Vision Mamba : Efficient Visual Representation Learning with Bidirectional State Space Model

Study Group vision
김윤서
2024/04/02

Contents

Intro

- Mamba에서의 성공적인 언어 모델링을 바탕으로, 시각 task에도 적용 가능하게 하고 싶음
- 문제점
    1) 단방향 모델링(unidirectional modeling)
    2) 위치 인식의 부족(lack of positional awareness)

- 해결
  : 양방향 SSM(bidirectional SSMs)과 위치 임베딩(positional embeddings)을 결합

- attention이 필요하지 않은 Vim은 ViT와 동일한 모델링 능력을 가지며, 제곱 시간 계산 및 선형 메모리 복잡도만 가진다.
- DeiT보다 2.8배 빠르며, 1248*1248 해상도의 이미지에서 특성을 추출하기 위해 배치 추론을 수행할 때 GPU 메모리를 86.8% 절약한다.

Method

1. SSM

$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t),$$
$$y(t) = \mathbf{C}h(t).$$

$x(t) \in \mathbb{R} \mapsto y(t) \in \mathbb{R}$

$h(t) \in \mathbb{R}^{\mathtt{N}}$

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

이산화(S4, Mamba)

->

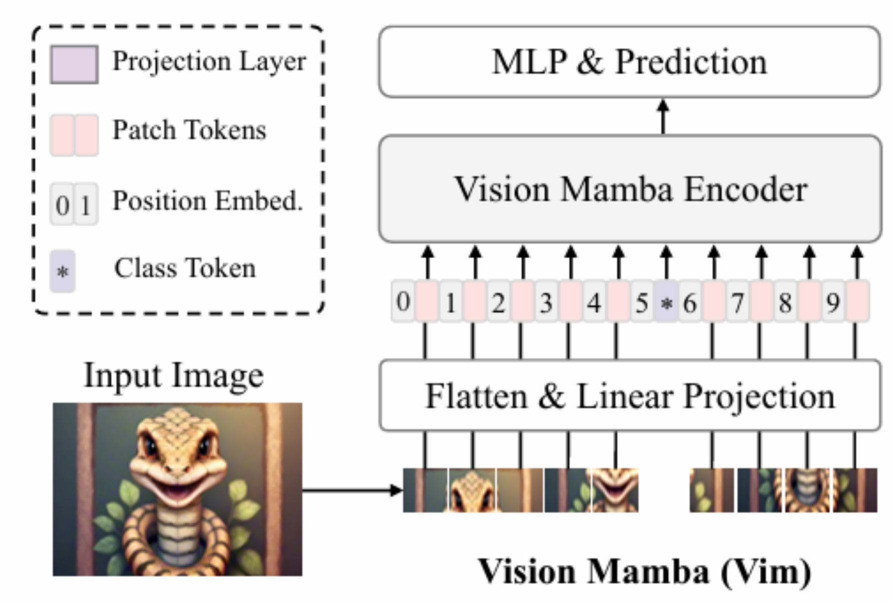
$$h_t = \overline{\mathbf{A}}h_{t-1} + \overline{\mathbf{B}}x_t,$$
$$y_t = \mathbf{C}h_t.$$

$$\overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{AB}}, \dots, \mathbf{C}\overline{\mathbf{A}}^{\mathtt{M}-1}\overline{\mathbf{B}})$$
$$\mathbf{y} = \mathbf{x} * \overline{\mathbf{K}},$$

Convolution kernel

Method

2. Vision Mamba



- Mamba 블록은 1차원 데이터만 처리 가능하므로 2차원인 이미지데이터에 일정한 처리를 해주어야 함.

2차원 이미지 t (H,W,C)

↓

2차원 패치 x_p $(J, (p^2 \cdot C))$

*(H,W)는 이미지 크기, C는 채널수, p는 이미지 패치크기.

↓

크기 D 벡터로 선형적으로 투영 ⊕ 위치임베딩 E_{pos} $(J+1, D)$를 추가

$$\mathbf{T}_0 = [\mathbf{t}_{cls}; \mathbf{t}_p^1\mathbf{W}; \mathbf{t}_p^2\mathbf{W}; \cdots; \mathbf{t}_p^J\mathbf{W}] + \mathbf{E}_{pos}$$

Method

2. Vision Mamba

2차원 이미지 ∈ (H,W,C)

↓

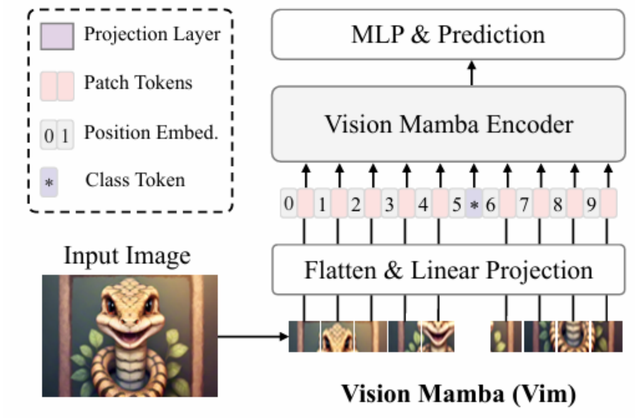
2차원 패치 x_p (J, (P²·C))

*(H,W)는 이미지 크기, C는 채널수, P는 이미지 패치크기.

↓

크기 D 벡터로 선형적으로 투영 ⊕위치임베딩 E_{pos} (J+1, D)를 추가

$$\mathbf{T}_0 = [\mathbf{t}_{cls}; \mathbf{t}_p^1\mathbf{W}; \mathbf{t}_p^2\mathbf{W}; \cdots; \mathbf{t}_p^J\mathbf{W}] + \mathbf{E}_{pos}$$




```python
class PatchEmbed(nn.Module):
    """ 2D Image to Patch Embedding
    """
    def __init__(self, img_size=224, patch_size=16, stride=16, in_chans=3, embed_dim=768, norm_layer=None, flatten=True):
        super().__init__()
        img_size = to_2tuple(img_size)
        patch_size = to_2tuple(patch_size)
        self.img_size = img_size
        self.patch_size = patch_size
        self.grid_size = ((img_size[0] - patch_size[0]) // stride + 1, (img_size[1] - patch_size[1]) // stride + 1)
        self.num_patches = self.grid_size[0] * self.grid_size[1]
        self.flatten = flatten

        self.proj = nn.Conv2d(in_chans, embed_dim, kernel_size=patch_size, stride=stride)
        self.norm = norm_layer(embed_dim) if norm_layer else nn.Identity()

    def forward(self, x):
        B, C, H, W = x.shape
        assert H == self.img_size[0] and W == self.img_size[1], \
            f"Input image size ({H}*{W}) doesn't match model ({self.img_size[0]}*{self.img_size[1]})."
        x = self.proj(x)
        if self.flatten:
            x = x.flatten(2).transpose(1, 2)  # BCHW -> BNC
        x = self.norm(x)
        return x
```

Method

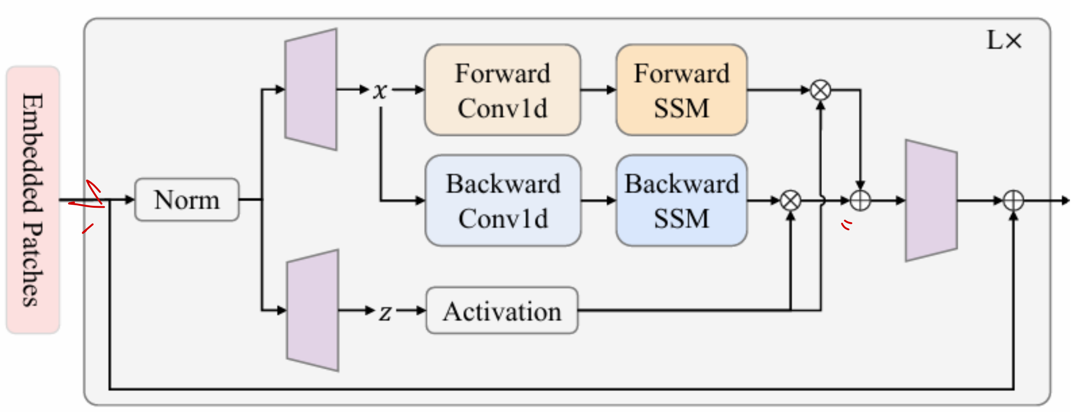
2. Vision Mamba

$$\mathbf{T}_l = \mathbf{Vim}(\mathbf{T}_{1-1}) + \mathbf{T}_{1-1},$$
$$\mathbf{f} = \mathbf{Norm}(\mathbf{T}_{\mathrm{L}}^{0}),$$
$$\hat{p} = \mathbf{MLP}(\mathbf{f}),$$

- 토큰 시퀀스 T_(l-1)를 Vim 인코더의 l번째 레이어로 보내고, 출력 T_t를 얻음(residual connection)
- 최종 출력 클래스 토큰을 정규화하고, MLP 헤드에 공급하여 최종 예측을 얻는다.

Method

3. Vim Block



Vision Mamba Encoder

Algorithm 1 Vim Block Process

Require: token sequence $\mathbf{T}_{l-1} : (\mathtt{B}, \mathtt{M}, \mathtt{D})$

Ensure: token sequence $\mathbf{T}_{l} : (\mathtt{B}, \mathtt{M}, \mathtt{D})$

/* normalize the input sequence $\mathbf{T}'_{l-1}$ */

$\mathbf{T}'_{l-1} : (\mathtt{B}, \mathtt{M}, \mathtt{D}) \leftarrow \mathbf{Norm}(\mathbf{T}_{l-1})$

$\mathbf{x} : (\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{Linear}^{\mathbf{x}}(\mathbf{T}'_{l-1})$

$\mathbf{z} : (\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{Linear}^{\mathbf{z}}(\mathbf{T}'_{l-1})$

/* process with different direction */

for o in {forward, backward} **do**

$\quad \mathbf{x}'_o : (\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{SiLU}(\mathbf{Conv1d}_o(\mathbf{x}))$

$\quad \mathbf{B}_o : (\mathtt{B}, \mathtt{M}, \mathtt{N}) \leftarrow \mathbf{Linear}^{\mathbf{B}}_o(\mathbf{x}'_o)$

$\quad \mathbf{C}_o : (\mathtt{B}, \mathtt{M}, \mathtt{N}) \leftarrow \mathbf{Linear}^{\mathbf{C}}_o(\mathbf{x}'_o)$

$\quad$ /* softplus ensures positive $\mathbf{\Delta}_o$ */

$\quad \mathbf{\Delta}_o : (\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \log(1 + \exp(\mathbf{Linear}^{\mathbf{\Delta}}_o(\mathbf{x}'_o) + \mathbf{Parameter}^{\mathbf{\Delta}}_o))$

$\quad$ /* shape of $\mathbf{Parameter}^{\mathbf{A}}_o$ is $(\mathtt{E}, \mathtt{N})$ */

$\quad \overline{\mathbf{A}}_o : (\mathtt{B}, \mathtt{M}, \mathtt{E}, \mathtt{N}) \leftarrow \mathbf{\Delta}_o \bigotimes \mathbf{Parameter}^{\mathbf{A}}_o$

$\quad \overline{\mathbf{B}}_o : (\mathtt{B}, \mathtt{M}, \mathtt{E}, \mathtt{N}) \leftarrow \mathbf{\Delta}_o \bigotimes \mathbf{B}_o$

$\quad \mathbf{y}_o : (\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{SSM}(\overline{\mathbf{A}}_o, \overline{\mathbf{B}}_o, \mathbf{C}_o)(\mathbf{x}'_o)$

end for

/* get gated $\mathbf{y}_o$ */

$\mathbf{y}'_{forward} : (\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{y}_{forward} \bigodot \mathbf{SiLU}(\mathbf{z})$

$\mathbf{y}'_{backward} : (\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{y}_{backward} \bigodot \mathbf{SiLU}(\mathbf{z})$

/* residual connection */

$\mathbf{T}_{l} : (\mathtt{B}, \mathtt{M}, \mathtt{D}) \leftarrow \mathbf{Linear}^{\mathbf{T}}(\mathbf{y}'_{forward} + \mathbf{y}'_{backward}) + \mathbf{T}_{l-1}$

Return: $\mathbf{T}_{l}$

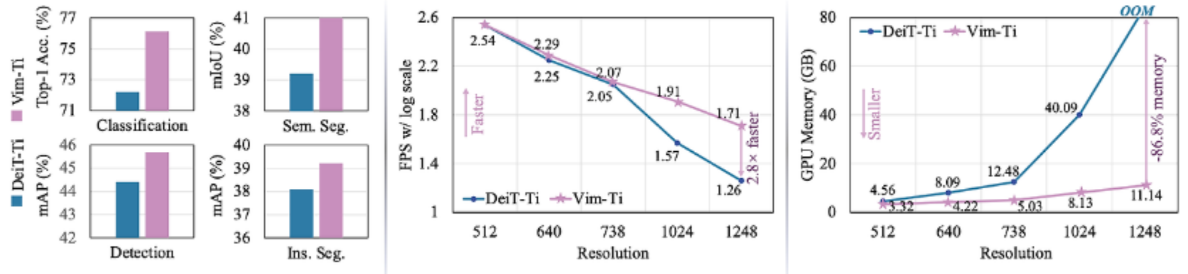
Experiments

1. Classification

- ImageNet 1K, input size=224,
(랜덤 크롭, 랜덤 플립, 라벨 스무딩, 믹스업),AdamW, cosine annealing,batch size=1024, epochs=300, lr=1e-03

추가로 long sequence fine-tuning 진행.
기존의 패치 사이즈 (224/4 = 56) 에서 8으로 대폭 줄여서 훨씬 긴 시퀀스를 만들고, 30 epoch 만큼 학습

Method	image size	#param.	ImageNet top-1 acc.
Convnets			
ResNet-18	224^2	12M	69.8
ResNet-50	224^2	25M	76.2
ResNet-101	224^2	45M	77.4
ResNet-152	224^2	60M	78.3
ResNeXt50-32×4d	224^2	25M	77.6
RegNetY-4GF	224^2	21M	80.0
Transformers			
ViT-B/16	384^2	86M	77.9
ViT-L/16	384^2	307M	76.5
DeiT-Ti	224^2	6M	72.2
DeiT-S	224^2	22M	79.8
DeiT-B	224^2	86M	81.8
SSMs			
S4ND-ViT-B	224^2	89M	80.4
Vim-Ti	224^2	7M	76.1
Vim-Ti†	224^2	7M	78.3 +2.2
Vim-S	224^2	26M	80.5
Vim-S†	224^2	26M	81.6 +1.1



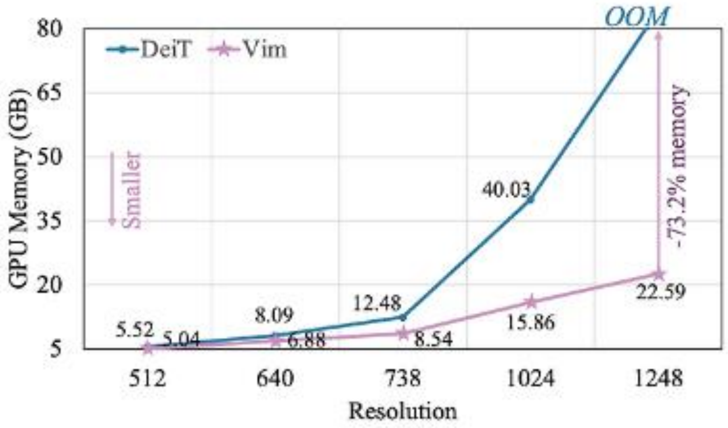
(a) Accuracy Comparison (b) Speed Comparison (c) GPU Memory Comparison

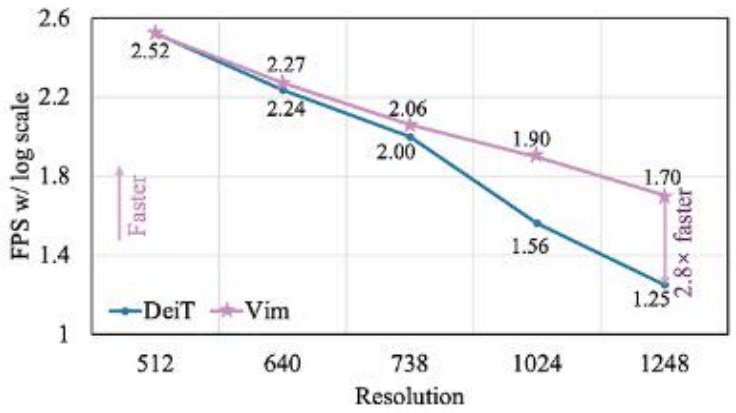
Experiments

2. Sementic segmentation

ADE20K, UperNet framework

Method	Backbone	image size	#param.	*val* mIoU
DeepLab v3+	ResNet-101	512^2	63M	44.1
UperNet	ResNet-50	512^2	67M	41.2
UperNet	ResNet-101	512^2	86M	44.9
UperNet	DeiT-Ti	512^2	11M	39.2
UperNet	DeiT-S	512^2	43M	44.0
UperNet	Vim-Ti	512^2	13M	41.0
UperNet	Vim-S	512^2	46M	44.9





Experiments

3. Object detection, Instance segmentation

COCO 2017, ViTDet framework

Backbone	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{box}_{s}	AP^{box}_{m}	AP^{box}_{l}
DeiT-Ti	44.4	63.0	47.8	26.1	47.4	61.8
Vim-Ti	45.7	63.9	49.6	26.1	49.0	63.2
Backbone	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}	AP^{mask}_{s}	AP^{mask}_{m}	AP^{mask}_{l}
DeiT-Ti	38.1	59.9	40.5	18.1	40.5	58.4
Vim-Ti	39.2	60.9	41.7	18.2	41.8	60.2

TRAIN AND TEST