Superkernel Neural Architecture Search for Image Denoising

Marcin Możejko Tomasz Latkowski Łukasz Treszczotko Michał Szafraniuk
Krzysztof Trojanowski

TCL Research Europe

Abstract

Recent advancements in Neural Architecture Search (NAS) resulted in finding new state-of-the-art Artificial Neural Network (ANN) solutions for tasks like image classification, object detection, or semantic segmentation without substantial human supervision. In this paper, we focus on exploring NAS for a dense prediction task that is image denoising. Due to a costly training procedure, most NAS solutions for image enhancement rely on reinforcement learning or evolutionary algorithm exploration, which usually take weeks (or even months) to train. Therefore, we introduce a new efficient implementation of various superkernel techniques that enable fast (6-8 RTX2080 GPU hours) single-shot training of models for dense predictions. We demonstrate the effectiveness of our method on the SIDD+ benchmark for image denoising [3].

1. Introduction

Neural architecture search (NAS) seeks to automate the process of choosing an optimal neural network architecture. Thanks to the trainable formulation of a model structure selection, it enables optimization not only for the underlying task but also for additional execution properties, *e.g*. memory constraints, or inference time. A number of search techniques have been proposed, including ones based on reinforcement learning [41], evolutionary algorithms [29], or gradient descent [6, 20, 22, 36].

The greatest challenge posed by most NAS algorithms is a trade-off between search time and memory requirements of the search procedure. In classical approaches, namely reinforcement learning or evolutionary algorithms, the statistics collected during parallel training of multiple samples from the architecture space drive the exploration process. As the exploration algorithm usually needs to train these samples from scratch, the search procedure requires hundreds [21] or thousands [7, 28, 42] of GPU hours to be performed. In order to alleviate this problem, multiple techniques were proposed. One of the most popular ones either narrows down the search to find a local cell structure which is later replicated throughout the network [42] or sharing weights across trained model samples [27]. It is crucial to notice that the memory requirements of a given training procedure are proportional to the most memory consuming model from the explored search space. This property makes these methods a good first shot at performing neural architecture search for image enhancement tasks, where the size of individual models often fills the entire GPU memory.

An alternative approach, which is usually called *single-shot* [4, 5, 15] search, seeks a different solution to the aforementioned issues. It models the search space in the form of a *supernetwork*, which is also a neural network itself. Then the search process is performed using gradient descent. These methods frequently reduce the overall search time from a couple of weeks to even a few hours [32]. Unfortunately, a huge downside of this approach is the fact that since *supernetwork* contains the whole search space, its memory requirements are significantly larger than those of typical network from the search space. This issue is even more important for the case of dense prediction tasks, where even a single model might be highly demanding in terms of memory use.

An efficient midway point is the *superkernel* approach. It reduces the structural *supernetwork* part to a single convolutional kernel, which cuts its memory requirements by order of magnitude. The improvement is achieved at the cost of narrowing down the search space selection to kernel sizes and the number of filters of a standard convolution operator.

A common solution to both prolonged training time and memory usage is using a *proxy* dataset. This dataset is usually smaller, both in terms of the number of examples and image size (*e.g*. using CIFAR10 instead of ImageNet for image classification). After initial training on a smaller dataset, the search procedure transfers the final architecture to the target task. Several approaches try to perform an architecture search procedure in a *proxyless* manner, directly on target data. It is achieved by either using a simplified search space [36] or by compressing / pruning *supernet* dur-

2160-7516/20/$31.00 ©2020 IEEE
DOI 10.1109/CVPRW50498.2020.00250

ing training [6]. The proxy-less approach may be particularly appealing for image enhancement tasks (like image denoising). For these tasks, using a *proxy* dataset is hard as proper training procedure requires high-resolution images.

In this work, we propose a new relaxed *superkernel* solution for image denoising, which is fast in training (6-8 GPU hours), memory-efficient (a *supernetwork* fits a single GPU with a batch size of 4) and might be trained in a *proxyless* manner using input image resolution of 128x128. We evaluate our models on a SIDD+ [3], (part of NTIRE 2020 Challenge on Real Image Denoising [1, 2, 3]) dataset for image denoising achieving state-of-the-art results.

2. Related work

2.1. Reinforcement Learning and Evolutionary algorithms for Neural architecture search

Initial approaches tackled Neural Architecture Search by using reinforcement learning and evolutionary algorithms. A seminal paper [41] used a reinforcement learning controller in order to produce network structures which were then trained from scratch and evaluated. Because of that, the whole training procedure took thousands of GPU hours to train a state-of-the-art model for the CIFAR10 dataset. In order to speed up computations, in the following work [42], the search space was narrowed down to only a small number of subnetwork structures, later repeated to form the final architecture. This approach was later improved by sharing weights across different subnetworks in order to transfer knowledge [27]. Similarly, [28] used a genetic algorithm instead of reinforcement learning to drive the optimization process, whereas [7] combined both of these approaches.

As memory requirements of these methods are proportional to the memory requirements of the biggest model in search space, this technique was used for memory demanding image enhancement tasks like super-resolution [14, 31, 8], medical image denoising [23], image restoration [37, 34, 16] and image inpainting [19].

2.2. *Single-shot* approaches

Another approach is to model the search space itself in the form of a neural network. Usually, it is done by a continuous sampling relaxation procedure, which approximates the discrete process of architecture selection in a differentiable manner. *e.g*. in [22] authors used softmax weights to approximate operation selection. More precisely, the approximation was achieved by combining outputs from each operation using softmax weights with learnable logits. The *supernetwork* was trained using a two-level gradient-based optimization process to separate training of convolutional weights and structural parameters. The final architecture was obtained by choosing operations by the highest logit value.

Authors of [20] pushed this approach even further and used softmax weights as a relaxation of both global CNN architecture (channels, strides, depth, connectivity) and local cell structure. The algorithm finds a network for a semantic image segmentation task that adjusts its architecture to a provided dataset. A search space forms a grid built of a local layer structure called a cell. Each cell is a general acyclic graph with learnable connections between k individually chosen and optimized operations. A grid consists of a replicated cell and a connectivity structure, which enables modeling multiple popular CNN designs like DeepLabv3, AutoEncoder, or Stacked Hourglass. The architecture search phase lasts several days on a P100 GPU.

Given that softmax weights can be an inaccurate approximation of discrete sampling when the entropy of modeled distribution is high, [36] introduced Gumbel Softmax approximation to neural architecture search [17]. In this technique, stochastic weights, used for combining different operations, seem to resemble one-hot coefficients better than softmax. The proposed solution is a computationally lightweight algorithm that finds a device-aware CNN architecture for an image classification task. The algorithm works on a predefined global CNN structure, /ie, channels, strides, and depth. The search space is spanned by a set of tensor operations, including different convolution setups, pooling, and skip connections. Optimal operation is found via gradient descent for each layer individually, taking into account classification metrics as well as `FLOPs`. The algorithm finds different architectures for different hardware setups, i.e., Samsung S10, and iPhone X. Again, the architecture search phase is time-consuming at hundreds of GPU hours.

The massive downside of the *single-shot* methods presented above is that they require computations of all possible operations in order to perform a single iteration of the search. In [6], authors try to circumvent this issue by introducing search space subsampling, which decreases memory cost. At each iteration, a training update is performed only on a randomly selected subset of possible operations. Still, these models usually require approximately twice as much memory as the most demanding model from the search space.

2.3. Superkernels

In order to alleviate memory issues, [32] introduces a new technique called a *superkernel*. In this method, the authors switch the search procedure from operation selection to kernel selection. It uses a concrete distribution in order to zero out parts of the maximal kernel, which is trained and shared across all models from the search space. As the size of the output of the convolution operation is usually of a few orders of magnitude greater than the size of its parameters, the additional cost of a search performed in the kernel space is negligible. In 4, we introduce a few new variations of this

technique based on the Gumbel Softmax relaxation [17].

2.4. Deep Learning for Image denoising

Several deep learning approaches were applied to image denoising task since the first successful MLP approach [30] achieved performance comparable to then state-of-the-art BM3D [10] approach. In [38], the authors used stacked convolutional blocks in order to approximate the residuum between noisy image and its cleaned version. Similarly, [25] introduced a residual connection not only between the input and the output of a network but also between model chained convolutional blocks. In Memnet [33], the authors introduced memory mechanisms based on *recurrent* and *gating* units in order to combine multi-level representations of an input image.

All of the methods presented above do not use any form of pooling/downsampling, which introduces a high computational cost. In order to alleviate this issue [24] used U-NET architecture. By applying the pooling / upsampling operations, they decreased the resolution of the inner convolutional volumes. It significantly cut the memory and computational burden of computations. In SGN [13], the authors noticed that using traditional pooling / upsample operations significantly decreased the quality of reconstruction as a lot of low-level image information is lost. Because of that, they introduced a shuffling (subpixel) rescaling designed to keep more details without significant growth in a computational cost. Moreover - thanks to a *top-down* self-guidance algorithm enables a light-weight combination of the multiscale image features.

3. Models for image denoising

In this section, we will introduce the base architectures that we used in our experiments. We transformed each of them into Neural Architecture Search *supernetworks* by introducing *superkernels* instead of standard convolutional filters.

3.1. Superkernel-based Multi Attentional Residual U-Net

The proposed architecture of the **Superkernel-based Multi Attentional Residual U-Net** network is shown in Figure 1. The network comprises multiple subnetworks named `SK-A-RES-UNET` trained simultaneously. The outcome of each subnetwork is passed to the channel attention block [39]. As the final layer, we used convolution with a kernel size of $k = 3$ to reduce the number of channels. The final output is additionally summed up with the input image.

The architecture of **Superkernel-based Attentional Residual U-Net** subnetwork is shown in Figure 2. This network is a modification of U-Net architecture, additionally equipped with attention mechanism and superkernel-based densely connected residual blocks (SK-DCRB). Each

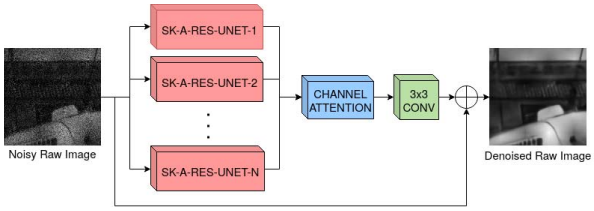

Figure 1. The architecture of **Superkernel-based Multi Attentional Residual U-Net** network

encoder layer has the same structure. Specifically, it consists of a convolutional layer with kernel size $k = 3$ followed by the ReLU activation function and SK-DCR block. As in vanilla U-Net architecture, the output of the encoder layer is passed to the decoder layer at the same spatial level. Each decoder layer takes the output from skip connection and the result from the previous layer. Unlike the common U-Net architectures, after the concatenation operation in each decoder layer, the channel attention block (CAB) is applied. We used the convolutions with a stride of 2 for down-sampling and shuffle/subpixel layers [13] for upsampling.

Figure 3 presents the schema of **Superkernel Densely Connected Residual Block**. This block has a similar structure as proposed in [26]. Each SK-DCR block comprises three convolutional layers followed by ReLU activation functions. We substituted the first two convolutional operations with superkernel. This approach allows the network to learn the most appropriate kernel size and growth rate. The output convolutional layer restores the number of filters.

3.2. Superkernel SkipInit Residual U-net

The architecture of the network is similar to the one delineated in Figure 2. The part that differs is the actual convolutional blocks. In **Superkernel-based SkipInit Residual U-net** there are multiple (2 in the chosen network) **Densely Connected Residual Blocks** (see for instance [40]) with a **SkipInit** scalar multiplication to stabilize learning as proposed in [11]. The outline of this block is presented in Figure 4. Briefly, the output of each residual branch is multiplied by a trainable scalar, which is initialized to zero, thus

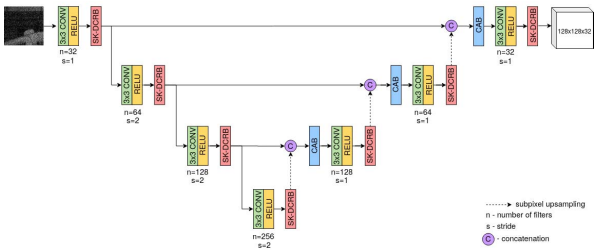

Figure 2. The architecture of **Superkernel-based Attentional Residual U-Net** subnetwork

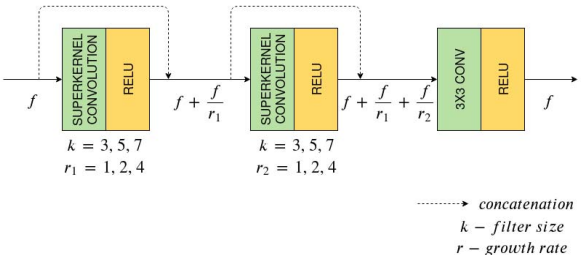

Figure 3. The outline of **Superkernel Densely Connected Residual Block**

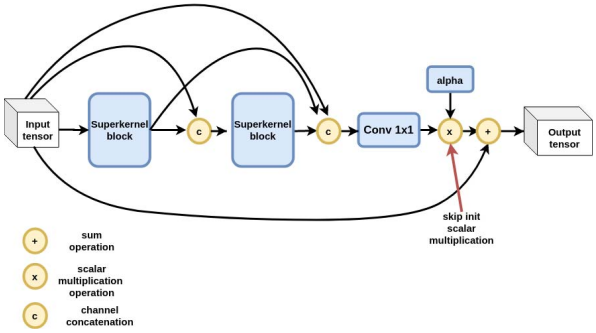

Figure 4. The architecture of **Densely Connected Residual Block with SkipInit Connection** subnetwork

making the block working like identity transformation at the beginning. If the optimization chooses to use the residual branch, then this scalar (denoted by **alfa** in Figure 4) will alter. The number of filters at a particular level is double the number of filters at the level directly above it, starting with 64 at level 0. The training of convolutional hyperparameters follows the Joint Superkernel regime described in Section 4.1.

4. Superkernels

In each of the following architectures, we applied a neural architecture search in order to find the optimal number of filters and kernel sizes for each convolution block. The search procedure was based on multiple modifications of a *superkernel* technique, namely:

- Factorized Gumbel Superkernel,
- Joint Gumbel Superkernel,
- Filterwise Gumbel Superkernel,
- Filterwise Attention-based Superkernel.

In every method, the neural architecture search explores different possibilities of kernel sizes and the number of filters. Each selection is obtained by appropriate slicing of a maximal kernel, called *superkernel*, and the way how this slicing is performed differs across different methods. In

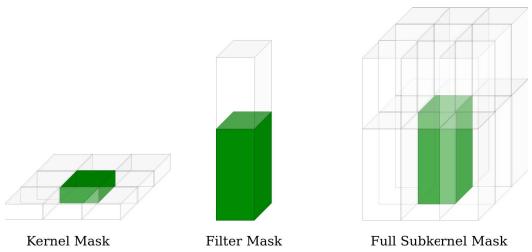

Figure 5. **The schema of Joint Superkernel and Factorized Superkernel slice generation.** A slice generated from choosing kernel size 1 from possible $k \in \{1, 3\}$ and filter size 32 from possible $f \in \{32, 48\}$.

each, the algorithm aims to optimize the structural distribution over a set of possible slices. We will later refer to the procedure of optimization of these structural parameters as *search*. Below one may find a detailed description of each NAS method.

4.1. Joint Superkernel

Using this technique, search is performed over a set of slices $S_{k,f}$ where $k \in \{k_1, \dots k_{n_k}\}$ is a size of a kernel and $f \in \{f_1, \dots, f_{n_f}\}$ is a number of filters. Both values f_i and k_i differ across different models and architectures. For each of these superkernels a maximal *superkernel* with a kernel size of $\max_{i=1,\dots n_k} k_i$ and $\max_{i=1,\dots n_f} f_i$ filter numbers is trained and shared across every subkernel. A slice $S_{k,f}$ consists of a centered subkernel with a kernel size k and first f filters of the maximal superkernel (see Figure 5).

The structural distribution is modeled as a softmax distribution over a set of all possible tuples $\{(k, f) : k \in \{k_1, \dots, k_{n_k}\}, f \in \{f_1, \dots, f_{n_f}\}\}$, with a single logit parameter $\theta_{(k,f)}$ per tuple.

4.2. Factorized Superkernel

In this technique the set of subkernels is the same as for Joint Superkernel in Section 4.1, but the structural distribution is factorized into two independent distributions $p_{k,f} = p_k p_f$, where p_k and p_f are softmax distributions over a set of possible kernels and filter sizes. In this scenario the optimized parameters form two sets of logit parameters: $\{\theta_{k_i} : k_i \in \{1, \dots n_k\}\}$ for kernel slices and $\{\theta_{f_i} : f_i \in \{1, \dots n_f\}\}$ for filter slices. In comparison to Joint Superkernel, such factorization significantly reduces number of parameters of structural distributions (from $n_f n_k$ to $n_f + n_k$) at the cost of smaller modeling flexibility.

4.3. Filterwise Superkernel

In this technique the search is performed over a set of slices $S_{k,M}$ where $k \in \{k_1, \dots, k_n\}$ - is a size of a kernel and $M \in \{0, 1\}^F$ is the set of all possible subsets of F potential filters. For each subkernel a maximal *superkernel* with kernel size of $\max k_i$ and F filters is trained and shared

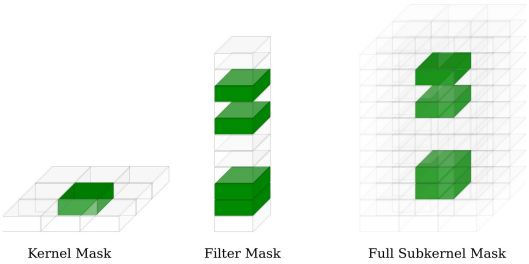

Figure 6. **The schema of Filterwise Superkernel and Filterwise Attention-based Superkernel slice generation.** A slice generated by choosing kernel size 1 from possible $k \in \{1, 3\}$ and with selected filters $f \in \{2, 3, 7, 9\}$ from 11 possible.

across every subkernel. A slice $S_{k,M}$ consists of a centered subkernel with kernel size k and and those filters for which $M_i = 1$ (see Figure 6).

The structural distribution is factorized into two independent distributions $p_{k,M} = p_k p_M$ where p_k is a softmax distribution over the set of possible kernels and p_M is modelled by F $i.i.d.$ Bernoulli distributions. p_k distribution is parametrized by a set of logit parameters $\{\theta_{k_i} : i \in \{1, \ldots, k_n\}\}$, where each of F independent Bernoulli filterwise distribution over M_i is controlled by a single logit parameter θ_{f_i}. .

4.4. Filterwise Attention-based Superkernel

In this technique, the set of subkernels is the same as in Filterwise Superkernel (Sec. 4.3). Similarly, the distribution over kernel size k is also a softmax distribution over possible kernels set. A substantial difference is in the way the distributions over masks $M \in \{0, 1\}^F$ are governed. Once again, each distribution over mask M_i is independent Bernoulli. This time, however, each of these distributions is parametrized by a single base logit parameter $\theta^b_{f_i}$ and an attention key vector $v^A_i \in \mathbb{R}^l$. The final logit of Bernoulli distribution θ_{f_i} is computed according to attention mechanism [35]:

$$\theta_{f_i} = softmax\left(v^A (v^A)^T\right) \theta^b_{f_i}. \tag{1}$$

4.5. Details of superkernel implementation

4.5.1 Sampling Relaxation

As slicing is a non-continuous operation, structural parameters of networks cannot be directly optimized using gradient descent. In order to enable this type of training, we applied a continuous approximation of sampling, namely Gumbel Softmax and Relaxed Bernoulli distributions [17]. We present an outline of this relaxation for the case of Factorized Superkernel.

We can rewrite a convolution on a volume x with a sampled subkernel $S_{k,f}$ in the following manner:

$$Conv\left(S_{k,f}, x\right) = \sum_{S_{k',f'}} Conv\left(S_{k',f'}, x\right) \mathbb{I}_{(k,f)=(k',f')}, \tag{2}$$

where $S_{k',f'}$ goes through the set of all possible superkernels and $\mathbb{I}$ is an indicator function. The Gumbel relaxation is equivalent to the following approximation:

$$\sum_{S_{k',f'}} Conv\left(S_{k',f'}, x\right) \mathbb{I}_{(k,f)=(k',f')} \approx \approx \sum_{S_{k',f'}} Conv\left(S_{k',f'}, x\right) GS(k', f'), \tag{3}$$

where GS is a sample from a Gumbel Softmax distribution with $\theta_{(k,f)}$ logit parametrization of a softmax distribution over a set of possible slices tuples (see Section 4.1).

4.5.2 Mask sampling reparametrization

In the approximation presented above, one still needs to compute $n_k n_f$ operations in order to perform the final summation. We decided to decrease the computational and memory burden by using a mask sampling trick. A superkernel slice might be reparametrized as $S_{k,f} = S * I_{k,f}$ where S is a superkernel shared across every subkernel, $I_{k,f}$ is the appropriate subkernel slice applied to *all-ones* kernel of the same shape as S and $*$ is the Hadamard product of two tensors. This reparametrization enables the following reformulation of the sampling operation:

$$\sum_{S_{k',f'}} Conv\left(S_{k',f'}, x\right) GS(k', f') = = Conv\left(S * \sum_{S_{k',f'}} I_{k',f'} GS(k', f'), x\right). \tag{4}$$

The RHS of the above equation needs only a single run of convolution operation with the maximal superkernel S masked with an average subkernel mask. This substantially decreases computational burden, both in terms of `FLOPs` and memory requirement, and enables full NAS treatment for image denoising.

4.5.3 Mask Sampling reparametrization - issue with a non-linear activation function

One may notice that the derivation above is not accurate for a search procedure which involves not only a $Conv$ operator but also a non-linear activation function. This is because of

the relation:

$$\sum_{S_{k',f'}} f\left(Conv\left(S_{k',f'}, x\right) GS(k', f')\right) = f\left(Conv\left(S * \sum_{S_{k',f'}} I_{k',f'} GS(k', f'), x\right)\right) \quad (5)$$

holds only for additive functions f. Both ReLU and PReLU, which were used in our experiments, do not have this property. Because of that, we tested two possibilities:

- **full** - where in order to keep computational feasibility we decided to ignore this fact and treat the RHS of the equation above as an approximation of the LHS,
- **separate** - where all components of the LHS were computed separately.

We have tested both approaches on a set of smaller architectures and did not notice any significant difference. If a model has a separate component in its name, it indicates it was trained using a separate approach. Otherwise it was trained using full treatment.

4.5.4 Bias sampling

We have applied a similar sampling technique as in the case of convolutional kernels to the biases. As the biases vector size is equal to the number of filters, the appropriate mask for bias slice is obtained by averaging kernel mask in all but filter dimensions.

4.5.5 Final model distillation

Once structural parameters and weights of a model are trained, we need to distill the model with the best kernel and filter sizes. We applied the following distillation strategy:

- for structural parameters governed by a **softmax** distribution, an option with a maximal probability was chosen,
- for structural masks governed by a **Bernoulli** distribution, a filter was selected if its logit was greater than 0.5.

This distillation procedure was done independently for each superkernel. We plan to use more sophisticated distillation strategies as part of our future work.

5. Experimental results

We performed several experiments in raw RGB image denoising task in order to test whether our approach to the search for the best kernel size/number of output channels combination provides noticeable benefits to the baseline (no-NAS) solution developed for the same task. In this section, we briefly describe the training procedure and the models used. We also provide the final results in terms of achieved PSNR and SSIM scores.

5.1. Training details

We trained all our models on the SSID+ [3] training data provided by NTIRE2020 Real Image Denoising rawRGB competition organizers. In our training procedure, we split the challenge images into two parts: 90% of images for a training set, and the rest 10% for validation. The split is stratified with respect to dataset mobile phone types. We used the validation data provided by the organizers of the competition as the final test set. We cut the training and validation images into 128x128 patches. We use Adam optimizer [18] with a learning rate of $2 * 10^{-5}$ for model training. The learning rate was kept constant during the learning process. As our pipeline comprised of a large set of models of different architectures and capacities, we decided to use a popular early stopping technique to constrain the training time. During the learning process, we monitored the PSNR value on a validation dataset. We evaluated our model after every 1K steps. In the case of no improvement in the PSNR within 30 validation evaluations, we stopped the training process. Depending on the hardware used (GeForce GTX 1080Ti or GeForce RTX 2080Ti), the training of our models lasted from a few hours up to 3 days.

5.2. Superkernel SkipInit Residual U-net

The overall architecture of the models in this section is based on the U-net presented in [26]. Model 1 in this subsection is the one described in Section 3.2 with U-net depth 3, three searchable growth rates 0.2, 0.4 and 0.6 in the DCR block. There are 2 DCR blocks of depth 2 at each U-net level (both for the encoder and decoder parts). At level 0, the maximum number of possible output channels is 32, and it increases with arithmetic progression at each layer. Model 2 differs from Model 1 only in the U-net depth (2 instead of 3) and the maximum number of channels at level 0 (64 instead of 32). The performance of theses models can be seen in Tables 4.5.3 and 4.5.3.

5.2.1 Chosen architecture

The search space for both models included the choice of kernel size (3 or 5) and growth rate (0.2, 0.4, 0.6) for the DCR block, see Figure 4 and Section 3.2. There where 28 DCR blocks in Model 1 and 20 in Model 2.

Model 1 in the **full** version of the **superkernel joint** framework, consecutively chose kernel size 5, with only a couple of exceptions. As regards growth rates, all but three

Superkernel Type	Network architecture			
	Model 1		Model 2	
	PSNR	**SSIM**	**PSNR**	**SSIM**
no superkernel	52.6738	0.99170	52.6883	0.99170
full	52.7330	0.99183	**52.7344**	**0.99183**
separate	**52.74498**	**0.99184**		
filterwise	52.6444	0.99179	52.6817	**0.99183**
filterwise with attention	52.7059	0.99173	52.7053	0.99191

Table 1. Superkernel-based SkipInit Residual U-Net model with two different architectures (with self-ensemble).

Superkernel Type	Network architecture			
	Model 1		Model 2	
	PSNR	**SSIM**	**PSNR**	**SSIM**
no superkernel	52.4937	**0.99157**	52.4490	0.99140
full	52.4638	0.99140	**52.4752**	0.99144
separate	**52.4957**	0.99148		
filterwise	52.4561	0.99147	52.4151	**0.99145**
filterwise with attention	52.4239	99138	52.4278	0.99152

Table 2. Superkernel-based SkipInit Residual U-Net model with two different architectures (without self-ensemble).

DCR blocks chose the largest growth rate possible. In the **separate** version, the story was similar, with kernel size 3 chosen only once.

In Model 2 with **full** version of **superkernel joint**, which had a higher baseline number of filters, but was shallower, as far as U-net architecture is concerned, chose kernel size 5 virtually everywhere. Growth rates were a bit different, with approximately 40% DCR blocks choosing a growth rate of 0.4 and the rest choosing 0.6.

5.3. Superkernel-based Multi Attentional Residual U-Net

Tables 5 and 5 contain PSNR and SSIM scores for architecture described in Section 3.1. We trained several models with a different number of subnetworks and superkernel types. Table 5 lists results without self-ensemble whereas models presented in Table 5 utilize self-ensemble technique. We use architectures with 1, 3, and 6 subnetworks. Each subnetwork has the same structure, as shown in Figure 2. As illustrated in Table 5, the application of the self-ensemble technique significantly improved the average PSNR and SSIM metrics. The best PSNR results were obtained for models containing six subnetworks. The mean PSNR value of models equipped with six subnetworks (with self-ensemble) outperformed the 1-subnetwork and 3-subnetwork models by 0.13 and 0.03, respectively.

5.3.1 Chosen architecture

The search space for the model consisted of kernel size and growth rate of densely connected residual blocks. For every first two convolutional layers in DCR block, the superkernel had to select a filter size in $k \in \{3, 5, 7\}$ and a growth rate in $1/r \in \{1, 0.5, 0.25\}$. The final superkernel choices differed depending on the part of the network. In the encoder part, the superkernel selected the kernel sizes of 5 and 7 and the growth rate of 1 and 0.5. In the decoder part, NAS selected the kernel sizes of 7 and the maximum number of filters ($r = 1$) with only some small exceptions.

6. Discussion

In all architectures introduced, at least one of Neural Architecture Search methods significantly improved over no-NAS baseline. The best single model - with 52.74 PSNR used the separate joint superkernel - and was achieved by SkipInit Residual U-net for Model 1. The difference between baseline architectures and searched ones is the most significant (approx. 0.12 of PSNR) for small models (with single subnetwork), which is promising from the perspective of model deployment on a mobile device.

The ablation study showed the importance of a self-ensembling technique. On average, it provided an improvement in both PSNR (0.21) and SSIM (0.0002) metrics. Interestingly, without self-ensembling, NAS models perform significantly worse as compared to their no-NAS baselines.

7. Potential improvements and future work

Although our search procedure shows the most significant PSNR improvement for small models, this comes at the cost of the selection of convolutional kernels with more spacious kernel sizes and higher the number of filters. How-

Superkernel Type	Number of subnetworks n					
	n=1		n=3		n=6	
	PSNR	**SSIM**	**PSNR**	**SSIM**	**PSNR**	**SSIM**
no superkernel	52.3517	**0.99156**	**52.4523**	0.99126	**52.4833**	0.99136
factorized	52.2690	0.99124	52.3799	0.99134	52.4032	**0.99150**
joint	**52.3925**	0.99138	52.4153	0.99140	52.4118	0.99147
filterwise	52.2887	0.99121	52.3503	**0.99145**	52.3516	0.99131
filterwise with attention	52.2932	0.99122	52.4365	0.99144	52.4195	0.99119
$\mu \pm \sigma$	52.3190 $\pm$0.0459	0.99132 $\pm$13.39e-5	52.4069 $\pm$0.0373	**0.99138** $\pm$7.05e-5	**52.4139** $\pm$0.0421	0.99137 $\pm$11.22e-5

Table 3. Superkernel-based Multi Attentional Residual U-Net model for different number of subnetworks and superkernel types (without self-ensemble).

Superkernel Type	Number of subnetworks n					
	n=1		n=3		n=6	
	PSNR	**SSIM**	**PSNR**	**SSIM**	**PSNR**	**SSIM**
no superkernel	52.4470	0.99157	52.5969	0.99150	52.6642	0.99159
factorized	52.4998	0.99157	52.6184	0.99167	**52.6981**	**0.99176**
joint	**52.5668**	**0.99163**	52.6016	0.99164	52.6189	0.99159
filterwise	52.5144	0.99150	52.5982	0.99180	52.6654	0.99163
filterwise with attention	52.5438	0.99157	**52.6880**	**0.99186**	52.5943	0.99150
$\mu \pm \sigma$	52.5144 $\pm$0.0409	0.99157 $\pm$4.12e-5	52.6206 $\pm$0.0346	**0.99169** $\pm$12.64e-5	**52.6482** $\pm$0.0369	0.99161 $\pm$8.45e-5

Table 4. Superkernel-based Multi Attentional Residual U-Net model for different number of subnetworks and superkernel types (with self-ensemble).

ever, a differentiable representation of kernel size enables the introduction of additional computational constraints to an optimization process. We expect that training equipped with additional computational regularization should result in finding models deployable on portable devices.

Our search procedure resembles variational inference methods for Bayesian Deep Learning; thus, our method might suffer from an insufficient exploration of a search space as reported in [12]. This issue might arise as our methods explore only points which are close to gradient descent trajectory, which narrows the explored region of the search space and makes it potentially sensitive to weight initialization. We plan to explore this sensitivity as our future work. Moreover, in our training procedure, both weights and structural parameters are optimized jointly. We plan to explore a two-level optimization (similar to [22]) in order to inspect how the coupling of these parameters affects the results.

As sampling within different superkernels is performed independently, there may arise issues connected to fairness [9]. Because of that, the network might quickly converge to a random sub-optimal solution, according to Matthew's law. We plan to test more advanced probabilistic models (*e.g.* based on Markov Random Fields, recurrent neural networks, or attention) in order to model more advanced search spaces and perform broader exploration.

Additionally, we observed that the training procedure could have a significant impact on the final results. Specifically, models equipped with superkernel might need longer training time (higher patience) in comparison to non-superkernel models. We suspect that this might be a reason behind performance drop for NAS models without self-ensembling. We plan to investigate this topic in future work.

8. Conclusion

In this work, we introduced a fast and light-weight algorithm for neural architecture search for image denoising. We proved that it could achieve state-of-the-art results exceeding no-NAS solution by a significant margin. We have shown that proposed superkernel techniques can achieve results comparable to the state-of-the-art architectures for image denoising within few GPU hours.

References

[1] Abdelrahman Abdelhamed, Stephen Lin, and Michael S. Brown. A high-quality denoising dataset for smartphone cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1692–1700, 2018.

[2] Abdelrahman Abdelhamed, Radu Timofte, and Michael S. Brown. Ntire 2019 challenge on real image denoising: Methods and results. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019.

[3] Abdelrahman Abdelhamed et al. NTIRE 2020 Challenge on Real Image Denoising: Dataset, Methods and Results. In *CVPR Workshops*, 2020.

[4] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 550–559, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

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

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

[7] Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in neural information processing systems*, pages 8699–8710, 2018.

[8] Xiangxiang Chu, Bo Zhang, Hailong Ma, Ruijun Xu, Jixiang Li, and Qingyuan Li. Fast, accurate and lightweight super-resolution with neural architecture search. *arXiv preprint arXiv:1901.07261*, 2019.

[9] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019.

[10] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 16:2080–95, 2007.

[11] Soham De and Samuel L. Smith. Batch Normalization Biases Deep Residual Networks Towards Shallow Paths. *arXiv preprint arXiv:2002.10444*, 2020.

[12] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep Ensembles: A Loss Landscape Perspective. *arXiv preprint arXiv:1912.02757*, 2019.

[13] Shuhang Gu, Yawei Li, Luc Van Gool, and Radu Timofte. Self-Guided Network for Fast Image Denoising. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct. 2019.

[14] Yong Guo, Yongsheng Luo, Zhenhao He, Jin Huang, and Jian Chen. Hierarchical Neural Architecture Search for Single Image Super-Resolution. *arXiv preprint arXiv:2003.04619*, 2020.

[15] 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. *arXiv preprint arXiv:1904.00420*, 2019.

[16] Kary Ho, Andrew Gilbert, Hailin Jin, and John Collomosse. Neural Architecture Search for Deep Image Prior. *arXiv preprint arXiv:2001.04776*, 2020.

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

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Yaoman Li and Irwin King. Architecture Search for Image Inpainting. In *International Symposium on Neural Networks*, pages 106–115. Springer, 2019.

[20] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 82–92, 2019.

[21] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

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

[23] Peng Liu, Mohammad D. El Basha, Yangjunyi Li, Yao Xiao, Pina C. Sanelli, and Ruogu Fang. Deep Evolutionary Networks with Expedited Genetic Algorithms for Medical Image Denoising. *Medical image analysis*, 54:306–315, 2019. Publisher: Elsevier.

[24] Pengju Liu, Hongzhi Zhang, Kai Zhang, Liang Lin, and Wangmeng Zuo. Multi-level wavelet-CNN for image restoration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 773–782, 2018.

[25] Xiaojiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections.

In *Advances in neural information processing systems*, pages 2802–2810, 2016.

[26] Bumjun Park, Songhyun Yu, and Jechang Jeong. Densely connected hierarchical network for image denoising. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019.

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

[28] 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.

[29] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org, 2017.

[30] Uwe Schmidt and Stefan Roth. Shrinkage Fields for Effective Image Restoration. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[31] Dehua Song, Chang Xu, Xu Jia, Yiyi Chen, Chunjing Xu, and Yunhe Wang. Efficient residual dense block search for image super-resolution. *arXiv preprint arXiv:1909.11409*, 2019.

[32] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. *arXiv preprint arXiv:1904.02877*, 2019.

[33] Ying Tai, Jian Yang, Xiaoming Liu, and Chunyan Xu. Memnet: A persistent memory network for image restoration. In *Proceedings of the IEEE international conference on computer vision*, pages 4539–4547, 2017.

[34] Gerard Jacques van Wyk and Anna Sergeevna Bosman. Evolutionary Neural Architecture Search for Image Restoration. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, \Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

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

[37] Haokui Zhang, Ying Li, Hao Chen, and Chunhua Shen. Memory-Efficient Hierarchical Neural Architecture Search for Image Denoising. *arXiv e-prints*, page arXiv:1909.08228, Sept. 2019.

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

[39] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 286–301, 2018.

[40] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2472–2481, 2018.

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

[42] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.