Memory-Efficient Hierarchical Neural Architecture Search for Image Denoising*

Haokui Zhang[†‡], Ying Li[†], Hao Chen[‡], Chunhua Shen[‡]
[†] Northwestern Polytechnical University, China [‡] The University of Adelaide, Australia

Abstract

Recently, neural architecture search (NAS) methods have attracted much attention and outperformed manually designed architectures on a few high-level vision tasks. In this paper, we propose HiNAS (Hierarchical NAS), an effort towards employing NAS to automatically design effective neural network architectures for image denoising. HiNAS adopts gradient based search strategies and employs operations with adaptive receptive field to build an flexible hierarchical search space. During the search stage, HiNAS shares cells across different feature levels to save memory and employ an early stopping strategy to avoid the collapse issue in NAS, and considerably accelerate the search speed. The proposed HiNAS is both memory and computation efficient, which takes only about 4.5 hours for searching using a single GPU. We evaluate the effectiveness of our proposed HiNAS on two different datasets, namely an additive white Gaussian noise dataset BSD500, and a realistic noise dataset SIM1800. Experimental results show that the architecture found by HiNAS has fewer parameters and enjoys a faster inference speed, while achieving highly competitive performance compared with state-of-the-art methods. We also present analysis on the architectures found by NAS. HiNAS also shows good performance on experiments for image de-raining.

1. Introduction

Single image denoising is an important task in low-level computer vision, which restores a clean image from a noisy one. Owing to the fact that noise corruption always occurs in the image sensing process and may degrade the visual quality of collected images, image denoising is needed for various computer vision tasks [2].

Traditional image denoising methods generally focus on modeling natural image priors and use the priors to restore the clean image, including sparse models [5, 26], Markov random field models [13], etc. One drawback of these methods is that most of them involve a complex optimization problem and can be time-consuming for inference [4, 9]. Recently, deep learning models have been successfully applied in various computer vision tasks and set new state-of-the-art. Motivated by this, most recent works on image denoising have shifted their approaches to deep learning, which builds a mapping function from noisy images to the desired corresponding clean images with deep learning models and have often outperformed conventional methods significantly [27, 33, 22]. Nonetheless, discovering state-of-the-art neural network architectures requires substantial efforts.

Recently a growing interest is witnessed in developing algorithmic solutions to automate the manual process of architecture design. Architectures automatically found by algorithms have achieved highly competitive performance in high-level vision tasks such as image classification [46], object detection [8, 36] and semantic segmentation [17, 29]. Inspired by this, here we design algorithms to automatically search for neural architectures *efficiently* for image denoising tasks. Our main contributions are summarized as follows.

1. Based on gradient based search algorithms, we propose a memory-efficient hierarchical neural architecture search approach for image denoising, termed HiNAS. To our knowledge, this is the first attempt to apply differentiable architecture search algorithms to low-level vision tasks.
2. The proposed HiNAS is able to search for both inner cell structures and outer layer widths. It is also memory and computation efficient, taking only about 4.5 hours for searching with a single GPU.
3. We apply our proposed HiNAS on two denoising datasets of different noise modes for evaluation. Experiments show that the networks found by our HiNAS achieves highly competitive performance compared with state-of-the-art algorithms, while having fewer parameters and a faster speed.
4. We conduct comparison experiments to analyse the network architectures found by our NAS algorithm in terms of the internal structure, offering some insights in architectures found by NAS.

*This work was done when H. Zhang was visiting The University of Adelaide. Correspondence: C. Shen (chunhua.shen@adelaide.edu.au).

1.1. Related Work

CNNs for image denoising. To date, due to the popularity of convolutional neural networks (CNNs), image denoising algorithms have achieved a significant performance boost. Recent network models such as DnCNN [41] and IrCNN [42] predict the residue presented in the image instead of the denoised image, showing promising performance. Lately, FFDNet [43] attempts to address spatially varying noise by appending noise level maps to the input of DnCNN. N3Net [35] formulates a differentiable version of nearest neighbor search to further improve DnCNN. DuRN-P [22] proposes a new style of residual connection, where two residual connections are employed to exploit the potential of paired operations. Some algorithms focus on denoising for real-noisy images. CBDNet [10] uses a simulated camera pipeline to supplement real training data. Similar work in [12] proposes a camera simulator that aims to accurately simulate the degradation and noise transformation performed by camera pipelines.

Network architecture search (NAS). NAS aims to design automated approaches for discovering high-performance neural architectures such that the procedure of tedious and heuristic manual design of neural architectures can be eliminated from the deep learning pipeline. Early attempts employ evolutionary algorithms (EAs) for optimizing neural architectures and parameters. The best architecture may be obtained by iteratively mutating a population of candidate architectures [19]. An alternative to EA is to use reinforcement learning (RL) techniques, e.g., policy gradients [47, 36] and Q-learning [44], to train a recurrent neural network that acts as a meta-controller to generate potential architectures—typically encoded as sequences—by exploring a predefined search space. However, EA and RL based methods are inefficient in search, often requiring a large amount of computations. Speed-up techniques are therefore proposed to remedy this issue. Exemplar works include hyper-networks [40], network morphism [6] and shared weights [30].

In terms of the design of search space and search strategies, our work is most closely related to DARTS [20], ProxylessNAS [1] and Auto-Deeplab [17]. DARTS is based on the continuous relaxation of the architecture representation, allowing efficient search of the cell architecture using gradient descent, which has achieved competitive performance. Motivated by this search efficiency, here we also use the gradient based approach as our search strategy. In addition, we employ convolution operations with adaptive receptive field in building our search space. We then extend the search space to include widths for cells by layering multiple candidate paths. Another optimization based NAS approach that has widths included in its search space is ProxylessNAS. However, it is limited to discover sequential structures and chooses kernel widths within manually designed blocks (Inverted Bottlenecks [11]). By introducing multiple paths of different widths, the search space of our HiNAS resembles Auto-Deeplab. The three major differences are: 1) to retain high resolution feature maps, we do not downsample the feature maps but reply on automatically selected dilated convolutions and deformable convolutions to adapt the receptive field; 2) we share the cell across different paths which leads to significant memory efficiency, only $1/3$ of that is needed by Auto-Deeplab counterparts; 3) to avoid the performance of the selected network degrading after a certain number of epochs (collapse problem), we employ a simple but effective early stopping search strategy. In addition, our HiNAS is proposed for low-level image restoration tasks, the three methods mentioned above are all proposed for high-level image understanding tasks. DARTS [20] and ProxylessNAS [1] are proposed for image classification. Auto-Deeplab [17] finds architectures for semantic segmentation.

Two more relevant works are E-CAE [32] and FALSR [3]. E-CAE [32] employs EA to search for an architectures of convolutional autoencoders for image inpainting and denoising. FALSR [3] is proposed for super resolution tasks. FALSR combines RL and EA and design a hybrid controller as its model generator. Both E-CAE and FALSR require a relatively large amount of computations and takes a large amout of GPU time for searching. Different from E-CAE and FALSR, our HiNAS employs gradient based strategies in searching for architectures for low-level image restoration tasks, probably for the first time, and shares cells across different feature levels to save memory. Our method only needs about 4.5 GPU hours to find a high-performing architecture on the BSD500 dataset (see Section 3.5).

2. Our Approach

Following [20, 1], we employ gradient-based architecture search strategies in our HiNAS and we search for a computation cell as the basic block then build the final architecture by stacking the found block with different widths. HiNAS defines a flexible hierarchical search space to design architectures for image denoising. In this section, we first introduce how to search for architectures of cells using continuous relaxation and adaptive search space. Then we explain how to determine the widths via multiple candidate paths and cell sharing. Last, we present our search strategy and our the loss functions.

2.1. Inner Cell Architecture Search

Continuous relaxation. For inner cell architecture search, we employ the continuous relaxation strategy proposed in DARTS [20]. More specifically, we build a supercell that integrates all possible layer types, which is show in the left side of Figure 1. This supercell is a directed acyclic graph

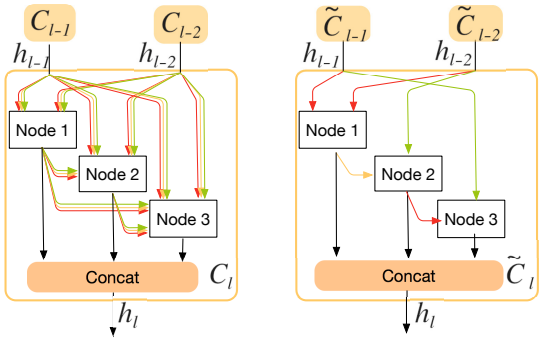

Figure 1: Inner cell architecture search. Left: supercell that contains all possible layer types. Right: the cell architecture search result, a compact cell, where each node only keeps the two most important inputs and each input is connected to the current node with a selected operation.

containing a sequence of N nodes. In Figure 1, we only show three nodes for clear exposition.

We denote the super cell in layer l as C_l, which takes outputs of previous cells and the cell before previous cells as inputs and outputs a tensor h_l. Inside C_l, each node takes the two inputs of the current cell and the outputs of all previous nodes as input and outputs a tensor. Taking the ith node in C_l as an example, the output of this node is calculated as:

$$x_{l,i} = \sum_{x_j \in I_{l,i}} O_{j\to i}(x_j), \tag{1}$$

where $I_{l,i} = \{h_{l-1}, h_{l-2}, x_{l,j<i}\}$ is the input set of node i. h_{l-1} and h_{l-2} are the outputs of cells in layers $l-1$ and $l-2$, respectively. $O_{j\to i}$ is the set of possible layer types. Here, to make the search space continuous, we operate each $O_{j\to i}$ in an continuous relaxation fashion, which is:

$$O_{j\to i}(x_j) = \sum_{k=1}^{S} \alpha_{j\to i}^{k} O^k(x_j), \tag{2}$$

where $\{O^1, O^2, \cdots, O^S\}$ correspond to S possible layer types. $\alpha_{j\to i}^{k}$ denotes the weight of operator O^k.

Adaptive search space. Following several recent image restoration networks [18, 31, 14], we do not reduce the spatial resolution of the input. To preserve pixel-level information for low-level image processing, we do not downsample the features but rely on operations with adaptive receptive field such as dilated convolutions and deformable convolutions. In this paper, we pre-define the following 6 types of basic operators:

- conv: 3×3 convolution;
- sep: 3×3 separable convolution;
- dil: 3×3 convolution with dilation rate of 2;
- def: 3×3 deformable convolution v2 [45];

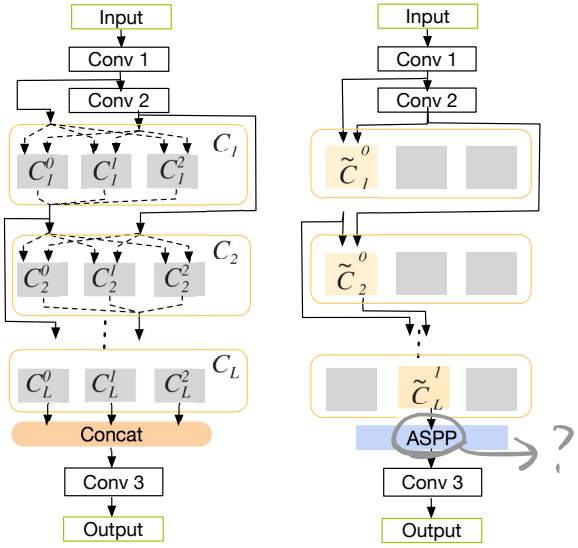

Figure 2: Outer layer width search. Left: network architecture search space, a supernet that consists of supercells and contains several supercells with different widths in each layer. Right: the final architecture obtained from the supernet, a compact network that consists of compact cells and only keeps one cell in each layer.

- skip: skip connection;
- none: no connection and return zero.

Each convolution operation starts with a ReLU activation layer and is followed by a batch normalization layer.

h_l is the concatenation of the outputs of N nodes and it can be expressed as:

$$\begin{aligned} h_l &= \text{Cell}(h_{l-1}, h_{l-2}) \\ &= \text{Concat}\{x_{l,i} | i \in \{1, 2, \cdots, N\}\}. \end{aligned} \tag{3}$$

In summary, the task of cell architecture search is to learn continuous weights α, which are updated via gradient descent. After the supercell is trained, for each node, we rank the corresponding inputs according to α values, then keep the top two inputs and remove the rest to obtain the compact cell, as shown in the right-side of Figure 1.

2.2. Memory-Efficient Width Search

Multiple candidate paths. Now we have presented the main idea of cell architecture search, which is used to design the specific architectures inside cells. As previously mentioned, the overall network is built by stacking several cells of different widths. To build the overall network, we still need to either *heuristically* set the width of each cell or search for a proper width for each cell *automatically*. In conventional CNNs, the change of widths of convolution layers is often related to the change of spatial resolutions. For instance, doubling the widths of following convolution layers after the features are downsampled. In our HiNAS,

instead of using downsample layers, we rely on operations with adaptive receptive field such as dilated convolutions and deformable convolutions to adjust the receptive field automatically. Thus the conventional experience of adjusting width no longer applies to our case.

To solve this problem, we employ the flexible hierarchical search space and leave the task of deciding width of each cell to the NAS algorithm itself, making the search space more general. In fact, several NAS algorithms in the literature also search for the outer layer width, mostly for high-level image understanding tasks. For example, FBNet [38] and MNASNet [34]consider different expansion rates inside their modules to discover compact networks for image classification.

In this section, we introduce the outer layer width search space which determines the widths of cells in different layers. Similarly, we build a supernet that contains several supercells with different widths in each layer. As illustrated in the left-side of Figure 2, the supernet mainly consists of three parts:

1) *start part*, consisting of input layer and two convolution layer;

2) *middle part*, containing L layers and each layer having three supercells of different widths;

3) *end part*, concatenating the outputs of C_L, then feeding them to a convolution layer to generate the output.

Our supernet provides three paths of cells with different widths. For each layer, the supernet decides to increase the width by twice, keeping previous width or reducing the width by two. After searching, only one cell at each layer is kept. The continuous relaxation strategy mentioned in the cell architecture search section is reused for inter cell search.

At each layer l, there are three cells C_l^0, C_l^1 and C_l^2 with widths W, $2W$ and $4W$, where W is the basic width and is set to 10 during search phase. The output feature of each layer is

$$h_l = \{h_l^0, h_l^1, h_l^2\}, \tag{4}$$

where h_l^i is the output of C_l^i. The channel width of h_l^i is $2^i NW$, where N is the number of nodes in the cells.

Cell sharing. Each cell C_l^i is connected to C_{l-1}^{i-1}, C_{l-1}^i and C_{l-1}^{i+1} in the previous layer and C_{l-2}^i two layers before. We first process the outputs h_{l-1} from those layers with a 1×1 convolution to form features f_{l-1} with width $2^i W$, matching the input of C_l^i. Then the output for the ith cell in layer l is computed with

$$h_l^i = C_l^i\left(\sum_{k=i-1}^{i+1} \beta_k^i f_{l-1}^k, f_{l-2}^i\right), \tag{5}$$

where β_k^i is the weight of f_{l-1}^k. We combine the three outputs of C_{l-1} according to corresponding weights then feed them to C_l^i as input. Here, features f_{l-1}^{i-1}, f_{l-1}^i and f_{l-1}^{i+1} come from different levels, but they share the cell C_l^i during computing h_l^i.

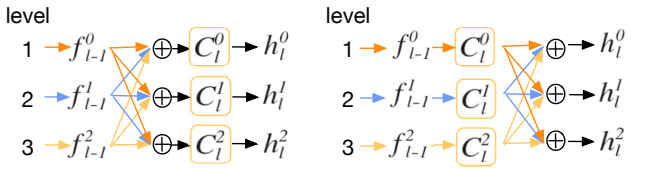

Figure 3: Comparison of cases of whether using cell sharing or not. Left: features from different levels share same cell. Using cell sharing; Right: features from different levels use different cells.

Note the similarity of this design with Auto-Deeplab, which is used to select feature strides for image segmentation. However, in Auto-Deeplab, the outputs from the three different levels are first processed by separate cells with different sets of weights before summing into the output:

$$h_l^i = \sum_{k=i-1}^{i+1} \beta_k^i C_l^k(f_{l-1}^k, f_{l-2}^i), \tag{6}$$

A comparison between Eqs. (5) and (6) is shown in Figure 3, where the inputs from layer $l-1$ are not shown out for simplicity.

For the hierarchical structure which has three candidate paths, the cell in each candidate path is used once with Eq. (5) and it is used three times with Eq. (6). By sharing the cell C_l^i, we are able to save the memory consumption by a factor of 3 in the supernet. Cell sharing has two main advantages: 1) improving applicability. NAS in general consumes much memory and computation. Improving memory efciency enables much broader applications. 2)improving searching efciency. As cell sharing saves memory consumption in the supernet, during search, we can use larger batch sizes during search to increase the search speed. We can also use a deeper and wider supernet for more accurate approximations.

Deriving the final architecture. Note that, different from the cell architecture search, we can not simply rank cells of different widths according to β values then keep the top one cell. In cell widths search, the channel widths of outputs of different cells in the same layer can be very different. Using the strategy that we have adopted in cell architecture search may lead to the widths of adjacent layers in the final network change drastically, which has a negative impact on the efficiency, as explained in [25]. In cell width search, we view the β values as probability, then use the Viterbi decoding algorithm to select the path with the maximum probability as the final result.

2.3. Searching Using Gradient Descent

Optimization function. In terms of the optimization method, our proposed HiNAS belongs to differentiable architecture search. The searching process is the optimization process. For image denoising, the two most widely used evaluation metrics are PSNR and SSIM [37]; and we design the following loss for optimizing supernet:

$$\text{loss} = \|\mathrm{f}_{\text{net}}(x) - y\|_2^2 + \lambda \cdot \mathrm{l}_{\text{ssim}}(\mathrm{f}_{\text{net}}(x), y), \quad (7)$$

where

$$\mathrm{l}_{\text{ssim}}(x, y) = \log_{10}(\text{ssim}(x, y)^{-1}), \quad (8)$$

Here x and y denote the input image and corresponding ground-truth. $\mathrm{l}_{\text{ssim}}(\cdot)$ is a loss item that is designed to enforce the visible structure of the result. $\mathrm{f}_{\text{net}}(\cdot)$ is the supernet. $\text{ssim}(\cdot)$ is structural similarity [37]. λ is a weighting coefficient and it is empirically set to 0.5 in all of our experiments.

Early stopping search. During optimization of the supernet with gradient descent, we find that the performance of network founded by HiNAS is often observed to *collapse* when the number of search epochs becomes large. The very recent method of Darts+ [16], which is concurrent to this work here, presents similar observations. Because of this collapse issue, it is hard to pre-set the number of search epochs. To solve this problem, we employ an early stopping search strategy. Specifically, we split the training set into three disjoint parts: Train W, Train A and Validation V. Sub-datasets W and A are used to optimize the weights of the supernet (kernels in convolution layers) and weights of different layer types and cells of different widths (α and β). During optimizing, we periodically evaluate the performance of the trained supernet on the validation dataset V. We stop the search procedure when the performance of supernet decreases for a pre-determined number of evaluations. Then we choose the supernet which offers the highest PSNR and SSIM scores on validation dataset V as the result of the architecture search. Details are presented in the search settings of Section 3.1.

3. Experiments

3.1. Datasets and Implementation Details

Datasets We carry out the denoising experiments on two datasets. The first one is BSD500 [28]. Following [27, 33, 18, 22], we use as the training set the combination of 200 images from the training set and 100 images from the validation set, and test on 200 images from the test set. On this dataset, we generate noisy images by adding white Gaussian noises to clean images with $\sigma = 30, 50, 70$.

The second one is SIM1800, built by ourselves. As the additive white noise models is not able to accurately reproduce the true noise in real world, by using the camera pipeline simulation method proposed in [12], we build this new denoising dataset SIM1800, which contains 1600 training samples and 212 test samples. More details of this dataset are introduced in supplementary.

Models	# parameters (M)	PSNR	SSIM
HiNAS-ws	0.63	29.14	0.8403
HiNAS-w40	0.96	29.15	0.8406
HiNAS-wm	1.13	28.89	0.8370

Table 1: Comparisons of different search settings.

Search settings. The supernet that we build for image denoising consists of 4 cells and each cell has 5 nodes. we perform architecture search on BSD500 and apply the networks found by HiNAS on both denoising datasets. Specifically, we randomly choose 2% of training samples as the validation set (Validation V). The rest are equally divided into two parts: one part is used to update the kernels of convolution layers (Train W) and the other part is used to optimize the parameters of the neural architecture (Train A).

We train the supernet at most 100 epochs with batch size of 12. We optimize the parameters of kernels and architecture with two optimizers. For learning the kernels of convolution layers, we employ the standard SGD optimizer. The momentum and weight decay are set to 0.9 and 0.0003, respectively. The learning rate decays from 0.025 to 0.001 with the cosine annealing strategy [23]. For learning the parameters of an architecture, we use the Adam optimizer, where both learning rate and weight decay are set to 0.001. In the first 20 epochs, we only update the parameters of kernels, then we start to alternately optimize the kernels of convolution layers and architecture parameters from epoch 21.

During the training process of searching, we randomly crop patches of 64×64 and feed them to the network. During evaluation, we split each image to some adjacent patches of 64×64 and then feed them to the network and finally join the corresponding patch results to otain final results of the whole test image. We evaluate the supernet for every epoch.

Training settings We train the network for 600k iterations with the Adam optimizer, where the initial learning rate, batchsize are set to 0.05 and 12, respectively. For data augmentation, we use random crop, random rotations $\in \{0°, 90°, 180°, 270°\}$, horizontal and vertical flipping. For random crop, the patches of 64×64 are randomly cropped from input images.

3.2. Benefits of Searching for the Outer Layer Width

In this section, to evaluate the benefits of searching outer layer width, we apply our HiNAS on BSD500 with three different search settings, which are denoted as HiNAS-ws, HiNAS-w40, HiNAS-wm. For HiNAS-ws, both inner cell

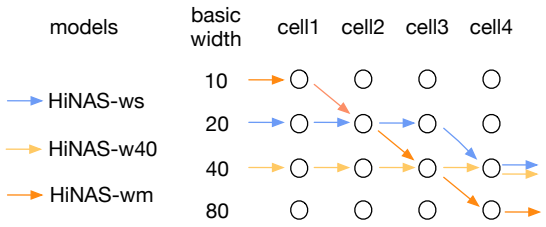

Figure 4: Comparisons of different search settings.

architectures and out layer width are found by our HiNAS algorithm. For the latter two settings, only the inner cell architectures are found by our algorithm and the outer layer widths are set manually. The basic width of each cell are set to 40 for HiNAS-w40. In HiNAS-wm, we set the basic width of the first cell to 10, then double the basic width cell by cell. The three settings are shown in Figure 4. The comparison results for denoising on BSD500 of $\sigma = 30$ are listed in Table 1.

As shown in Table 1, from HiNAS-ws to HiNAS-w40, PSNR and SSIM show slight improvement, 0.01 for PSNR and 0.0003 for SSIM. Meanwhile the corresponding number of parameters is increased by 52%. HiNAS-wm shows the worst performance, and yet it contains the most parameters. With searching for the outer layer width, HiNAS-ws achieves the best trade-off between the number of parameters and accuracy.

3.3. Benefits of Using l_{ssim} Loss

Methods	$\sigma = 30$		$\sigma = 50$		$\sigma = 70$	
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
N3Net [31]	28.66	0.8220	26.50	0.7490	25.18	0.6960
HiNAS*	29.03	0.8254	26.77	0.7498	25.42	0.6962
HiNAS**	**29.14**	**0.8403**	**26.77**	**0.7635**	**25.48**	**0.7129**

Table 2: Ablation study on BSD500. HiNAS* is trained with single loss MSE and HiNAS** is trained with the combination loss MSE and l_{ssim}.

Here we analyze how our designed loss item l_{ssim} improves image restoration results. We implement two baselines: 1) HiNAS* trained with single MSE loss; and 2) HiNAS** trained with the combination MSE loss and l_{ssim}. Table 2 shows the results of these two methods and that of N3Net on the BSD500 dataset. It is clear that both HiNAS* and HiNAS** outperform the competitive model, while HiNAS** trained with the combination loss shows even better results over HiNAS*.

3.4. Architecture Analysis

Now let us analyse the architectures designed by HiNAS. Figures 5 (a) and (b) show the search results in outer network level and the details inside cells, respectively. From Figures 5 (a) and (b), we can see that:

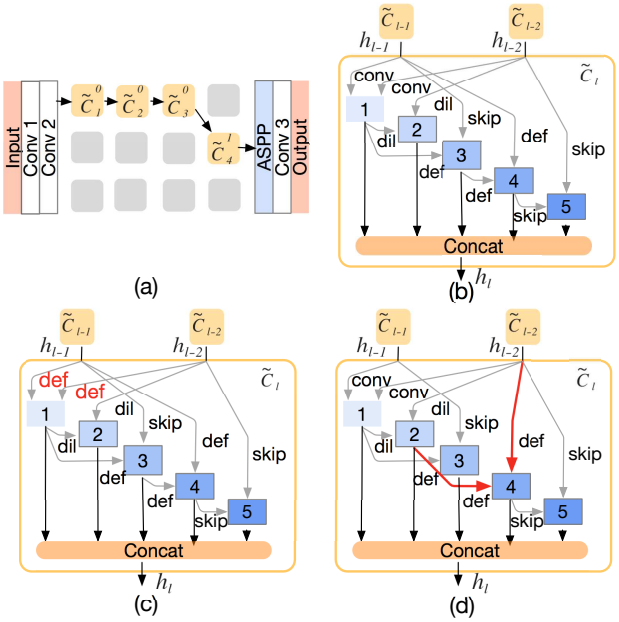

Figure 5: Architecture analysis. 'Conv', 'def' and 'dil' denote conventional, deformable and dilated convolutions. 'Skip' is skip connection. (a) Outer layer architecture; (b) inner cell architecture; (c) modified cells, $R1$; (d) modified cells, $R2$.

Methods	HiNAS	HiNAS, $R1$	HiNAS, $R2$
PSNR	**29.14**	29.06	29.13
SSIM	**0.8403**	0.8398	0.8400

Table 3: Architecture analysis.

1. In the denoising network found by our HiNAS, the width of cell that is most close to output layer has the maximum number of channels. This is consist with previous manually designed networks.
2. Generally speaking, with the same widths, deformable convolution is more flexible and powerful than other convolution operations. Even so, inside cells, instead of connecting all the nodes with the powerful deformable convolution, HiNAS connects different nodes with different types of operators, such as conventional convolution, dilated convolution and skip connection. We believe that these results prove that HiNAS is able to select proper operators.
3. Separable convolutions are not included in the searched results. We conjecture that this is caused by the fact that we do not limit FLOPS or number of parameters during search. Interestingly, the networks found by our HiNAS still have fewer parameters than other manual models.

Methods	Cell sharing	# param. (M)	σ = 30		σ = 50		σ = 70		search cost		training cost		search method
			PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	GPU	hours	GPU	hours	
E-CAE [32]	-	1.05	28.23	0.8047	26.17	0.7255	24.83	0.6636	4 V100	44.0	1 V100	3.6	EA
HiNAS	✗	0.63	29.13	0.8403	26.78	0.7636	25.44	0.7123	1 V100	21.6	1 V100	12	gradient
HiNAS	✓	0.63	29.14	0.8403	26.77	0.7635	25.48	0.7129	1 V100	4.5	1 V100	12	gradient

Table 4: Comparisons with E-CAE on BSD500. For E-CAE, the search and training time costs computed on V100 GPUs are provided by authors.

Methods	# parameters (M)	time cost (s)	$\sigma = 30$		$\sigma = 50$		$\sigma = 70$	
			PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
BM3D [4]	-	-	27.31	0.7755	25.06	0.6831	23.82	0.6240
WNNM [9]	-	-	27.48	0.7807	25.26	0.6928	23.95	0.3460
RED [27]	0.99	-	27.95	0.8056	25.75	0.7167	24.37	0.6551
MemNet [33]	4.32	-	28.04	0.8053	25.86	0.7202	24.53	0.6608
NLRN [18]	0.98	10411.49	28.15	**0.8423**	25.93	0.7214	24.58	0.6614
E-CAE [32]	1.05	-	28.23	0.8047	26.17	0.7255	24.83	0.6636
DuRN-P [22]	0.78	-	28.50	0.8156	26.36	0.7350	25.05	0.6755
N3Net [31]	0.68	121.11	28.66	0.8220	26.50	0.7490	25.18	0.6960
HiNAS	**0.63**	**83.25**	**29.14**	0.8403	**26.77**	**0.7635**	**25.48**	**0.7129**

Table 5: Denoising experiments. Comparisons with state-of-the-arts on the BSD500 dataset. We show our results in the last row. Time cost means GPU-seconds for inference on the 200 images from the test set of BSD500 using one single GTX 980 graphic card.

From Figure 5 (b), we can see that the networks found by HiNAS consist of many fragmented branches, which might be the main reason why the designed networks have better performance than previous denoising models. As explained in [25], the fragmentation structure is beneficial for accuracy. Here we verify if HiNAS improves the accuracy by designing a proper architecture or by simply integrating various branch structures and convolution operations. We modify the architecture found by our HiNAS in two different ways and then compare the modified architectures with unmodified architectures.

The first modification is replacing conventional convolutions in the searched architectures with deformable convolutions as shown in Figure 5 (c). As mentioned above, deformable convolution is more flexible than conventional convolution, replacing conventional convolutions with deformable convolutions *in theory* should improve the capacity of networks. The other modification is to change the connection relationships between nodes inside each cell, as shown in Figure 5 (d), which is aiming to verify if the connection relationship built by our HiNAS is indeed appropriate.

Following the two proposed modifications, we modify different operations and connections in different nodes. Modied architectures achieve lower performance. However, limited by space, we only show two examples here. The modification parts are marked in red in Figure 5 (c) and (d). The comparison results are listed in Table 3, where the two mentioned modification operations, are denoted as $R1$ and $R2$. From Table 3, we can see that both modifications reduce the accuracy. Replacing convolution operation reduces the PSNR and SSIM by 0.08 and 0.0005, respectively. Changing connection relationships decreases the PSNR and SSIM to 29.13 and 0.8400, respectively.

From the comparison results, we can draw a conclusion: HiNAS does find a proper structure and select proper convolution operations, instead of simply integrating a complex network with various operations. *The fact that a slight perturbation to the found architecture deteriorates the accuracy indicates that the found architecture is indeed a local optimum in the architecture search space.*

3.5. Comparisons with Other NAS Methods

Inspired by recent advances in NAS, three NAS methods have been proposed for low-level image restoration tasks [32, 3, 21]. E-CAE [32] is proposed for image inpainting and denoising. FALSR [3] is proposed for super resolution. EvoNet [21] searches for networks for medical image denoising. All three methods are based on EA and require a large amount of compuational resources and are GPU time hungry. By using four P100 GPUs, E-CAE takes four days (384 GPU hours) to execute the evolutionary algorithm and fine-tune the best model for denoising on BSD500. FALSR takes about 3 days on 8 Tesla-V100 GPUs (576 GPU hours) to find the best architecture. EvoNet uses 4 Geforce TITAN GPUs and takes 135 hours for finding the best gene. Here we mainly focus on comparing our HiNAS with E-CAE, because both them are proposed for searching for architectures for the task of denoising on BSD500. Table 4 shows the details.

Compared with E-CAE [32], FALSR [3] and EvoNet [21], our HiNAS is much faster in searching. By using a single Tesla V100, HiNAS takes about 4.5 hours in searching (4.5 hours is for three levels of σ) and 12 hours for training

Methods	PSNR	SSIM
NLRN [18]	27.53	0.8081
N3Net [31]	**27.62**	0.8191
HiNAS	27.23	**0.8326**

Table 6: Denoising results on SIM1800.

the network found by our algorithm. The fast search speed of our HiNAS benefits from the following three advantages.

1. HiNAS uses a gradient based search strategy. EA based NAS methods generally need to train a large number of children networks (genes) to update their populations. For instance, FALSR trained about 10k models during its searching process. In sharp contrast, our HiNAS only needs to train one supernet in the search stage.
2. In searching for the outer layer width, we share cells across different feature levels, saving memory consumption in the supernet. As a result, we can use larger batch sizes for training the supernet, which further speeds up search. Comparing the last two rows of Table 4, we can see that the proposed cell sharing strategy significantly accelerates the search speed without any negative influence to performance.
3. By using a simple early-stopping search strategy, HiNAS further saves 0.5 to 1.5 hours in the search stage.

3.6. Comparisons with State-of-the-art

Now we compare the HiNAS designed networks with a number of recent methods and use PSNR and SSIM to quantitatively measure the restoration performance of those methods. The comparison results on BSD500 and SIM1800 are listed in Table 5 and Table 6, respectively. Refer to supplementary materials to see visual results.

Table 5 shows that N3Net and HiNAS beat other models by a clear margin. Our proposed HiNAS achieves the best performance when σ is set to 50 and 70. When the noise level σ is set to 30, the SSIM of NLRN is slightly higher (0.002) than that of our HiNAS, but the PSNR of NLRN is much lower (nearly 1dB) than that of HiNAS.

Overall our HiNAS achieves better performance than others. In addition, compared with the second best model N3Net, the network designed by HiNAS has fewer parameters and is faster in inference. As listed in Table 5, the HiNAS designed network has 0.63M parameters, which is 92.65% that of N3Net and 60% that of E-CAE. Compared with N3Net, the HiNAS designed network reduces the inference time on the test set of BSD500 by **31.26%**.

We compare the network designed by HiNAS with NLRN and N3Net on SIM1800. Table 6 lists the results, from which we can see that the SSIM of the HiNAS designed network is much higher than that of NLRN and N3Net. However, PSNR of the HiNAS designed network is slightly lower than that of NLRN and N3Net. In summary, the performance of the HiNAS designed network is competitive with that of NLRN and N3Net on SIM1800. The corresponding visual result is shown in the supplementary material.

Methods	PSNR	SSIM
DSC [24]	18.56	0.5996
LP [15]	20.46	0.7297
DetailsNet [7]	21.16	0.7320
JORDER [39]	22.24	0.7763
JORDER-R [39]	22.29	0.7922
SCAN [14]	23.45	0.8112
RESCAN [14]	24.09	0.8410
HiNAS	**26.31**	**0.8685**

Table 7: De-raining results on Rain800. With a GTX 980 graphic card, RESCAN and **HiNAS** respectively cost 44.35, **21.80** GPU-seconds for inference on the test set of Rain800.

Additional experiments We apply the proposed HiNAS on a challenging de-raining dataset Rain800. The supernet that we build for image de-raining contains 3 cells and each cell is made up of 4 nodes. Search and training setting are consistent with that of the denoising experiments, except that we use random crop and horizontal flipping for augmentation.

The results are listed in Table 7. Corresponding visual results are included in supplementary materials. As shown in Table 7, the de-raining network designed by HiNAS achieves much better performance than others. Comparing RESCAN to the network designed by HiNAS, PSNR and SSIM are improved by 2.22 and 0.0275, respectively. In addition, the inference speed of HiNAS designed de-raining network is 2.03$\times$ that of RESCAN.

4. Conclusion

In this work, we have proposed HiNAS, an memory-efficient hierarchical architecture search algorithm for the low-level image restoration task image denoising. HiNAS adopts differentiable architecture search algorithms and a cell sharing strategy. It is both memory and computation efficient, taking only about 4.5 hours to search using a single GPU. In addition, a simple but effective early stopping strategy is used to avoid the NAS collapse problem. Our proposed HiNAS achieves highly competitive or better performance compared with previous state-of-the-art methods with fewer parameters and a faster inference speed. We believe that the proposed method can be applied to many other low-level image processing tasks.

Acknowledgments C. Shen's participation was in part supported by the ARC Grant "Deep learning that scales". H. Zhang and Y. Li's participation was in part supported by the National Natural Science Foundation of China (61871460, 61876152) and Fundamental Research Funds for the Central Universities (3102019ghxm016).

References

[1] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv: Comp. Res. Repository*, abs/1812.00332, 2018. 2

[2] Priyam Chatterjee and Peyman Milanfar. Is denoising dead? *IEEE Trans. Image Process.*, 19(4):895–911, 2009. 1

[3] Xiangxiang Chu, Bo Zhang, Hailong Ma, Ruijun Xu, Jixiang Li, and Qingyuan Li. Fast, accurate and lightweight super-resolution with neural architecture search. *arXiv: Comp. Res. Repository*, abs/1901.07261, 2019. 2, 7

[4] K Dabov, A Foi, V Katkovnik, and K Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Trans. Image Process.*, 16(8):2080–2095, 2007. 1, 7

[5] Weisheng Dong, Lei Zhang, Guangming Shi, and Xin Li. Nonlocally centralized sparse representation for image restoration. *IEEE Trans. Image Process.*, 22(4):1620–1630, 2012. 1

[6] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv: Comp. Res. Repository*, abs/1804.09081, 2018. 2

[7] Xueyang Fu, Jiabin Huang, Delu Zeng, Yue Huang, Xinghao Ding, and John Paisley. Removing rain from single images via a deep detail network. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 3855–3863, 2017. 8

[8] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc Le. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 7036–7045, 2019. 1

[9] Shuhang Gu, Lei Zhang, Wangmeng Zuo, and Xiangchu Feng. Weighted nuclear norm minimization with application to image denoising. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 2862–2869, 2014. 1, 7

[10] Shi Guo, Zifei Yan, Kai Zhang, Wangmeng Zuo, and Lei Zhang. Toward convolutional blind denoising of real photographs. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1712–1722, 2019. 2

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proc. Eur. Conf. Comp. Vis.*, pages 630–645. Springer, 2016. 2

[12] Ronnachai Jaroensri, Camille Biscarrat, Miika Aittala, and Frédo Durand. Generating training data for denoising real rgb images via camera pipeline simulation. *arXiv: Comp. Res. Repository*, abs/1904.08825, 2019. 2, 5

[13] Xiangyang Lan, Stefan Roth, Daniel Huttenlocher, and Michael Black. Efficient belief propagation with learned higher-order Markov random fields. In *Proc. Eur. Conf. Comp. Vis.*, pages 269–282. Springer, 2006. 1

[14] Xia Li, Jianlong Wu, Zhouchen Lin, Hong Liu, and Hongbin Zha. Recurrent squeeze-and-excitation context aggregation net for single image deraining. In *Proc. Eur. Conf. Comp. Vis.*, pages 254–269, 2018. 3, 8

[15] Yu Li, Robby T Tan, Xiaojie Guo, Jiangbo Lu, and Michael Brown. Rain streak removal using layer priors. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 2736–2744, 2016. 8

[16] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019. 5

[17] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 82–92, 2019. 1, 2

[18] Ding Liu, Bihan Wen, Yuchen Fan, Chen Change Loy, and Thomas S Huang. Non-local recurrent network for image restoration. In *Proc. Advances in Neural Inf. Process. Syst.*, pages 1673–1682, 2018. 3, 5, 7, 8

[19] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv: Comp. Res. Repository*, abs/1711.00436, 2017. 2

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv: Comp. Res. Repository*, abs/1806.09055, 2018. 2

[21] Peng Liu, Mohammad El Basha, Yangjunyi Li, Yao Xiao, Pina Sanelli, and Ruogu Fang. Deep evolutionary networks with expedited genetic algorithms for medical image denoising. *Medical Image Analysis*, 54:306–315, 2019. 7

[22] Xing Liu, Masanori Suganuma, Zhun Sun, and Takayuki Okatani. Dual residual networks leveraging the potential of paired operations for image restoration. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 7007–7016, 2019. 1, 2, 5, 7

[23] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *Proc. Int. Conf. Learn. Representations*, 2017. 5

[24] Yu Luo, Yong Xu, and Hui Ji. Removing rain from a single image via discriminative sparse coding. In *Proc. IEEE Int. Conf. Comp. Vis.*, pages 3397–3405, 2015. 8

[25] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proc. Eur. Conf. Comp. Vis.*, pages 116–131, 2018. 4, 7

[26] Julien Mairal, Francis R Bach, Jean Ponce, Guillermo Sapiro, and Andrew Zisserman. Non-local sparse models for image restoration. In *Proc. IEEE Int. Conf. Comp. Vis.*, volume 29, pages 54–62. Citeseer, 2009. 1

[27] Xiaojiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In *Proc. Advances in Neural Inf. Process. Syst.*, pages 2802–2810, 2016. 1, 5, 7

[28] David Martin, Charless Fowlkes, Doron Tal, Jitendra Malik, et al. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. IEEE Int. Conf. Comp. Vis.*, pages 416–423, 2001. 5

[29] Vladimir Nekrasov, Hao Chen, Chunhua Shen, and Ian Reid. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 9126–9135, 2019. 1

[30] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv: Comp. Res. Repository*, abs/1802.03268, 2018. 2

[31] Tobias Plötz and Stefan Roth. Neural nearest neighbors networks. In *Proc. Advances in Neural Inf. Process. Syst.*, pages 1087–1098, 2018. 3, 6, 7, 8

[32] Masanori Suganuma, Mete Ozay, and Takayuki Okatani. Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. In *Proc. Int. Conf. Mach. Learn.*, 2018. 2, 7

[33] Ying Tai, Jian Yang, Xiaoming Liu, and Chunyan Xu. Memnet: A persistent memory network for image restoration. In *Proc. IEEE Int. Conf. Comp. Vis.*, pages 4539–4547, 2017. 1, 5, 7

[34] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 2820–2828, 2019. 4

[35] Stefan Roth Tobias Plötz. Neural nearest neighbors networks. In *Proc. Advances in Neural Inf. Process. Syst.*, pages 1673–1682, 2018. 2

[36] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, and Chunhua Shen. NAS-FCOS: Fast neural architecture search for object detection. *arXiv: Comp. Res. Repository*, abs/1906.04423, 2019. 1, 2

[37] Zhou Wang, Alan Bovik, Hamid Sheikh, and Eero Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4):600–612, 2004. 5

[38] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 10734–10742, 2019. 4

[39] Wenhan Yang, Robby T Tan, Jiashi Feng, Jiaying Liu, Zongming Guo, and Shuicheng Yan. Deep joint rain detection and removal from a single image. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1357–1366, 2017. 8

[40] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. *arXiv: Comp. Res. Repository*, abs/1810.05749, 2018. 2

[41] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Trans. Image Process.*, 26(7):3142–3155, 2017. 2

[42] Kai Zhang, Wangmeng Zuo, Shuhang Gu, and Lei Zhang. Learning deep cnn denoiser prior for image restoration. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 3929–3938, 2017. 2

[43] Kai Zhang, Wangmeng Zuo, and Lei Zhang. Ffdnet: Toward a fast and flexible solution for cnn-based image denoising. *IEEE Trans. Image Process.*, 27(9):4608–4622, 2018. 2

[44] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 2423–2432, 2018. 2

[45] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 9308–9316, 2019. 3

[46] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv: Comp. Res. Repository*, abs/1611.01578, 2016. 1

[47] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 8697–8710, 2018. 2