VMamba: Visual State Space Model

문학준

gloriel621@g.skku.edu

TNT Vision

2024/4/30

TRAIN AND TEST

Contents

TRAIN AND TEST

Introduction

1. ViT의 단점
    a. Self Attention의 Complexity
2. VMamba의 contribution
    a. SSM에 기반한 novel network architecture 제시.
    b. Cross-Scan Module (CSM) 제시 -> bridge the gap between 1d array scanning and 2d plain traversing
    c. Classification, Object Detection, Semantic Segmentation에서 좋은 성능을 보임.

TRAIN AND TEST

Preliminaries

- SSM(State Space Models)

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t) \qquad (1a)$$
$$y(t) = \boldsymbol{C}h(t) \qquad (1b)$$

- Discretization of SSM (이산화) : $(\Delta, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) \mapsto (\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})$

$$h_t = \overline{\boldsymbol{A}}h_{t-1} + \overline{\boldsymbol{B}}x_t \qquad (2a)$$
$$y_t = \boldsymbol{C}h_t \qquad (2b)$$

$$\overline{\boldsymbol{A}} = \exp(\Delta \boldsymbol{A}) \qquad \overline{\boldsymbol{B}} = (\Delta \boldsymbol{A})^{-1}(\exp(\Delta \boldsymbol{A}) - \boldsymbol{I}) \cdot \Delta \boldsymbol{B}$$

- Kernel을 Precomputing 하여 값 계산, Convolution 가능.

$$\overline{\boldsymbol{K}} = (\boldsymbol{C}\overline{\boldsymbol{B}}, \boldsymbol{C}\overline{\boldsymbol{A}\boldsymbol{B}}, \dots, \boldsymbol{C}\overline{\boldsymbol{A}}^{k}\overline{\boldsymbol{B}}, \dots) \qquad (3a)$$
$$y = x * \overline{\boldsymbol{K}} \qquad (3b)$$

TRAIN AND TEST

Preliminaries

- Linear Time Invariant(선형 시간 불변성)

$$\overline{A} = \exp(\Delta A) \qquad \overline{B} = (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B$$

($\bar{A}, \bar{B}, C$는 모두 고정적인 값)
(변화량 Δ는 항상 일정한 값)

$$h_t = \overline{A}h_{t-1} + \overline{B}x_t \qquad (2a)$$

$$y_t = Ch_t \qquad (2b)$$

TRAIN AND TEST

Preliminaries

- Structured State Space Model (S4)
  a. A의 값을 어떻게 정의하는 것이 좋은가?
  b. 연산 효율을 어떻게 높일 수 있는가?

  ➡ 분해 가능하고, 좋은 성능을 보이는 A를 사용하여 Convolution Kernel $\bar{K}$ 를 구한다.

- 1. S4에서 Random A보다 HiPPO A를 사용했을 때 더 나은 성능을 보임.

$$(\textbf{HiPPO Matrix}) \qquad \boldsymbol{A}_{nk} = -\begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}$$

- 2. 이 HiPPO A는 Normal Plus Low Rank(NPLR) 로 decompose 할 수 있다.
  (= A를 정규 행렬과 low-rank 행렬의 합의 형태로 표현할 수 있다.)
  V = unitary matrix, Λ = diagonal matrix, P, Q = low-rank matrix

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^* - \boldsymbol{P}\boldsymbol{Q}^\top = \boldsymbol{V}\left(\boldsymbol{\Lambda} - \left(\boldsymbol{V}^*\boldsymbol{P}\right)\left(\boldsymbol{V}^*\boldsymbol{Q}\right)^*\right)\boldsymbol{V}^*$$

TRAIN AND TEST

Preliminaries

- Structured State Space Model (S4)
  a. A의 값을 어떻게 정의하는 것이 좋은가?
  b. 연산 효율을 어떻게 높일 수 있는가?

- 3. $\overline{A}, \overline{B}$ 를 A, B에 대한 식으로 나타낼 수 있다.
  그리고 iFFT를 통해 $\overline{K}$도 구할 수 있다.

$$\overline{\boldsymbol{A}} = \boldsymbol{A_1 A_0}$$

$$\overline{\boldsymbol{B}} = \frac{2}{\Delta}\boldsymbol{A_1}\Delta\boldsymbol{B} = 2\boldsymbol{A_1 B}.$$

$$\begin{aligned} x_k &= \overline{\boldsymbol{A}}x_{k-1} + \overline{\boldsymbol{B}}u_k \\ &= \boldsymbol{A_1 A_0}x_{k-1} + 2\boldsymbol{A_1 B}u_k \\ y_k &= \boldsymbol{C}x_k. \end{aligned}$$

Algorithm 1 S4 Convolution Kernel (Sketch)

Input: S4 parameters $\boldsymbol{\Lambda}, \boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{C}^N$ and step size Δ
Output: SSM convolution kernel $\overline{\boldsymbol{K}} = \mathcal{K}_L(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}})$ for $\boldsymbol{A} = \boldsymbol{\Lambda} - \boldsymbol{P}\boldsymbol{Q}^*$ (equation (5))

1: $\widetilde{\boldsymbol{C}} \leftarrow \left(\boldsymbol{I} - \overline{\boldsymbol{A}}^L\right)^* \overline{\boldsymbol{C}}$ ▷ Truncate SSM generating function (SSMGF) to length L

2: $\begin{bmatrix} k_{00}(\omega) & k_{01}(\omega) \\ k_{10}(\omega) & k_{11}(\omega) \end{bmatrix} \leftarrow \left[\widetilde{\boldsymbol{C}}\ \boldsymbol{Q}\right]^* \left(\frac{2}{\Delta}\frac{1-\omega}{1+\omega} - \boldsymbol{\Lambda}\right)^{-1} [\boldsymbol{B}\ \boldsymbol{P}]$ ▷ Black-box Cauchy kernel

3: $\hat{\boldsymbol{K}}(\omega) \leftarrow \frac{2}{1+\omega}\left[k_{00}(\omega) - k_{01}(\omega)(1 + k_{11}(\omega))^{-1}k_{10}(\omega)\right]$ ▷ Woodbury Identity

4: $\hat{\boldsymbol{K}} = \{\hat{\boldsymbol{K}}(\omega) : \omega = \exp(2\pi i \frac{k}{L})\}$ ▷ Evaluate SSMGF at all roots of unity $\omega \in \Omega_L$

5: $\overline{\boldsymbol{K}} \leftarrow \mathsf{iFFT}(\hat{\boldsymbol{K}})$ ▷ Inverse Fourier Transform

TRAIN AND TEST

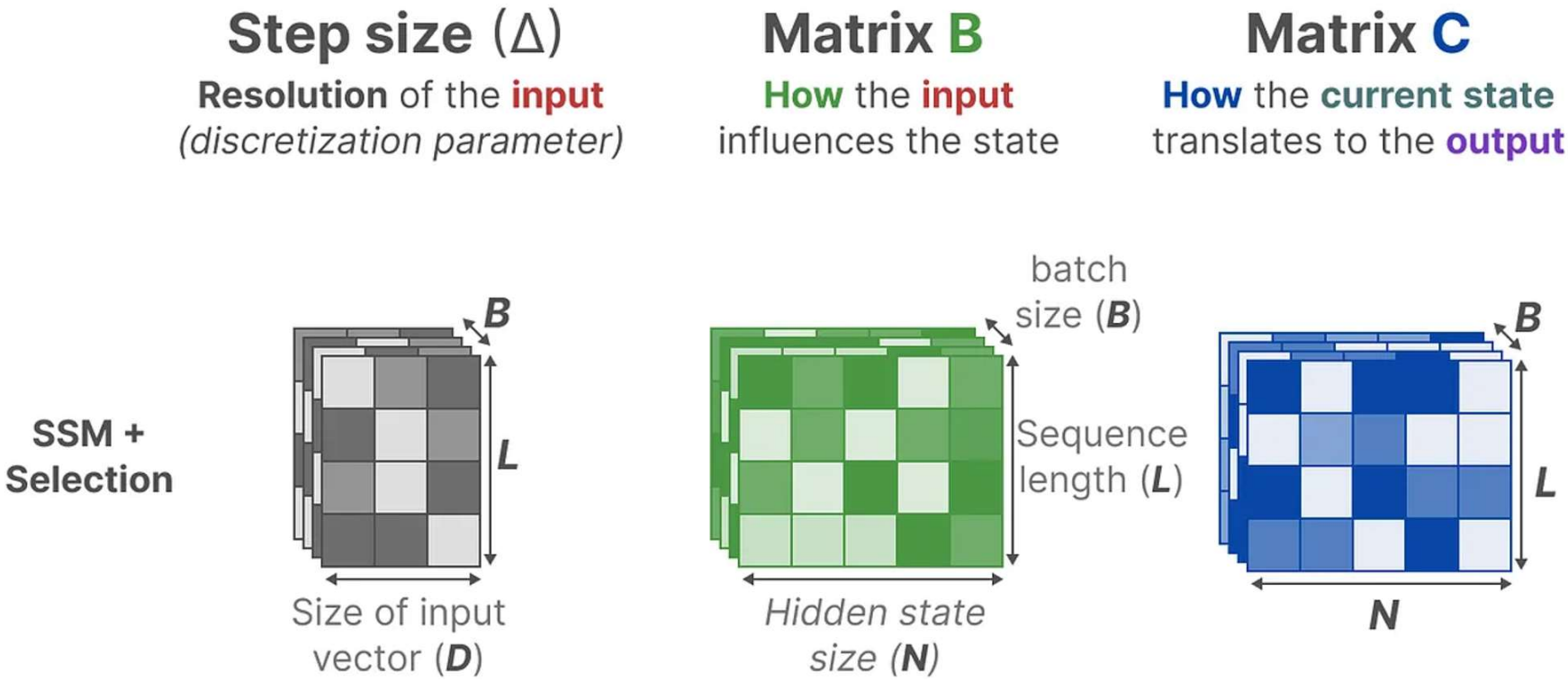
Preliminaries

- Mamba(S6)
  a. Linear-Time Invariant -> Time Variant Model 로 변환.
     - LTI 모델(S4)의 한계 : Selective copying, Inductive head task를 잘 수행하지 못함. (Selectivity가 부족)
     - 따라서, 정보를 선택적으로 저장해야 할 필요가 있음.
     - S4에서는 입력 x에 상관없이, A, B, C는 항상 같은 정적인 특성을 가진다.
     - Mamba는 입력에 따라 B, C, Δ 가 달라진다. (A는 여전히 고정된 값)
     - Δ가 작을 경우, 입력의 비중이 낮아지고, 이전 맥락을 사용하는 데에 초점을 맞춘다.

TRAIN AND TEST

Preliminaries

- Mamba(S6)



Preliminaries

- Mamba(S6)

  b. Selective Scan

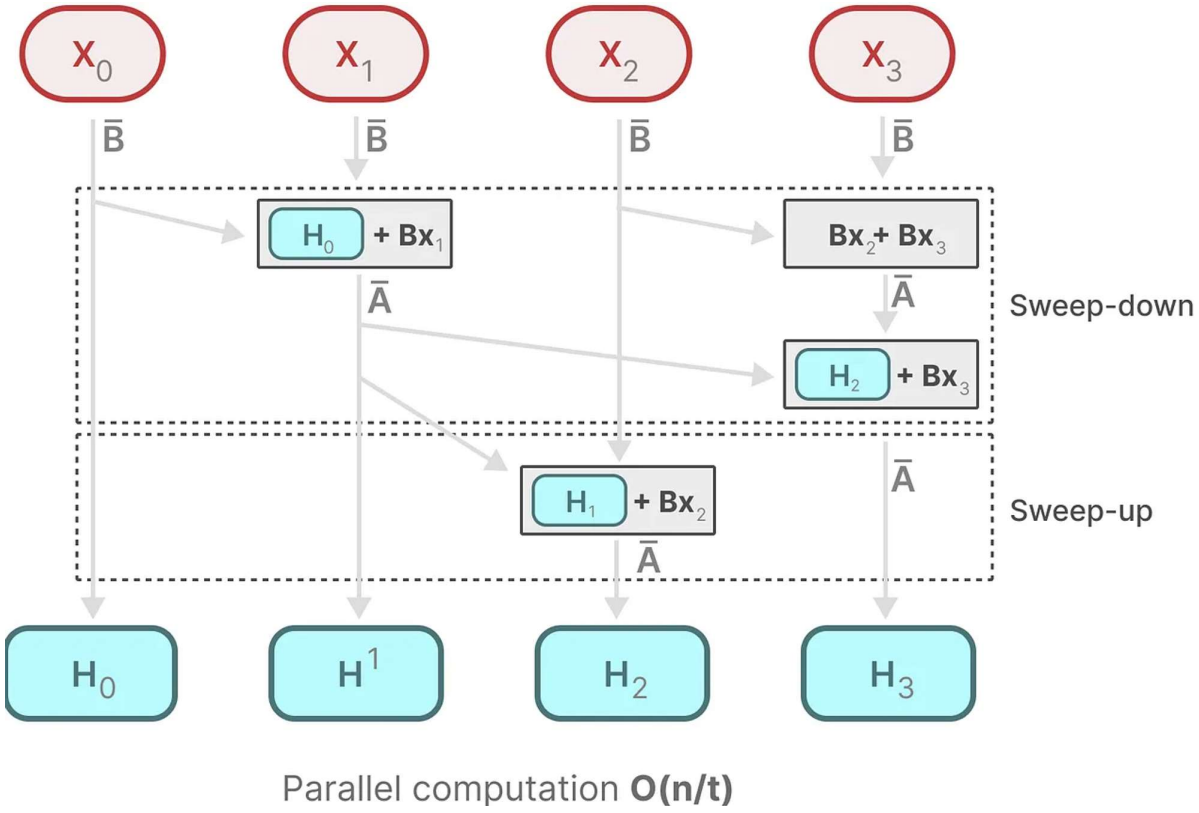
  - 입력에 따라 B, C, Δ 가 달라지기 때문에, Kernel을 미리 계산해 둘 수 없다. (Convolution 사용 불가)
  - Parallel Associative Scan, Kernel Fusion, Recomputation을 사용하여 해당 단점을 극복.

TRAIN AND TEST

Preliminaries

- Mamba(S6)
  - Parallel Associative Scan

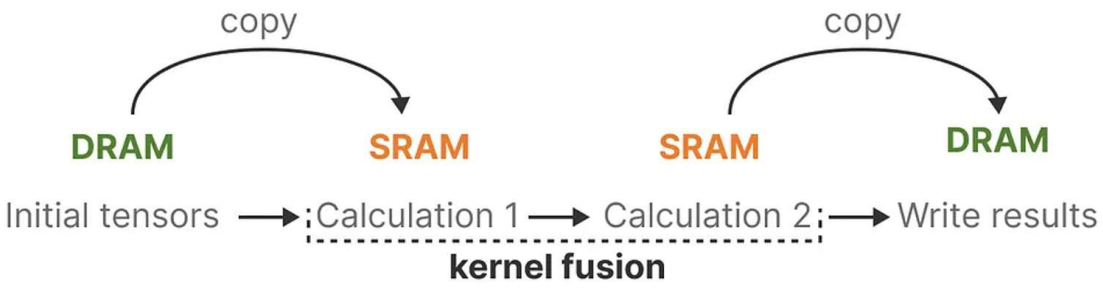
    (Convolution 대신 사용)

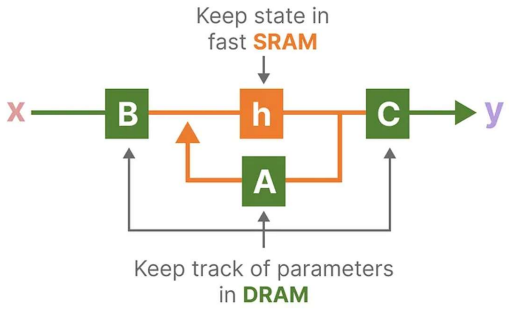


TRAIN AND TEST

Preliminaries

- Mamba(S6)
  - Kernel Fusion & Recomputation

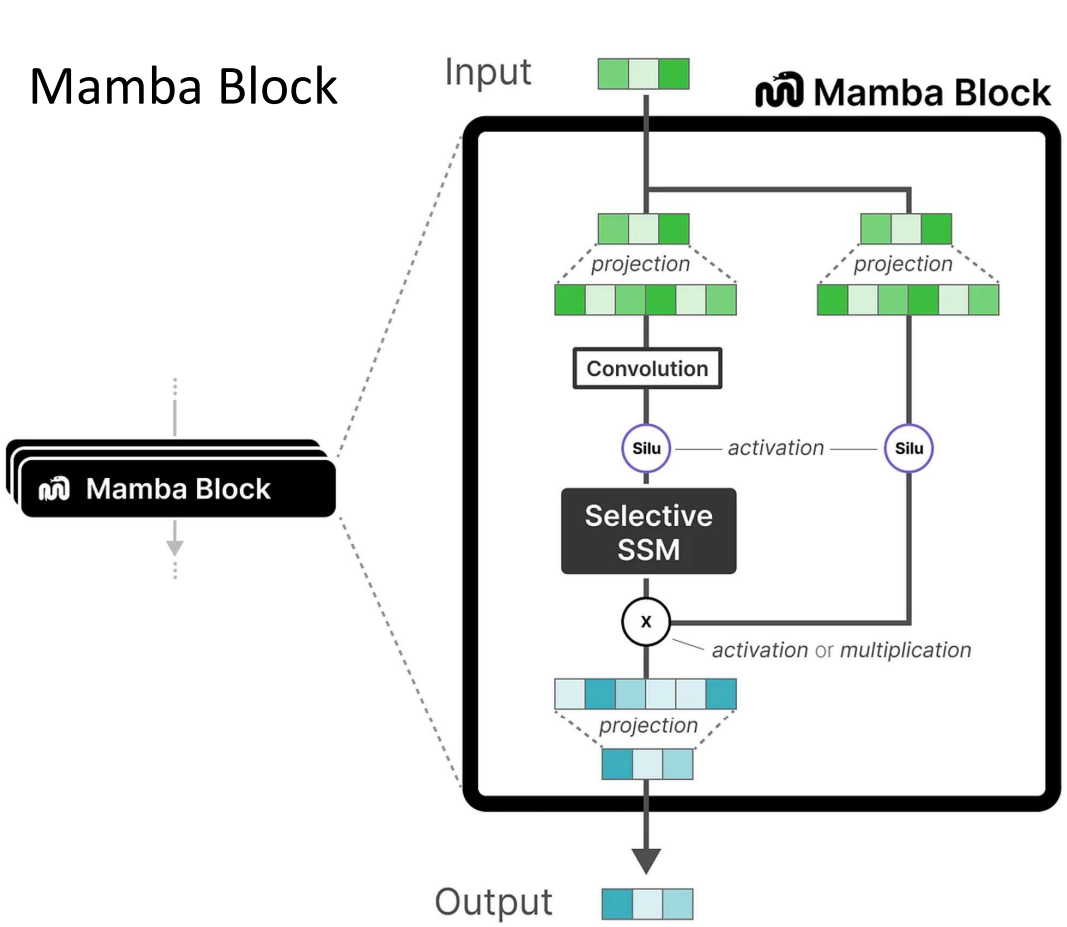


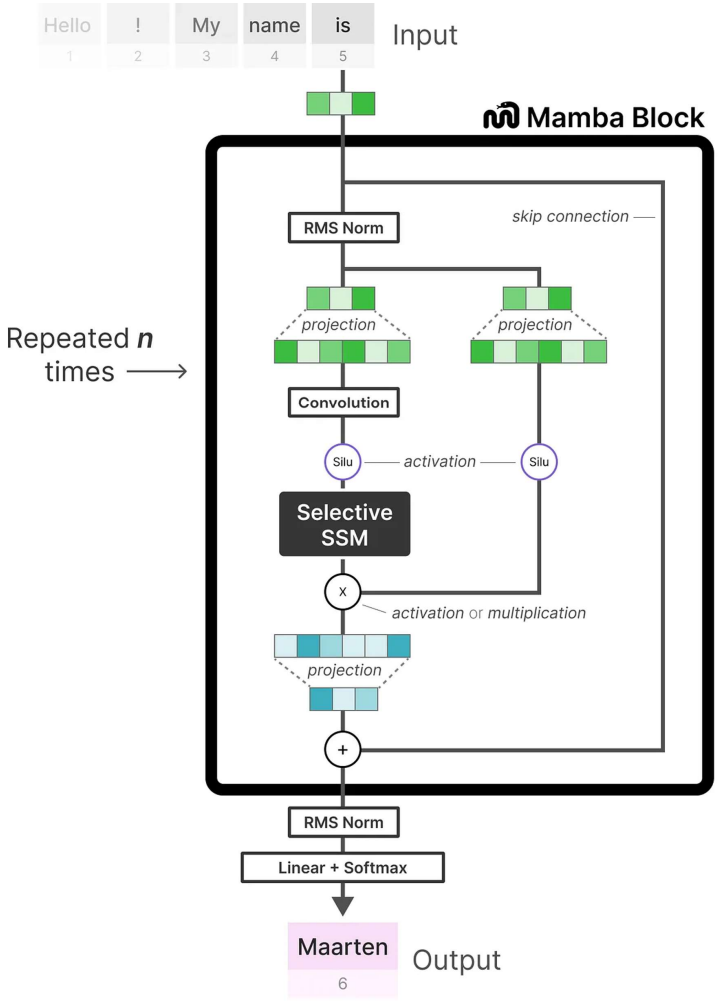


TRAIN AND TEST

Preliminaries

- Mamba Block



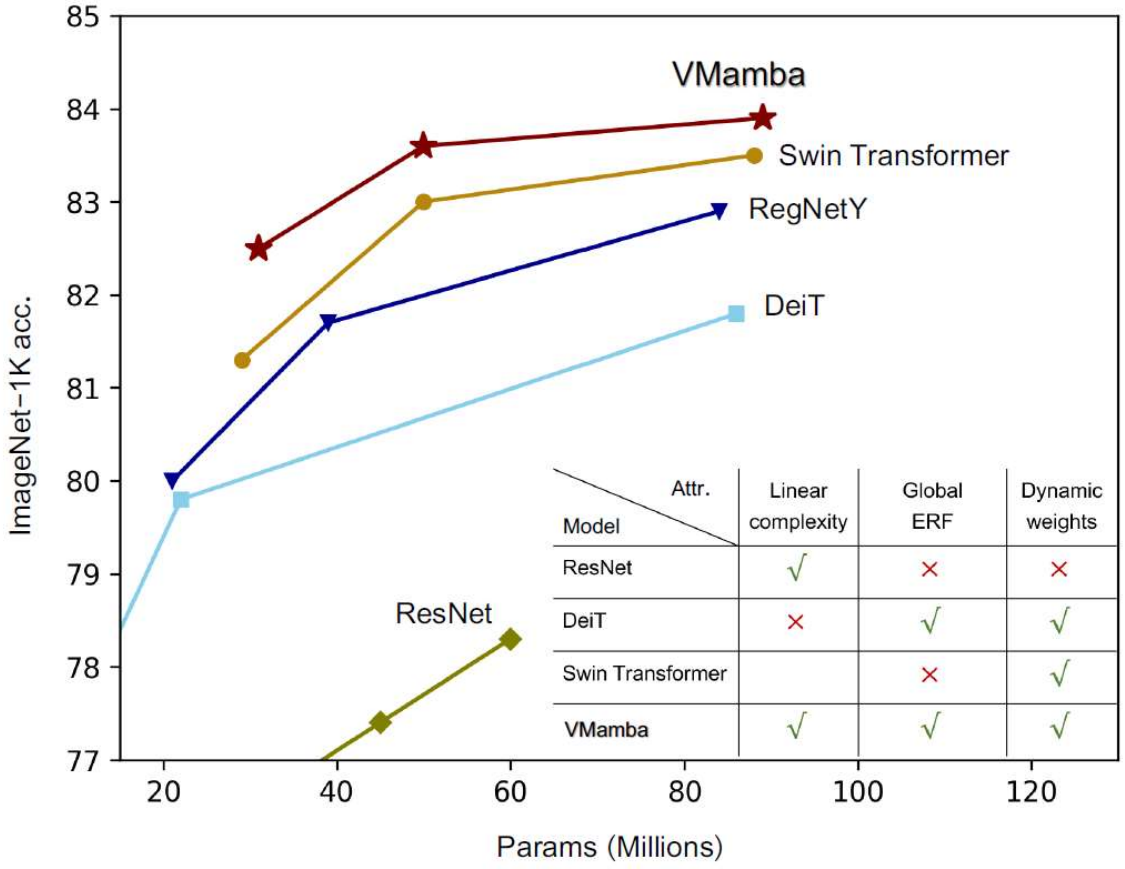


VMamba: Visual State Space Model

TRAIN AND TEST

VMamba Performance

- ImageNet 1K



Model \ Attr.	Linear complexity	Global ERF	Dynamic weights
ResNet	√	×	×
DeiT	×	√	√
Swin Transformer		×	√
VMamba	√	√	√

TRAIN AND TEST

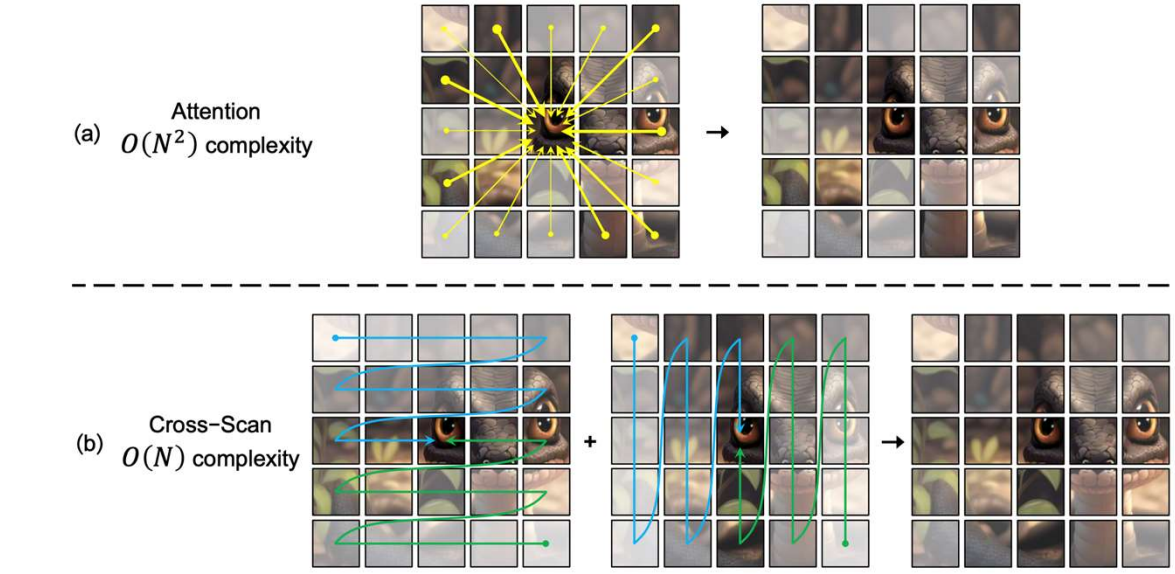
Cross-Scan Module(CSM)

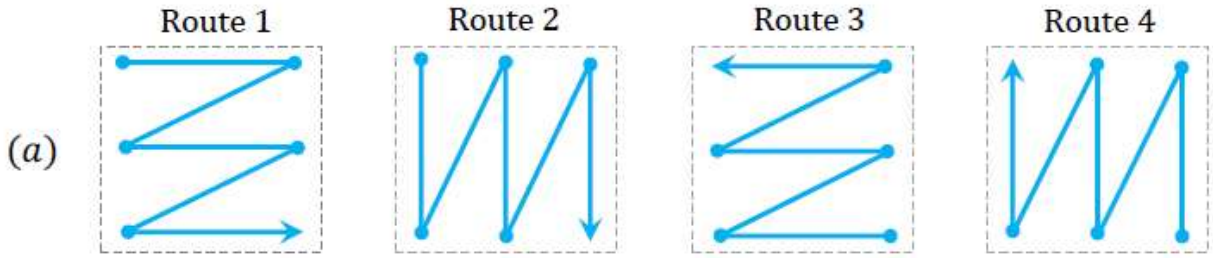
S4ND: S4를 Vision Task에 적용시킨 논문

- Convolution 사용
- Weight이 input independent.
- Limited capacity.

Attention VS Cross-Scan Module

- 한 patch를 다른 patch들과 모두 attention 함. O(N^2)
- CSM은 기존의 SSM 기반 방식과 달리 4방향의 Cross Scan을 적용하여 처리함.





TRAIN AND TEST

2D-Selective-Scan (SS2D)

1. Cross-Scan
2. Parallel S6 block processing
3. Cross-Merge

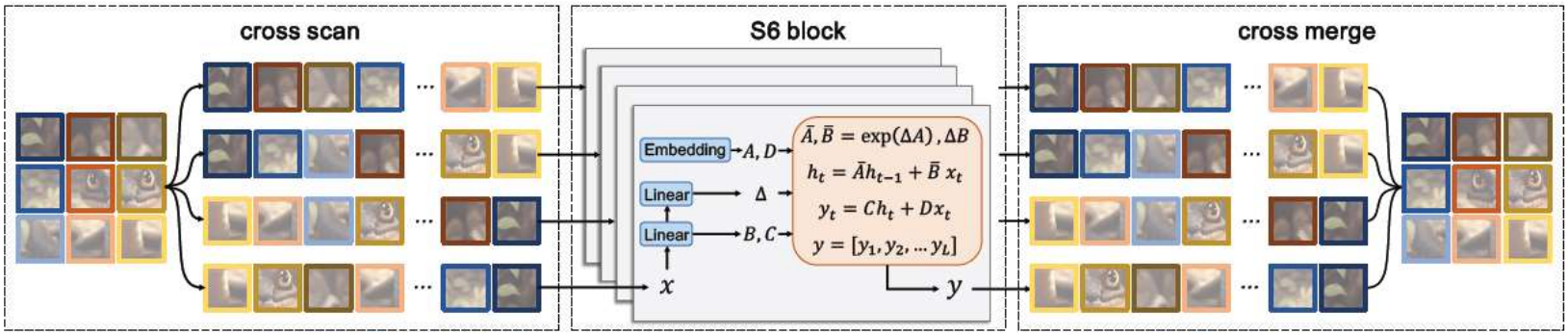


Figure 3: **Illustration of the 2D-Selective-Scan (SS2D) operation.** The input image is first divided into patches and then flattened along four different scanning paths (*Cross-Scan*). The resulting sequences of image patches are then separately processed by distinct S6 blocks. Subsequently, the outputs are merged to construct the 2D feature map as the final output (*Cross-Merge*).

TRAIN AND TEST

The Relationship between SS2D and Self-attention

In this section, we clarify the relationship between SS2D and the self-attention mechanism commonly employed in existing backbone models. Subsequently, visualization results are provided to substantiate our explanation.

Notation Definition. Let T denote the length of the sequence with indices from a to b, we define the following variables

$$\boldsymbol{V} \coloneqq [\boldsymbol{V_1}; \dots; \boldsymbol{V_T}] \in \mathbb{R}^{T \times D_v}, \text{ where } \boldsymbol{V_i} \coloneqq \boldsymbol{u_{a+i-1}} \boldsymbol{\Delta_{a+i-1}} \in \mathbb{R}^{1 \times D_v} \tag{10}$$

$$\boldsymbol{K} \coloneqq [\boldsymbol{K_1}; \dots; \boldsymbol{K_T}] \in \mathbb{R}^{T \times D_k}, \text{ where } \boldsymbol{K_i} \coloneqq \boldsymbol{B_{a+i-1}} \in \mathbb{R}^{1 \times D_k} \tag{11}$$

$$\boldsymbol{Q} \coloneqq [\boldsymbol{Q_1}; \dots; \boldsymbol{Q_T}] \in \mathbb{R}^{T \times D_k}, \text{ where } \boldsymbol{Q_i} \coloneqq \boldsymbol{C_{a+i-1}} \in \mathbb{R}^{1 \times D_k} \tag{12}$$

$$\boldsymbol{w} \coloneqq [\boldsymbol{w_1}; \dots; \boldsymbol{w_T}] \in \mathbb{R}^{T \times D_k \times D_v}, \text{ where } \boldsymbol{w_i} \coloneqq \prod_{j=1}^{i} e^{\boldsymbol{A}\boldsymbol{\Delta_{a-1+j}}} \in \mathbb{R}^{D_k \times D_v} \tag{13}$$

$$\boldsymbol{H} \coloneqq [\boldsymbol{h_a}; \dots; \boldsymbol{h_b}] \in \mathbb{R}^{T \times D_k \times D_v}, \text{ where } \boldsymbol{h_i} \in \mathbb{R}^{D_k \times D_v} \tag{14}$$

$$\boldsymbol{Y} \coloneqq [\boldsymbol{y_a}; \dots; \boldsymbol{y_b}] \in \mathbb{R}^{T \times D_v}, \text{ where } \boldsymbol{y_i} \in \mathbb{R}^{D_v} \tag{15}$$

TRAIN AND TEST

The Relationship between SS2D and Self-attention

Mathematical Derivation. Based on these notations, the discretized solution to time-varying SSMs (Eq. 3) can be written as

$$h_b = w_T \odot h_a + \sum_{i=1}^{T} \frac{w_T}{w_i} \odot \left({K_i}^\top V_i \right) \tag{16}$$

where $\odot$ denotes the element-wise product between matrices, and the division is also elements-wise. Therefore, the first term of the output of SSM can be computed by

$$y_b = Q_T h_b \tag{17}$$

$$= Q_T (w_T \odot h_a) + Q_T \sum_{i=1}^{T} \frac{w_T}{w_i} \odot \left({K_i}^\top V_i \right) \tag{18}$$

Therefore, the j-th slice along dimension D_v of $\boldsymbol{Y}$, denoted as $\boldsymbol{X} := \boldsymbol{Y}^{(j)} \in \mathbb{R}^{T \times 1}$, can be expressed as

$$\boldsymbol{X} = (Q_T \odot w^{(j)}) {h_a}^{(j)} + \left[\left(Q_T \odot w^{(j)} \right) \left(\frac{K}{w^{(j)}} \right)^\top \odot M \right] V^{(j)} \tag{19}$$

TRAIN AND TEST

Activation Map

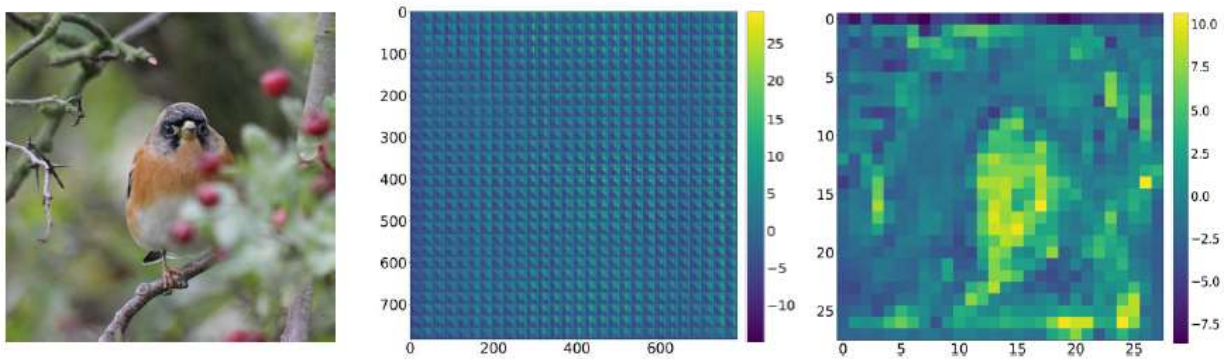
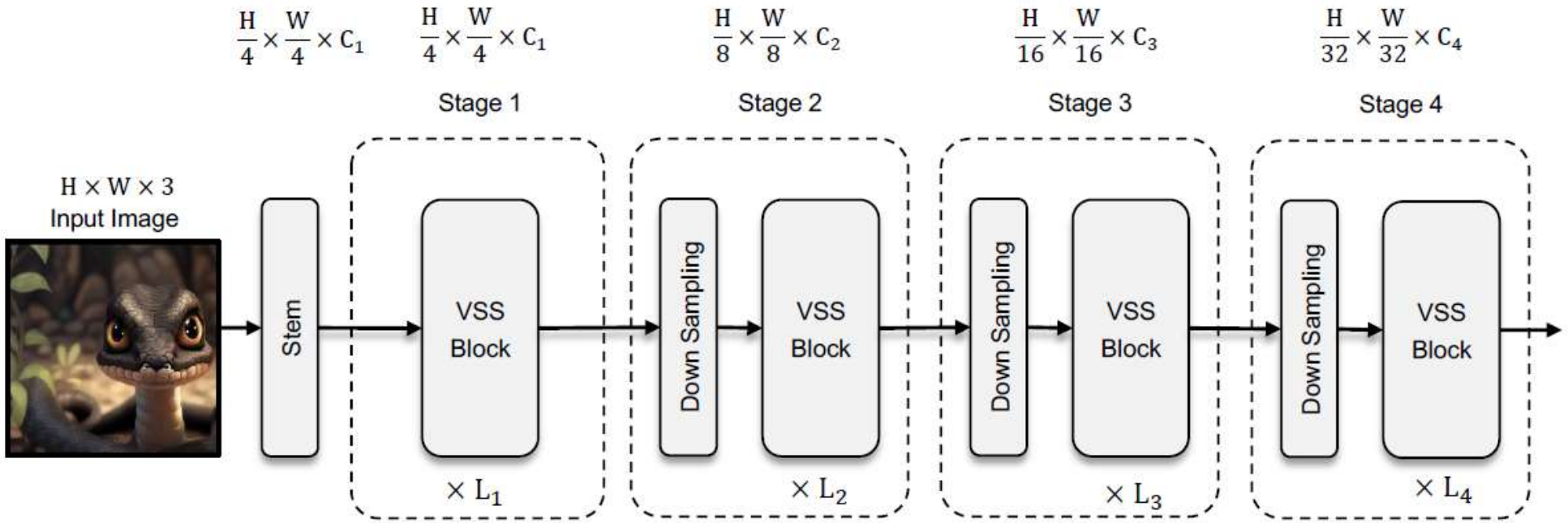


Figure 5: The visualization of the (a) input image, (b) overall activation map, and (c) heat map of CSM.

Activation Map Visualization. We have demonstrated that the computational process of selective SSMs encompasses mechanisms resembling self-attention, and this enables us to probe the internal operations of SS2D by visualizing its weight matrices.

TRAIN AND TEST

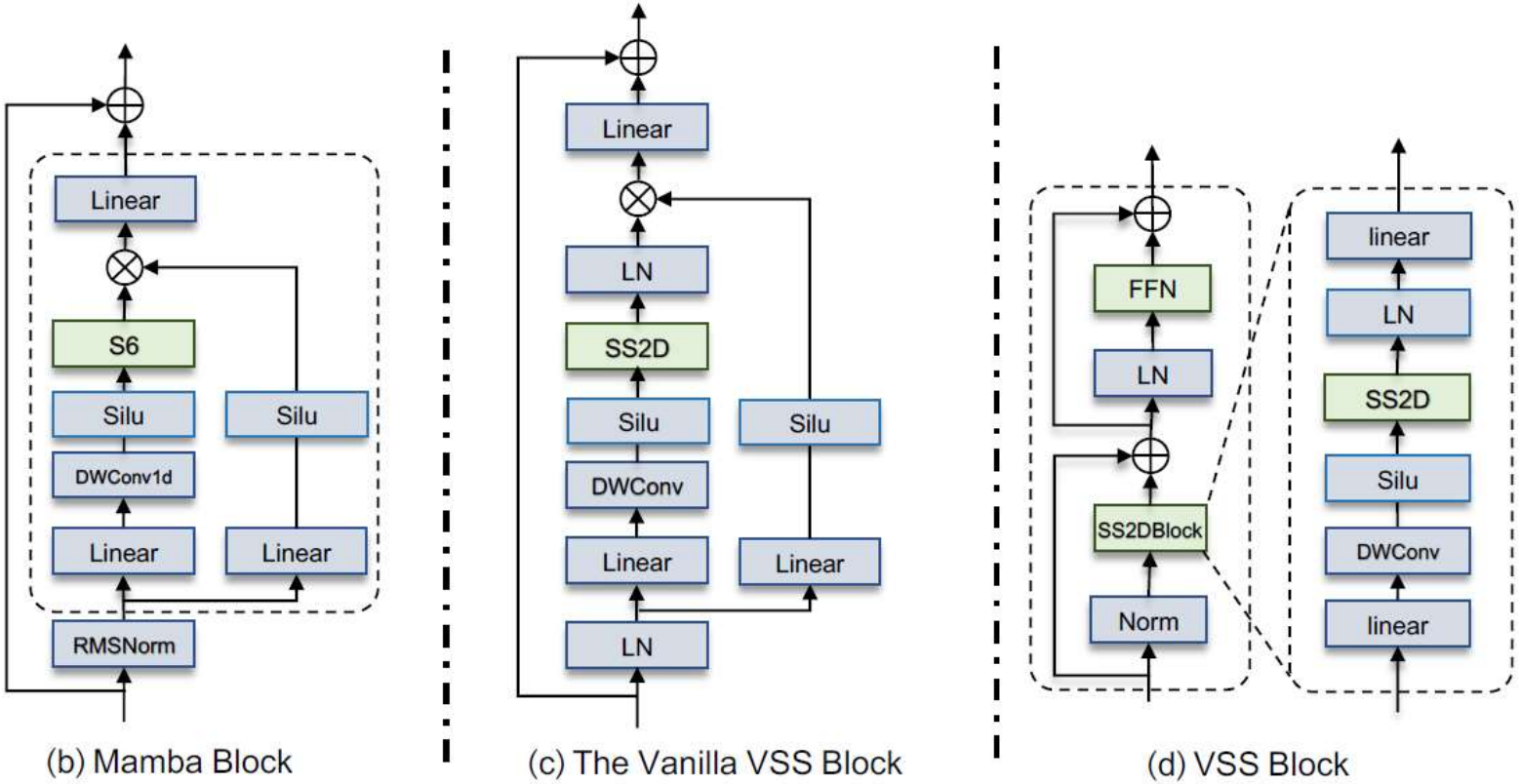
Overall Architecture



(a) Architecture of VMamba

VSS Block

VSS: Visual State Space

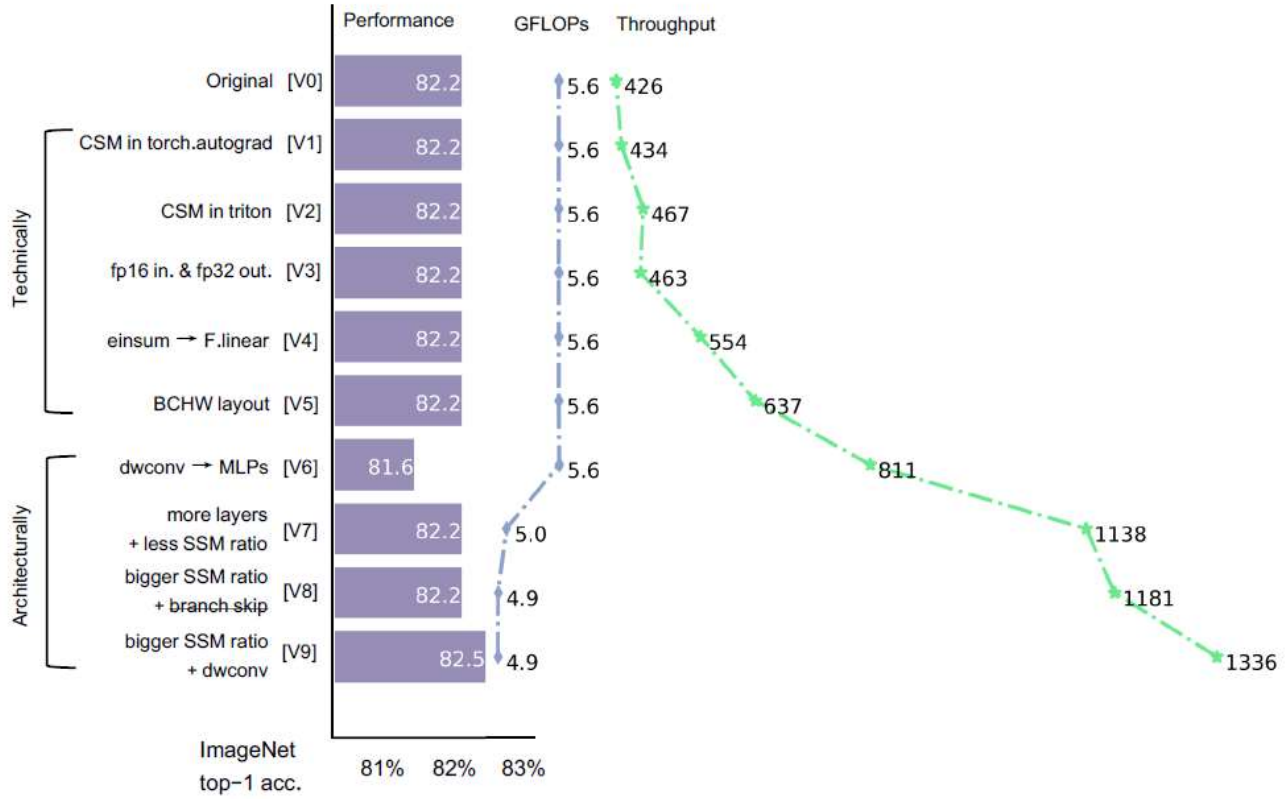


(b) Mamba Block (c) The Vanilla VSS Block (d) VSS Block

TRAIN AND TEST

Accelerating VMamba

(= 많은 시행착오를 통해 조금씩 성능을 높였습니다)



TRAIN AND TEST

Experiments

ViT 구조 참조하여 ResNet 수정

	Method	Image size	#Param.	FLOPs	Throughput	Train Throughput	ImageNet top-1 acc.
CNN	RegNetY-4G [41]	224^2	21M	4.0G	–	–	80.0
	RegNetY-8G [41]	224^2	39M	8.0G	–	–	81.7
	RegNetY-16G [41]	224^2	84M	16.0G	–	–	82.9
CNN	EffNet-B3 [47]	300^2	12M	1.8G	–	–	81.6
	EffNet-B4 [47]	380^2	19M	4.2G	–	–	82.9
	EffNet-B5 [47]	456^2	30M	9.9G	–	–	83.6
	EffNet-B6 [47]	528^2	43M	19.0G	–	–	84.0
	ViT-B/16 [12]	384^2	86M	55.4G	–	–	77.9
	ViT-L/16 [12]	384^2	307M	190.7G	–	–	76.5
ViT + Knowledge Distillation	DeiT-S [50]	224^2	22M	4.6G	1759	2397	79.8
	DeiT-B [50]	224^2	86M	17.5G	500	1024	81.8
	DeiT-B [50]	384^2	86M	55.4G	498	344	83.1
Swin: ViT + Hierarchical Structure + Shifted Window Self-Attention	ConvNeXt-T [33]	224^2	29M	4.5G	1189	701	82.1
	ConvNeXt-S [33]	224^2	50M	8.7G	682	444	83.1
	ConvNeXt-B [33]	224^2	89M	15.4G	435	334	83.8
	HiViT-T [64]	224^2	19M	4.6G	1391	1300	82.1
	HiViT-S [64]	224^2	38M	9.1G	711	697	83.5
	HiViT-B [64]	224^2	66M	15.9G	456	541	83.8
	Swin-T [32]	224^2	28M	4.6G	1247	985	81.3
	Swin-S [32]	224^2	50M	8.7G	719	640	83.0
	Swin-B [32]	224^2	88M	15.4G	457	494	83.5
	S4ND-ConvNeXt-T [40]	224^2	30M	-	684	331	82.2
	S4ND-ViT-B [40]	224^2	89M	-	404	340	80.4
	ViM-S [68]	224^2	26M	-	811	232†	80.5
	VMamba-T	224^2	31M	4.9G	1335	464	82.5
	VMamba-S	224^2	50M	8.7G	874	313	83.6
	VMamba-B	224^2	89M	15.4G	645	246	83.9

Table 1: **Performance comparison on ImageNet-1K**. Throughput values are measured with an A100 GPU and an AMD EPYC 7542 CPU, using the toolkit released by [56], following the protocol proposed in [32]. † denotes that the training process only supports float32 datatype.

TRAIN AND TEST

Experiments

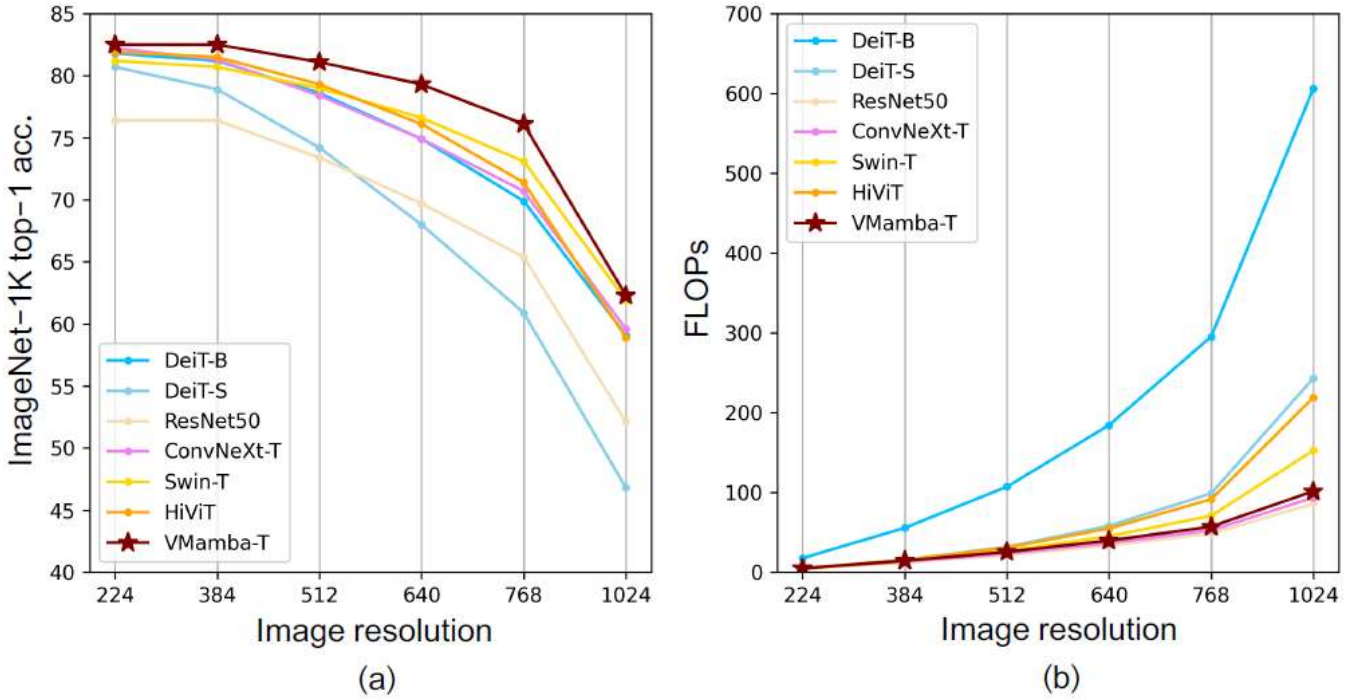


Figure 11: Illustration of the change in (a) classification accuracy and (b) FLOPs with progressively larger test image resolutions.

TRAIN AND TEST

Experiments

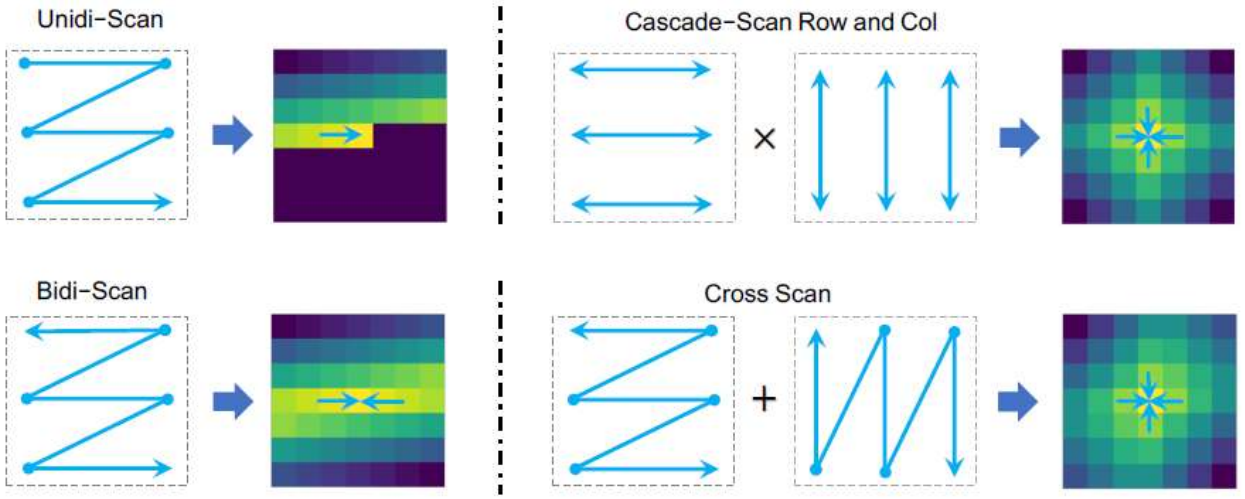


Figure 12: Illustration of different scanning methods for selective scan.

Method	#Param.	FLOPs	Throughput	Train Throughput.	ImageNet top-1 acc.
Unidi-Scan	30.70	4.86	1342	464	82.2
Bidi-Scan	30.70	4.86	1344	465	74.8†
Cascade-Scan	30.70	4.86	817	253	–
CSM	30.70	4.86	1343	464	82.5

Table 5: Performance comparison of different scanning approaches. The proposed CSM achieves

TRAIN AND TEST

Inference Results

```
[2024-04-27 18:26:27 vssm1_tiny_0230](main.py 346): INFO Test: [60/391] Time 0.129 (0.472)      Loss 0.7212 (0.5593)    Acc@1 83.594 (88.589)   Acc@5 95.312 (97.797)   Mem 10417MB
[2024-04-27 18:26:28 vssm1_tiny_0230](main.py 346): INFO Test: [70/391] Time 0.128 (0.424)      Loss 0.6528 (0.5886)    Acc@1 84.375 (87.797)   Acc@5 99.219 (97.700)   Mem 10417MB
[2024-04-27 18:26:29 vssm1_tiny_0230](main.py 346): INFO Test: [80/391] Time 0.129 (0.387)      Loss 0.6416 (0.6034)    Acc@1 85.156 (87.375)   Acc@5 96.094 (97.598)   Mem 10417MB
[2024-04-27 18:26:30 vssm1_tiny_0230](main.py 346): INFO Test: [90/391] Time 0.128 (0.359)      Loss 0.8384 (0.6052)    Acc@1 72.656 (87.303)   Acc@5 96.875 (97.588)   Mem 10417MB
[2024-04-27 18:26:32 vssm1_tiny_0230](main.py 346): INFO Test: [100/391]       Time 0.128 (0.336)      Loss 0.4221 (0.6045)    Acc@1 92.188 (87.106)   Acc@5 97.656 (97.649)   Mem 10417MB
[2024-04-27 18:26:33 vssm1_tiny_0230](main.py 346): INFO Test: [110/391]       Time 0.128 (0.317)      Loss 1.0830 (0.6134)    Acc@1 60.938 (86.691)   Acc@5 99.219 (97.677)   Mem 10417MB
[2024-04-27 18:26:34 vssm1_tiny_0230](main.py 346): INFO Test: [120/391]       Time 0.129 (0.302)      Loss 0.5562 (0.6121)    Acc@1 89.062 (86.848)   Acc@5 98.438 (97.708)   Mem 10417MB
[2024-04-27 18:26:36 vssm1_tiny_0230](main.py 346): INFO Test: [130/391]       Time 0.129 (0.289)      Loss 0.2549 (0.6049)    Acc@1 96.875 (87.083)   Acc@5 100.000 (97.722)  Mem 10417MB
[2024-04-27 18:26:37 vssm1_tiny_0230](main.py 346): INFO Test: [140/391]       Time 0.129 (0.277)      Loss 1.0137 (0.6041)    Acc@1 77.344 (86.963)   Acc@5 96.875 (97.800)   Mem 10417MB
[2024-04-27 18:26:38 vssm1_tiny_0230](main.py 346): INFO Test: [150/391]       Time 0.129 (0.267)      Loss 0.7349 (0.6128)    Acc@1 78.125 (86.796)   Acc@5 98.438 (97.724)   Mem 10417MB
[2024-04-27 18:26:39 vssm1_tiny_0230](main.py 346): INFO Test: [160/391]       Time 0.129 (0.259)      Loss 0.5918 (0.6161)    Acc@1 85.156 (86.753)   Acc@5 97.656 (97.656)   Mem 10417MB
[2024-04-27 18:26:41 vssm1_tiny_0230](main.py 346): INFO Test: [170/391]       Time 0.129 (0.251)      Loss 1.5020 (0.6349)    Acc@1 66.406 (86.230)   Acc@5 89.844 (97.492)   Mem 10417MB
[2024-04-27 18:26:42 vssm1_tiny_0230](main.py 346): INFO Test: [180/391]       Time 0.129 (0.244)      Loss 1.6621 (0.6506)    Acc@1 58.594 (85.855)   Acc@5 89.844 (97.302)   Mem 10417MB
[2024-04-27 18:26:43 vssm1_tiny_0230](main.py 346): INFO Test: [190/391]       Time 0.129 (0.238)      Loss 1.3262 (0.6703)    Acc@1 70.312 (85.320)   Acc@5 92.969 (97.129)   Mem 10417MB
[2024-04-27 18:26:45 vssm1_tiny_0230](main.py 346): INFO Test: [200/391]       Time 0.129 (0.233)      Loss 0.4934 (0.6863)    Acc@1 88.281 (84.966)   Acc@5 99.219 (96.984)   Mem 10417MB
[2024-04-27 18:26:46 vssm1_tiny_0230](main.py 346): INFO Test: [210/391]       Time 0.130 (0.228)      Loss 0.8276 (0.6998)    Acc@1 79.688 (84.575)   Acc@5 97.656 (96.901)   Mem 10417MB
[2024-04-27 18:26:47 vssm1_tiny_0230](main.py 346): INFO Test: [220/391]       Time 0.129 (0.224)      Loss 0.2949 (0.7015)    Acc@1 94.531 (84.523)   Acc@5 98.438 (96.847)   Mem 10417MB
[2024-04-27 18:26:49 vssm1_tiny_0230](main.py 346): INFO Test: [230/391]       Time 0.129 (0.220)      Loss 1.2549 (0.7075)    Acc@1 71.875 (84.409)   Acc@5 89.844 (96.774)   Mem 10417MB
[2024-04-27 18:26:50 vssm1_tiny_0230](main.py 346): INFO Test: [240/391]       Time 0.129 (0.216)      Loss 0.7974 (0.7079)    Acc@1 85.938 (84.430)   Acc@5 92.969 (96.713)   Mem 10417MB
[2024-04-27 18:26:51 vssm1_tiny_0230](main.py 346): INFO Test: [250/391]       Time 0.129 (0.212)      Loss 0.4014 (0.7225)    Acc@1 94.531 (83.945)   Acc@5 99.219 (96.592)   Mem 10417MB
[2024-04-27 18:26:52 vssm1_tiny_0230](main.py 346): INFO Test: [260/391]       Time 0.129 (0.209)      Loss 0.7847 (0.7338)    Acc@1 81.250 (83.675)   Acc@5 96.875 (96.483)   Mem 10417MB
[2024-04-27 18:26:54 vssm1_tiny_0230](main.py 346): INFO Test: [270/391]       Time 0.130 (0.206)      Loss 1.4209 (0.7398)    Acc@1 67.188 (83.525)   Acc@5 90.625 (96.411)   Mem 10417MB
[2024-04-27 18:26:55 vssm1_tiny_0230](main.py 346): INFO Test: [280/391]       Time 0.130 (0.204)      Loss 0.8794 (0.7422)    Acc@1 77.344 (83.502)   Acc@5 96.875 (96.391)   Mem 10417MB
[2024-04-27 18:26:56 vssm1_tiny_0230](main.py 346): INFO Test: [290/391]       Time 0.129 (0.201)      Loss 1.1279 (0.7476)    Acc@1 53.906 (83.336)   Acc@5 97.656 (96.333)   Mem 10417MB
[2024-04-27 18:26:58 vssm1_tiny_0230](main.py 346): INFO Test: [300/391]       Time 0.129 (0.199)      Loss 0.6025 (0.7519)    Acc@1 89.062 (83.285)   Acc@5 95.312 (96.257)   Mem 10417MB
[2024-04-27 18:26:59 vssm1_tiny_0230](main.py 346): INFO Test: [310/391]       Time 0.129 (0.196)      Loss 0.8398 (0.7558)    Acc@1 80.469 (83.224)   Acc@5 94.531 (96.184)   Mem 10417MB
[2024-04-27 18:27:00 vssm1_tiny_0230](main.py 346): INFO Test: [320/391]       Time 0.130 (0.194)      Loss 0.4866 (0.7615)    Acc@1 91.406 (83.100)   Acc@5 98.438 (96.111)   Mem 10417MB
[2024-04-27 18:27:01 vssm1_tiny_0230](main.py 346): INFO Test: [330/391]       Time 0.130 (0.192)      Loss 1.1514 (0.7716)    Acc@1 67.969 (82.841)   Acc@5 94.531 (96.009)   Mem 10417MB
[2024-04-27 18:27:03 vssm1_tiny_0230](main.py 346): INFO Test: [340/391]       Time 0.129 (0.191)      Loss 0.5874 (0.7740)    Acc@1 89.062 (82.790)   Acc@5 98.438 (95.988)   Mem 10417MB
[2024-04-27 18:27:04 vssm1_tiny_0230](main.py 346): INFO Test: [350/391]       Time 0.129 (0.189)      Loss 0.7593 (0.7753)    Acc@1 82.812 (82.737)   Acc@5 97.656 (95.989)   Mem 10417MB
[2024-04-27 18:27:05 vssm1_tiny_0230](main.py 346): INFO Test: [360/391]       Time 0.130 (0.187)      Loss 1.1582 (0.7828)    Acc@1 71.875 (82.535)   Acc@5 96.094 (95.947)   Mem 10417MB
[2024-04-27 18:27:07 vssm1_tiny_0230](main.py 346): INFO Test: [370/391]       Time 0.130 (0.186)      Loss 0.9873 (0.7798)    Acc@1 68.750 (82.560)   Acc@5 98.438 (95.999)   Mem 10417MB
[2024-04-27 18:27:08 vssm1_tiny_0230](main.py 346): INFO Test: [380/391]       Time 0.129 (0.184)      Loss 0.7271 (0.7823)    Acc@1 82.031 (82.495)   Acc@5 98.438 (95.977)   Mem 10417MB
[2024-04-27 18:27:09 vssm1_tiny_0230](main.py 346): INFO Test: [390/391]       Time 0.082 (0.183)      Loss 1.3779 (0.7805)    Acc@1 57.500 (82.492)   Acc@5 95.000 (95.996)   Mem 10417MB
[2024-04-27 18:27:10 vssm1_tiny_0230](main.py 353): INFO  * Acc@1 82.492 Acc@5 95.996
[2024-04-27 18:27:10 vssm1_tiny_0230](main.py 197): INFO Accuracy of the network ema on the 50000 test images: 82.5%
```

TRAIN AND TEST

TRAIN AND TEST