A. Index

The supplementary materials are organized as follows.

- We present the details of the searched FP-NAS models and visualize them in Section B.
- Details of our training recipe, and comparisons with those used by other methods are presented in Section C.
- Details of FP-NAS search spaces are presented in Section D.
- A study on the supernet warmup is presented in Section E.
- The derivation of gradients for updating architecture parameters is shown in Section F.

B. Understanding FP-NAS Architectures

We use the proposed FP-NAS method to search for a family of models in different sizes. The complete results are shown in Table B.1.

We also visualize two representative FP-NAS models, including FP-NAS-S1++ and FP-NAS-L2 model, in Fig B.1. Compared with hand-crafted models, the searched architectures select more non-uniform choices along kernel size, non-linearity, feature channel and number of splits over MB-Conv blocks.

For example, small kernel size 3 is more favored in the early blocks while large kernel size 5 is more often chosen in the later blocks. This is likely because large kernel size is more computationally expensive and we can only afford to use it in the later blocks where the spatial size of feature map is small (e.g. \(14^2, 7^2\)). Also in the later blocks, convolution with large kernel size can more effectively capture the global context.

We also find large choices of the number of splits in SA block, such as 2 and 4, are more often used in the later blocks. This is likely because high-level semantic features only emerge in the later blocks, and attention with multiple splits is more needed to attend to certain semantic features relevant to the image content, compared with low-level features in the early blocks where attention is less useful.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Input size</th>
<th>FLOPS(M)</th>
<th>Params(M)</th>
<th>Distill</th>
<th>Top-1 acc (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1++</td>
<td>128(^2)</td>
<td>66</td>
<td>5.9</td>
<td>×</td>
<td>72.0</td>
</tr>
<tr>
<td>S2++</td>
<td>160(^2)</td>
<td>98</td>
<td>5.8</td>
<td>×</td>
<td>72.2</td>
</tr>
<tr>
<td>S3++</td>
<td>192(^2)</td>
<td>147</td>
<td>5.8</td>
<td>×</td>
<td>74.2</td>
</tr>
<tr>
<td>S4++</td>
<td>224(^2)</td>
<td>268</td>
<td>6.4</td>
<td>×</td>
<td>76.6</td>
</tr>
<tr>
<td>L0</td>
<td>224(^2)</td>
<td>399</td>
<td>11.3</td>
<td>×</td>
<td>78.0</td>
</tr>
<tr>
<td>L1</td>
<td>240(^2)</td>
<td>728</td>
<td>15.8</td>
<td>✓</td>
<td>80.9</td>
</tr>
<tr>
<td>L2</td>
<td>256(^2)</td>
<td>997</td>
<td>20.7</td>
<td>×</td>
<td>81.6</td>
</tr>
</tbody>
</table>

Table B.1: The family of FP-NAS models. All results are obtained using one model and a single crop on ImageNet-1K. Column Distill denotes whether model distillation [2] is used to train the model.

Table C.2: ImageNet top-1 accuracy (%) of FP-NAS models trained with different training recipes. We start with using LS only, and sequentially add AA, EMA and SD.

C. Training Recipes

When training FP-NAS models, we adopt label smoothing [4] (LS), Auto-Augment [1] (AA), and Exponential Model Averaging (EMA). We study the impact of the training recipe on the testing accuracy by training FP-NAS models under different training recipes. The results are shown in Table C.2.

For EMA, it does not improve our smallest S1++ model,
For training large FP-NAS-L models, we use LS, AA, EMA, and SD, which are also used by EfficientNet.

For SD, it actually reduces the accuracy of our small models such as S1++ and S4++. Therefore, we do not use SD to train our small models. On the other side, on our large models, such as L0 and L2, SD slightly improves the accuracy by 0.1%.

We also compare our training recipe with that from other methods, and the results are shown in Table C.3. For small models under consideration, including MnasNet [5] and FBNetV2 [7], they are different in whether AA and EMA are used. Although FP-NAS-S++ models are trained with both AA and EMA, the improvement by using either one is much less significant compared with the improvement of FP-NAS-S++ models over other models (See Table 8 in the paper). For training large FP-NAS-L models, we use LS, AA, EMA and SD, which are also used by EfficientNet.

Table C.3: Comparing training recipe used by different methods. We use the following short-hand notations. LS: label smoothing [4], AA: auto-augmentation [1], EMA: exponential model averaging [6], and SD: stochastic depth [3].

but can improve S4++ model by 0.2%. EMA is more effective on large models, including L0 and L2 models, and improve their accuracy by 0.3% and 1.4%, respectively.

For SD, it actually reduces the accuracy of our small models such as S1++ and S4++. Therefore, we do not use SD to train our small models. On the other side, on our large models, such as L0 and L2, SD slightly improves the accuracy by 0.1%.

D. FP-NAS Search Spaces

In the Table 3, we introduce three FP-NAS search spaces from L0 to L2. They share the same FP-NAS micro-architecture, but have different macro-architectures, which are shown in Table D.4, D.5, and D.6, respectively.

E. SuperNet Warmup

In our experiments, we fix the architecture hyperparameters while only updating the model weights at the beginning of search for a number of epochs. This is to warm-up the supernet by uniformly sampling architectures from the initial distribution, and update model weights associated with them. Compared with randomly initialized model weights, the updated weights lead to a better estimation of the data likelihood $P(y|X, \omega, A_k)$ of the sampled architectures $\{A_k\}$ and therefore a better estimation of the gradients for updating architecture parameters.
Accuracy (59.67.8) when the architecture parameters are freezed in the beginning. Comparing searched architectures when supernet warmup is used or not.

The macro-architecture of FP-NAS-L1 search space is defined as follows. First, the cost-aware loss function derivation is shown below. First, the cost-aware loss function is defined as follows.

\[ \mathcal{L}(\omega, \alpha) = -\log P(y|X, \omega, \alpha) + \beta \log C(\alpha) \]  

(1)

Then, the gradient w.r.t architecture parameters can be derived as follows.

\[ \nabla_{\alpha} \mathcal{L}(\omega, \alpha) = \frac{1}{P(y|X, \omega, \alpha)} \int P(y|X, \omega, \alpha) \nabla_{\alpha} \log P(y|X, \omega, A) dA \]
\[ + \beta \frac{1}{C(\alpha)} \int C(A) \nabla_{\alpha} P(A|\alpha) dA \]
\[ = \frac{1}{K} \sum_{k=1}^{K} P(y|X, \omega, A_k) - \frac{C(A_k)}{C(\alpha)} \nabla_{\alpha} \log P(A_k|\alpha) \]
\[ \approx \sum_{k} m^k_{\alpha} \nabla_{\alpha} - \log P(A_k|\alpha) \]

(2)

Table D.5: The macro-architecture of FP-NAS-L1 search space. It is used to search for FP-NAS-L1 architecture.

Table D.6: The macro-architecture of FP-NAS-L2 search space. It is used to search for FP-NAS-L2 architecture.

Table E.7: Comparing searched architectures when supernet warmup is used or not.

In a study where architecture is searched in FBNetV2-F space, we use 315 epochs and compare the searched model when the architecture parameters are frozen in the beginning 45 epochs or not. The results are shown in Table E.7. The architecture searched without supernet warmup has inferior accuracy, and thus we always warmup supernet to search FP-NAS models.

F. Derivation of Gradients for Updating Architecture Parameters

In Eqn (7) of the paper, we presented how to compute the gradient w.r.t architecture hyper-parameters \( \alpha \). The full derivation is shown below. First, the cost-aware loss function is defined as follows.