Vision Mamba

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

Lianghui Zhu[1*], Bencheng Liao[1*], Qian Zhang[2], Xinlong Wang[3], Wenyu Liu[1], Xinggang Wang[1] ✉
[1] Huazhong University of Science and Technology
[2] Horizon Robotics [3] Beijing Academy of Artificial Intelligence
Code & Models: hustvl/Vim

Motivation

- Mamba DL models - potential in long sequence modeling

Challenge:

- position sensitivity of visual data and global context

Previous models in Vision using SSM combined SSM with convolution or attention (otherwise task-specific)

Vision Mamba is pure SSM-based model.

Mamba process

$$\begin{aligned} h'(t) &= \mathbf{A}h(t) + \mathbf{B}x(t), \\ y(t) &= \mathbf{C}h(t). \end{aligned} \tag{1}$$

SSM-based models

$$\begin{aligned} \overline{\mathbf{A}} &= \exp(\mathbf{\Delta A}), \\ \overline{\mathbf{B}} &= (\mathbf{\Delta A})^{-1}(\exp(\mathbf{\Delta A}) - \mathbf{I}) \cdot \mathbf{\Delta B}. \end{aligned} \tag{2}$$

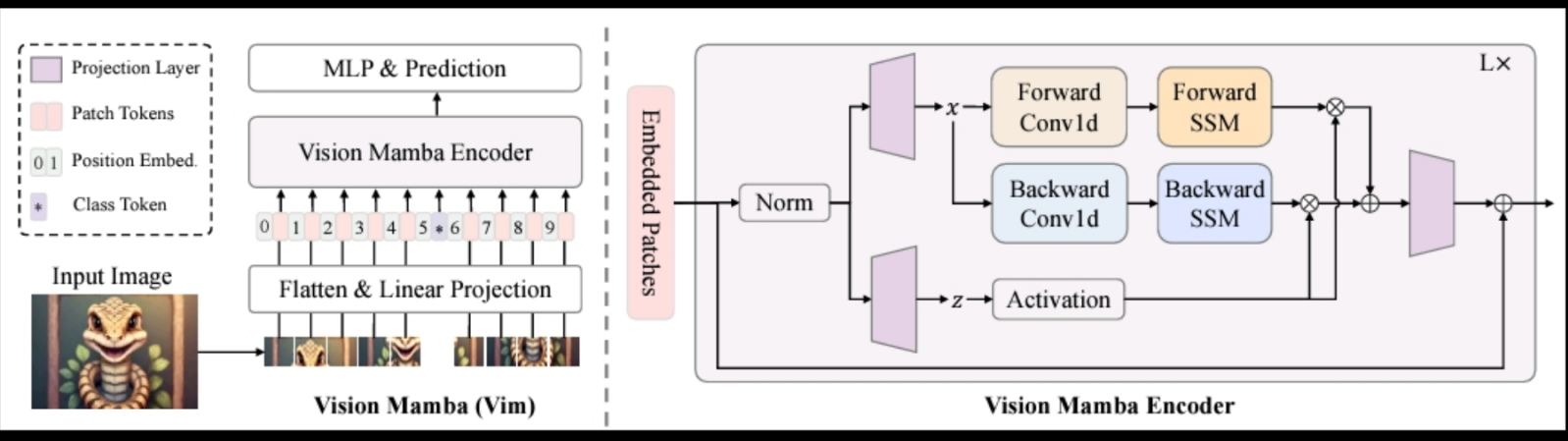
Mamba's discretization

$$\begin{aligned} h_t &= \overline{\mathbf{A}}h_{t-1} + \overline{\mathbf{B}}x_t, \\ y_t &= \mathbf{C}h_t. \end{aligned} \tag{3}$$

evolution.

$$\overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{M-1}\overline{\mathbf{B}}), \tag{4}$$

Vision Mamba



Class token – inspired by ViT and BERT

$$
\begin{aligned}
\mathbf{T}_l &= \mathbf{Vim}(\mathbf{T}_{l-1}) + \mathbf{T}_{l-1}, \\
\mathbf{f} &= \mathbf{Norm}(\mathbf{T}_{\mathrm{L}}^{0}), \\
\hat{p} &= \mathbf{MLP}(\mathbf{f}),
\end{aligned} \tag{6}
$$

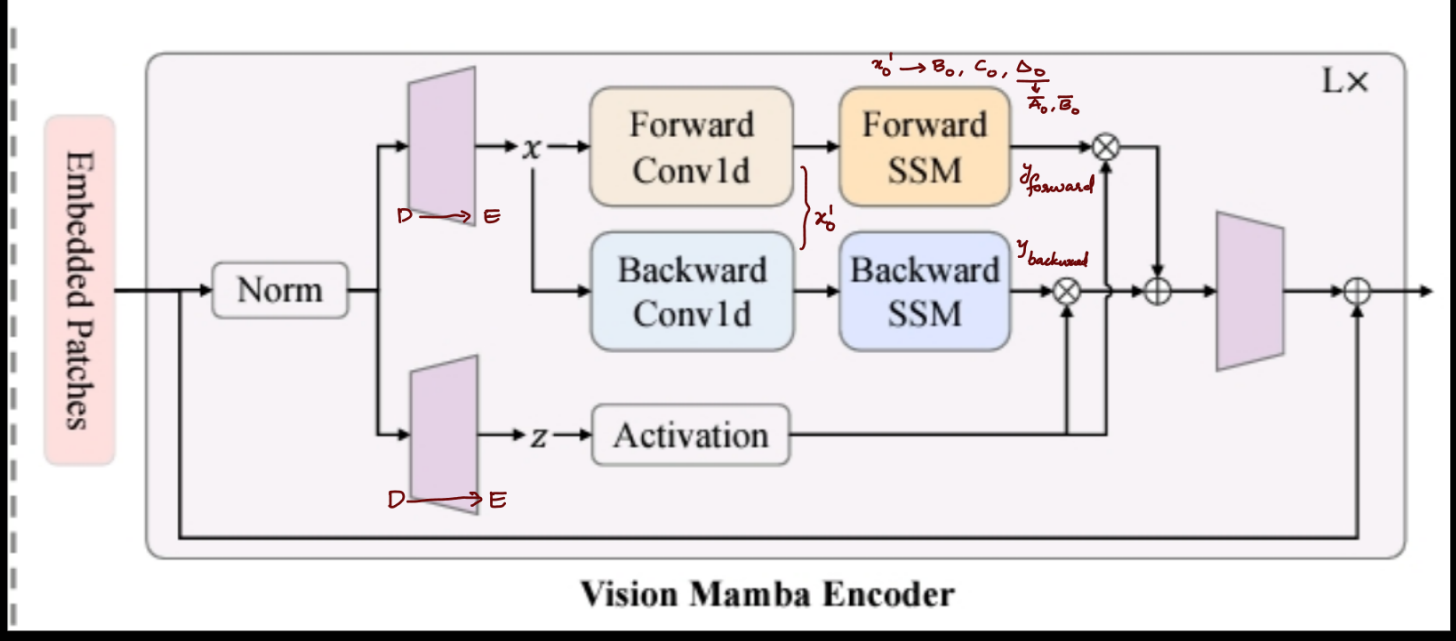
Vim – proposed block

T_L^0 – class token of output

Norm – Normalization.

Vim Block

→ incorporates bidirectional sequence modelling for vision.



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**
  $\mathbf{x}'_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{SiLU}(\mathbf{Conv1d}_o(\mathbf{x}))$
  $\mathbf{B}_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{N}) \leftarrow \mathbf{Linear}^{\mathbf{B}}_o(\mathbf{x}'_o)$
  $\mathbf{C}_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{N}) \leftarrow \mathbf{Linear}^{\mathbf{C}}_o(\mathbf{x}'_o)$
  /* softplus ensures positive $\mathbf{\Delta}_o$ */
  $\mathbf{\Delta}_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \log(1 + \exp(\mathbf{Linear}^{\mathbf{\Delta}}_o(\mathbf{x}'_o) + \mathbf{Parameter}^{\mathbf{\Delta}}_o))$
  /* shape of $\mathbf{Parameter}^{\mathbf{A}}_o$ is $(\mathtt{E}, \mathtt{N})$ */
  $\overline{\mathbf{A}}_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}, \mathtt{N}) \leftarrow \mathbf{\Delta}_o \bigotimes \mathbf{Parameter}^{\mathbf{A}}_o$
  $\overline{\mathbf{B}}_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}, \mathtt{N}) \leftarrow \mathbf{\Delta}_o \bigotimes \mathbf{B}_o$
  $\mathbf{y}_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{SSM}(\overline{\mathbf{A}}_o, \overline{\mathbf{B}}_o, \mathbf{C}_o)(\mathbf{x}'_o)$
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: $\mathbf{T}_{l}$

L: the number of blocks,
D: the hidden state dimension,
E: expanded state dimension,
N: SSM dimension.

Following ViT [13] and DeiT [60], we first employ 16×16 kernel size projection layer to get a 1-D sequence of non-overlapping patch embeddings. Subsequently, we directly stack L Vim blocks. By default, we set the number of blocks L to 24, SSM dimension N to 16. To align with the model sizes of DeiT series, we set the hidden state dimension D to 192 and expanded state dimension E to 384 for the tiny-size variant. For the small-size variant, we set D to 384 and E to 768.

Vision Mamba adopts the efficient use of modern hardware from Mamba.

Without this, Vim's SSM operation,

$$O(BMEN)$$

Now,

process

Slow HBM (large memory size) $\xrightarrow{\Delta_o, A_o, B_o, C_o}$ fast SRAM (high bandwidth)

then

$\Delta_o, A_o, B_o \xrightarrow{\text{discretize}} \overline{A}_o, \overline{B}_o$

(B, M, E, N)

$\overline{A}_o, \overline{B}_o, C_o, x'_o \xrightarrow{\text{Vim}} y_o$

(B, M, E)

reduces IOs $O(BMEN)$ to $O(BME + EN)$

→ uses same memory methods as in Mamba

→ for intermediate states $(B, M, E, N$ size)

Vim recomputes them at network backward pass.

→ intermediate activations & convolution are recomputed.

(activation values memory intensive) - fast for recomputation

Computationally, $(E = 2D)$

M = token length, D = token dimension

$$\Omega(\text{self-attention}) = 4MD^2 + 2M^2D, \quad (7)$$ → quadratic to sequence length

$$\Omega(\text{SSM}) = 3M(2D)N + M(2D)N, \quad (8)$$ → linear to sequence length

Experiments

4.1. Image Classification

Settings. We benchmark Vim on the ImageNet-1K dataset [9], which contains 1.28M training images and 50K validation images from 1,000 categories. All models are trained on the training set, and top-1 accuracy on the validation set is reported. For fair comparisons, our training settings mainly follow DeiT [60]. Specifically, we apply random cropping, random horizontal flipping, label-smoothing regularization, mixup, and random erasing as data augmentations. When training on 224^2 input images, we employ AdamW [43] with a momentum of 0.9, a total batch size of 1024, and a weight decay of 0.05 to optimize models. We train the Vim models for 300 epochs using a cosine schedule, 1×10^{-3} initial learning rate, and EMA. During testing, we apply a center crop on the validation set to crop out 224^2 images. Experiments are performed on 8 A800 GPUs.

Long Sequence Fine-tuning To make full use of the efficient long sequence modeling power of Vim, we continue to fine-tune Vim with a long sequence setting for 30 epochs after ImageNet pretraining. Specifically, we set a patch extraction stride of 8 while keeping the patch size unchanged, a constant learning rate of 10^{-5}, and a weight decay of 10^{-8}.

Settings for Semantic Segmentation. We conduct experiments for semantic segmentation on the ADE20K [73] dataset. ADE20K contains 150 fine-grained semantic categories, with 20K, 2K, and 3K images for training, validation, and testing, respectively. We choose UperNet [69] as our base framework. In training, we employ AdamW with a weight decay of 0.01, and a total batch size of 16 to optimize models. The employed training schedule uses an initial learning rate of 6×10^{-5}, linear learning rate decay, a linear warmup of 1,500 iterations, and a total training of 160K iterations. The data augmentations follow common settings, including random horizontal flipping, random re-scaling within the ratio range [0.5, 2.0], and random photometric distortion. During evaluation, we rescale the image to have a shorter side of 512.

Settings for Object Detection and Instance Segmentation. We conduct experiments for object detection and instance segmentation on the COCO 2017 dataset [38]. The COCO 2017 dataset contains 118K images for training, 5K images for validating, and 20K images for testing. We use the canonical Cascade Mask R-CNN [4] as the base framework. For ViT-based backbones, we apply extra configurations (*e.g.*, interleaved window & global attention) to handle the high-resolution images following ViTDet [37]. For SSM-based Vim, we directly use it without any modifications. Other training and evaluation settings are just the same. During training, we employ AdamW with a weight decay of 0.1, and a total batch size of 64 to optimize models. The employed training schedule uses an initial learning rate of 1×10^{-4}, linear learning rate decay, and a total training of 380K iterations. The data augmentations use large-scale jitter data augmentation [18] to 1024×1024 input images. During evaluation, we rescale the image to have a shorter side of 1024.

Results

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

Classification

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

Object detection & image segmentation.

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

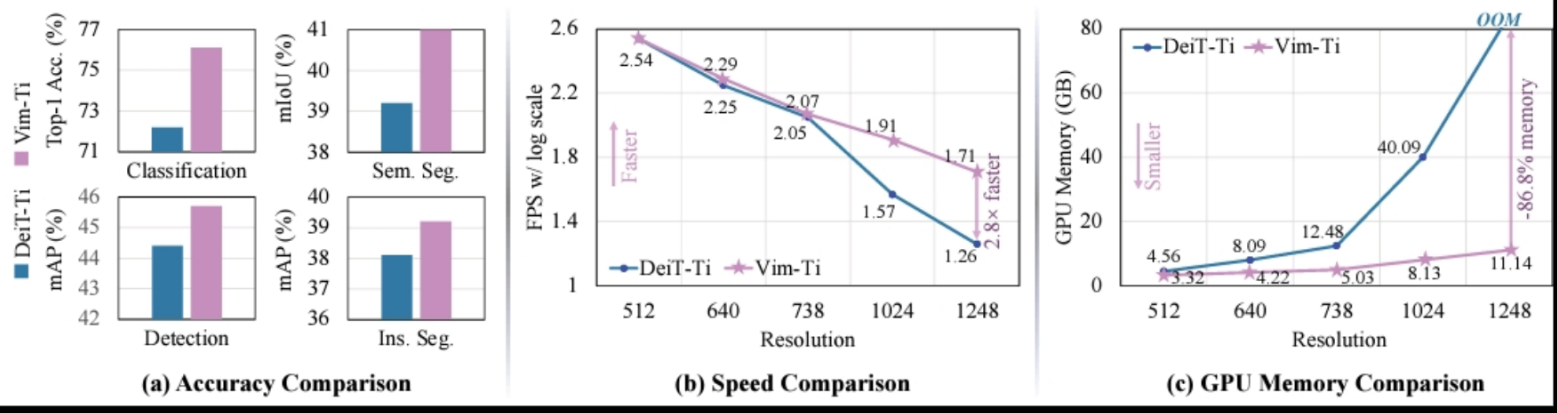
Semantic segmentation.

† = long sequence fine tuning

DeiT -

[60] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021. 2, 4, 5, 6, 8, 11

Exp Moving. Average



combine backbone with commonly used FPN (feature Pyramid) also gave comparable. results. &

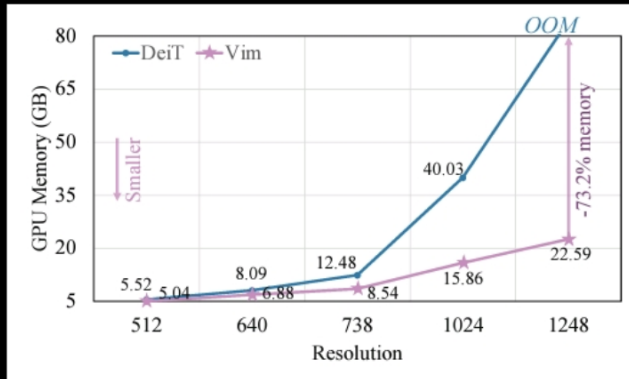


Figure 3. GPU memory efficiency comparison between DeiT-Ti [59] and our Vim-Ti on the commonly used downstream framework. We perform batch inference and benchmark the GPU memory on the architecture with the backbone and FPN. Vim requires comparable GPU memory to DeiT with a small resolution, *i.e.*, 512×512. As the input image resolution increases, Vim will use significantly less GPU memory.

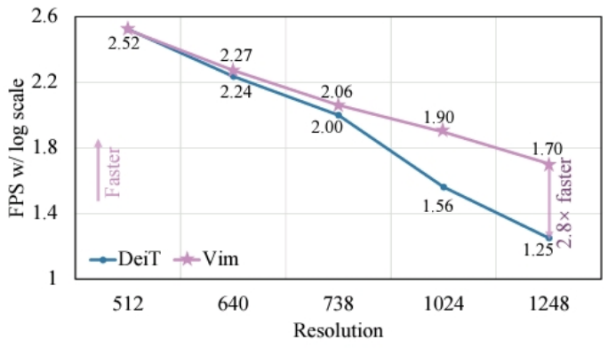


Figure 4. FPS comparison between DeiT-Ti [59] and our Vim-Ti on the commonly used downstream framework. We perform batch inference and benchmark the log-scaled FPS on the architecture with the backbone and FPN. Vim achieves comparable performance to DeiT with a small resolution, *i.e.*, 512×512. As the input image resolution increases, Vim has a higher FPS.

We highlight that the accuracy superiority is non-trivial since DeiT is equipped with window attention while Vim works in a pure sequence modeling manner. Specifically, to perform representation learning on high-resolution images (*i.e.*, 1024×1024), we follow ViTDet [37] and modify the DeiT backbone with the use of 2D window attention, which injects 2D prior and breaks the sequential modeling nature of Transformer. Thanks to the efficiency illustrated in Sec. 3.5, Fig. 1 and Fig. 3, we can directly apply Vim on 1024×1024 input images and learn sequential visual representation for object detection and instance segmentation without need for 2D priors in the backbone.

[37] Yanghao Li, Hanzi Mao, Ross Girshick, and Kaiming He. Exploring plain vision transformer backbones for object detection. In *ECCV*, 2022. 2, 7, 11

Study on Bidirection layer.

Bidirectional strategy	ImageNet top-1 acc.	ADE20K mIoU
None	73.2	32.3
Bidirectional Layer	70.9	33.6
Bidirectional SSM	72.8	33.2
Bidirectional SSM + Conv1d	73.9	35.9

Table 4. Ablation study on the bidirectional design. To ensu fair comparison, we do not use the class token for each experime The default setting for Vim is marked in blue.

- None. We directly adopt the Mamba block to process visual sequence with only the forward direction.
- Bidirectional Sequence. During training, we randomly flip the visual sequence. This works like data augmentation.
- Bidirectional Block. We pair the stacked blocks. The first block of each pair processes visual sequence in the forward direction and the second block of each pair processes in the backward direction.
- Bidirectional SSM. We add an extra SSM for each block to process the visual sequence in the backward direction.
- Bidirectional SSM + Conv1d. Based on Bidirectional SSM, we further add a backward Conv1d before the backward SSM (Fig. 2).

Study on Classification design.

Classification strategy	ImageNet top-1 acc.
Mean pool	73.9
Max pool	73.4
Head class token	75.2
Double class token	74.3
Middle class token	76.1

Table 5. Ablation study on the classification design. The defau setting for Vim is marked in blue.

Classification Design. We ablate the classification design of Vim, benchmarking on ImageNet-1K classification. We study the following classification strategies:

- Mean pool. We adopt mean pooling on the output feature from the last Vim block and perform classification on this pooled feature.
- Max pool. We first adapt the classification head on each token of the visual sequence and then perform max pooling on the sequence to get the classification prediction result.
- Head class token. Following DeiT [60], we concatenate the class token at the head of the visual sequence and perform classification.
- Double class token. Based on the head class token strategy, we additionally add a class token at the tail of the visual sequence.
- Middle class token. We add a class token at the middle of the visual sequence and then perform classification on the final middle class token.