth-grad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[58] Akshayvarun Subramanya, Vipin Pillai, and Hamed Pirsiavash. Fooling network interpretation in image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2020–2029, 2019.

[59] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*, 2019.

[60] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[61] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[62] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

[63] Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*, 2019.

[64] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[65] Yulin Wang, Jiayi Guo, Shiji Song, and Gao Huang. Meta-semi: A meta-learning approach for semi-supervised learning. *CoRR*, abs/2007.02394, 2020.

[66] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[67] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *ICLR*, 2019.

[68] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: partial channel connections for memory-efficient architecture search. In *ICLR*, 2020.

[69] Junjie Yang, Kaiyi Ji, and Yingbin Liang. Provably faster algorithms for bilevel optimization. *Advances in Neural Information Processing Systems*, 34, 2021.

[70] Yibo Yang, Hongyang Li, Shan You, Fei Wang, Chen Qian, and Zhouchen Lin. Ista-nas: Efficient and consistent neural architecture search by sparse coding, 2020.

[71] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

[72] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[73] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. *arXiv preprint arXiv:1909.09656*, 2019.

[74] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020.

[75] Guoqing Zheng, Ahmed Hassan Awadallah, and Susan T. Dumais. Meta label correction for learning with weak supervision. *CoRR*, abs/1911.03809, 2019.

[76] Pan Zhou, Caiming Xiong, Richard Socher, and Steven C. H. Hoi. Theory-inspired path-regularized differential network architecture search. *CoRR*, abs/2006.16537, 2020.

[77] Jihong Zhu and Jihong Pei. Progressive kernel pruning with saliency mapping of input-output channels. *Neurocomputing*, 467:360–378, 2022.

[78] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *International Conference on Learning Representations*, 2017.

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

[80] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The contributions are summarized in Lines 37-42, which are consistent with the main claims made in the abstract and introduction.
   (b) Did you describe the limitations of your work? [Yes] The limitations are described in Lines 352-354.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] The potential negative societal impacts are discussed in Lines 354-356.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The data is publicly available. The code is proprietary.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] The major training details are described in Lines 170-229. Additional training details are described in the supplements.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Error bars (standard deviations) are reported in Table 1, 2, 4, and 5.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] They are included in the supplements.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] They are cited in Line 199.

(b) Did you mention the license of the assets? [N/A] These datasets are public.

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] These datasets are public.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] These datasets are well-known public datasets, which do not contain personally identifiable information or offensive content.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [Yes] The instructions are included in the supplements.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] Our study does not need IRB approvals.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] The study is a voluntary study, which does not involve compensation.

A Appendix

A.1 More discussion of limitations

Apply SANAS to non-differentiable NAS methods. To apply SANAS to non-differentiable NAS methods, we have to change the current gradient-based optimization algorithm to some other non-gradient-based optimization algorithms (such as REINFORCE [66] for reinforcement learning), which might incur higher computational costs. To apply SANAS to reinforcement learning (RL) based NAS methods, we perform the following procedures. First, we use an RL controller [79] to generate a set of candidate architectures. Second, given a candidate architecture, we train its weight parameters on a training dataset, similar to stage I in SANAS. Third, given the trained model, we perform adversarial attacks to detect the saliency maps of the training data, similar to stage II in SANAS. Fourth, we use saliency maps to reweight training data and retrain the model on reweighted data, similar to stage III in SANAS. Fifth, we evaluate the retrained model on a validation set and use validation accuracy as a reward for this architecture. We repeat steps 2-5 for every candidate architecture, calculate the mean reward on all candidate architectures, and update the RL controller by maximizing the mean reward using policy gradient [79]. These procedures repeat until convergence. Similar procedures can be conducted to perform saliency-aware architecture search in evolutionary algorithm based NAS methods.

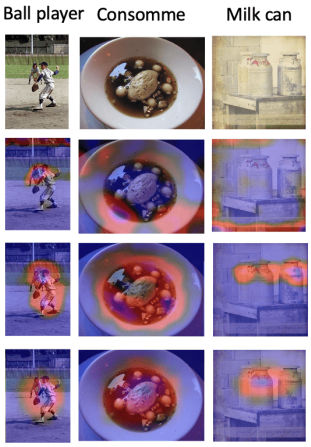

Figure 5: More examples of visual saliency maps.

When to use our method and when not. It is recommended to use SANAS in applications that strongly need high-performance neural architectures capable of generating sensible saliency maps but do not have strong requirements on the time spent on architecture search. For example, imaging-based disease diagnosis is a good application scenario of SANAS, for two reasons. First, disease diagnosis needs to be highly accurate and needs high-fidelity saliency maps for interpreting predicted diagnosis outcomes. Second, to use an automatically searched neural architecture in hospitals, FDA approval is needed, which usually takes several months. To successfully pass the FDA approval, it is acceptable to take some extra time to search for a high-quality neural architecture. For applications which have high restrictions on search cost but allow sacrificing some performance and ignoring saliency maps, other NAS methods that have higher search efficiency but lower performance and weaker saliency-generation capability than our method might be better choices. Examples of such applications are online learning applications which need to update architectures in real time.

A.2 More examples of visual saliency maps

Figure 5 shows more examples of visual saliency maps. The saliency maps detected by our method are more sensible than those detected by baselines.

A.3 More examples of salient word detection

Table 10 shows more examples of salient words detected by different methods. In each example, the top-2 words detected by our method are more salient than baselines. The prediction task corresponding to Table 9 and 10 is sentiment classification. A word is more salient if it has a stronger correlation with a sentiment (either positive or negative). For example, the word "entertaining" in Table 9 implies a positive sentiment, and therefore is considered to be salient. In contrast, the word "mix" is a neutral word that is irrelevant to sentiments, and therefore is not considered to be salient.

A.4 Improve computational efficiency of SANAS

We improved the computational efficiency of our method from both the algorithm side and implementation side. On the algorithm side, we speed up computation by approximating the optimal solution at each stage using a one-step gradient descent update [41] and reducing the frequencies of these updates. Specifically, we update the architecture A every 5 mini-batches. In contrast, baselines (including Darts, Parts, Pcdarts, Prdarts) update A on every mini-batch. We update model weights W_2 and perturbations δ every 3 mini-batches, and update W_1 on every mini-batch. We empirically found that reducing the update frequencies of certain parameters can significantly speed up convergence without sacrificing performance. Besides, when calculating hypergradients of A, we recursively

Table 10: Top-2 salient words (marked with red color) detected by different methods.

EC	an entertaining mix of period drama and flat-out farce that should please history fans.
GMPGC	an entertaining mix of period drama and flat-out farce that should please history fans.
Ours	an entertaining mix of period drama and flat-out farce that should please history fans.
EC	a very witty take on change, risk and romance, and the film uses humor to make its points about acceptance and growth.
GMPGC	a very witty take on change, risk and romance, and the film uses humor to make its points about acceptance and growth.
Ours	a very witty take on change, risk and romance, and the film uses humor to make its points about acceptance and growth.
EC	with youthful high spirits, tautou remains captivating throughout michele's religious and romantic quests, and she is backed by a likable cast.
GMPGC	with youthful high spirits, tautou remains captivating throughout michele's religious and romantic quests, and she is backed by a likable cast.
Ours	with youthful high spirits, tautou remains captivating throughout michele's religious and romantic quests, and she is backed by a likable cast.
EC	the title's lameness should clue you in on how bad the movie is.
GMPGC	the title's lameness should clue you in on how bad the movie is.
Ours	the title's lameness should clue you in on how bad the movie is.

approximate matrix-vector multiplications using finite-difference calculations [41], which reduces the computation cost from being quadratic in matrix dimensions down to linear.

On the implementation side, we speed up computation by leveraging techniques and tricks including 1) automatic mixed precision [44], 2) using multiple (4, specifically) workers and pinned memory in PyTorch DataLoader, 3) using cudNN autotuner, 4) kernel fusion, etc.

A.5 Hyperparameter tuning strategies

Most hyperparameters in our method follow their default values used in baseline methods. The only hyperparameter needing to be tuned is the tradeoff parameter γ. To tune γ on CIFAR-100, we randomly sample 2.5K data from the 25K training set and sample 2.5K data from the 25K validation set. Then we use the 5K sampled data as a hyperparameter tuning set. γ is tuned in 0.1, 0.5, 1, 2, 3. For each configuration of γ, we use the remaining 22.5K training data and 22.5K validation data to perform architecture search and use their combination to perform architecture evaluation (retraining a larger stacked network from scratch). Then we measure the performance of the stacked network on the 5K sampled data. γ value yielding the best performance on the 5K sampled data is selected. For γ in CIFAR-10 and ImageNet experiments, we simply used the value tuned on CIFAR-100 and did not conduct further tuning.