1. Pruning residual connection in the standard form

In our ensemble form, we cut out short paths to achieve better performance. Here, we explore whether we can obtain the same effect by pruning residual connection in the standard form. We experiment with the DeiT-S deleting the residual connection in the shallow layers and the results are shown in Table 1. We can see that cutting out residual connection affects the performance and convergence, which demonstrates that the success of our path pruning is not from cutting out residual connection and cannot be achieved in the standard form.

2. Our ensemble form of hierarchical ViTs

We visualize our ensemble form of hierarchical ViTs in Figure 1. The LayerNorm expression in our model is \( E[x]/\sqrt{Var[x]} \cdot \gamma + \beta \). In Figure 1, we observe that the same downsampling layer \( D_n \) in different paths compute individual standard deviations, namely asynchronous standard deviation, causing different forward propagation result with standard form. Neglecting the influence of bias, to achieve consistent forward propagation, we need to synchronize standard deviations in different paths, namely synchronous standard deviation. For example, the input of \( D_1 \) in \( p_0, p_1, p_2 \), and \( p_3 \) are different, leading to different standard deviations. The input of \( D_1 \) in \( p_3 \) is the same as the standard form. Therefore, if we want to achieve the same forward propagation, we can synchronize all the standard deviations of \( D_1 \) with the standard deviation in \( p_3 \). However, we find that using either asynchronous or synchronous standard deviation yields similar performance when we train them from scratch.

3. Self-distillation in the standard form

We apply our self-distillation method in the standard form to make low-level feature maps mimic high-level feature maps in Table 2 and find out that it is difficult to work. The models suffer from an accuracy drop or divergence. We try to explain this issue from an ensemble perspective.

Assuming that we select \( x_t \) and \( x_s \) (\( t > s \)) which are the output of any intermediate transformers \( T_t \) and \( T_s \) as the teacher and the student, respectively. There are \( t - s \) transformers between \( x_t \) and \( x_s \). According to the Eq. 5, we can find a function \( \mathcal{F} \) and denote the \( x_t \) as

\[
x_t = x_s + \mathcal{F}(x_s),
\]

where a student component is in the teacher feature map. Then we force the \( x_s \) to mimic the \( x_s + \mathcal{F}(x_s) \), i.e., \( x_t \). The model may be optimized to an unexpected direction by the KD loss, such as enlarging the weight of \( x_s \) in \( x_t \) and decreasing the \( \mathcal{F}(x_s) \) to 0. When we use \( l_2 \) loss as the KD loss, the effect is most obvious where the model diverges directly. Therefore, we speculate that the inherent ensemble property of ViTs limits the application of self-distillation in the standard form. In contrast, our ensemble view avoids this issue. Our ensemble form decouples the linear combination and the paths do not contain the linear components of previous paths.

4. Path combination for 2N+1 paths

In Eq. 5, we combine the MHSA and FFN paths into an \( f \) path and obtain \( N + 1 \) paths in a ViT, where \( N \) is the number of transformer layers. If we do not combine them, we will get \( 2N + 1 \) paths. We conduct experiments to explore the path combination for \( 2N + 1 \) paths. According to the previous works [2, 29, 32, 39], self-attention and FFN
Table 1: Pruning the residual connection in the shallow transformer layers.

<table>
<thead>
<tr>
<th>Model</th>
<th>No. of layers w/o shortcut</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeiT-S</td>
<td>0 (Baseline)</td>
<td>79.8</td>
</tr>
<tr>
<td>DeiT-S</td>
<td>1</td>
<td>77.7</td>
</tr>
<tr>
<td>DeiT-S</td>
<td>2</td>
<td>Loss NAN</td>
</tr>
</tbody>
</table>

Table 2: Applying our self-distillation method to distill feature maps in the standard ViT form.

<table>
<thead>
<tr>
<th>Model</th>
<th>KD Loss</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeiT-S</td>
<td>-</td>
<td>79.8 (Baseline)</td>
</tr>
<tr>
<td>DeiT-S</td>
<td>$l_2$ Loss</td>
<td>Loss NAN</td>
</tr>
<tr>
<td>DeiT-S</td>
<td>KL Loss</td>
<td>79.6</td>
</tr>
</tbody>
</table>

Table 3: Applying path pruning and EnsembleScale to DeiT-S with $2N + 1$ paths. ES is short for Ensemble. Note that $x_0$ path is not contained in any experiments.

<table>
<thead>
<tr>
<th>SA path</th>
<th>FFN path</th>
<th>ES</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_8 - p_{12}$</td>
<td>$p_1 - p_{12}$</td>
<td></td>
<td>80.0</td>
</tr>
<tr>
<td>$p_8 - p_{12}$</td>
<td>$p_3 - p_{12}$</td>
<td></td>
<td>80.1</td>
</tr>
<tr>
<td>$p_1 - p_{12}$</td>
<td>$p_1 - p_{12}$</td>
<td>✓</td>
<td>80.3</td>
</tr>
</tbody>
</table>

5. The demo code of our ensemble form

The demo code of our ensemble form is summarized in Algorithm 1. We only require a few modifications in the code of the standard form, demonstrating our ensemble form is implementation- and deployment-friendly.

```
Algorithm 1 Demo code of our ensemble form (PyTorch-like)

# N: the number of transformer layers
# self_attention: the function of self attention
# ffn: the function of FFN
# patch_embedding: the function of patch embedding

class Block:
    def forward(input):
        sa_path = self_attention(norm(input))
        ffn_path = ffn(norm(input + sa_path))
        return input + sa_path + ffn_path

class ViT:
    def init(self):
        blocks = [Block() for i in range(N)]

    def forward(input):
        x = patch_embedding(input)
        paths = [x]
        for i in range(N):
            x, f = blocks[i](x)
            paths.append(f)
        return sum(paths)
```