Densely Connected Search Space for More Flexible Neural Architecture Search

Jiemin Fang\textsuperscript{1,}, Yuzhu Sun\textsuperscript{1,}, Qian Zhang\textsuperscript{2,}, Yuan Li\textsuperscript{2,}, Wenyu Liu\textsuperscript{1,}, Xinggang Wang\textsuperscript{1†}
\textsuperscript{1}School of EIC, Huazhong University of Science and Technology \textsuperscript{2}Horizon Robotics
\{jaminfong, yzsun, liuwj, xgwang\}@hust.edu.cn
\{qian01.zhang, yuan.li\}@horizon.ai

Abstract

Neural architecture search (NAS) has dramatically advanced the development of neural network design. We revisit the search space design in most previous NAS methods and find the number and widths of blocks are set manually. However, block counts and block widths determine the network scale (depth and width) and make a great influence on both the accuracy and the model cost (FLOPs/latency). In this paper, we propose to search block counts and block widths by designing a densely connected search space, i.e., DenseNAS. The new search space is represented as a dense super network, which is built upon our designed routing blocks. In the super network, routing blocks are densely connected and we search for the best path between them to derive the final architecture. We further propose a chained cost estimation algorithm to approximate the model cost during the search. Both the accuracy and model cost are optimized in DenseNAS. For experiments on the MobileNetV2-based search space, DenseNAS achieves 75.3\% top-1 accuracy on ImageNet with only 361MB FLOPs and 17.9ms latency on a single TITAN-XP. The larger model searched by DenseNAS achieves 76.1\% accuracy with only 479M FLOPs. DenseNAS further promotes the ImageNet classification accuracies of ResNet-18, -34 and -50-B by 1.5\%, 0.5\% and 0.3\% with 200M, 600M and 680M FLOPs reduction respectively. The related code is available at \url{https://github.com/JaminFong/DenseNAS}.

1. Introduction

In recent years, neural architecture search (NAS) \cite{cnnnas, googlenet, mobilenetv2, darts} has demonstrated great successes in designing neural architectures automatically and achieved remarkable performance gains in various tasks such as image classification \cite{cnnnas, googlenet}, semantic segmentation \cite{segnas, unet} and object detection \cite{maskrcnn, fasterrcnn}. NAS has been a critically important topic for architecture designing.

In NAS research, the search space plays a crucial role that constrains the architectures in a prior-based set. The performance of architectures produced by NAS methods is strongly associated with the search space definition. A more flexible search space has the potential to bring in architectures with more novel structures and promoted performance. We revisit and analyze the search space design in most previous works \cite{searchspace, darts, dartsplus, dis}. For a clear illustration, we review the following definitions. \textit{Block} denotes a set of layers/operations in the network which output feature maps with the same spatial resolution and the same width (number of channels). \textit{Stage} denotes a set of sequential \textit{blocks} whose outputs are under the same spatial resolution settings. Different \textit{blocks} in the same stage are allowed to have various widths. Many recent works \cite{mobilenetv2, dartsplus, efficientnet} stack the inverted residual convolution modules (MBConv) defined in MobileNetV2 \cite{mobilenetv2} to construct the search space. They

\textsuperscript{†}The work is performed during the internship at Horizon Robotics.
\textsuperscript{††}Corresponding author.
search for different kernel sizes and expansion ratios in each MBConv. The depth is searched in terms of layer numbers in each block. The searched networks with MBConvs show high performance with low latency or few FLOPs.

In this paper, we aim to perform NAS in a more flexible search space. Our motivation and core idea are illustrated in Fig. 1. As the upper part of Fig. 1 shows, the number of blocks in each stage and the width of each block are set manually and fixed during the search process. It means that the depth search is constrained within the block and the width search cannot be performed. It is worth noting that the scale (depth and width) setting is closely related to the performance of a network, which has been demonstrated in many previous theoretical studies [35, 32] and empirical results [17, 40]. Inappropriate width or depth choices usually cause drastic accuracy degradation, significant computation cost, or unsatisfactory model latency. Moreover, we find that recent works [37, 3, 41] manually tune width settings to obtain better performance, which indicates the design of network width demands much prior-based knowledge and trial-and-error.

We propose a densely connected search space to tackle the above obstacles and name our method as DenseNAS. We show our novelly designed search space schematically in the bottom part of Fig. 1. Different from the search space design principles in the previous works [3, 41], we allow more blocks with various widths in one stage. Specifically, we design the routing blocks to construct the densely connected super network which is the representation of the search space. From the beginning to the end of the search space, the width of the routing block increases gradually to cover more width options. Every routing block is connected to several subsequent ones. This formulation brings in various paths in the search space and we search for the best path to derive the final architecture. As a consequence, the block widths and counts in each stage are allocated automatically. Our method extends the depth search into a more flexible space. Not only the number of layers within one block but also the number of blocks within one stage can be searched. The block width search is enabled as well. Moreover, the positions to conduct spatial down-sampling operations are determined along with the block counts search.

We integrate our search space into the differentiable NAS framework by relaxing the search space. We assign a probability parameter to each output path of the routing block. During the search process, the distribution of probabilities is optimized. The final block connection paths in the super network are derived based on the probability distribution. To optimize the cost (FLOPs/latency) of the network, we design a chained estimation algorithm targeted at approximating the cost of the model during the search.

Our contributions can be summarized as follows.

- We propose a densely connected search space that enables network/block widths search and block counts search. It provides more room for searching better networks and further reduces expert designing efforts.

- We propose a chained cost estimation algorithm to precisely approximate the computation cost of the model during search, which makes the DenseNAS networks achieve high performance with low computation cost.

- In experiments, we demonstrate the effectiveness of our method by achieving SOTA performance on the MobileNetV2 [37]-based search space. Our searched network achieves 75.3% accuracy on ImageNet [9] with only 361MB FLOPs and 17.9ms latency on a single TITAN-XP.

- DenseNAS can further promote the ImageNet classification accuracies of ResNet-18, -34 and -50-B [18] by 1.5%, 0.5% and 0.3% with 200M, 600M, 680M FLOPs and 1.5ms, 2.4ms, 6.1ms latency reduction respectively.

2. Related Work

Search Space Design

NASNet [49] is the first work to propose a cell-based search space, where the cell is represented as a directed acyclic graph with several nodes inside. NASNet searches for the operation types and the topological connections in the cell and repeat the searched cell to form the whole network architecture. The depth of the architecture (i.e., the number of repetitions of the cell), the widths and the occurrences of down-sampling operations are all manually set. Afterwards, many works [28, 34, 36, 30] adopt a similar cell-based search space. However, architectures generated by cell-based search spaces are not friendly in terms of latency or FLOPs. Then MnasNet [39] stacks MBConvs defined in MobileNetV2 [37] to construct a search space for searching efficient architectures. Some works [3, 13, 41, 8] simplify the search space by searching for the expansion ratios and kernel sizes of MBConv layers.

Some works study more about the search space. Liu et al. [29] proposes a hierarchical search space that allows flexible network topologies (directed acyclic graphs) at each level of the hierarchies. Auto-DeepLab [27] creatively designs a two-level hierarchical search space for semantic segmentation networks. CAS [46] customizes the search space design for real-time segmentation networks. RandWire [42] explores randomly wired architectures by designing network generators that produce new families of models for searching. Our proposed method designs a densely connected search space beyond conventional search constrains to generate the architecture with a better trade-off between accuracy and model cost.
various paths to transmit tensors.

### 3.1.1 Basic Layer

We define the basic layer to be the elementary structure in our search space. One basic layer represents a set of candidate operations which include MBConvs and the skip connection. MBConvs are with kernel sizes of \{3, 5, 7\} and expansion ratios of \{3, 6\}. The skip connection is for the depth search. If the skip connection is chosen, the corresponding layer is removed from the resulting architecture.

### 3.1.2 Routing Block

For the purpose of establishing various paths in the super network, we propose the routing block with the ability of aggregating tensors from preceding routing blocks and transmit tensors to subsequent ones. We divide the routing block into two parts, shape-alignment layers and basic layers.

Shape-alignment layers exist in the form of several parallel branches, while every branch is a set of candidate operations. They take input tensors with different shapes (including widths and spatial resolutions) which come from multiple preceding routing blocks and transform them into tensors with the same shape. As shape-alignment layers are required for all routing blocks, we exclude the skip connection in candidate operations of them. Then tensors processed by shape-alignment layers are aggregated and sent to several basic layers. The subsequent basic layers are used for feature extraction whose depth can also be searched.

### 3.1.3 Dense Super Network

Many previous works \[39, 3, 41\] manually set a fixed number of blocks, and retain all the blocks for the final architecture. Benefiting from the aforementioned structures of routing blocks, we introduce more routing blocks with various widths to construct the dense super network which is the representation of the search space. The final searched architecture is allowed to select a subset of the routing blocks and discard the others, giving the search algorithm more room.

We define the super network as \(N_{\text{sup}}\) and assume it to consist of \(N\) routing blocks, \(N_{\text{sup}} = \{B_1, B_2, ..., B_N\}\). The network structure is shown in Fig. 2. We partition the entire network into several stages. As Sec. 1 defines, each stage contains routing blocks with various widths and the same spatial resolution. From the beginning to the end of the super network, the widths of routing blocks grow gradually. In the early stage of the network, we set a small growing stride for the width because large width settings in the early network stage will cause huge computational cost. The growing stride becomes larger in the later stages. This design principle of the super network allows more possibilities of block counts and block widths.

We assume that each routing block in the super network connects to \(M\) subsequent ones. We define the connection between the routing block \(B_i\) and its subsequent routing block \(B_j\) (\(j > i\)) as \(C_{ij}\). The spatial resolutions of \(B_i\) and \(B_j\) are \(H_i \times W_i\) and \(H_j \times W_j\) respectively (normally \(H_i = W_i\) and \(H_j = W_j\)). We set some constraints on the connections to avoid the stride of the spatial down-sampling exceeding 2. Specifically, \(C_{ij}\) only exists when \(j - i \leq M\) and \(H_i/H_j \leq 2\). Following the above paradigms, the search space is constructed as a dense super network based on the connected routing blocks.

### 3.2. Relaxation of Search Space

We integrate our search space by relaxing the architectures into continuous representations. The relaxation is implemented on both the basic layer and the routing block. We can search for architectures via back-propagation in the relaxed search space.

**3.2.1 Relaxation in the Basic Layer**

Let \(O\) be the set of candidate operations described in Sec. 3.1.1. We assign an architecture parameter \(\alpha^o_i\) to the candidate operation \(o \in O\) in basic layer \(\ell\). We relax the basic layer by defining it as a weighted sum of outputs from all candidate operations. The architecture weight of the operation is computed as a softmax of architecture parameters over all operations in the basic layer:

\[
\omega^\ell_o \equiv \frac{\exp(\alpha^o_i)}{\sum_{o' \in O} \exp(\alpha^{o'}_i)}. 
\] (1)

The output of basic layer \(\ell\) can be expressed as

\[
x_{\ell+1} = \sum_{o \in O} \omega^\ell_o \cdot o(x_\ell), \quad (2)
\]

where \(x_\ell\) denotes the input tensor of basic layer \(\ell\).

**3.2.2 Relaxation in the Routing Block**

We assume that the routing block \(B_i\) outputs the tensor \(b_i\) and connects to \(m\) subsequent blocks. To relax the block connections as a continuous representation, we assign each output path of the block an architecture parameter. Namely the path from \(B_i\) to \(B_j\) has a parameter \(\beta_{ij}\). Similar to how we compute the architecture weight of each operation above, we compute the probability of each path using a softmax function over all paths between the two routing blocks:

\[
p_{ij} = \frac{\exp(\beta_{ij})}{\sum_{k=1}^m \exp(\beta_{ik})}. \quad (3)
\]

For routing block \(B_i\), we assume it takes input tensors from its \(m'\) preceding routing blocks \((B_{i-m'}, B_{i-m'+1}, \ldots)\).
\( B_{i-m'+2} \ldots B_{i-1} \). As shown in Fig. 2, the input tensors from these routing blocks differ in terms of width and spatial resolution. Each input tensor is transformed to the same size by the corresponding branch of shape-alignment layers in \( B_i \). Let \( H_{ik} \) denote the \( k \)th transformation branch in \( B_i \) which is applied to the input tensor from \( B_{i-k} \), where \( k = 1 \ldots m' \). Then the input tensors processed by shape-alignment layers are aggregated by a weighted-sum using the path probabilities,

\[
x_i = \sum_{k=1}^{m'} p_{i-k,i} \cdot H_{ik}(x_{i-k}).
\]  

(4)

It is worth noting that the path probabilities are normalized on the output dimension but applied on the input dimension (more specifically on the branches of shape-alignment layers). One of the shape-alignment layers is essentially a weighted-sum mixture of the candidate operations. The layer-level parameters \( \alpha \) control which operation to be selected, while the outer block-level parameters \( \beta \) determine how blocks connect.

### 3.3. Chained Cost Estimation Algorithm

We propose to optimize both the accuracy and the cost (latency/FLOPs) of the model. To this end, the model cost needs to be estimated during the search. In conventional (latency/FLOPs) of the model. To this end, the model cost needs to be estimated during the search. In conventional

3.4. Search Procedure

Benefiting from the continuously relaxed representation of the search space, we can search for the architecture by updating the architecture parameters (introduced in Sec. 3.2) using stochastic gradient descent. We find that at the beginning of the search process, all the weights of the operations are under-trained. The operations or architectures which converge faster are more likely to be strengthened, which leads to shallow architectures. To tackle this, we split our search procedure into two stages. In the first stage, we only optimize the weights for enough epochs to get operations sufficiently trained until the accuracy of the model is not too low. In the second stage, we activate the architecture optimization. We alternatively optimize the operation weights by descending \( \nabla_w L_{train}(w, \alpha, \beta) \) on the training set, and optimize the architecture parameters by descending \( \nabla_{\alpha,\beta} L_{val}(w, \alpha, \beta) \) on the validation set. Moreover, a dropping-path training strategy \([1, 3]\) is adopted to decrease memory consumption and decouple different architectures in the super network.

When the search procedure terminates, we derive the final architecture based on the architecture parameters \( \alpha, \beta \). At the layer level, we select the candidate operation with the maximum architecture weight, i.e., \( \text{arg max}_{o \in \mathcal{O}} \alpha^o \). At the network level, we use the Viterbi algorithm \([15]\) to derive the paths connecting the blocks with the highest total transition probability based on the output path probabilities. Every block in the final architecture only connects to the next one.

### 4. Experiments

In this section, we first show the performance with the MobileNetV2 \([37]\)-based search space on ImageNet \([9]\) classification. Then we apply the architectures searched on ImageNet to object detection on COCO \([25]\). We further extend our DenseNAS to the ResNet \([18]\)-based search space. Finally, we conduct some ablation studies and analysis. The implementation details are provided in the appendix.

\[ \text{cost} = \text{cost}^1. \]  

(7)

We design a loss function with the cost-based regularization to achieve the multi-objective optimization:

\[
\mathcal{L}(w, \alpha, \beta) = \mathcal{L}_{CE} + \lambda \log_{10} \text{cost},
\]  

(8)

where \( \lambda \) and \( \tau \) are the hyper-parameters to control the magnitude of the model cost term.
Table 1: Our results on the ImageNet classification with the MobileNetV2-based search space compared with other methods. Our models achieve higher accuracies with lower latencies. For GPU latency, we measure all the models with the same setup (on one TITAN-XP with a batch size of 32).

<table>
<thead>
<tr>
<th>Model</th>
<th>FLOPs</th>
<th>GPU Latency</th>
<th>Top-1 Acc(%)</th>
<th>Search Time (GPU hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0-MobileNetV2 [37]</td>
<td>585M</td>
<td>28.0ms</td>
<td>74.7</td>
<td>-</td>
</tr>
<tr>
<td>NASNet-A [49]</td>
<td>564M</td>
<td>-</td>
<td>74.0</td>
<td>48K</td>
</tr>
<tr>
<td>AmoebaNet-A [36]</td>
<td>555M</td>
<td>-</td>
<td>74.5</td>
<td>76K</td>
</tr>
<tr>
<td>DARTS [30]</td>
<td>574M</td>
<td>36.0ms</td>
<td>73.3</td>
<td>96</td>
</tr>
<tr>
<td>RandWire-WS [42]</td>
<td>583M</td>
<td>-</td>
<td>74.7</td>
<td>-</td>
</tr>
<tr>
<td>DenseNAS-Large</td>
<td>479M</td>
<td>28.9ms</td>
<td><strong>76.1</strong></td>
<td>64</td>
</tr>
<tr>
<td>1.0-MobileNetV1 [21]</td>
<td>575M</td>
<td>16.8ms</td>
<td>70.6</td>
<td>-</td>
</tr>
<tr>
<td>1.0-MobileNetV2 [37]</td>
<td>300M</td>
<td>19.5ms</td>
<td>72.0</td>
<td>-</td>
</tr>
<tr>
<td>FBNet-A [41]</td>
<td>249M</td>
<td>15.8ms</td>
<td>73.0</td>
<td>216</td>
</tr>
<tr>
<td>DenseNAS-A</td>
<td>251M</td>
<td>13.6ms</td>
<td>73.1</td>
<td>64</td>
</tr>
<tr>
<td>MnasNet [39]</td>
<td>317M</td>
<td>19.7ms</td>
<td>74.0</td>
<td>91K</td>
</tr>
<tr>
<td>FBNet-B [41]</td>
<td>295M</td>
<td>18.9ms</td>
<td>74.1</td>
<td>216</td>
</tr>
<tr>
<td>Proxyless(mobile) [3]</td>
<td>320M</td>
<td>21.3ms</td>
<td>74.6</td>
<td>200</td>
</tr>
<tr>
<td>DenseNAS-B</td>
<td>314M</td>
<td><strong>15.4ms</strong></td>
<td><strong>74.6</strong></td>
<td>64</td>
</tr>
<tr>
<td>MnasNet-92 [39]</td>
<td>388M</td>
<td>-</td>
<td>74.8</td>
<td>91K</td>
</tr>
<tr>
<td>FBNet-C [41]</td>
<td>375M</td>
<td>22.1ms</td>
<td>74.9</td>
<td>216</td>
</tr>
<tr>
<td>Proxyless(GPU) [3]</td>
<td>465M</td>
<td>22.1ms</td>
<td>75.1</td>
<td>200</td>
</tr>
<tr>
<td>DenseNAS-C</td>
<td><strong>361M</strong></td>
<td><strong>17.9ms</strong></td>
<td>75.3</td>
<td>64</td>
</tr>
<tr>
<td>Random Search</td>
<td>360M</td>
<td>26.9ms</td>
<td>74.3</td>
<td>64</td>
</tr>
</tbody>
</table>

4.1. Performance on MobileNetV2-based Search Space

We implement DenseNAS on the MobileNetV2 [37]-based search space, set the GPU latency as our secondary optimization objective, and search models with different sizes under multiple latency optimization magnitudes (defined in Eq. 8). The ImageNet results are shown in Tab. 1. We divide Tab. 1 into several parts and compare DenseNAS models with both manually designed models [21, 37] and NAS models. DenseNAS achieves higher accuracies with both fewer FLOPs and lower latencies. Note that for FBNet-A, the group convolution in the 1×1 conv and the channel shuffle operation are used, which do not exist in FBNet-B, -C and Proxyless. In the compared NAS methods [39, 3, 41], the block counts and block widths in the search space are set and adjusted manually. DenseNAS allocates block counts and block widths automatically. We further visualize the results in Fig. 3, which clearly demonstrates that DenseNAS achieves a better trade-off between accuracy and latency. The searched architectures are shown in Fig. 4.

4.2. Generalization Ability on COCO Object Detection

We apply the searched DenseNAS networks on the COCO [25] object detection task to evaluate the generalization ability of DenseNAS networks and show the results.
Table 2: Object detection results on COCO. The FLOPs are calculated with 1088 × 800 input.

<table>
<thead>
<tr>
<th>Method</th>
<th>Params</th>
<th>FLOPs</th>
<th>mAP(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNetV2 [37]</td>
<td>11.49M</td>
<td>133.05B</td>
<td>32.8</td>
</tr>
<tr>
<td>DenseNAS-B</td>
<td>12.69M</td>
<td>133.09B</td>
<td>34.3</td>
</tr>
<tr>
<td>FBNNet-C [41]</td>
<td>12.65M</td>
<td>134.17B</td>
<td>34.9</td>
</tr>
<tr>
<td>Proxyless(GPU) [3]</td>
<td>14.62M</td>
<td>135.81B</td>
<td>35.1</td>
</tr>
<tr>
<td>DenseNAS-C</td>
<td>13.24M</td>
<td>133.91B</td>
<td>35.1</td>
</tr>
<tr>
<td>MobileNetV2 [37]</td>
<td>4.33M</td>
<td>0.8B</td>
<td>22.1</td>
</tr>
<tr>
<td>Mnasnet-92 [59]</td>
<td>5.3M</td>
<td>1.0B</td>
<td>22.9</td>
</tr>
<tr>
<td>FBNNet-C [41]</td>
<td>6.27M</td>
<td>1.06B</td>
<td>22.9</td>
</tr>
<tr>
<td>Proxyless(GPU) [3]</td>
<td>7.91M</td>
<td>1.24B</td>
<td>22.8</td>
</tr>
<tr>
<td>DenseNAS-C</td>
<td>6.87M</td>
<td>1.05B</td>
<td>23.1</td>
</tr>
</tbody>
</table>

Table 3: ImageNet classification results of ResNets and DenseNAS networks searched on the ResNet-based search spaces.

<table>
<thead>
<tr>
<th>Model</th>
<th>Params</th>
<th>FLOPs</th>
<th>GPU Latency</th>
<th>Top-1 Acc(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18 [18]</td>
<td>11.7M</td>
<td>1.8IB</td>
<td>13.5ms</td>
<td>72.0</td>
</tr>
<tr>
<td>DenseNAS-R1</td>
<td>11.1M</td>
<td>1.61B</td>
<td>12.0ms</td>
<td>73.5</td>
</tr>
<tr>
<td>ResNet-34 [19]</td>
<td>21.8M</td>
<td>3.66B</td>
<td>24.6ms</td>
<td>75.3</td>
</tr>
<tr>
<td>DenseNAS-R2</td>
<td>19.5M</td>
<td>3.06B</td>
<td>22.2ms</td>
<td>75.8</td>
</tr>
<tr>
<td>ResNet-50-B</td>
<td>25.6M</td>
<td>4.09B</td>
<td>47.8ms</td>
<td>77.7</td>
</tr>
<tr>
<td>RandWire-WS, C=109 [12]</td>
<td>31.9M</td>
<td>4.0B</td>
<td>-</td>
<td>79.0</td>
</tr>
<tr>
<td>DenseNAS-R3</td>
<td>24.7M</td>
<td>3.41B</td>
<td>41.7ms</td>
<td>78.0</td>
</tr>
</tbody>
</table>

4.3. Performance on ResNet-based Search Space

We apply our DenseNAS framework on the ResNet [18]-based search space to further evaluate the generalization ability of our method. It is convenient to implement DenseNAS on ResNet [18] as we set the candidate operations in the basic layer as the basic block defined in ResNet [18] and the skip connection. The ResNet-based search space is also constructed as a densely connected super network.

We search for several architectures with different FLOPs and compare them with the original ResNet models on the ImageNet [9] classification task in Tab. 3. We further replace all the basic blocks in DenseNAS-R2 with the bottleneck blocks and obtain DenseNAS-R3 to compare with ResNet-50-B and the NAS model RandWire-WS, C=109 [42] (WS, C=109). Though WS, C=109 achieves a higher accuracy, the FLOPs increases 600M, which is a great number, 17.6% of DenseNAS-R3. Besides, WS C=109 uses separable convolutions which greatly decrease the FLOPs while DenseNAS-R3 only contains plain convolutions. Moreover, RandWire networks are unfriendly to inference on existing hardware for the complicated connection patterns. Our proposed DenseNAS promotes the accuracy of ResNet-18, -34 and -50-B by 1.5%, 0.5% and 0.3% with 200M, 600M, 680M fewer FLOPs and 1.5ms, 2.4ms, 6.1ms lower latency respectively. We visualize the comparison results in Fig. 5 and the performance on the ResNet-

Figure 5: Graphical comparisons between ResNets and DenseNAS networks on the ResNet-based search space.

based search space further demonstrates the great generalization ability and effectiveness of DenseNAS.

4.4. Ablation Study and Analysis

Comparison with Other Search Spaces To further demonstrate the effectiveness of our proposed densely connected search space, we conduct the same search algorithm used in DenseNAS on the search spaces of FBNNet and ProxylessNAS as well as a new search space which is constructed following the settings of block counts and block widths in MobileNetV2. The three search spaces are denoted as FBNNet-SS, Proxyless-SS and MBV2-SS respectively. All the search training settings and hyper-parameters are the same as that we use for DenseNAS. The results are shown in Tab. 4 and DenseNAS achieves the highest accuracy with the lowest latency.

Table 4: Comparisons with other search spaces on ImageNet. SS: Search Space. MBV2: MobileNetV2.

<table>
<thead>
<tr>
<th>Search Space</th>
<th>FLOPs</th>
<th>GPU Latency</th>
<th>Top-1 Acc(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FBNNet-SS [41]</td>
<td>369M</td>
<td>25.6ms</td>
<td>74.9</td>
</tr>
<tr>
<td>Proxyless-SS [3]</td>
<td>398M</td>
<td>18.9ms</td>
<td>74.8</td>
</tr>
<tr>
<td>MBV2-SS [37]</td>
<td>383M</td>
<td>32.1ms</td>
<td>74.7</td>
</tr>
<tr>
<td>DenseNAS</td>
<td>361M</td>
<td>17.9ms</td>
<td>75.3</td>
</tr>
</tbody>
</table>

Comparison with Random Search As random search [24, 38] is treated as an important baseline to validate NAS methods. We conduct random search experiments and show the results in Tab. 1. We randomly sample 15 models in our search space whose FLOPs are similar to DenseNAS-C. Then we train every model for 5 epochs on ImageNet. Finally, we select the one with the highest validation accuracy and train it under the same settings as DenseNAS. The total search cost of the random search is the same as DenseNAS. We observe that DenseNAS-C is 1% accuracy higher compared with the randomly searched model, which proves the effectiveness of DenseNAS.
Table 5: Comparison with different connection number settings on ImageNet.

<table>
<thead>
<tr>
<th>Connect Num</th>
<th>GPU Latency</th>
<th>Top-1 Acc(%)</th>
<th>Search Epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>18.9ms</td>
<td>74.8</td>
<td>150</td>
</tr>
<tr>
<td>4</td>
<td>17.9ms</td>
<td>75.3</td>
<td>150</td>
</tr>
<tr>
<td>5</td>
<td>19.2ms</td>
<td>74.6</td>
<td>150</td>
</tr>
<tr>
<td>5</td>
<td>16.7ms</td>
<td>74.9</td>
<td>200</td>
</tr>
</tbody>
</table>

**The Number of Block Connections** We explore the effect of the maximum number of connections between routing blocks in the search space. We set the maximum connection number as 4 in DenseNAS. Then we try more options and show the results in Tab. 5. When we set the connection number to 3, the searched model gets worse performance. We attribute this to the search space shrinkage which causes the loss of many possible architectures with good performance. As we set the number to 5 and the search process takes the same number of epochs as DenseNAS, i.e., 150 epochs. The performance of the searched model is not good, even worse than that of the connection number 3. Then we increase the search epochs to 200 and the search process achieves a comparable result with DenseNAS. This phenomenon indicates that larger search spaces need more search cost to achieve comparable/better results with/than smaller search spaces with some added constraints.

**Cost Estimation Method** As the super network is densely connected and the final architecture is derived based on the total transition probability, the model estimation needs to take the effects of all the path probabilities on the whole network into consideration. We try a **local cost estimation** strategy that does not involve the global connection effects on the whole super network. Specifically, we compute the cost of the whole network by summing the cost of every routing block during the search as follows, while the transition probability \( p_{ji} \) is only used for computing the cost of each individual block rather than the whole network.

\[
\text{cost} = \sum_{i} \left( \sum_{j=i-m}^{i+1} p_{ji} \cdot \text{cost}_{\text{align}} + \text{cost}_{i} \right),
\]

where all definitions in the equation are the same as that in Eq. 6. We randomly generate the architecture parameters (\( \alpha \) and \( \beta \)) to derive the architectures. Then we draw the approximated cost values computed by **local cost estimation** and our proposed **chained cost estimation** respectively, and compare with the real cost values in Fig. 6. In this experiment, we take FLOPs as the model cost because the FLOPs is easier to measure than latency. 1,500 models are sampled in total. The results show that the predicted cost values computed by our **chained cost estimation** algorithm has a much stronger correlation with the real values and approximate more to the real ones. As the predicted values are computed based on the randomly generated architecture parameters which are not binary parameters, there are still differences between the predicted and real values.

**Architecture Analysis** We visualize the searched architectures in Fig. 4. It shows that DenseNAS-B and -C have one more block in the last stage than other architectures, which indicates enlarging the depth in the last stage of the network tends to obtain a better accuracy. Moreover, the smallest architecture DenseNAS-A whose FLOPs is only 251M has one fewer block than DenseNAS-B and -C to decrease the model cost. The structures of the final searched architectures show the great flexibility of DenseNAS.

5. Conclusion

We propose a densely connected search space for more flexible architecture search, DenseNAS. We tackle the limitations in previous search space design in terms of the block counts and widths. The newly designed routing blocks are utilized to construct the search space. The proposed chained cost estimation algorithm aims at optimizing both accuracy and model cost. The effectiveness of DenseNAS is demonstrated on both MobileNetV2- and ResNet- based search spaces. We leave more applications, e.g., semantic segmentation, face detection, pose estimation, and more network-based search space implementations, e.g., MobileNetV3 [20], ShuffleNet [45] and VarGNet [44], for future work.

Acknowledgement

This work was supported by National Key R&D Program of China (No. 2018YFB1402600), National Natural Science Foundation of China (NSFC) (No. 61876212, No. 61733007 and No. 61572207), and HUST-Horizon Computer Vision Research Center. We thank Liangchen Song, Kangjian Peng and Yingqing Rao for the discussion and assistance.
References


[28] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Huang, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In ECCV, 2018. 2


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


