Supplemental Material for Joint Token Pruning and Squeezing Towards More Aggressive Compression of Vision Transformers

Siyuan Wei1* Tianzhu Ye2* Shen Zhang1 Yao Tang1 Jiajun Liang1†
1MEGVII Technology 2Tsinghua University
{weisiyuan, zhangshen, tangyao02, liangjiajun}@megvii.com, ytz20@mails.tsinghua.edu.cn

1. Overview
In the supplemental materials, we show the following details of our joint Token Pruning & Squeezing (TPS):
• Visualizations.
• Details of two variants.
• TPS on hybrid ViTs.
• Detailed experiment settings.
• TPS on larger models and with larger input size.
• TPS under different keep ratios.
• More ablations about TPS design.

2. Visualizations
We demonstrate the additional cases from ImageNet1K [1], which our TPS-DeiT and DeiT predict correctly but dynamicViT-DeiT predict wrongly. As Fig 2 shows, we found that the imperfect pruning policy brings the loss of background context and incomplete subject, which puzzles the model and leads to a close but incorrect prediction. However, our TPS conquers these cases by squeezing the information of pruned tokens into similar reserved tokens.

3. Details of Two Variants
We design two variants of TPS: dTPS and eTPS, to show our flexibility and compare fairly with dynamicViT [7] and EViT [5]. Theoretically, our TPS can be incorporated with any token pruning method. In this paper, we choose dynamicViT and EViT as baselines for their strong performance and concise forms. The major disparities between the two variants are as follows:

Forward procedure. The TPS module drops the tokens from inputs practically except for the training stage of dTPS. In each pruning stage, the dTPS module employs the gumbel-softmax [3] to sample binary decision mask randomly during training and maintain the presently reserved mask and pruned mask to avoid previously pruned tokens from participating in the matching and fusing step. In the subsequent attention layer, the attention masking strategy from dynamicViT [7] is employed to erase the effects of dropped tokens. The implementation details can be found in the code file.

Position to insert. As mentioned in the paper, dTPS and eTPS cut down tokens in inter-block and intra-block ways, respectively. The dTPS module is inserted before the transformer block, while the eTPS module is inserted after the multi-head attention layer. The distinction derives from the different token scoring methods. The learnable score prediction employed by dynamicViT and dTPS does not rely on any internal operation of transformer blocks, while the scoring based on the class-token attentions requires the results from the multi-head attention block.

Parameters. The eTPS module is parameter-free, while the dTPS module increases the total number of parameters by a small amount due to its learnable token score prediction head.

Performances. The performances of dTPS and eTPS modules are close but can be slightly different when training epochs changes. According to our experiments, the eTPS module outperforms the dTPS module under 30 epochs. The opposite results were observed under 100 epochs. The difference demonstrates that extra parameters of dTPS modules endow the model with a higher upper limit.

4. TPS on Hybrid ViTs
We conduct experiments on PVT [10] and CvT [11] to prove our design is compatible with hybrid ViTs.

4.1. PVT
Generally, We insert dTPS modules between the patch embedding layer and the subsequent basic block of each pruned stage. Unlike TPS on vanilla ViTs, we reserve atten-
Figure 1. Note that dynamicViT can't converge in the two most aggressive pruning settings in (b).

Figure 2. Comparisons between dynamicViT [7] and our joint Token Pruning & Squeezing. The results are given by dynamicViT-DeiT-small and our TPS-DeiT-small with the same pruning setup (pruning locations are \{4\textsuperscript{th}, 7\textsuperscript{th}, 8\textsuperscript{th}\}) and token keeping ratio of each pruning stage \(\rho\) is 0.5. The cases of dynamicViT and our TPS are displayed based on practical operations of the first stage with pruning applied. For dynamicViT, the blank tokens denote pruned ones; for our TPS, we mask each group of matched tokens in the TPS as the same color for visualization clarity.
respectively.

the complete spatial structure during training and inference, and utilize the masking or padding operations to maintain
tive tokens from the whole tokens set in each pruned stage and utilize the masking or padding operations to maintain
the complete spatial structure during training and inference, respectively.

Training. The spatial reduction layer of the basic block in PVT requires input with a complete spatial structure. For
the basic block with a spatial reduction block, given the policy $M$, we maintain the complete spatial structure and mask
the dropped tokens in key $K$ and in value $V$ with zeros before the spatial reduction layer as follows:

$$SR^*(Q, K, V) = MHA(Q, SR(K \odot M), SR(V \odot M)) \quad (1)$$

Here, $SR^*$ is the modified spatial reduction attention, $MHA$ is the multi-head attention operation, and $SR$ is
the spatial reduction layer. Moreover, we perform the same masking operation on dropped tokens before the patch em-
bedding layer of next stage.

For the basic block without a spatial reduction layer, no masking operation is needed, and we conduct the attention
masking strategy from dynamicViT [7] to erase the effects of the dropped tokens.

Inference. The inference procedure of dTPS is adjusted slightly to practically accelerate the spatial reduction layer. The input tokens are pruned by a top-k selection operation based on the scoring results. For the block with a spatial
reduction layer, we pad the previously dropped tokens with zero in the $SR^*$ to maintain the complete spatial structure.

$$K' = Pad(K, M) \quad (2)$$

$$V' = Pad(V, M) \quad (3)$$

$$SR^*(Q, K, V) = MHA(Q, SR(M'), SR(V')) \quad (4)$$

Also, the same padding operation is utilized before the patch embedding layer of the next stage. For the block without
a spatial reduction layer, no padding operation is needed either. The requirement of complete spatial structure leads
to less shrinkage of computations.

4.2. CvT

The last stage of CvT [11] contains most of its blocks; therefore we only modify the last stage with our dTPS. The
operations remain the same for other stages as the original CvT [11].

Training. The convolutional projection operation in CvT requires the input with a complete spatial structure.
Given the policy $M$, we mask the dropped tokens with zeros before the convolution projection:

$$Q, K, V = ConvolutionalProjection(X \odot M) \quad (5)$$

Inference. The input tokens are pruned by a top-k selection operation based on the scoring results. To maintain
the complete spatial structure, we pad the previously dropped tokens with zeros in the convolutional projection layer.

$$X' = Pad(X, M) \quad (6)$$

$$Q, K, V = ConvolutionalProjection(X') \quad (7)$$

convolutional projection in stage 3. It only performs token pruning in stage 3 to avoid the extra operation to main-
tain the structured spatial input. Compared to ATS [2], our method utilizes masking and padding during training and
inference to keep the spatial structure.

5. Detailed Experiment Settings

5.1. ImageNet-1K Classification

All experiments follow the same data augmentations used in DeiT $^1$ [8]. All the model is initialized with pre-
trained models’ weights and fine-tuned with different token pruning location and token keeping ratio. We adopt the AdamW [6] as the optimizer and a cosine learning rate scheduler.

TPS on DeiT $^2$. The experiment settings of dTPS-DeiT follows dynamicViT $^2$ except for basic learning rate

<table>
<thead>
<tr>
<th>Matching Method</th>
<th>Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:1</td>
<td>71.90</td>
</tr>
<tr>
<td>1:1</td>
<td>69.02</td>
</tr>
</tbody>
</table>

(a) Different matching methods on dTPS-DeiT-T. The keep ratio is 0.7.

<table>
<thead>
<tr>
<th>Fusing Method</th>
<th>Policy</th>
<th>Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weighting</td>
<td>Original</td>
<td>70.58</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>65.56 (-5.02)</td>
</tr>
<tr>
<td>Average</td>
<td>Original</td>
<td>70.47</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>65.173 (-5.30)</td>
</tr>
</tbody>
</table>

(b) Robustness comparison of fusing methods on dTPS-DeiT-T. The keep ratio is 0.5.

Table 1. The pruned layers includes $4^{th}$, $7^{th}$, $8^{th}$. (a) N:1 matching employed by TPS finds the nearest reserved token for each pruned token to inject multiple tokens into the same token, while 1:1 matching finds the nearest pruned token for each reserved token. (b) The similarity-weighting fusing obtains a lower accuracy drop under random token squeezing (see more details in the main paper: Sec. 4.3 ) than average fusing.

---

1. [https://github.com/facebookresearch/DeiT](https://github.com/facebookresearch/DeiT)
2. [https://github.com/raoyongming/DynamicViT](https://github.com/raoyongming/DynamicViT)
is set to $\frac{\text{batchsize}}{1024} \times 2.5 \times 10^{-4}$ and no stage of fixing backbone weights. The experiment settings of eTPS-DeiT follow EViT\(^3\). The pruning settings include combinations of three multi-layer pruning settings: $\text{prune_locs} \in \{[4, 7, 10], [3, 5, 7, 9], [4, 6, 8, 10]\}$, and two token keeping ratios $\rho \in \{0.5, 0.7\}$. The token keeping ratio remains the same in all pruning stages.

**TPS on LV-ViT [4].** The experiments of dTPS and eTPS on LV-ViT [4] follow the same training settings of dTPS and eTPS on DeiT, except for the basic learning rate is set to $\frac{\text{batchsize}}{1024} \times 1.0 \times 10^{-4}$ for the stable convergence. For LV-ViT-T, the pruning settings include combinations of three multi-layer pruning settings: $\text{prune_locs} \in \{[4, 7, 10], [3, 5, 7, 9], [4, 6, 8, 10]\}$, and two token keeping ratios $\rho \in \{0.5, 0.7\}$. For LV-ViT-S, the pruning settings include combinations of three multi-layer pruning settings: $\text{prune_locs} \in \{[5, 9, 13], [3, 6, 9, 12], [4, 7, 10, 13]\}$, and two token keeping ratios $\rho \in \{0.5, 0.7\}$.

**TPS on PS-ViT [12].** The experiments of dTPS on PS-ViT [12] follow the same training settings as ATS [2] on PS-ViT [12]. The basic learning rate is set to $\frac{\text{batchsize}}{768} \times 5.0 \times 10^{-4}$, $\text{prune_locs}$ is set to $\{3, 6, 9\}$ and token keeping ratio $\rho$ is 0.5.

**TPS on PVT [10].** The pruning stages include stage 2, stage 3, and stage 4. The token keeping ratio for all dTPS modules is set to 0.7. Basic learning rate is set to $\frac{\text{batchsize}}{1024} \times 2.5 \times 10^{-4}$.

**TPS on CvT [11].** The basic learning rate is set to $\frac{\text{batchsize}}{1024} \times 5.0 \times 10^{-5}$. The dTPS modules are only inserted into stage 3, and the pruning locations include $\{3, 6, 9\}$ for CvT-13,$\{5, 9, 13\}$ for CvT-21. The token keeping ratio for all TPS modules is set to 0.5.

### 5.2. iNaturalist 2019 Classification

**TPS on DeiT [8].** For the experiment on iNaturalist 2019 Classification [9], we re-train DeiT and fine-tune the model with dynamicViT or dTPS applied.

In the training step, we initialize DeiT [8] with weights of ImageNet1K pre-trained model and re-train them for 300 epochs. The basic learning rate is set to $\frac{\text{batchsize}}{1024} \times 10^{-3}$. The other settings follow DeiT [8].

In the fine-tuning step, we initialize dynamicViT-DeiT and dTPS-DeiT with weights from the last step and fine-tune them for 30 epochs with the same pruning setup. The token pruning location is set to $[4, 7, 10]$, and the token keeping ratio is 0.5. We follow the same fine-tuning settings as the experiments on ImageNet1K, except for no distillation loss.

![Figure 3. TPS under different keep ratios on DeiT-S.](image)

**6. TPS on Larger Models and with Larger Input Size**

We conduct experiments of TPS on DeiT-B as shown in Fig. 1a to demonstrate it is compatible with large models. We also prove TPS can perform well with larger input size such as $384 \times 384$, as shown in Fig. 1b.

**7. TPS under Different Keep Ratios**

Experiments of TPS under different keep ratios are shown in Fig. 3.

**8. More Ablations About TPS Design**

More matching & fusing methods are shown in Tab. 1 as ablations about our TPS design. Tab. 1a indicates that TPS performance improvement benefits from compressing pruned tokens’ information while unmatched reserved tokens remain unchanged. Token scoring can be proved necessary for squeezing under random token division meets a significant drop as shown in Tab. 1b and robustness analysis in the main paper: Sec. 4.3.

---

\(^3\) https://github.com/youweiliang/evit


References


