Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

Ze Liu[1,2†*] Yutong Lin[1,3†*] Yue Cao[1*] Han Hu[1*‡] Yixuan Wei[1,4†]
Zheng Zhang[1] Stephen Lin[1] Baining Guo[1]
[1]Microsoft Research Asia [2]University of Science and Technology of China
[3]Xian Jiaotong University [4]Tsinghua University
{v-zeliu1,v-yutlin,yuecao,hanhu,v-yixwe,zhez,stevelin,bainguo}@microsoft.com

Abstract

This paper presents a new vision Transformer, called Swin Transformer, that capably serves as a general-purpose backbone for computer vision. Challenges in adapting Transformer from language to vision arise from differences between the two domains, such as large variations in the scale of visual entities and the high resolution of pixels in images compared to words in text. To address these differences, we propose a hierarchical Transformer whose representation is computed with ***S****hifted* ***win****dows. The shifted windowing scheme brings greater efficiency by limiting self-attention computation to non-overlapping local windows while also allowing for cross-window connection. This hierarchical architecture has the flexibility to model at various scales and has linear computational complexity with respect to image size. These qualities of Swin Trans-*

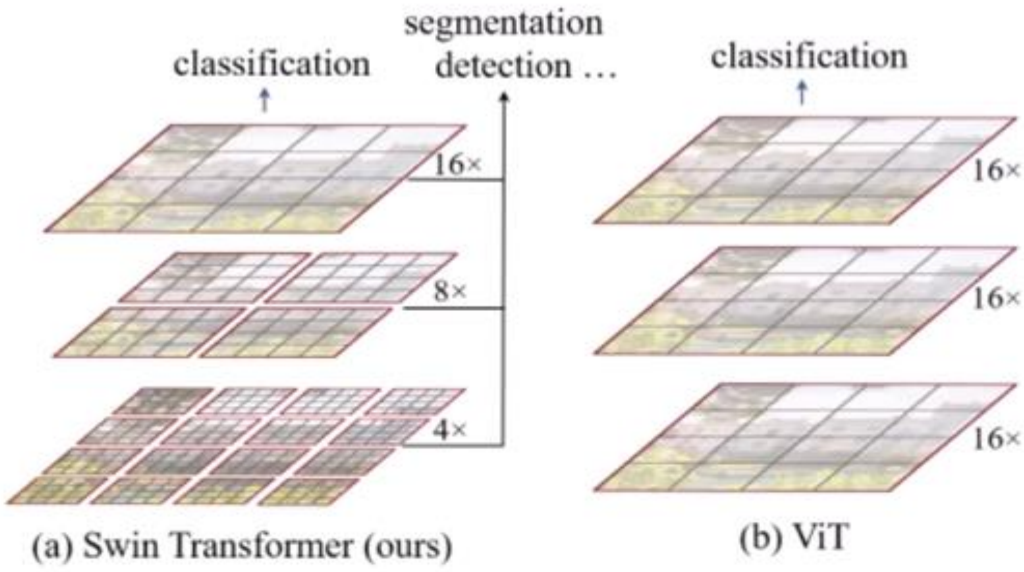
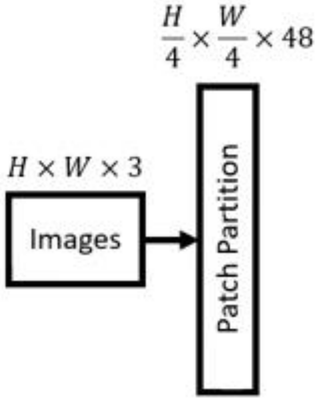
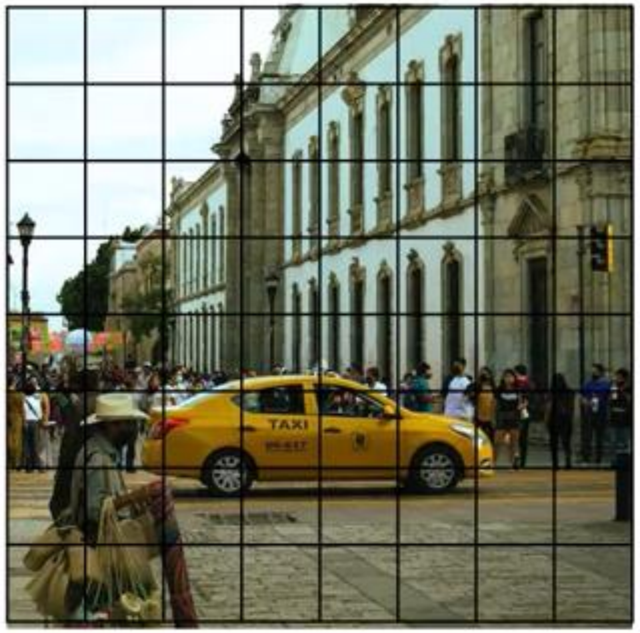
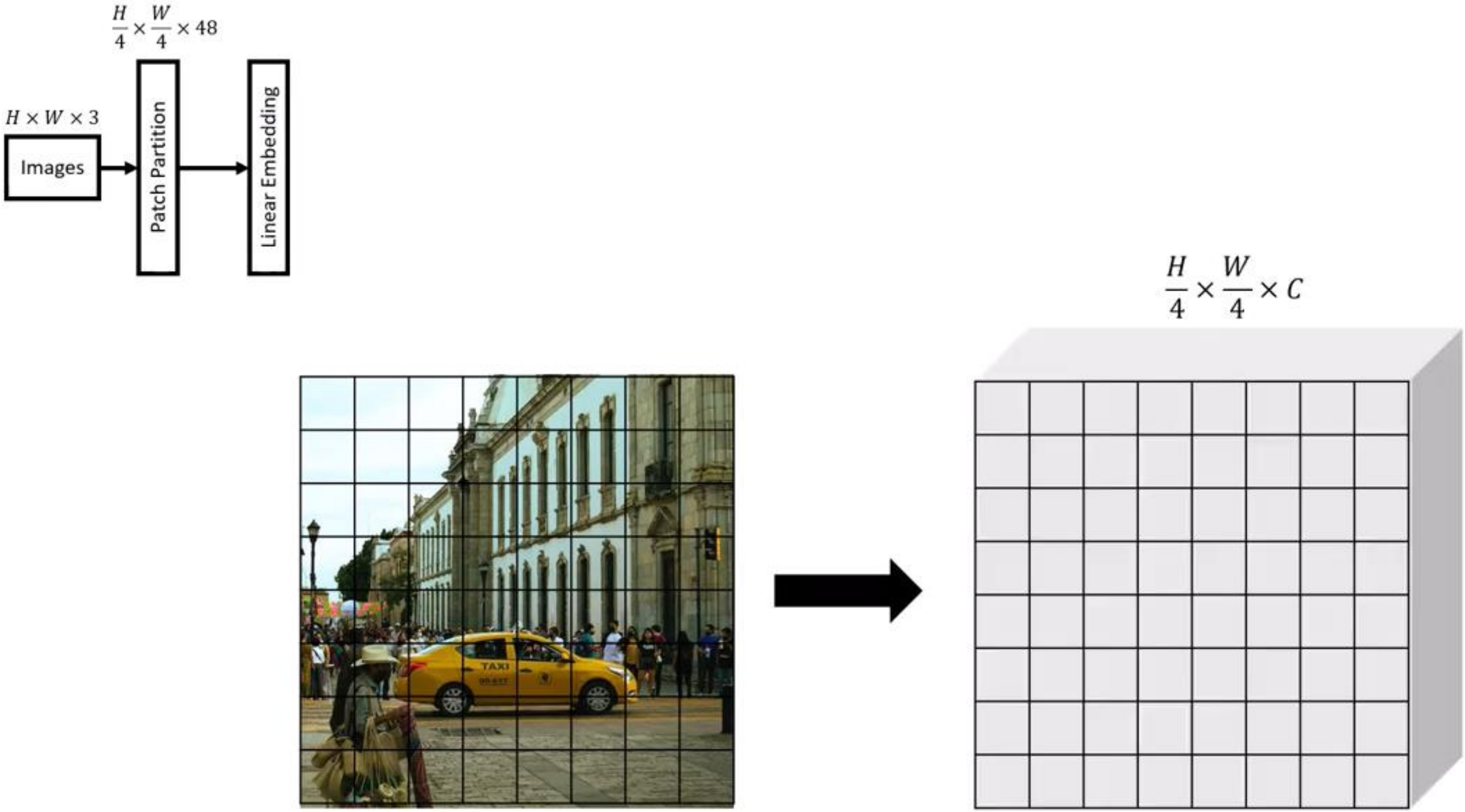
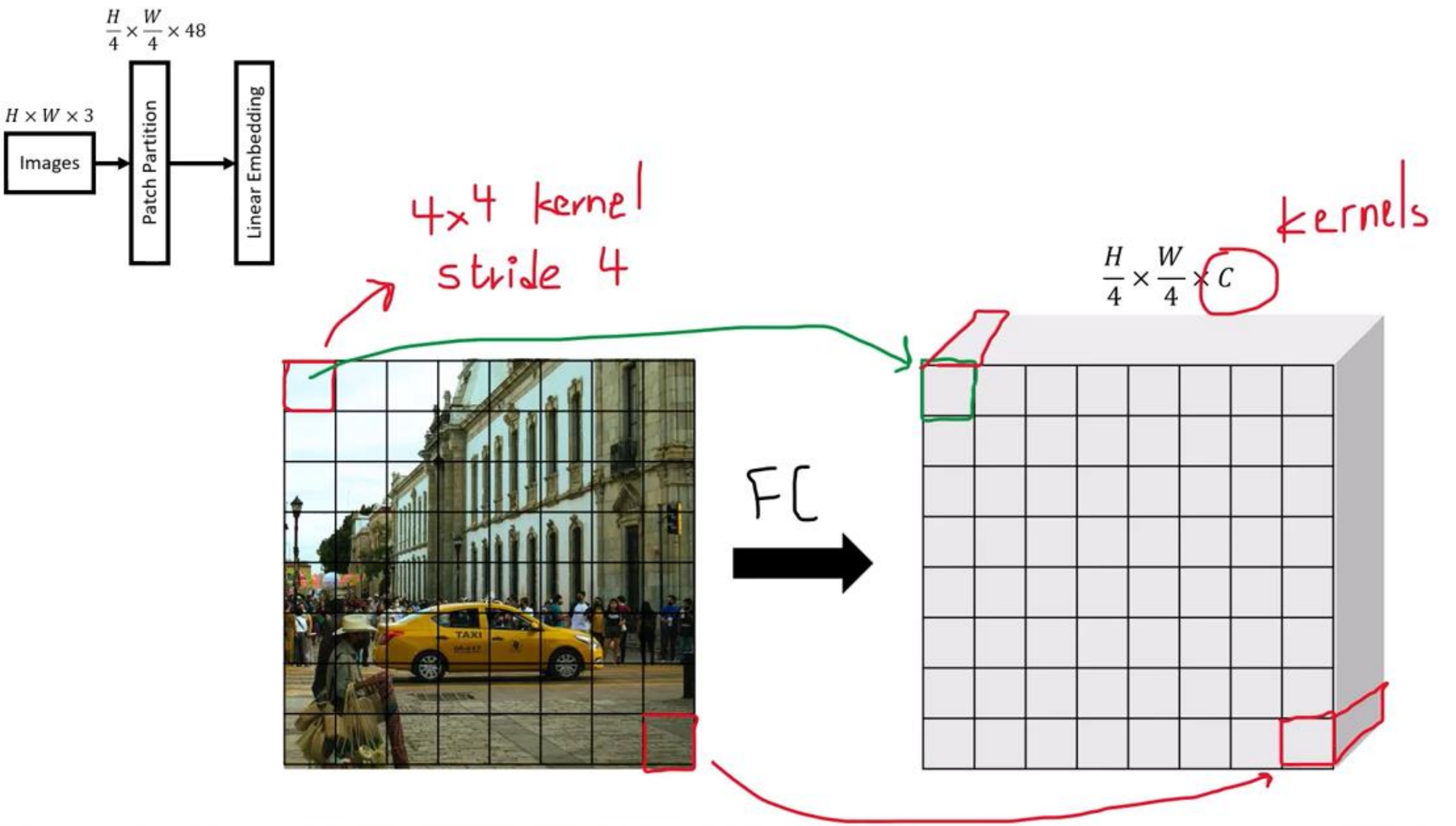


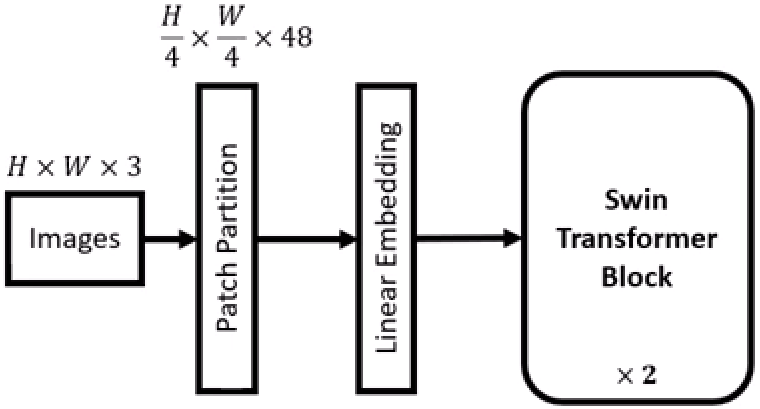
Figure 1. (a) The proposed Swin Transformer builds hierarchical feature maps by merging image patches (shown in gray) in deeper layers and has linear computation complexity to input image size due to computation of self-attention only within each local window (shown in red). It can thus serve as a general-purpose back-

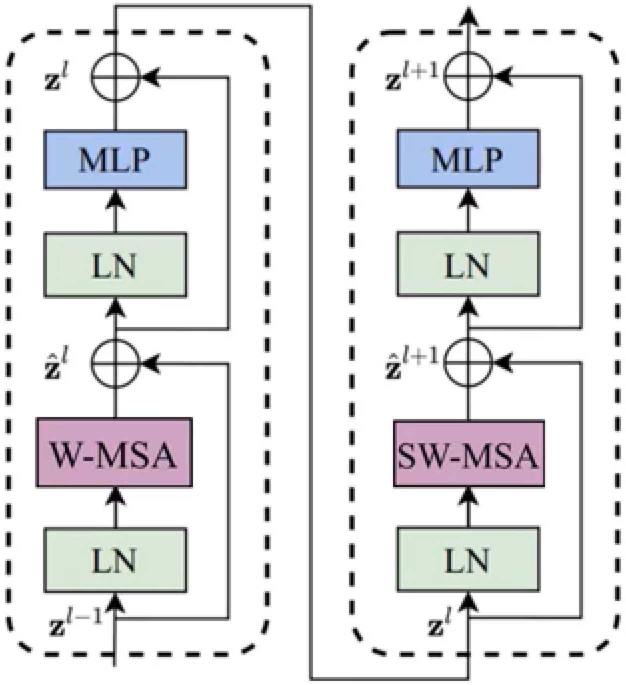

$\frac{H}{4} \times \frac{W}{4} \times 48$
$H \times W \times 3$
Images
Patch Partition


TAXI

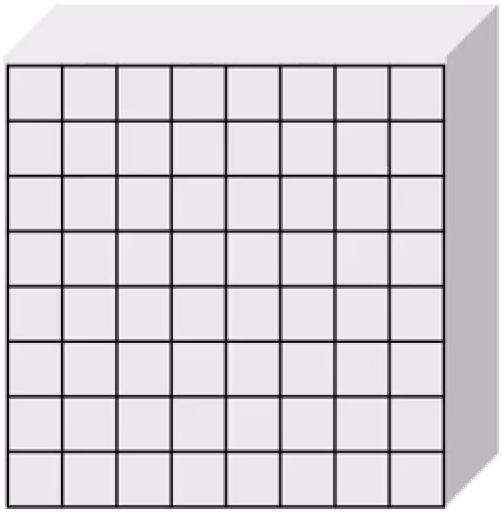

$\frac{H}{4} \times \frac{W}{4} \times 48$
$H \times W \times 3$
Images
Patch Partition
Linear Embedding
TAXI
$\frac{H}{4} \times \frac{W}{4} \times C$

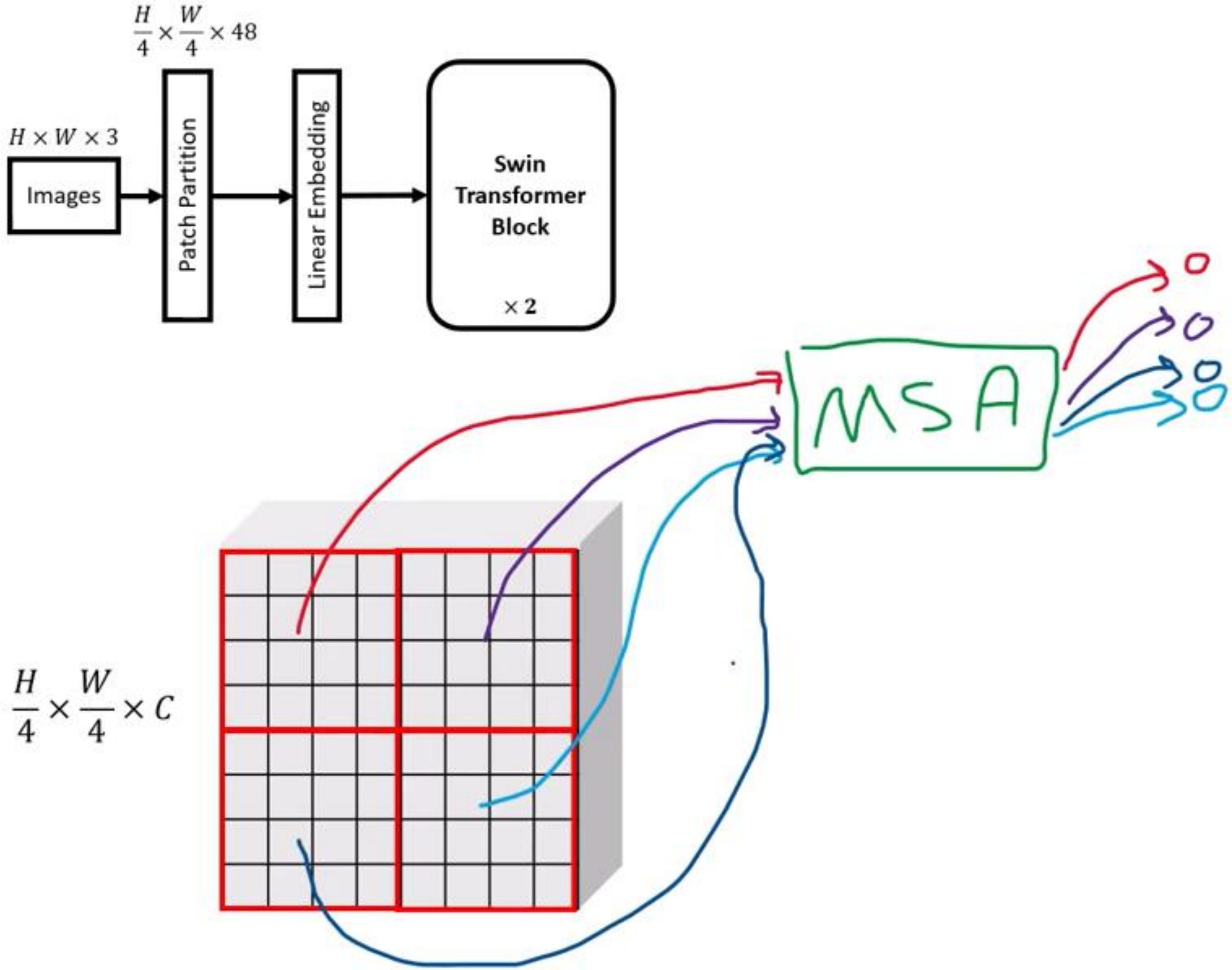

$\frac{H}{4} \times \frac{W}{4} \times 48$
$H \times W \times 3$
Images
Patch Partition
Linear Embedding
4x4 kernel
stride 4
FC
$\frac{H}{4} \times \frac{W}{4} \times C$
kernels
TAXI

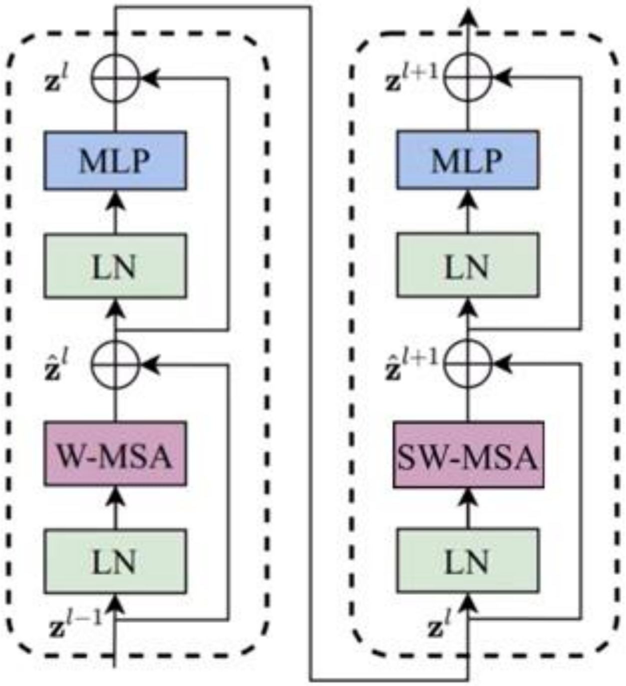




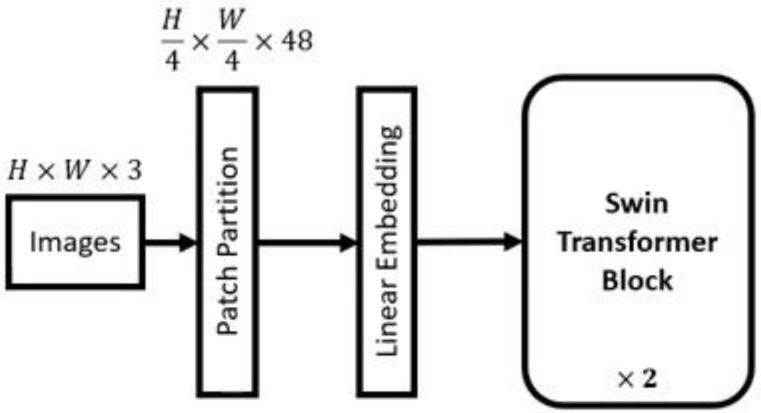
$\frac{H}{4} \times \frac{W}{4} \times C$



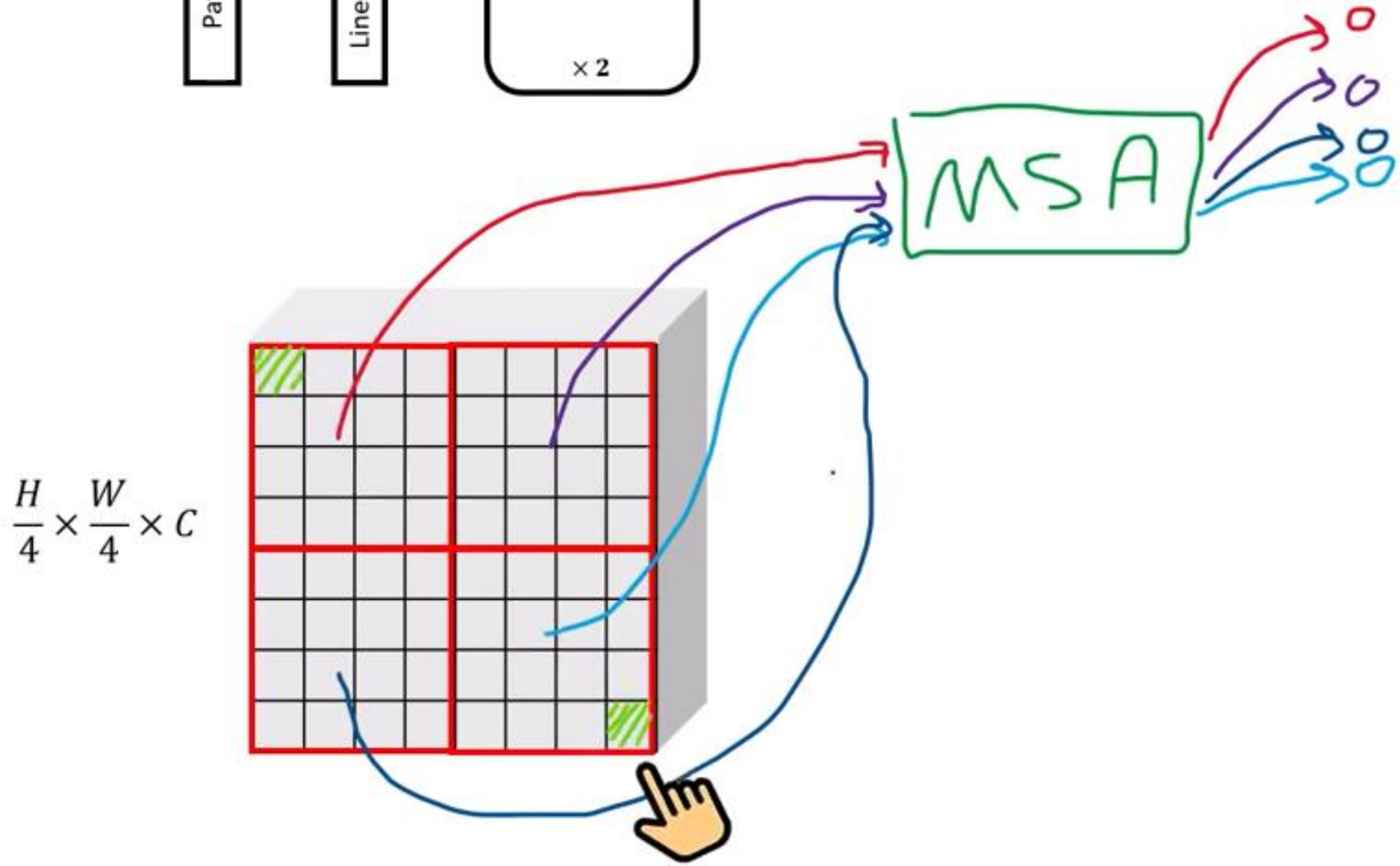

$\frac{H}{4} \times \frac{W}{4} \times 48$
$H \times W \times 3$
Images
Patch Partition
Linear Embedding
Swin Transformer Block
$\times 2$
MSA
$\frac{H}{4} \times \frac{W}{4} \times C$


$\mathbf{z}^l$
MLP
LN
$\hat{\mathbf{z}}^l$
W-MSA
LN
$\mathbf{z}^{l-1}$
$\mathbf{z}^{l+1}$
MLP
LN
$\hat{\mathbf{z}}^{l+1}$
SW-MSA
LN
$\mathbf{z}^l$

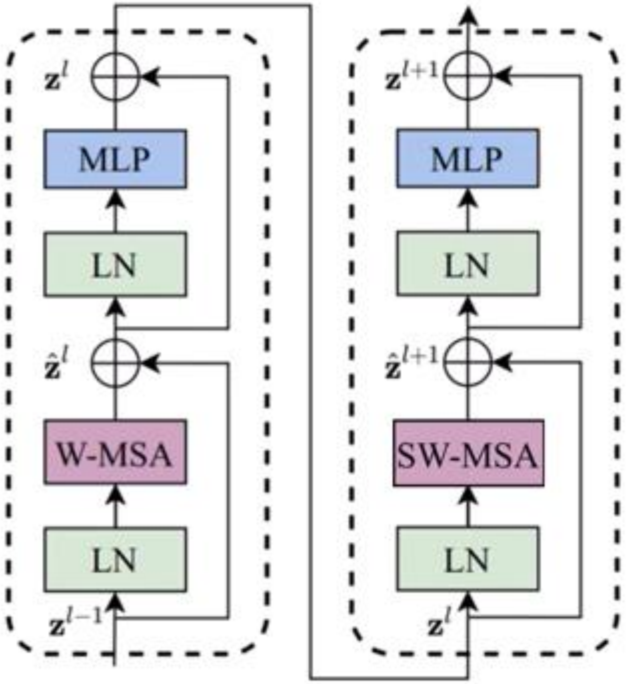
W-MSA



$\frac{H}{4} \times \frac{W}{4} \times 48$
$H \times W \times 3$
Images
Patch Partition
Linear Embedding
Swin Transformer Block
$\times$ 2

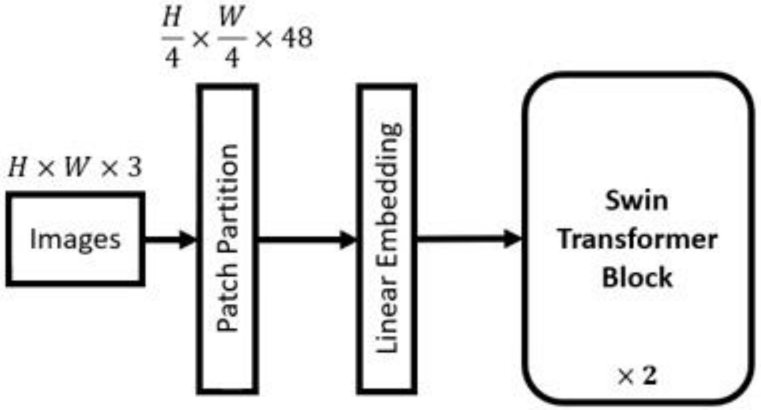


$\frac{H}{4} \times \frac{W}{4} \times C$
MSA

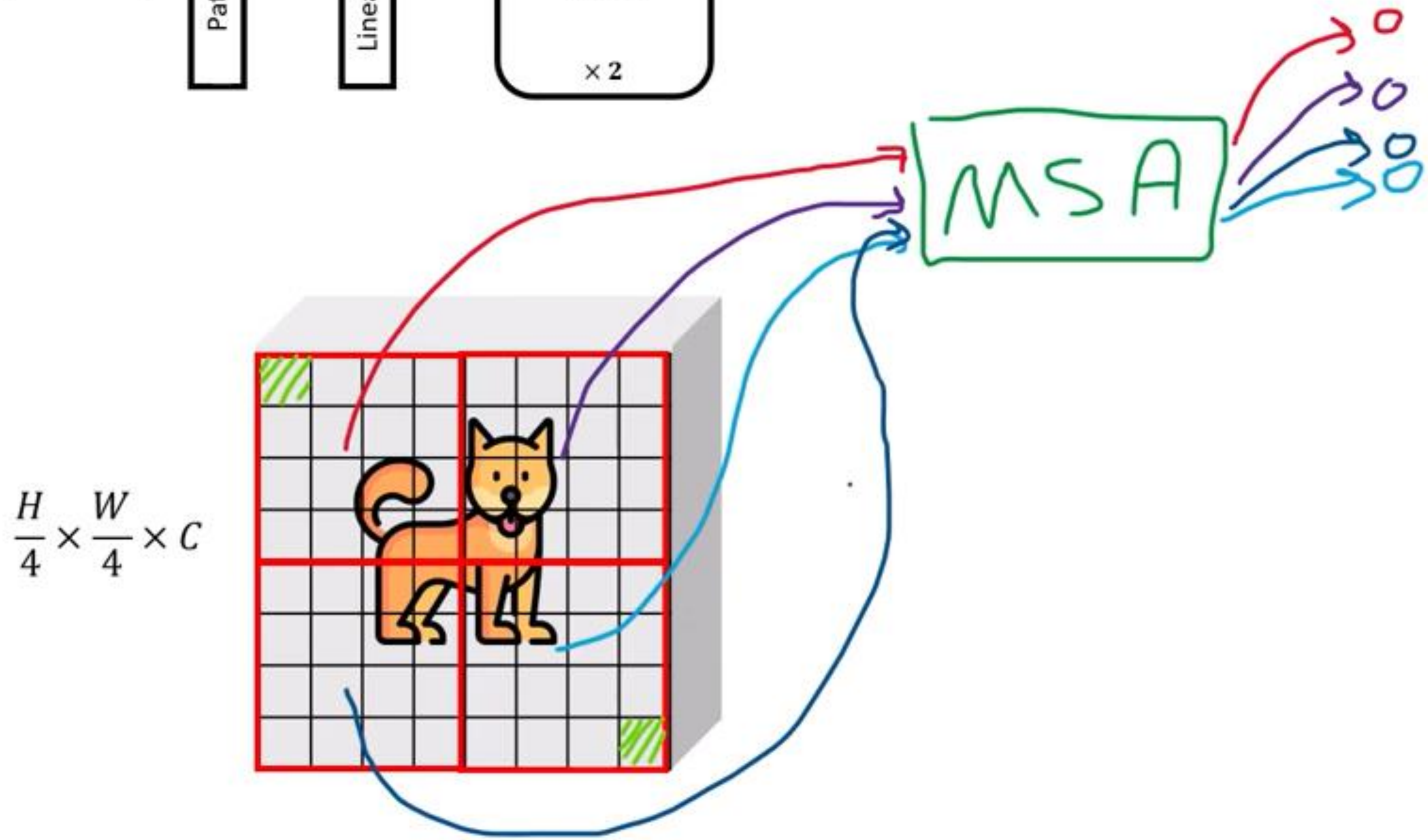


$\mathbf{z}^l$
MLP
LN
$\hat{\mathbf{z}}^l$
W-MSA
LN
$\mathbf{z}^{l-1}$
$\mathbf{z}^{l+1}$
MLP
LN
$\hat{\mathbf{z}}^{l+1}$
SW-MSA
LN
$\mathbf{z}^l$

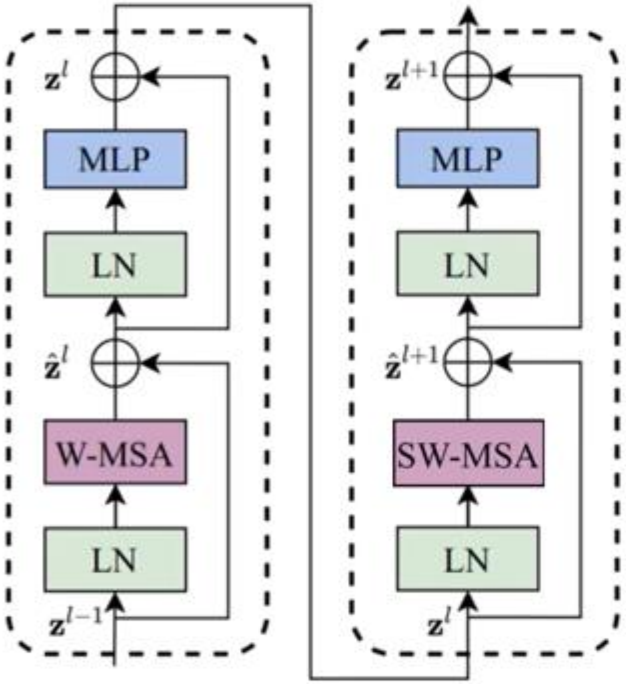
W-MSA



$\frac{H}{4} \times \frac{W}{4} \times 48$
$H \times W \times 3$
Images
Patch Partition
Linear Embedding
Swin Transformer Block
$\times 2$

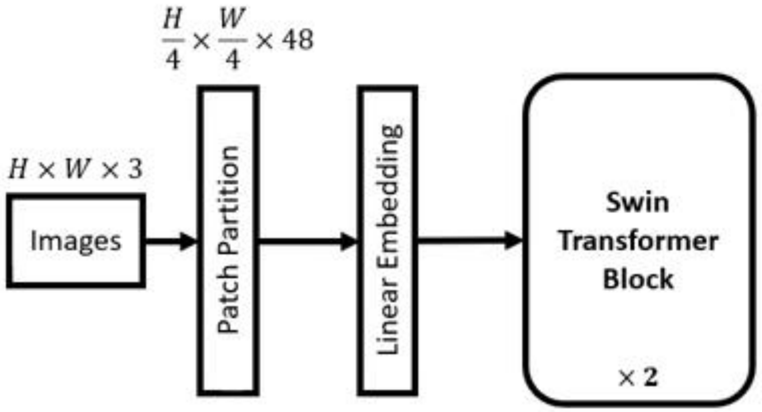


$\frac{H}{4} \times \frac{W}{4} \times C$
MSA

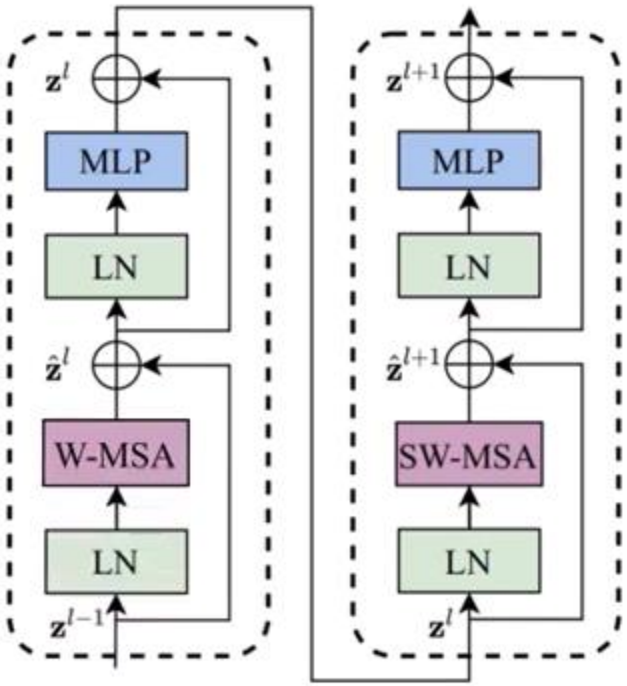


$\mathbf{z}^l$
MLP
LN
$\hat{\mathbf{z}}^l$
W-MSA
LN
$\mathbf{z}^{l-1}$
$\mathbf{z}^{l+1}$
MLP
LN
$\hat{\mathbf{z}}^{l+1}$
SW-MSA
LN
$\mathbf{z}^l$

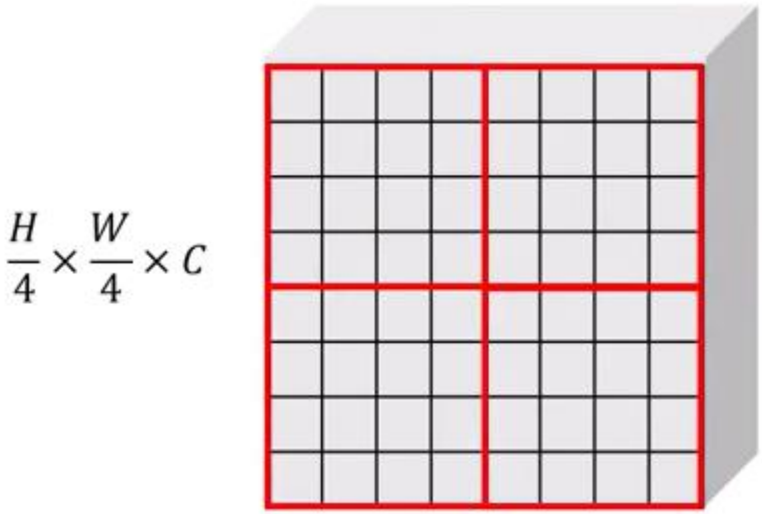
W-MSA



$\frac{H}{4} \times \frac{W}{4} \times 48$
$H \times W \times 3$
Images
Patch Partition
Linear Embedding
Swin Transformer Block
$\times 2$

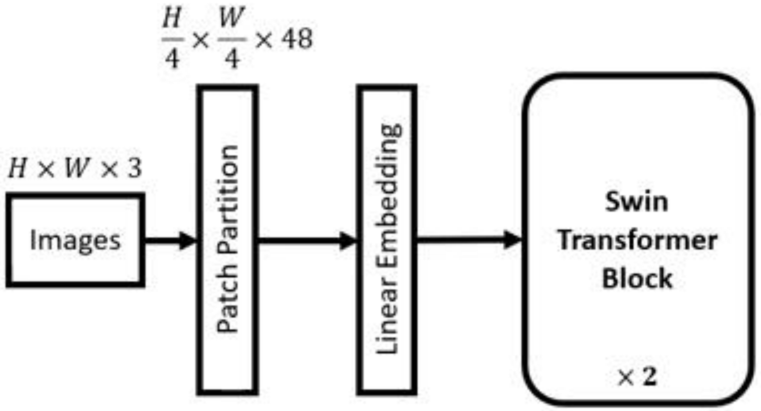


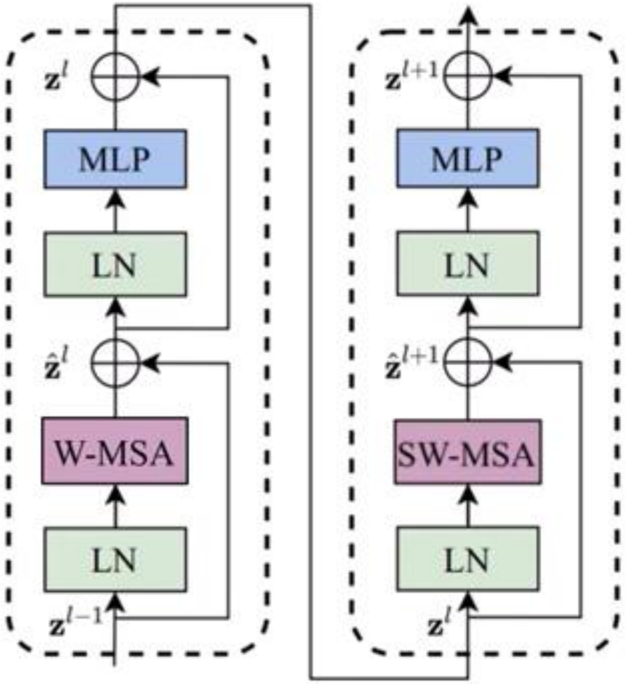
$\mathbf{z}^l$
MLP
LN
$\hat{\mathbf{z}}^l$
W-MSA
LN
$\mathbf{z}^{l-1}$
$\mathbf{z}^{l+1}$
MLP
LN
$\hat{\mathbf{z}}^{l+1}$
SW-MSA
LN
$\mathbf{z}^l$

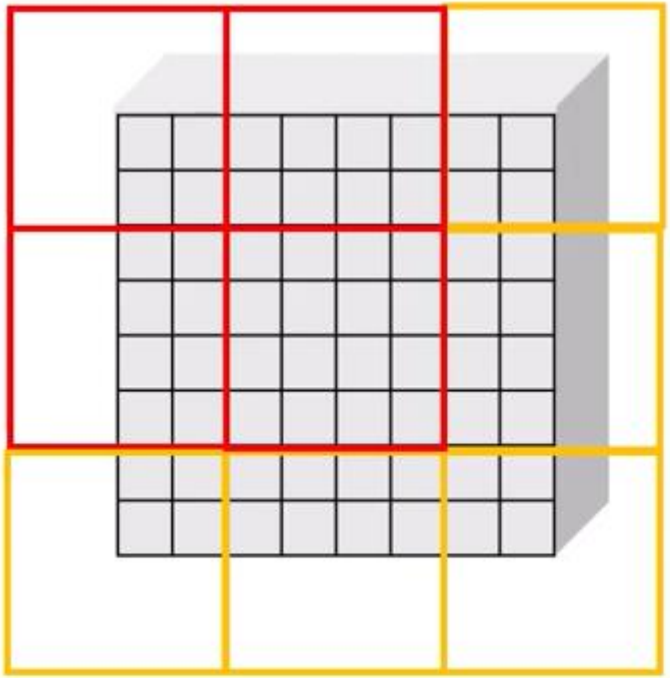


$\frac{H}{4} \times \frac{W}{4} \times C$

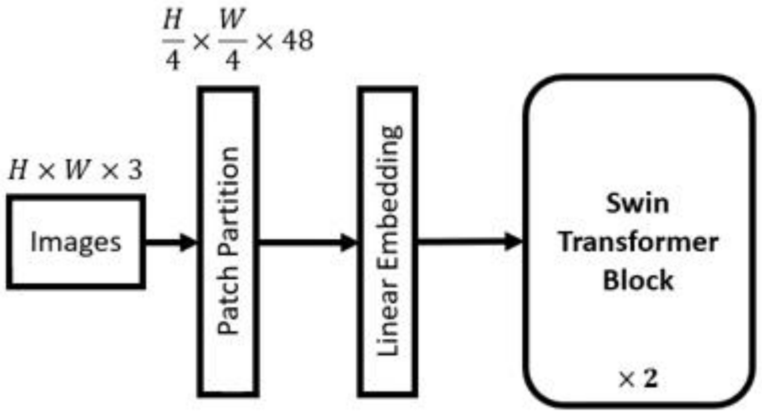
SW-MSA

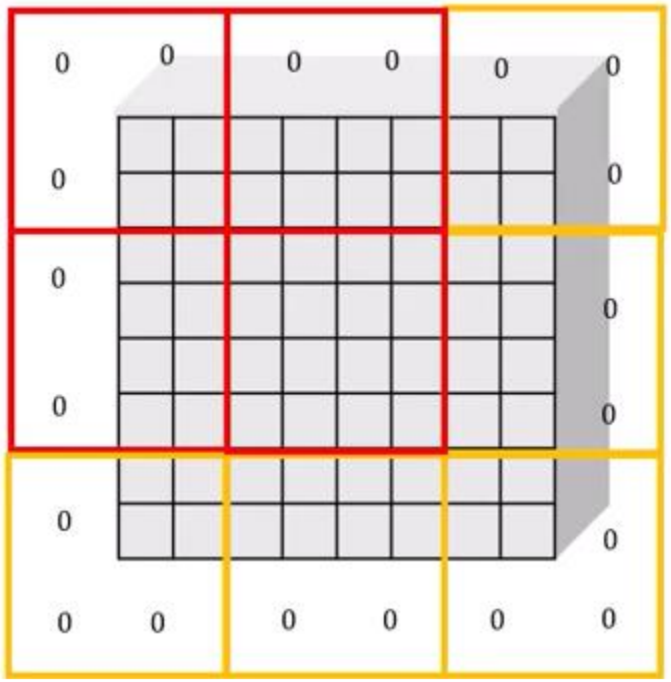

$H \times W \times 3$
Images
$\frac{H}{4} \times \frac{W}{4} \times 48$
Patch Partition
Linear Embedding
Swin Transformer Block
$\times$ 2

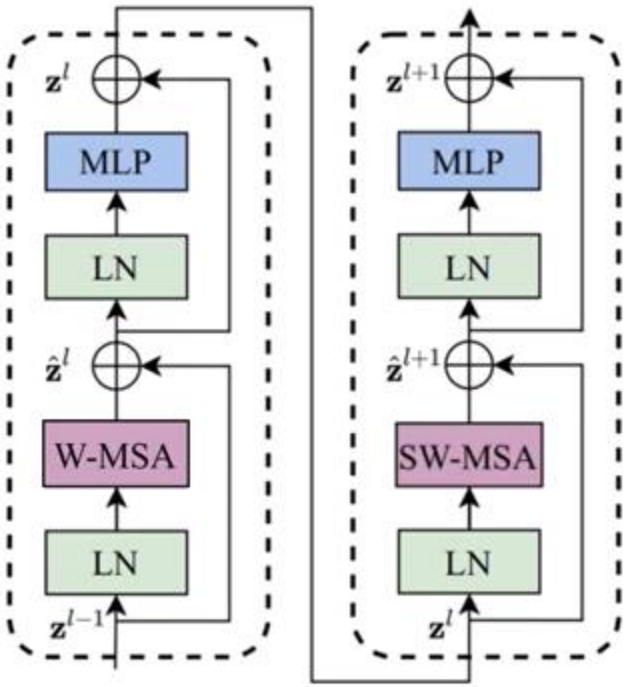

$\mathbf{z}^l$
MLP
LN
$\hat{\mathbf{z}}^l$
W-MSA
LN
$\mathbf{z}^{l-1}$
$\mathbf{z}^{l+1}$
MLP
LN
$\hat{\mathbf{z}}^{l+1}$
SW-MSA
LN
$\mathbf{z}^l$



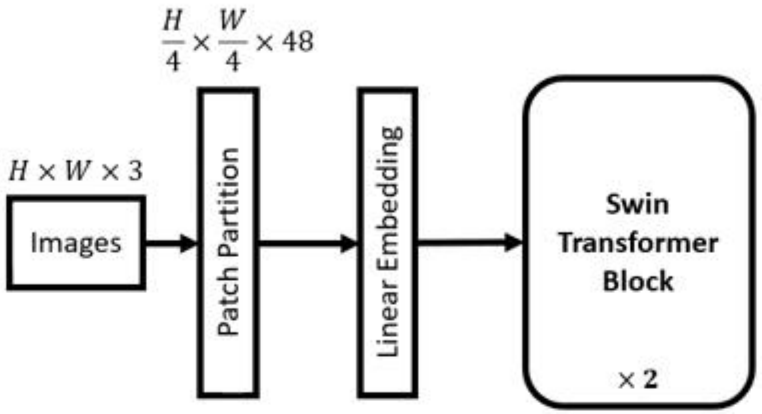

SW-MSA

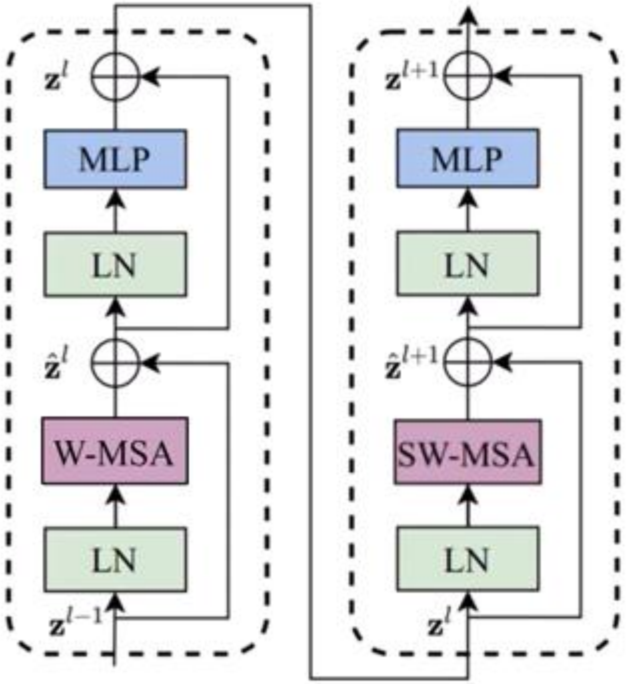

$\frac{H}{4} \times \frac{W}{4} \times 48$
$H \times W \times 3$
Images
Patch Partition
Linear Embedding
Swin Transformer Block
$\times$ 2

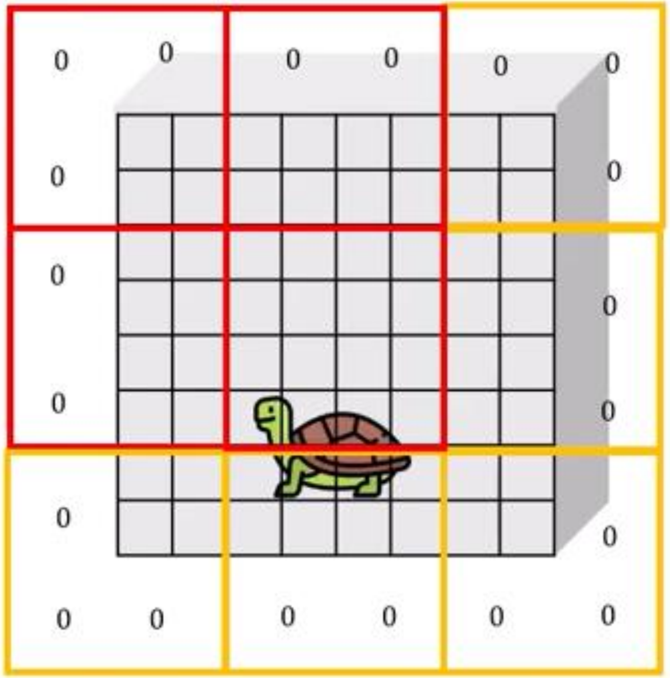

0 0 0 0 0 0
0 0
0 0
0 0
0 0
0 0 0 0 0 0


$\mathbf{z}^l$
MLP
LN
$\hat{\mathbf{z}}^l$
W-MSA
LN
$\mathbf{z}^{l-1}$
$\mathbf{z}^{l+1}$
MLP
LN
$\hat{\mathbf{z}}^{l+1}$
SW-MSA
LN
$\mathbf{z}^l$

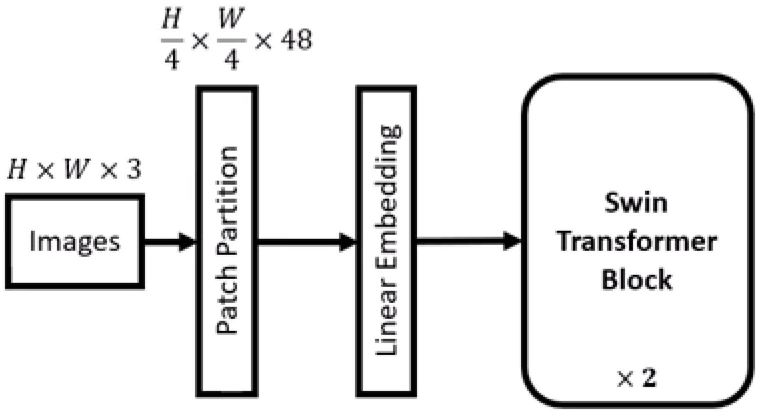

SW-MSA

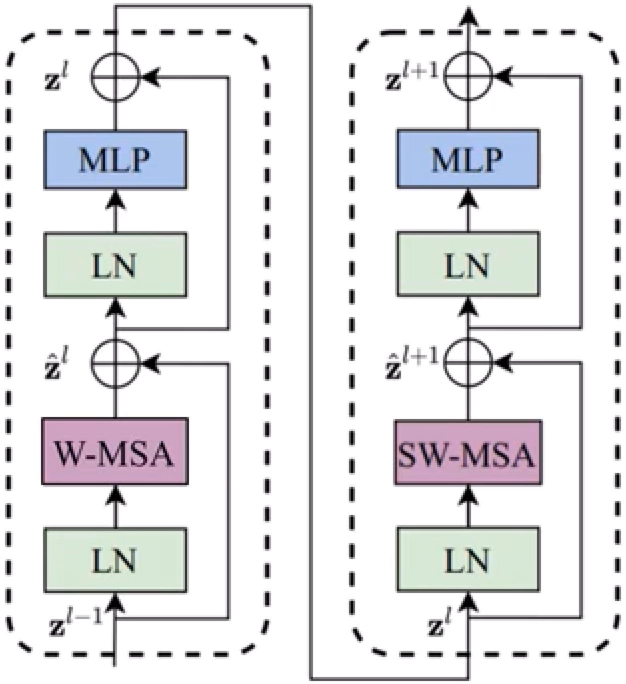

$\frac{H}{4} \times \frac{W}{4} \times 48$
$H \times W \times 3$
Images
Patch Partition
Linear Embedding
Swin Transformer Block
$\times 2$

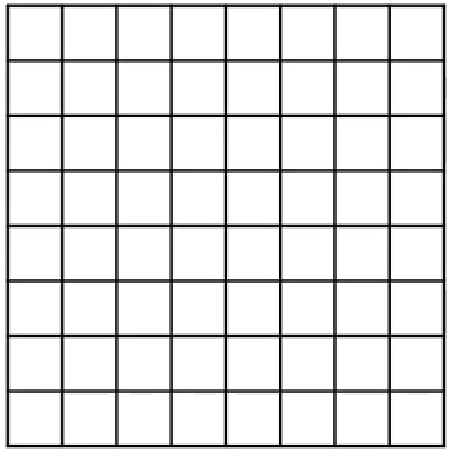

$\mathbf{z}^l$
MLP
LN
$\hat{\mathbf{z}}^l$
W-MSA
LN
$\mathbf{z}^{l-1}$
$\mathbf{z}^{l+1}$
MLP
LN
$\hat{\mathbf{z}}^{l+1}$
SW-MSA
LN
$\mathbf{z}^l$


0 0 0 0 0 0
0 0
0 0
0 0
0 0
0 0 0 0 0 0

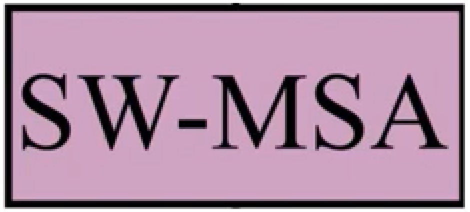

SW-MSA

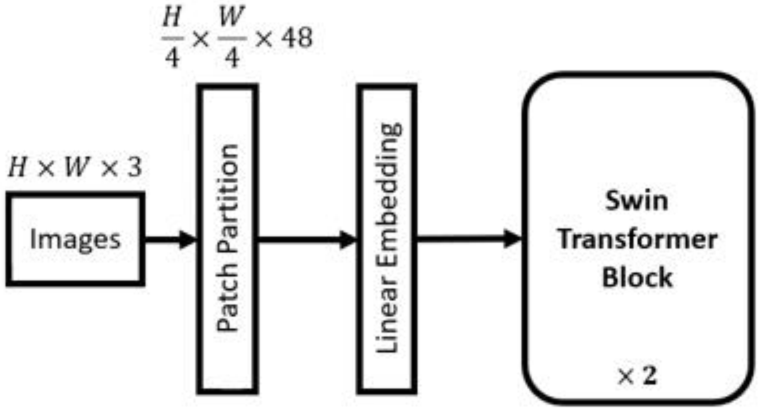


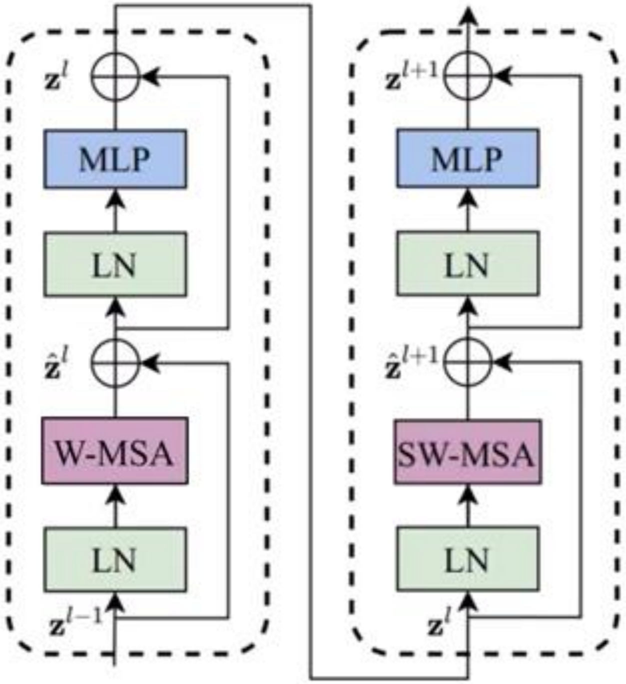


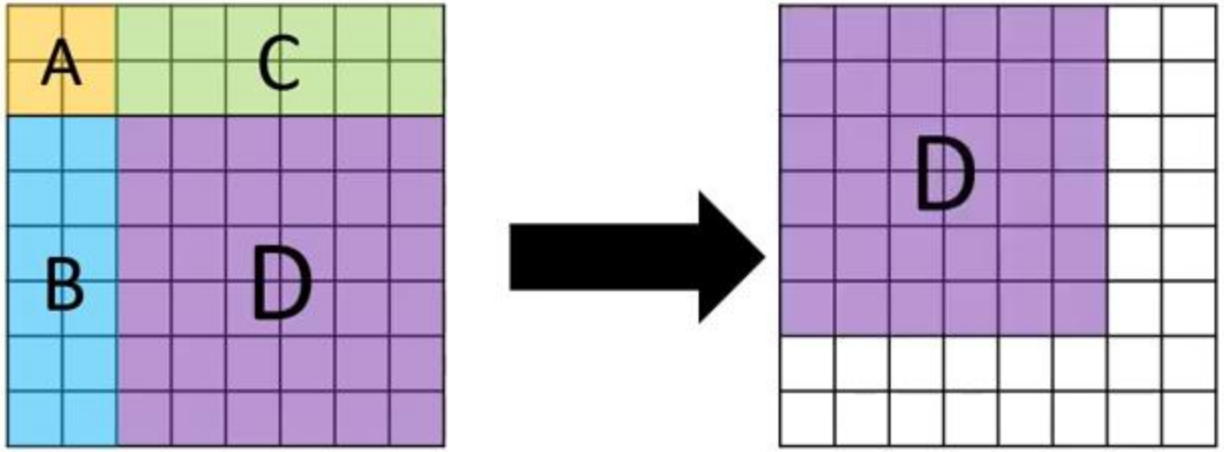


Original Patches



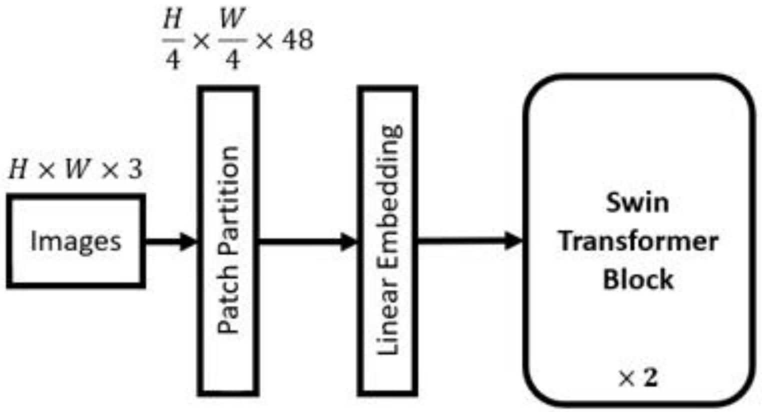


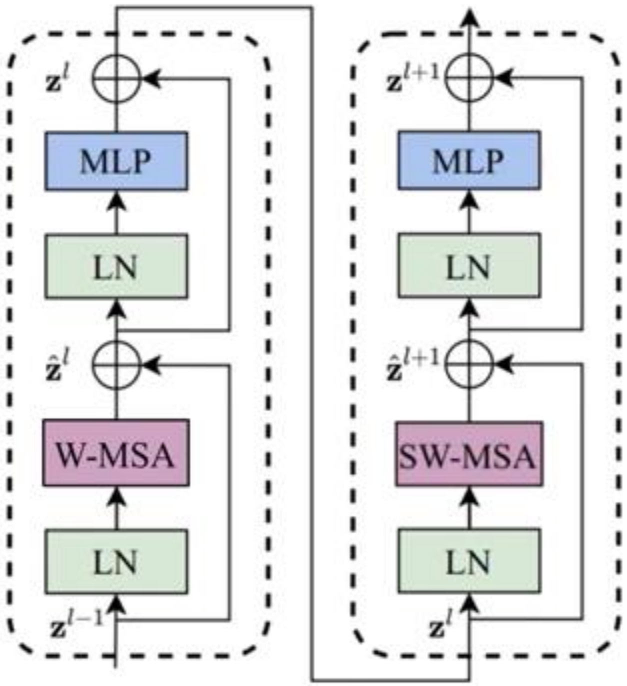


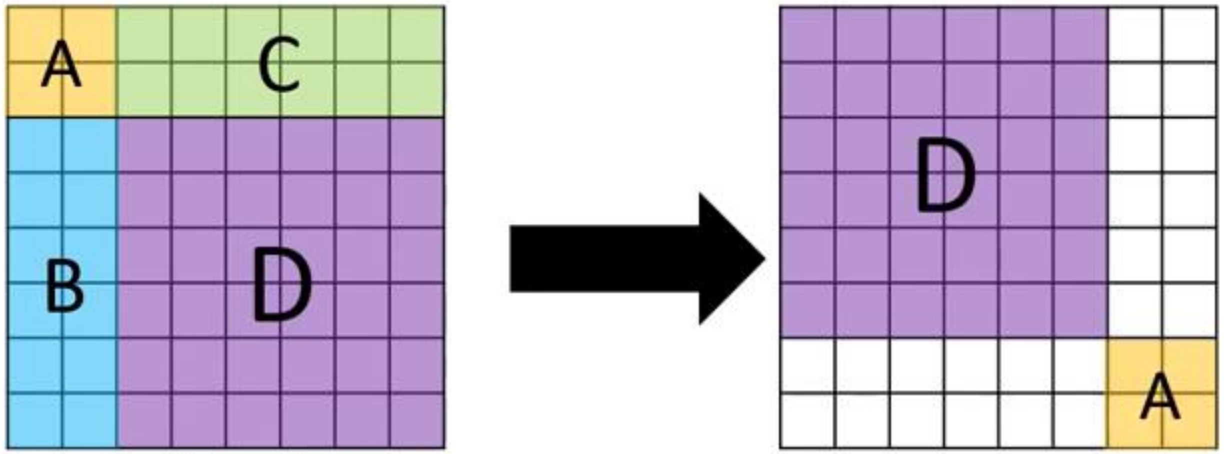


Original Patches

Shifted Patches

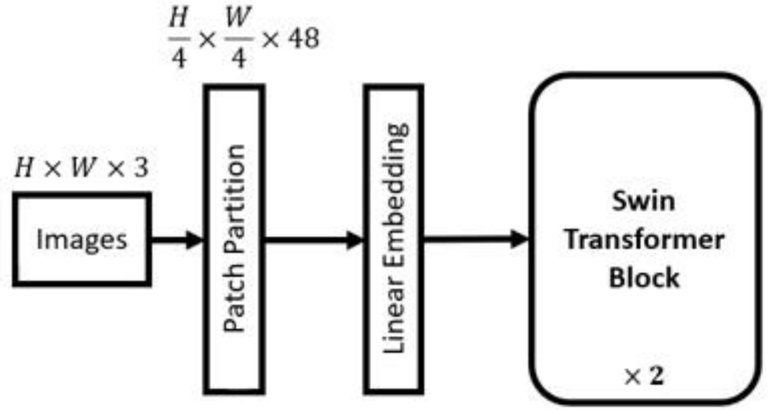


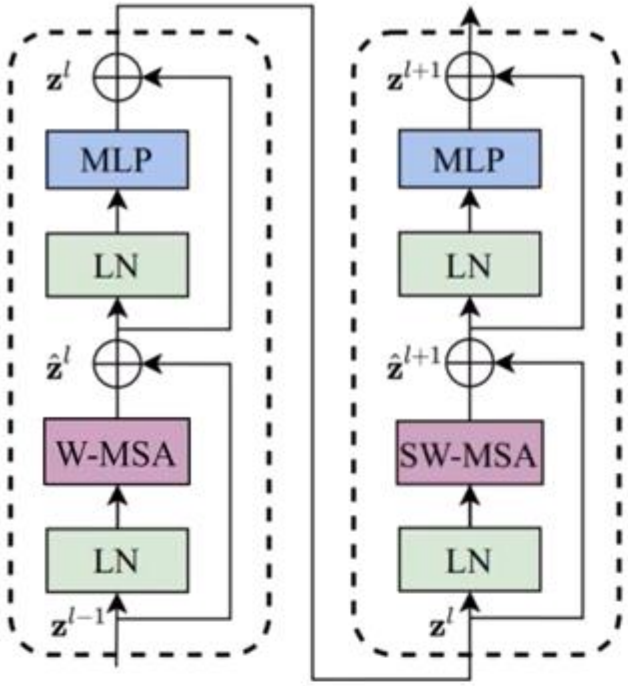


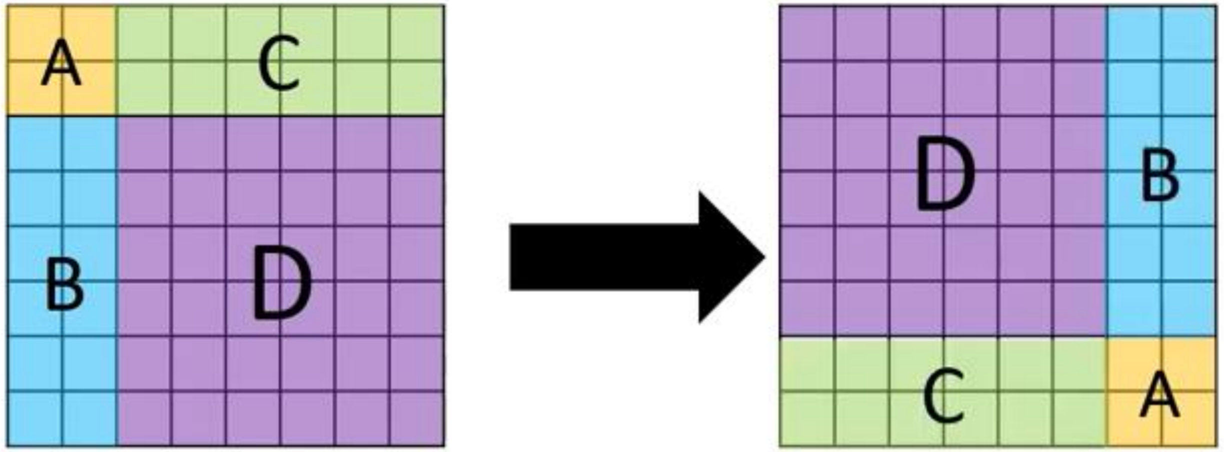


Original Patches

Shifted Patches



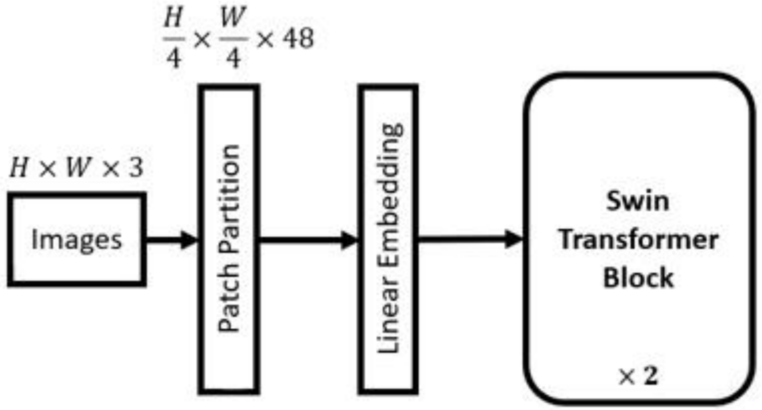


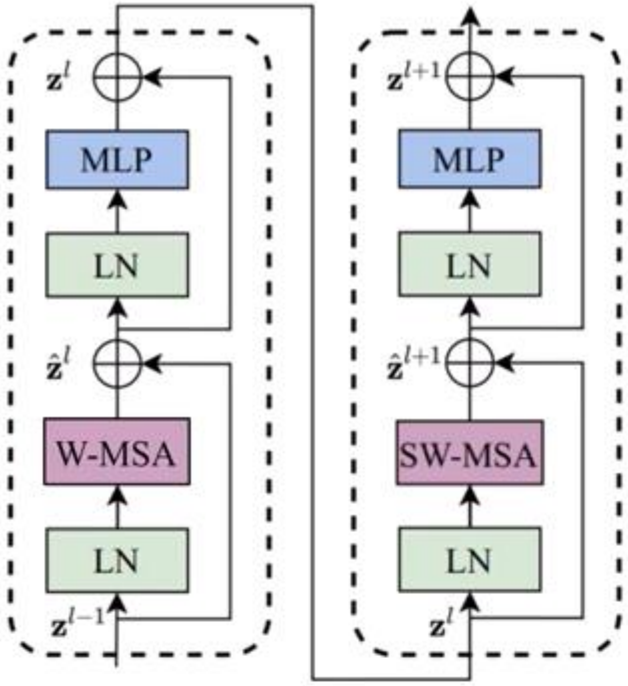


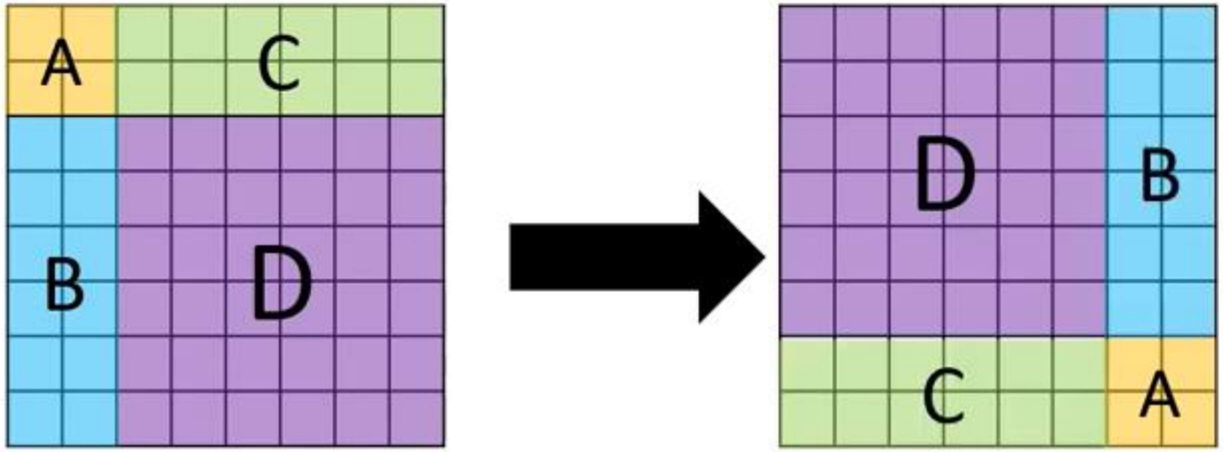
Original Patches

Shifted Patches

SW-MSA



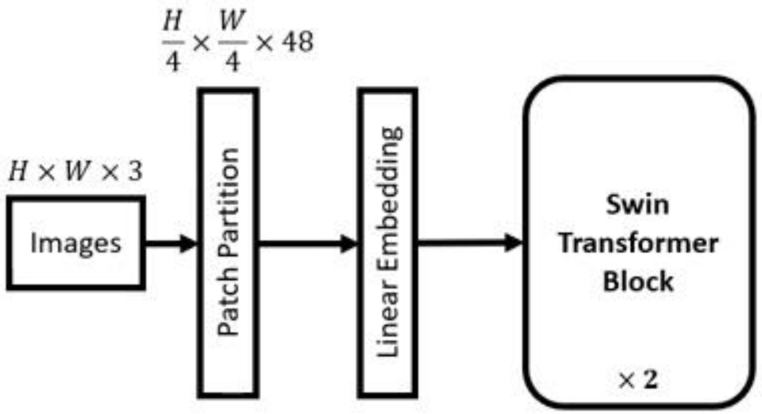




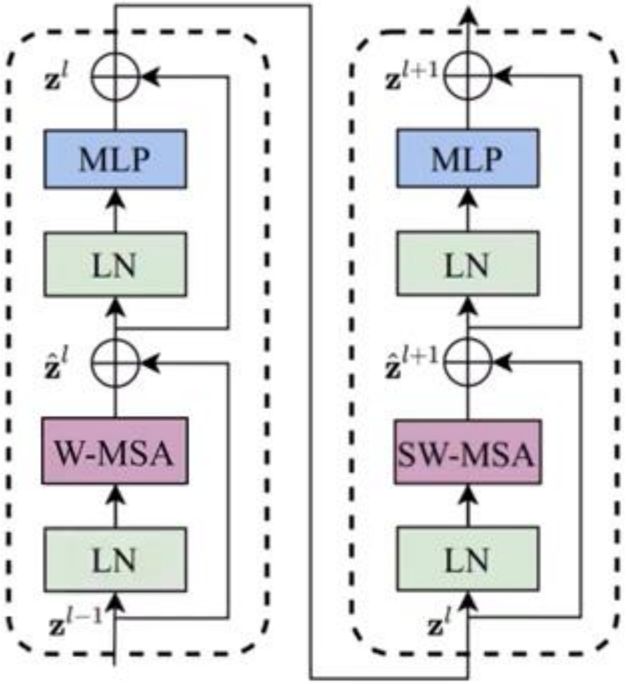
Original Patches

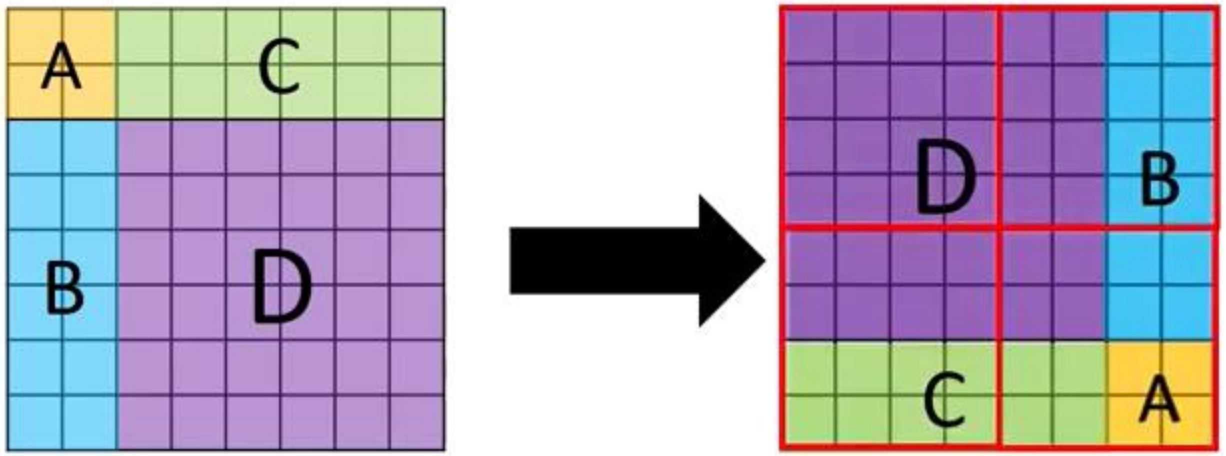
Shifted Patches

SW-MSA



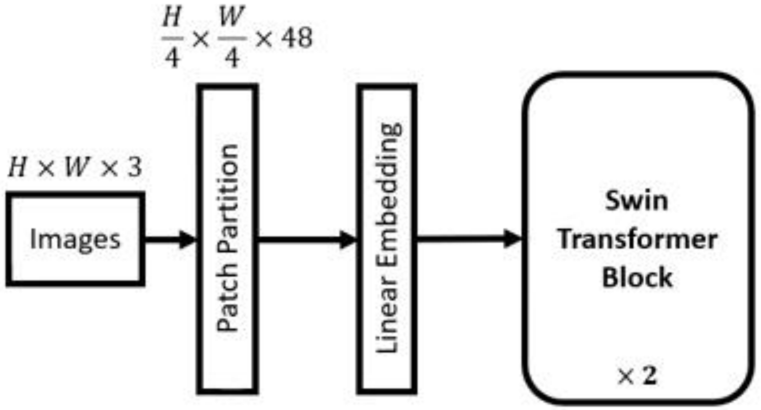
SW-MSA





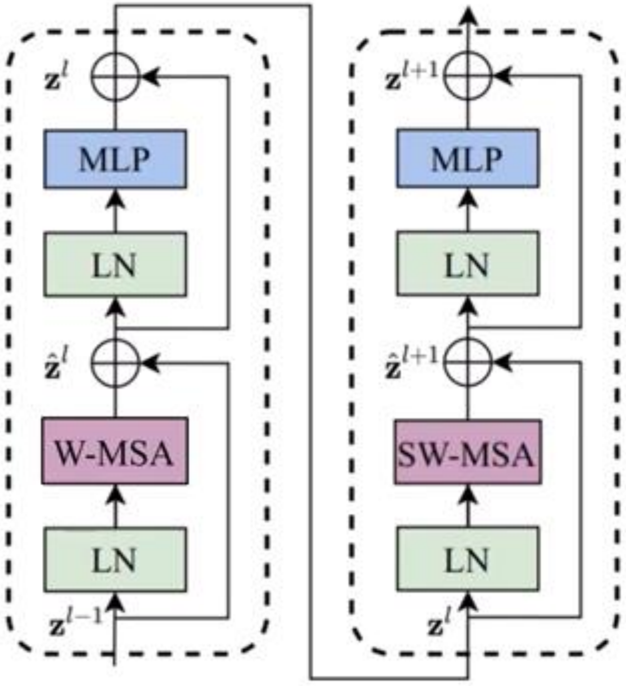
Original Patches

Shifted Patches

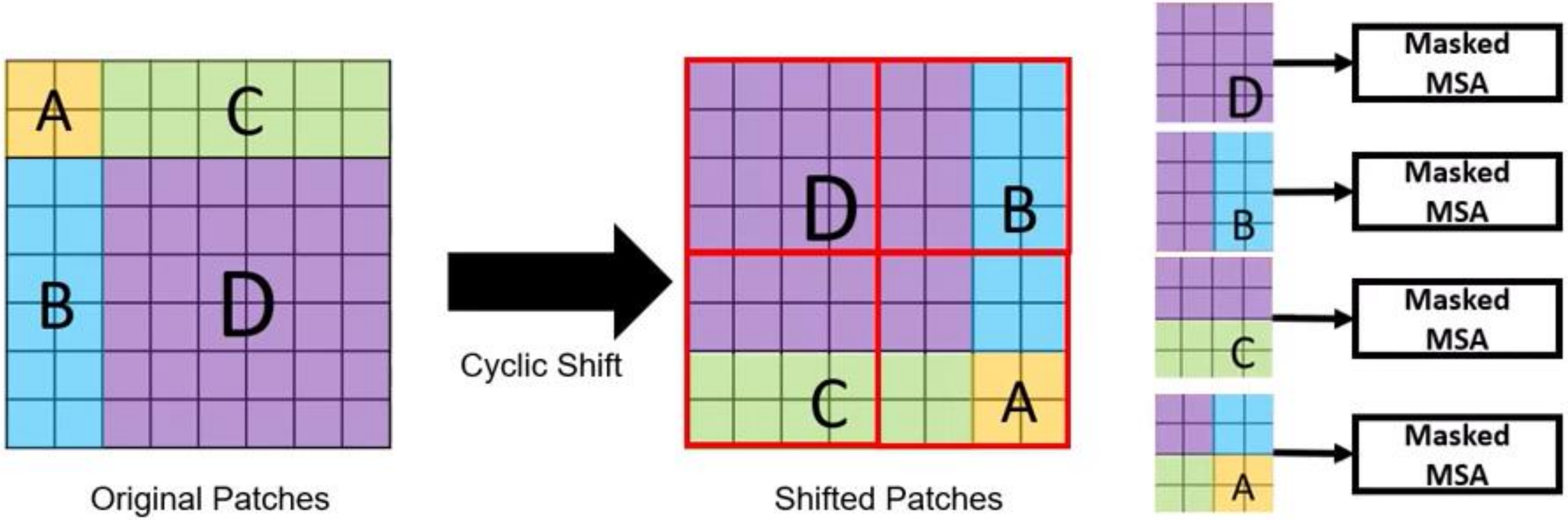


$\frac{H}{4} \times \frac{W}{4} \times 48$
$H \times W \times 3$
Images
Patch Partition
Linear Embedding
Swin Transformer Block
× 2

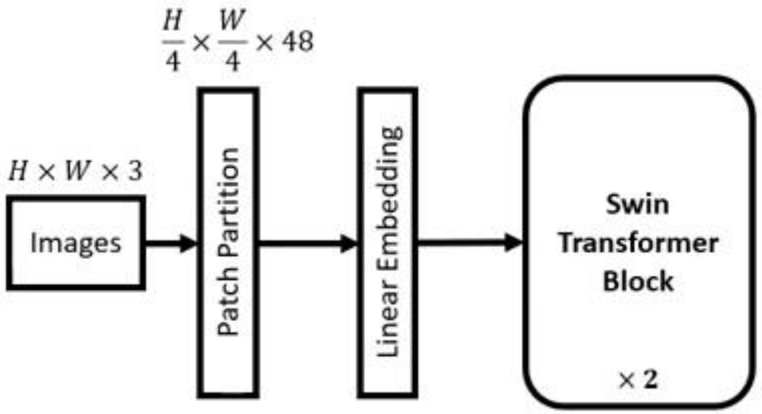
SW-MSA



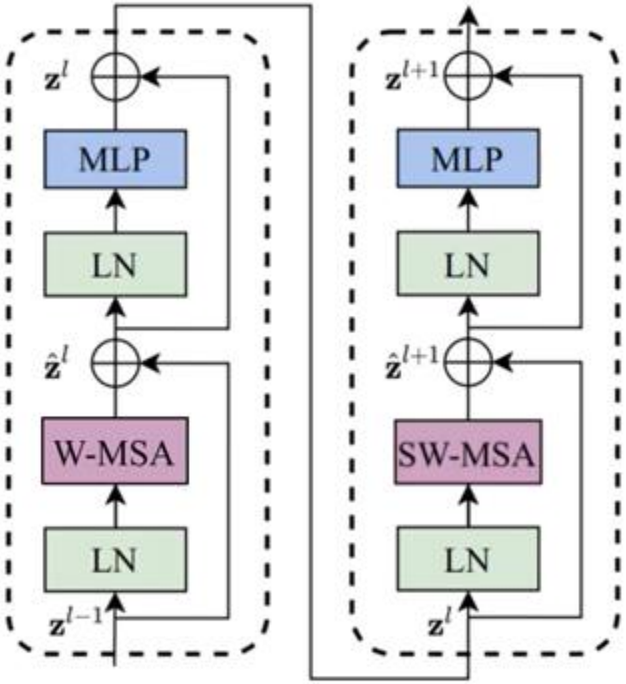
$\mathbf{z}^l$
MLP
LN
$\hat{\mathbf{z}}^l$
W-MSA
LN
$\mathbf{z}^{l-1}$
$\mathbf{z}^{l+1}$
MLP
LN
$\hat{\mathbf{z}}^{l+1}$
SW-MSA
LN
$\mathbf{z}^l$

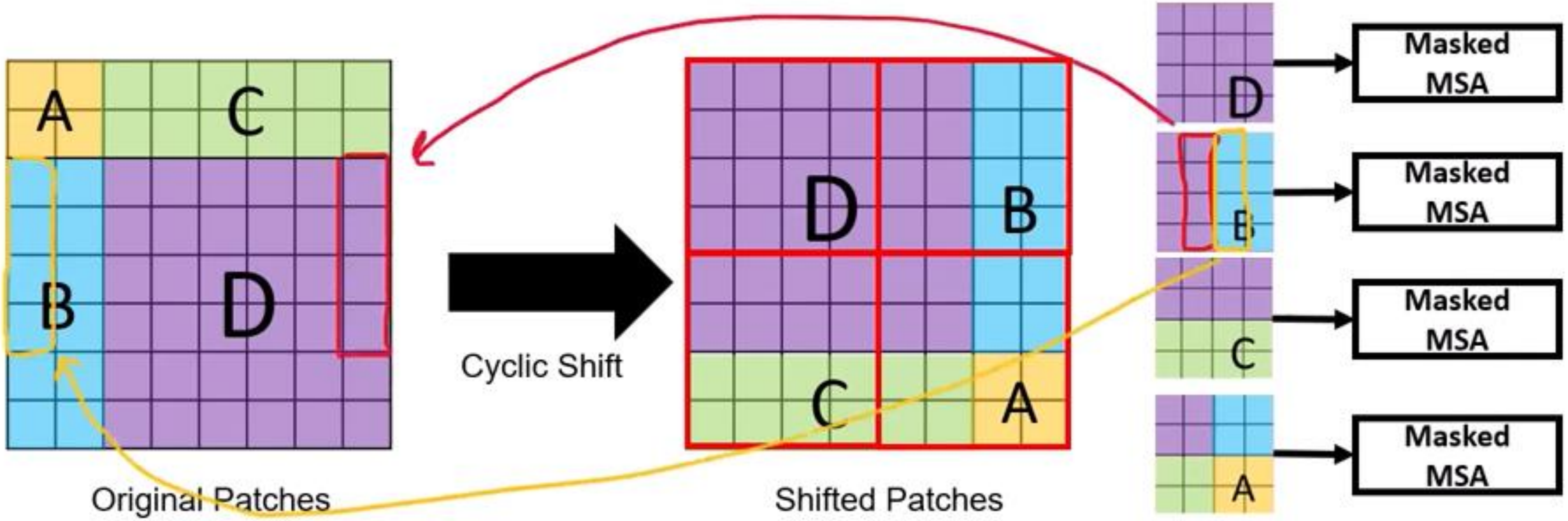


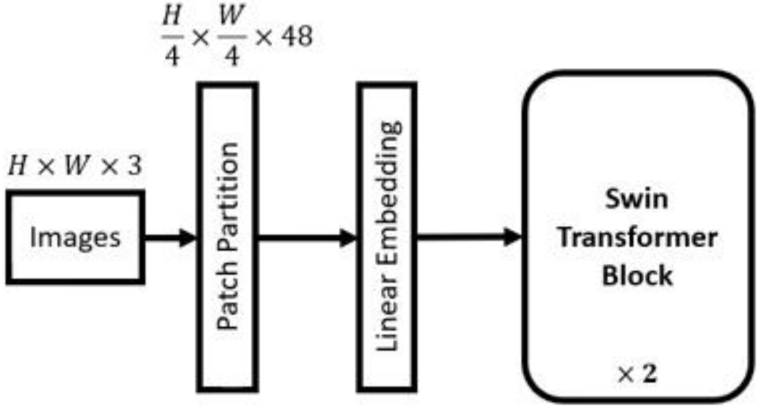
A
C
B
D
Original Patches
Cyclic Shift
D
B
C
A
Shifted Patches
D
Masked MSA
B
Masked MSA
C
Masked MSA
A
Masked MSA


$\frac{H}{4} \times \frac{W}{4} \times 48$
$H \times W \times 3$
Images
Patch Partition
Linear Embedding
Swin Transformer Block
$\times$ 2

SW-MSA

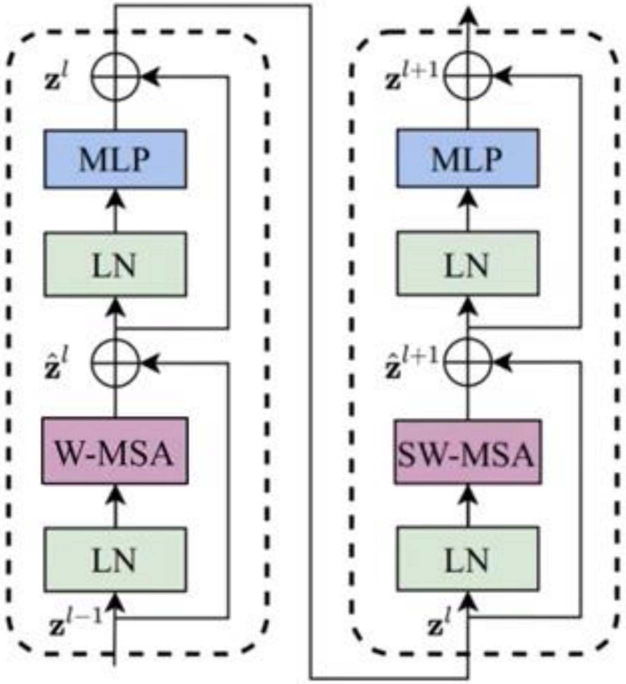

$\mathbf{z}^l$
MLP
LN
$\hat{\mathbf{z}}^l$
W-MSA
LN
$\mathbf{z}^{l-1}$
$\mathbf{z}^{l+1}$
MLP
LN
$\hat{\mathbf{z}}^{l+1}$
SW-MSA
LN
$\mathbf{z}^l$


A
C
B
D
Cyclic Shift
D
B
C
A
Original Patches
Shifted Patches
D
B
C
A
Masked MSA
Masked MSA
Masked MSA
Masked MSA

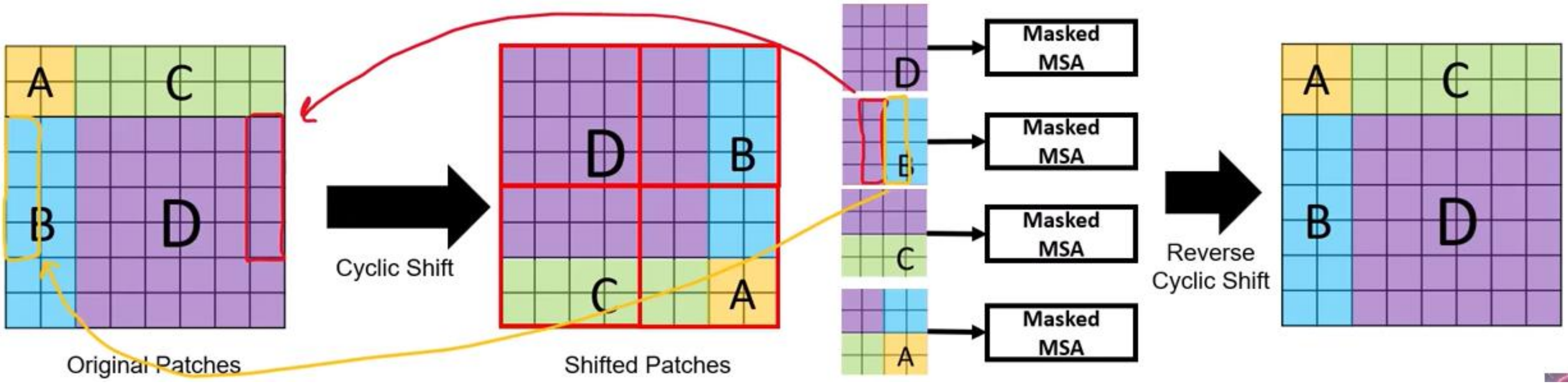


$\frac{H}{4} \times \frac{W}{4} \times 48$
$H \times W \times 3$
Images
Patch Partition
Linear Embedding
Swin Transformer Block
$\times$ 2

SW-MSA



$\mathbf{z}^l$
MLP
LN
$\hat{\mathbf{z}}^l$
W-MSA
LN
$\mathbf{z}^{l-1}$
$\mathbf{z}^{l+1}$
MLP
LN
$\hat{\mathbf{z}}^{l+1}$
SW-MSA
LN
$\mathbf{z}^l$



A
C
B
D
Cyclic Shift
D
B
C
A
D
B
C
A
Masked MSA
Masked MSA
Masked MSA
Masked MSA
Reverse Cyclic Shift
A
C
B
D
Original Patches
Shifted Patches

```python
        if self.shift_size > 0:
            # calculate attention mask for SW-MSA
            H, W = self.input_resolution
            img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
            h_slices = (slice(0, -self.window_size),
                        slice(-self.window_size, -self.shift_size),
                        slice(-self.shift_size, None))
            w_slices = (slice(0, -self.window_size),
                        slice(-self.window_size, -self.shift_size),
                        slice(-self.shift_size, None))
            cnt = 0
            for h in h_slices:
                for w in w_slices:
                    img_mask[:, h, w, :] = cnt
                    cnt += 1

            mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
            mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
            attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
            attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
        else:
            attn_mask = None
```

```python
        if self.shift_size > 0:
            # calculate attention mask for SW-MSA
            H, W = self.input_resolution
            img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
            h_slices = (slice(0, -self.window_size), [0: -4]
                        slice(-self.window_size, -self.shift_size), [-4: -2]
                        slice(-self.shift_size, None)) [-2: ]
            w_slices = (slice(0, -self.window_size), [0: -4]
                        slice(-self.window_size, -self.shift_size), [-4: -2]
                        slice(-self.shift_size, None)) [-2: ]
            cnt = 0
            for h in h_slices:
                for w in w_slices:
                    img_mask[:, h, w, :] = cnt
                    cnt += 1

            mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
            mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
            attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
            attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
        else:
            attn_mask = None
```

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Assumptions:

H = W = 8

window_size = 4

shift_size = 2

```python
if self.shift_size > 0:
    # calculate attention mask for SW-MSA
    H, W = self.input_resolution
    img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
    h_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    w_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1

    mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
else:
    attn_mask = None
```

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Assumptions:

H = W = 8

window_size = 4

shift_size = 2

```python
if self.shift_size > 0:
    # calculate attention mask for SW-MSA
    H, W = self.input_resolution
    img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
    h_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    w_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1

    mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
else:
    attn_mask = None
```

0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Assumptions:

H = W = 8

window_size = 4

shift_size = 2

```python
        if self.shift_size > 0:
            # calculate attention mask for SW-MSA
            H, W = self.input_resolution
            img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
            h_slices = (slice(0, -self.window_size), [0: -4]
                        slice(-self.window_size, -self.shift_size), [-4: -2]
                        slice(-self.shift_size, None)) [-2: ]
            w_slices = (slice(0, -self.window_size), [0: -4]
                        slice(-self.window_size, -self.shift_size), [-4: -2]
                        slice(-self.shift_size, None)) [-2: ]
            cnt = 0
            for h in h_slices:
                for w in w_slices:
                    img_mask[:, h, w, :] = cnt
                    cnt += 1

            mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
            mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
            attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
            attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
        else:
            attn_mask = None
```

0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Assumptions:

H = W = 8

window_size = 4

shift_size = 2

```
if self.shift_size > 0:
    # calculate attention mask for SW-MSA
    H, W = self.input_resolution
    img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
    h_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    w_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1

    mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
else:
    attn_mask = None
```

0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
3	3	3	3	0	0	0	0
3	3	3	3	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Assumptions:

H = W = 8

window_size = 4

shift_size = 2

```python
        if self.shift_size > 0:
            # calculate attention mask for SW-MSA
            H, W = self.input_resolution
            img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
            h_slices = (slice(0, -self.window_size), [0: -4]
                        slice(-self.window_size, -self.shift_size), [-4: -2]
                        slice(-self.shift_size, None)) [-2: ]
            w_slices = (slice(0, -self.window_size), [0: -4]
                        slice(-self.window_size, -self.shift_size), [-4: -2]
                        slice(-self.shift_size, None)) [-2: ]
            cnt = 0
            for h in h_slices:
                for w in w_slices:
                    img_mask[:, h, w, :] = cnt
                    cnt += 1

            mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
            mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
            attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
            attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
        else:
            attn_mask = None
```

0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
3	3	3	3	4	4	0	0
3	3	3	3	4	4	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Assumptions:

H = W = 8

window_size = 4

shift_size = 2

```
if self.shift_size > 0:
    # calculate attention mask for SW-MSA
    H, W = self.input_resolution
    img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
    h_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    w_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1

    mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
else:
    attn_mask = None
```

0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
3	3	3	3	4	4	5	5
3	3	3	3	4	4	5	5
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Assumptions:

H = W = 8

window_size = 4

shift_size = 2

```
if self.shift_size > 0:
    # calculate attention mask for SW-MSA
    H, W = self.input_resolution
    img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
    h_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    w_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1

    mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
else:
    attn_mask = None
```

0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
3	3	3	3	4	4	5	5
3	3	3	3	4	4	5	5
6	6	6	6	0	0	0	0
6	6	6	6	0	0	0	0

Assumptions:

H = W = 8

window_size = 4

shift_size = 2

```python
if self.shift_size > 0:
    # calculate attention mask for SW-MSA
    H, W = self.input_resolution
    img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
    h_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    w_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1

    mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
else:
    attn_mask = None
```

0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
3	3	3	3	4	4	5	5
3	3	3	3	4	4	5	5
6	6	6	6	7	7	0	0
6	6	6	6	7	7	0	0

Assumptions:

- H = W = 8
- window_size = 4
- shift_size = 2

```python
if self.shift_size > 0:
    # calculate attention mask for SW-MSA
    H, W = self.input_resolution
    img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
    h_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    w_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1

    mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
else:
    attn_mask = None
```

0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
3	3	3	3	4	4	5	5
3	3	3	3	4	4	5	5
6	6	6	6	7	7	8	8
6	6	6	6	7	7	8	8

Assumptions:

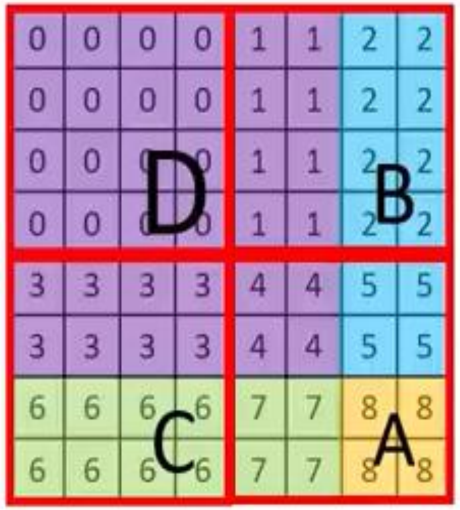
H = W = 8

window_size = 4

shift_size = 2

```python
if self.shift_size > 0:
    # calculate attention mask for SW-MSA
    H, W = self.input_resolution
    img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
    h_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    w_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1

    mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
else:
    attn_mask = None
```



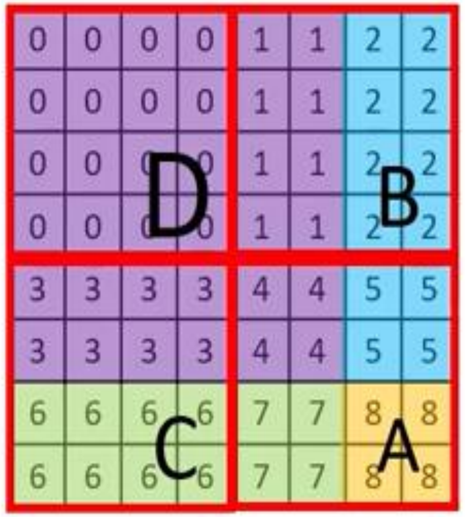
Assumptions:

H = W = 8

window_size = 4

shift_size = 2

```
if self.shift_size > 0:
    # calculate attention mask for SW-MSA
    H, W = self.input_resolution
    img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
    h_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    w_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1
                                                            4    4            4
    mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
else:
    attn_mask = None
```



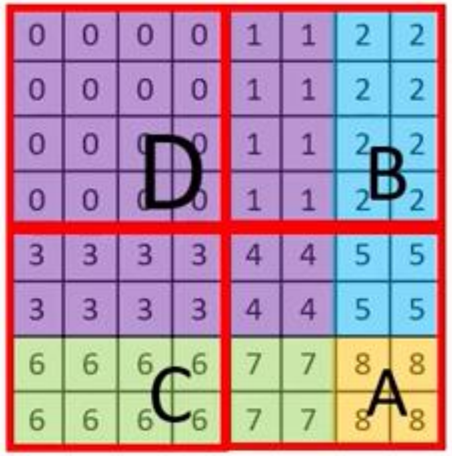
Assumptions:

H = W = 8

window_size = 4

shift_size = 2

```python
        if self.shift_size > 0:
            # calculate attention mask for SW-MSA
            H, W = self.input_resolution
            img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
            h_slices = (slice(0, -self.window_size),  [0: -4]
                        slice(-self.window_size, -self.shift_size),  [-4: -2]
                        slice(-self.shift_size, None))  [-2: ]
            w_slices = (slice(0, -self.window_size),  [0: -4]
                        slice(-self.window_size, -self.shift_size),  [-4: -2]
                        slice(-self.shift_size, None))  [-2: ]
            cnt = 0
            for h in h_slices:
                for w in w_slices:
                    img_mask[:, h, w, :] = cnt
                    cnt += 1
                                                                        4      4            4
            mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
            mask_windows = mask_windows.view(-1, self.window_size * self.window_size)  → [4,16]
            attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
            attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
        else:
            attn_mask = None
```



Assumptions:

H = W = 8

window_size = 4

shift_size = 2

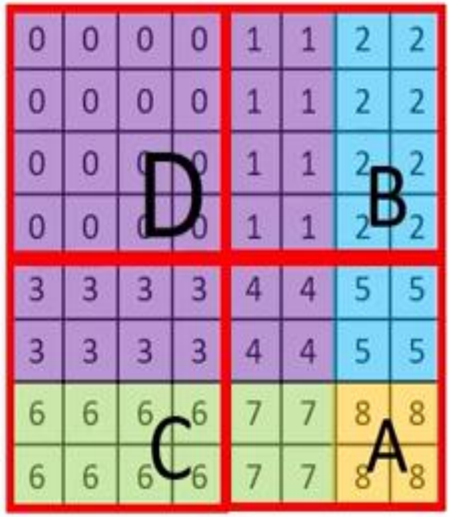
```
if self.shift_size > 0:
    # calculate attention mask for SW-MSA
    H, W = self.input_resolution
    img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
    h_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    w_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1

    mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size) → [4,16]
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
else:
    attn_mask = None
```

[4,1,16]

[4,16,1]

4 4 4



Assumptions:

H = W = 8

window_size = 4

shift_size = 2

```python
if self.shift_size > 0:
    # calculate attention mask for SW-MSA
    H, W = self.input_resolution
    img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
    h_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    w_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1

    mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size)
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
else:
    attn_mask = None
```

[4,1,16]

[4,16,1]

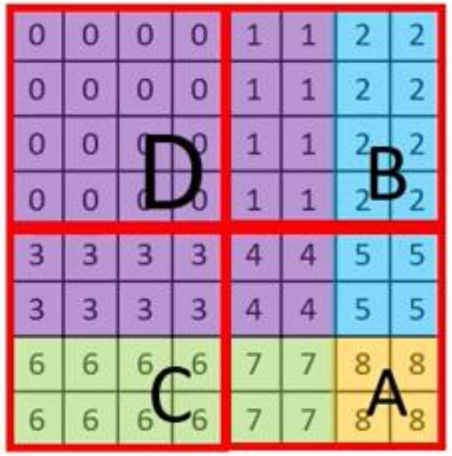
4 4 4

[4,16]

Different

Same

why?



Assumptions:

H = W = 8

window_size = 4

shift_size = 2

```
if self.shift_size > 0:
    # calculate attention mask for SW-MSA
    H, W = self.input_resolution
    img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
    h_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    w_slices = (slice(0, -self.window_size), [0: -4]
                slice(-self.window_size, -self.shift_size), [-4: -2]
                slice(-self.shift_size, None)) [-2: ]
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1

    mask_windows = window_partition(img_mask, self.window_size)  # nW, window_size, window_size, 1
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size) → [4,16]
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
else:
    attn_mask = None
```

4 4 4

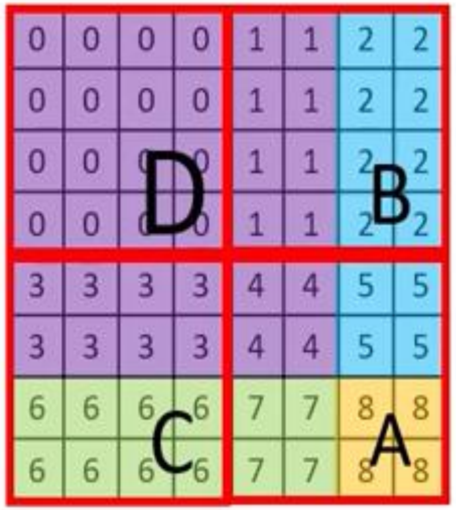
[4,1,16]

[4,16,1]

Different

Same

why?



Assumptions:

H = W = 8

window_size = 4

shift_size = 2

$$Attention(Q,K,V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

+attn_mask

3. Apply the softmax function to each input:

$$\text{softmax}(-100) = \frac{e^{-100}}{1} \approx 3.72 \times 10^{-44}$$

$$\text{softmax}(0) = \frac{e^{0}}{1} = 1$$

4. Normalize the outputs so that they sum to 1:

$$\text{softmax}(-100) \approx \frac{3.72 \times 10^{-44}}{1 + 3 \times 3.72 \times 10^{-44}} = \frac{3.72 \times 10^{-44}}{1} \approx 3.72 \times 10^{-44}$$

$$\text{softmax}(0) \approx \frac{1}{1 + 3 \times 3.72 \times 10^{-44}} = \frac{1}{1} = 1$$

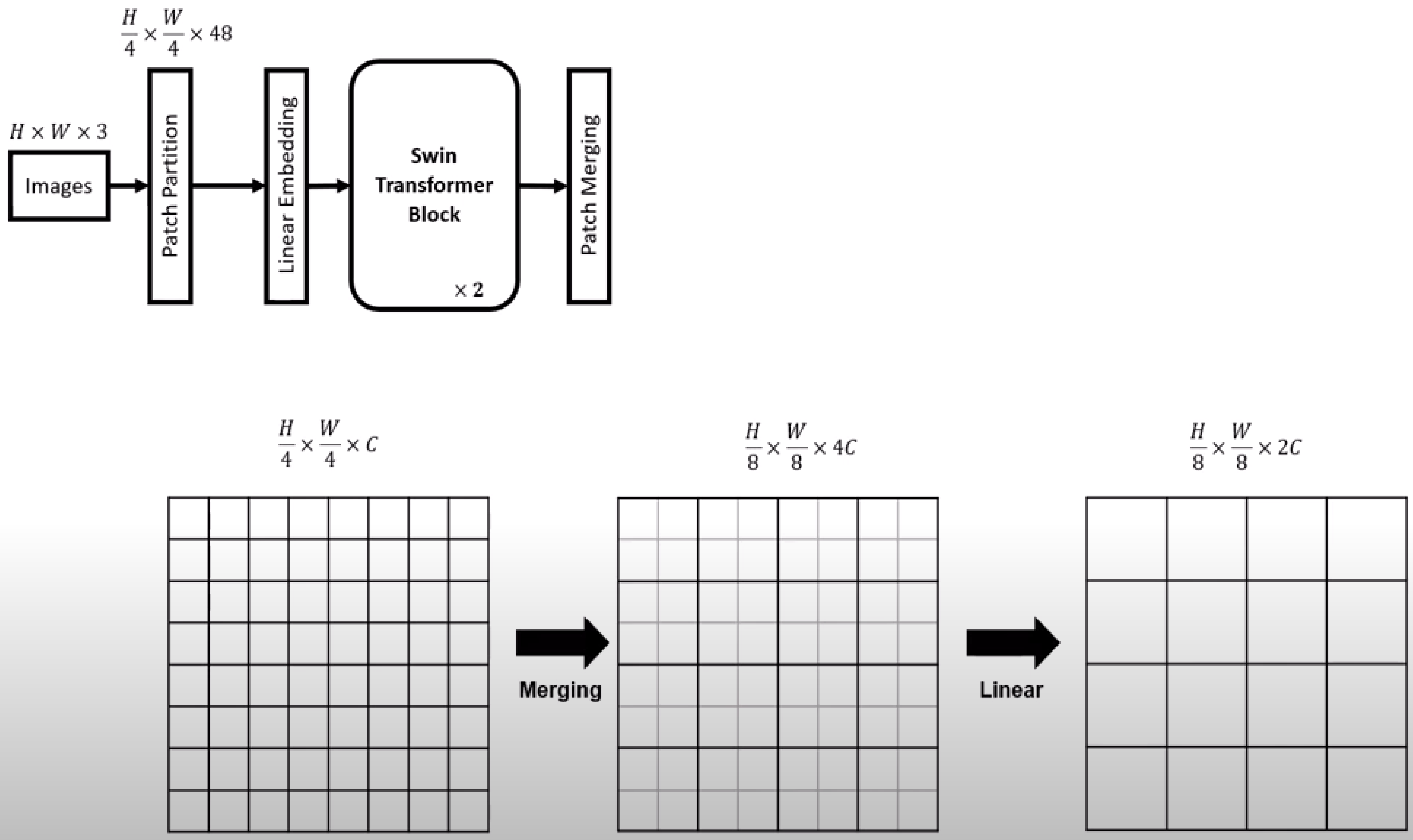
Since the terms 3.72×10^{-44} are exceedingly small compared to 1, the output for the softmax will effectively be:

$$\text{softmax}([-100, -100, -100, 0]) = [3.72 \times 10^{-44}, 3.72 \times 10^{-44}, 3.72 \times 10^{-44}, 1]$$

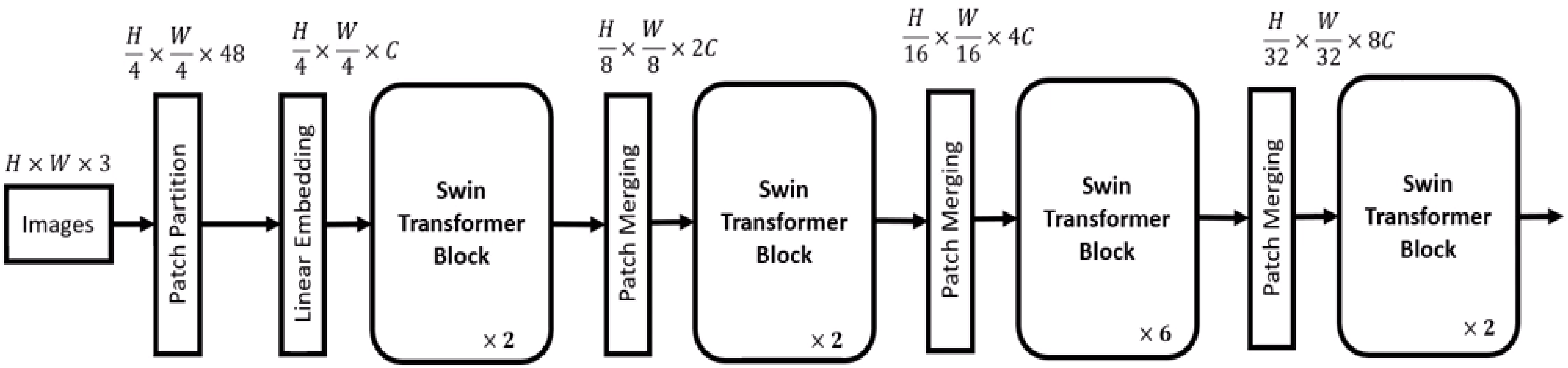
When normalized, the values will sum to 1:

$$\text{softmax}([-100, -100, -100, 0]) \approx [0, 0, 0, 1]$$

This indicates that the probability is almost entirely concentrated on the last input, which is 0, as it is significantly larger than the others.



$\frac{H}{4} \times \frac{W}{4} \times 48$
$H \times W \times 3$
Images
Patch Partition
Linear Embedding
Swin Transformer Block
$\times 2$
Patch Merging
$\frac{H}{4} \times \frac{W}{4} \times C$
$\frac{H}{8} \times \frac{W}{8} \times 4C$
$\frac{H}{8} \times \frac{W}{8} \times 2C$
Merging
Linear



$H \times W \times 3$
Images
$\frac{H}{4} \times \frac{W}{4} \times 48$
Patch Partition
$\frac{H}{4} \times \frac{W}{4} \times C$
Linear Embedding
Swin Transformer Block
$\times 2$
$\frac{H}{8} \times \frac{W}{8} \times 2C$
Patch Merging
Swin Transformer Block
$\times 2$
$\frac{H}{16} \times \frac{W}{16} \times 4C$
Patch Merging
Swin Transformer Block
$\times 6$
$\frac{H}{32} \times \frac{W}{32} \times 8C$
Patch Merging
Swin Transformer Block
$\times 2$

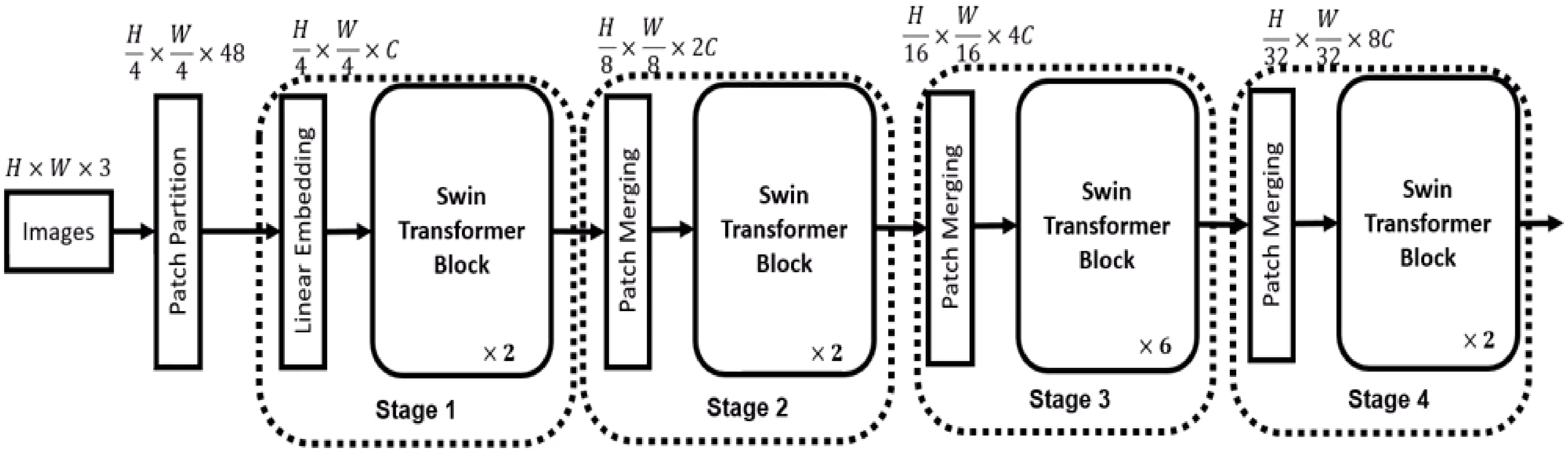


Image classification: Output of the last stage is passed through a linear layer.

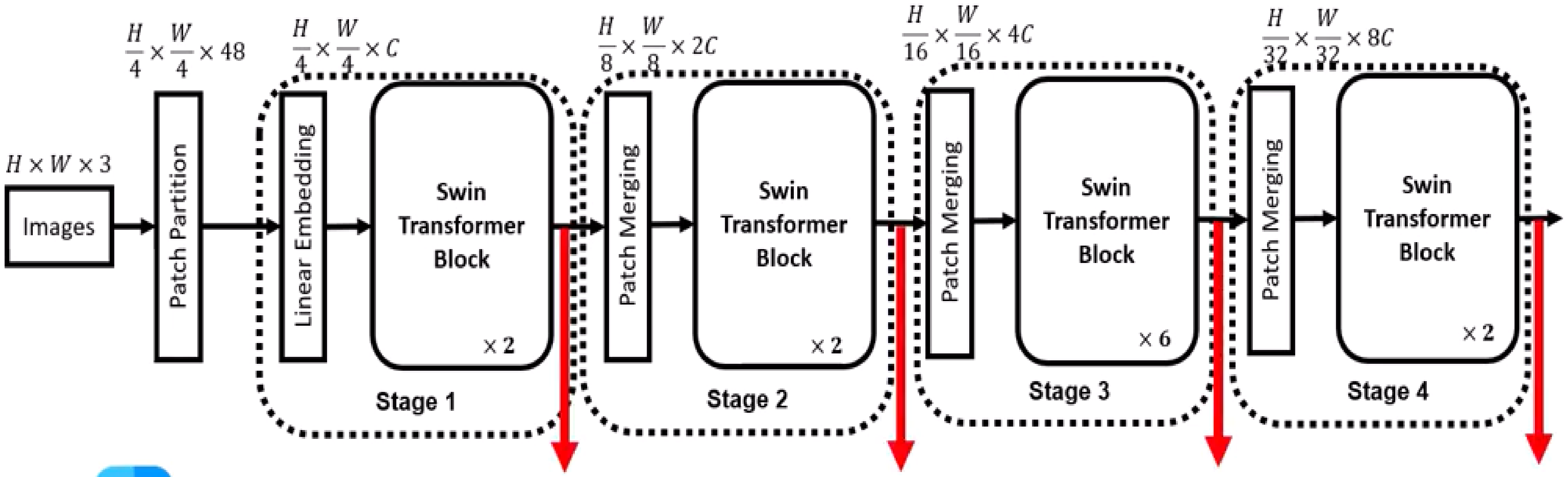


Image classification: Output of the last stage is passed through a linear layer.

Object detection and Image segmentation: Output of all the stages are used as features.

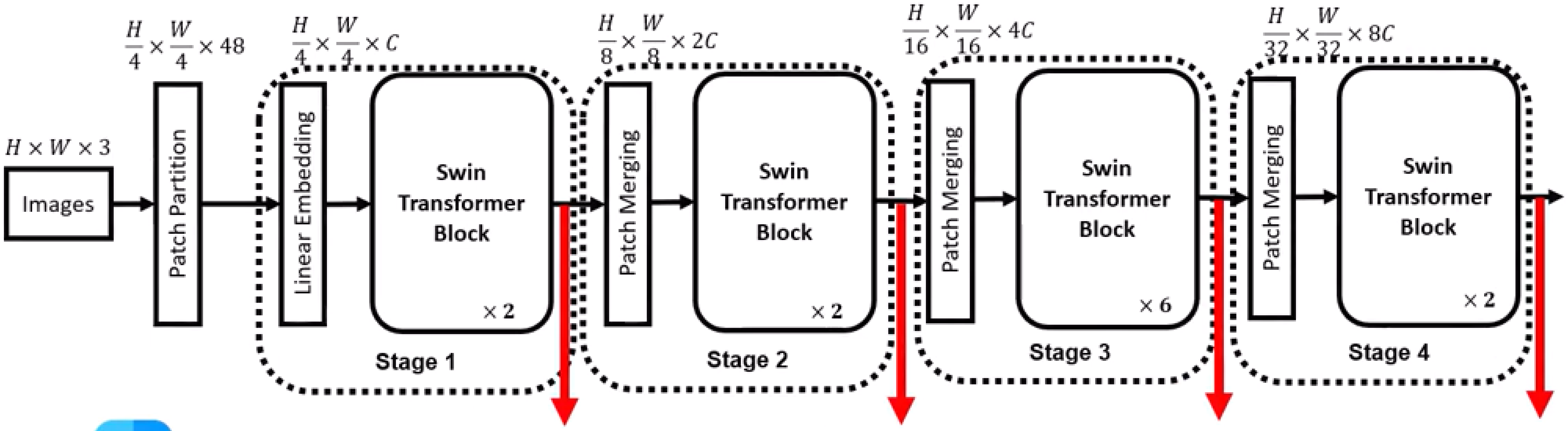


Image classification: Output of the last stage is passed through a linear layer.

Object detection and Image segmentation: Output of all the stages are used as features.

- Swin-T: $C = 96$, layer numbers = $\{2, 2, 6, 2\}$
- Swin-S: $C = 96$, layer numbers =$\{2, 2, 18, 2\}$
- Swin-B: $C = 128$, layer numbers =$\{2, 2, 18, 2\}$
- Swin-L: $C = 192$, layer numbers =$\{2, 2, 18, 2\}$

(a) Regular ImageNet-1K trained models

method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
RegNetY-4G [44]	224^2	21M	4.0G	1156.7	80.0
RegNetY-8G [44]	224^2	39M	8.0G	591.6	81.7
RegNetY-16G [44]	224^2	84M	16.0G	334.7	82.9
ViT-B/16 [19]	384^2	86M	55.4G	85.9	77.9
ViT-L/16 [19]	384^2	307M	190.7G	27.3	76.5
DeiT-S [57]	224^2	22M	4.6G	940.4	79.8
DeiT-B [57]	224^2	86M	17.5G	292.3	81.8
DeiT-B [57]	384^2	86M	55.4G	85.9	83.1
Swin-T	224^2	29M	4.5G	755.2	81.3
Swin-S	224^2	50M	8.7G	436.9	83.0
Swin-B	224^2	88M	15.4G	278.1	83.5
Swin-B	384^2	88M	47.0G	84.7	84.5

(b) ImageNet-22K pre-trained models

method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
R-101x3 [34]	384^2	388M	204.6G	-	84.4
R-152x4 [34]	480^2	937M	840.5G	-	85.4
ViT-B/16 [19]	384^2	86M	55.4G	85.9	84.0
ViT-L/16 [19]	384^2	307M	190.7G	27.3	85.2
Swin-B	224^2	88M	15.4G	278.1	85.2
Swin-B	384^2	88M	47.0G	84.7	86.4
Swin-L	384^2	197M	103.9G	42.1	87.3

	ImageNet		COCO		ADE20k
	top-1	top-5	AP^{box}	AP^{mask}	mIoU
w/o shifting	80.2	95.1	47.7	41.5	43.3
shifted windows	**81.3**	**95.6**	**50.5**	**43.7**	**46.1**
no pos.	80.1	94.9	49.2	42.6	43.8
abs. pos.	80.5	95.2	49.0	42.4	43.2
abs.+rel. pos.	81.3	95.6	50.2	43.4	44.0
rel. pos. w/o app.	79.3	94.7	48.2	41.9	44.1
rel. pos.	**81.3**	**95.6**	**50.5**	**43.7**	**46.1**

Table 4. Ablation study on the *shifted windows* approach and different position embedding methods on three benchmarks, using the Swin-T architecture. w/o shifting: all self-attention modules adopt regular window partitioning, without *shifting*; abs. pos.: absolute position embedding term of ViT; rel. pos.: the default settings with an additional relative position bias term (see Eq. (4)); app.: the first scaled dot-product term in Eq. (4).