Spectral State Space Model for Rotation-Invariant Visual Representation Learning

Supplementary Material

A. Implementation Details

In this section, we give the pseudo-code for the Rotational Feature Normalizer (RFN) and Spectral Traversal Scan (STS) modules. This pseudo-code provides a concise summary of the key steps involved in our approach, offering a high-level abstraction of the implementation. It is designed to complement the detailed explanations in the main paper and can serve as a reference for reproducing our results.

We begin with Algorithm 1, which outlines the steps for implementing the RFN module. As discussed in the paper, this module ensures a consistent representation of image features across different orientations. This module comprises four key steps: rotation, patchification, back-rotation, and a max operation to aggregate features across all orientations.

Algorithm 1 Rotational Feature Normalizer Module

Require: Input image $\mathcal{I} \in \mathbb{R}^{H\times W\times 3}$; rotation angles: $\{\theta_r \mid r = 1, \ldots, R\}$; stem module: Stem; patch size: p
Ensure: Aggregated feature map $\mathcal{F}$.
Initialize: $\{\theta_r\}$ and $\{-\theta_r\}$.
for $r \in \{1, \ldots, R\}$ **do**
  $\mathcal{I}_r \leftarrow \mathcal{R}_{\theta_r}(\mathcal{I})$ ▷ Rotate image by θ_r
  $\mathcal{F}_r \leftarrow \text{Stem}(\mathcal{I}_r)$ ▷ Extract features
  $\mathcal{F}_r^{\text{unrotated}} \leftarrow \mathcal{R}_{-\theta_r}(\mathcal{F}_r)$ ▷ Back-rotate feature map
  Append $\mathcal{F}_r^{\text{unrotated}}$ to $\mathcal{F}_r$.
end for
$\mathcal{F} \leftarrow \max_{r\in\{1,\ldots,R\}} \mathcal{F}_r$ ▷ Max operation
Output: Aggregated feature map $\mathcal{F}$.

Algorithm 2 represents the STS strategy for determining the traversal order of image patches based on spectral decomposition. The process begins by constructing the adjacency matrix $\mathbf{W}$ using the k-Nearest Neighbors (KNN) algorithm, based on the Euclidean distances between image patches. This matrix captures the relationships and similarities between the patches. Subsequently, the degree matrix $\mathbf{D}$ is computed, where each diagonal entry D_{ii} represents the sum of the weights of edges connected to node i. Using $\mathbf{W}$ and $\mathbf{D}$, the symmetric normalized Laplacian matrix $\mathbf{L}_{\text{sym}}$ is calculated. Spectral decomposition is then applied to $\mathbf{L}_{\text{sym}}$, yielding the eigenvalues U and eigenvectors V. Finally, the patches are reordered based on the spectral information, resulting in the ordered sequence of patches P.

Algorithm 2 Spectral Traversal Scan Module

Require: patch feature $\mathbf{f}$, number of neighbors k, and eigenvectors m
Ensure: Traversal sequence $\mathcal{P}$
Step 1: *Compute Weighted Adjacency Matrix*
for each pair of patches $(\mathbf{x}_i, \mathbf{x}_j)$ **do**
  if $i \in knn_j$ OR $j \in knn_i$ **then**
    $W_{ij} = \exp\left(-\frac{\|\mathbf{f}_i - \mathbf{f}_j\|^2}{2\sigma^2}\right)$
  else
    $W_{ij} = 0$
  end if
end for
Step 2: *Compute Normalized Laplacian*
$\mathbf{L}_{\text{sym}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$
Step 3: *Compute m Smallest Eigenvectors of $\mathbf{L}_{sym}$*
$[U, V] = \text{eigenSolver}(L_{\text{sym}}, m)$
Step 4: *Building Traversal Sequences*
Sort U from smallest to largest.
Select the corresponding m eigenvalues from V.
for $j = 1$ to m **do**
  P_1^j = sorting f using increasing order of $\mathbf{v}^{(j)}$
  P_2^j = sorting f using decreasing order of $\mathbf{v}^{(j)}$
  $\mathcal{P} = \left[P_1^{(j)},\ P_2^{(j)}\right]^{j=1,\ldots,m}$
end for
Output: Traversal sequence $\mathcal{P}$.

B. Downsampling Strategy

Similar to VMamba, the architecture of our network consists of four layers. Each layer contains a different number of Spectral VMamba blocks. For example, in the tiny scale configuration, we use the setup [2, 2, 5, 2], indicating that the first layer has two Spectral VMamba blocks, the second layer also has two blocks, the third layer has five blocks, and the final layer contains two blocks. As previously mentioned, the STS module is applied only once, in the first Spectral VMamba block of the first layer. Additionally, downsampling occurs at the end of each layer.

The eigenvectors are initially computed in the STS using the original 14×14 spatial features. However, after downsampling, the eigenvectors derived from the 14×14 patches no longer align with the downsampled patches in subsequent layers, which requires careful handling to maintain consis-

tency. To resolve this alignment issue, we save the indices used during the max pooling operation, which is responsible for downsampling the spatial features. By leveraging the saved indices, we extract the corresponding eigenvectors, generating a new set that matches the shape of the downsampled patches. This alignment ensures that the eigenvectors and patches remain correctly paired, enabling us to order the downsampled patches using eigenvectors of compatible dimensions. We repeat this process at each subsequent layer to maintain proper alignment throughout the network.

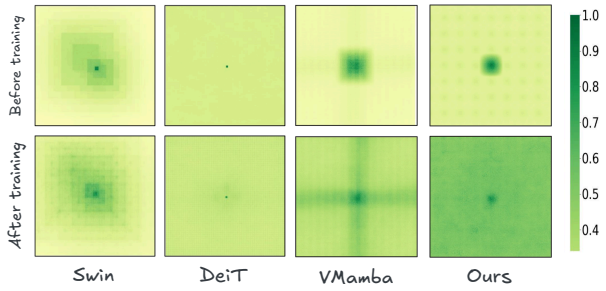

Figure 1. Comparison of Effective Receptive Fields (ERF) Before and After Training.

C. Visualization of Activation Maps

Effective Receptive Field (ERF) in Convolutional Neural Networks (CNNs) refer to the region of the input image that significantly influences the activation of a particular neuron in a deeper layer of the network. Unlike the theoretical receptive field, which considers the full extent of influence regardless of intensity, the ERF focuses on the practical impact, often showing that only a central portion of the theoretical receptive field has substantial influence. Analyzing ERFs helps in refining network designs to ensure that neurons capture relevant information efficiently, enhancing performance in tasks such as object detection and image segmentation.

We conducted experiments to compare our model with VMamba, focusing on the ERF of the central pixel before and after training. Our results in Figure 1 indicate that our model exhibits more global ERFs compared to VMamba. Specifically, after training, the areas colored dark green, which represent regions of high influence, are more extensive in our method than in VMamba. This suggests that our model is better at capturing broader contextual information from the input image. The increased dark green regions in our model demonstrate its enhanced capability to integrate information over larger portions of the input, potentially leading to improved performance in tasks requiring a comprehensive understanding of the image content.

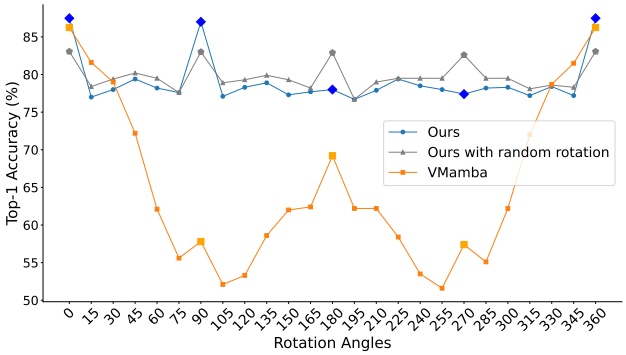

Figure 2. Effect of Random Rotation in RFN Module.

D. Training Dynamics

Figure 3 illustrates the comparison of training dynamics, measured using the maximum accuracy with an Exponential Moving Average, across three model scales: Tiny, Small, and Base. Each plot represents the accuracy progression over 300 epochs for our proposed method and the VMamba model. The plots represents our method demonstrates faster convergence across all scales, achieving high accuracy earlier in training compared to VMamba. The faster convergence not only highlights the efficiency of our approach but also reduces training time, making it highly suitable for practical applications where computational resources or time are constrained.

E. RFN with Random Rotations

For the RFN module, we utilized four *canonical* angles: $0°, 90°, 180°, 270°$, corresponding to quarter turns. Additionally, we tested the method with four random rotations selected at each iteration from the ranges [0, 90], [91, 180], [181, 270], and [271, 360]. Unlike quarter turns, most random rotations necessitate interpolation. The results, presented in Figure 2, demonstrate that even with random rotations, the model can achieve a high level of invariance. However, a slight accuracy drop was observed when using random rotations (83.06%) compared to the canonical quarter turns (87.48%), which can be due to the loss of details resulting from interpolation.

F. Downstream tasks

To demonstrate the versatility of our pre-trained model and its applicability to tasks beyond classification, we extended its use to semantic segmentation. Specifically, we fine-tuned the model—originally pre-trained on the *mini*ImageNet dataset—to perform segmentation tasks on ADE20K dataset [**?**]. ADE20K includes 150 fine-grained semantic categories and comprises 20,000 training images, 2,000 validation images, and 3,000 test images. For optimization, we use AdamW with a weight decay of 0.01 and a total batch

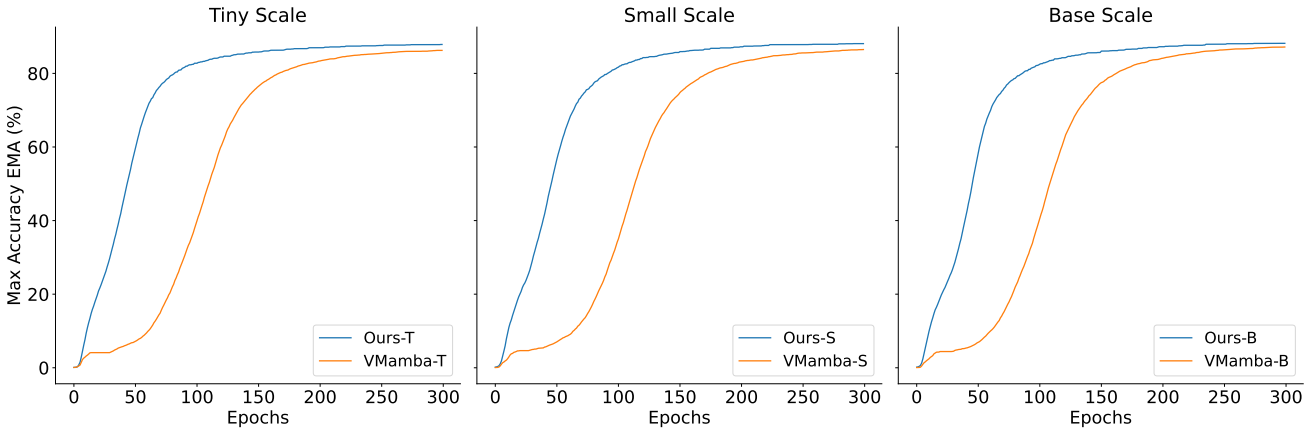

Figure 3. Training Speed Comparison.

size of 2 per GPU. The training schedule features an initial learning rate of 6×10^{-5}, linear decay, a 1500-iteration linear warmup, and a total of 160,000 iterations. We apply standard data augmentations such as random horizontal flipping, random scaling within a ratio range of 0.5 to 2.0, and random photometric distortion.

Table 1 shows that our method outperforms VMamba by +2.81 (tiny), +1.68 (small), and +1.49 (base) on single-scale (SS) testing, and by +3.74 (tiny), +2.21 (small), and +0.58 (base) on multi-scale testing. The backbone used for the segmentation task was initialized from a classification checkpoint which was trained on the mini-ImageNet dataset. This explains the relatively lower range of segmentation performance compared to other papers, which often use classification models pre-trained on ImageNet-1k.

Method	**mIoU (SS)**	**mIoU (MS)**
VMamba-T	22.77	23.98
VMamba-S	25.84	27.13
VMamba-B	26.32	28.41
Ours-T	25.58 **(+2.81)**	27.72 **(+3.74)**
Ours-S	27.52 **(+1.68)**	29.34 **(+2.21)**
Ours-B	27.81 **(+1.49)**	28.99 **(+0.58)**

Table 1. Results of semantic segmentation on ADE20K. SS and MS denote single-scale and multi-scale testing, respectively.