서울여자대학교 소프트웨어융합학과 김세희

Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

Microsoft Research Asia

2021

1. Introduction

- ✓ Transformer is notable for its use of attention to model long-range dependencies in the data.
- ✓ In this paper, we seek to expand the applicability of Transformer such that it can serve as a general-purpose backbone for computer vision.
- ✓ **There are two challenges:**
  - ➢ Differences of scale(variability in scale)
  - ➢ Higher resolution of pixels in images compared to words in passages of text
- ✓ To overcome these issues, we propose a general-purpose Transformer backbone **"Swin Transformer"**

2. Related Work

- ✓ CNN and variants
  - ➢ Served as the standard network model throughout computer vision.
  - ➢ VGG, GoogleNet, ResNet, DenseNet …
- ✓ Self-attention based backbone architectures
  - ➢ some works employ self-attention layers to replace some or all of the spatial convolution layers in the popular ResNet.
- ✓ Self-attention/Transformers to complement CNNs
- ✓ Transformer based vision backbones
  - ➢ Vision Transformer(ViT)
  - ➢ An image is 11 worth 16x16 words: Transformers for image recognition at scale(2021)

3. Method

❖ Structure

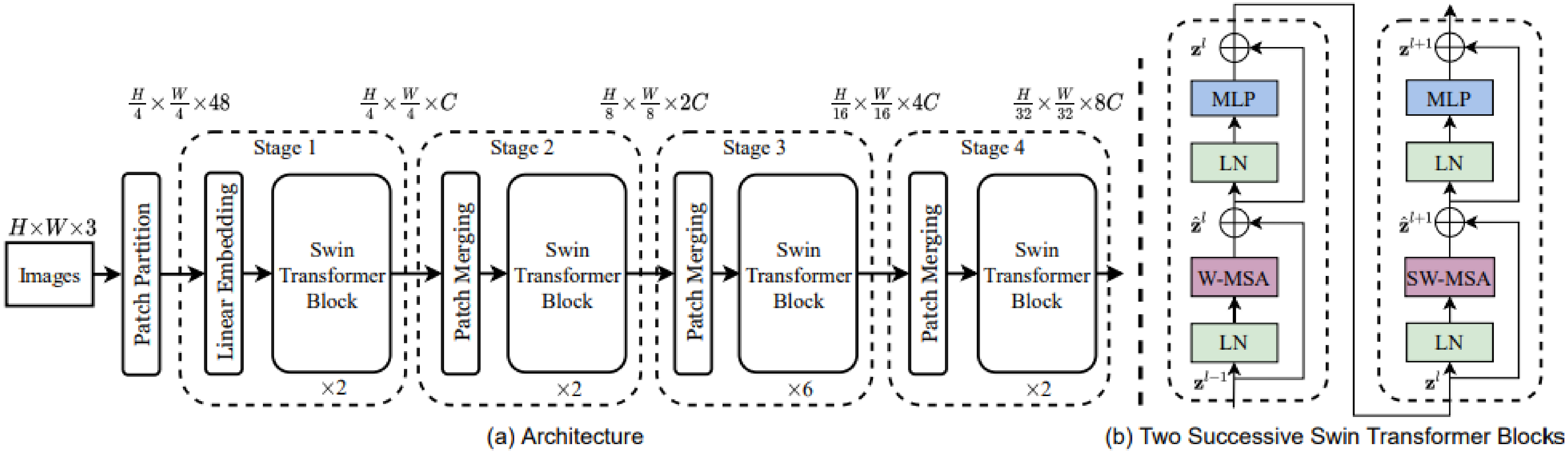
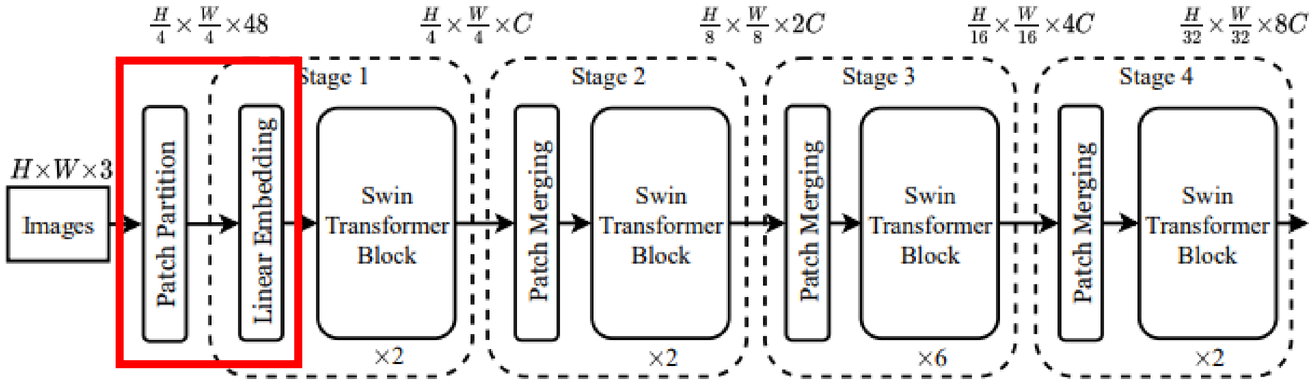


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

3. Method

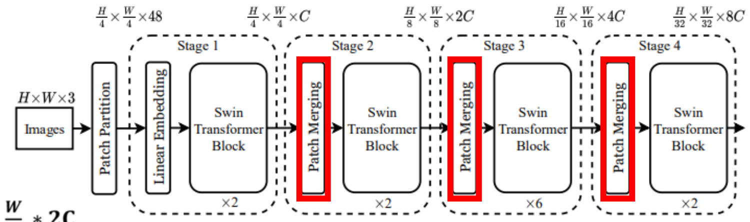
❖ Patch Partition + Linear Embedding

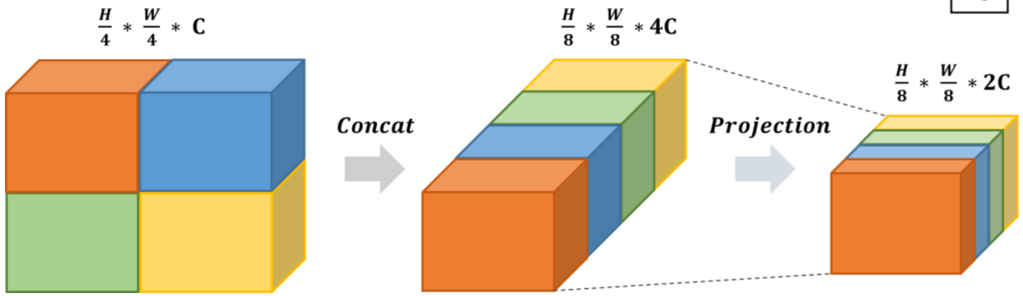


- ✓ It first splits an input RGB image into non-overlapping, like ViT.
- ✓ Used a patch size of 4 × 4(in Tiny).
  - ➢ Thus the feature dimension of each patch is 4 × 4 × 3 = 48
- ✓ A linear embedding layer is applied on this raw-valued feature to project it to an arbitrary dimension.

3. Method

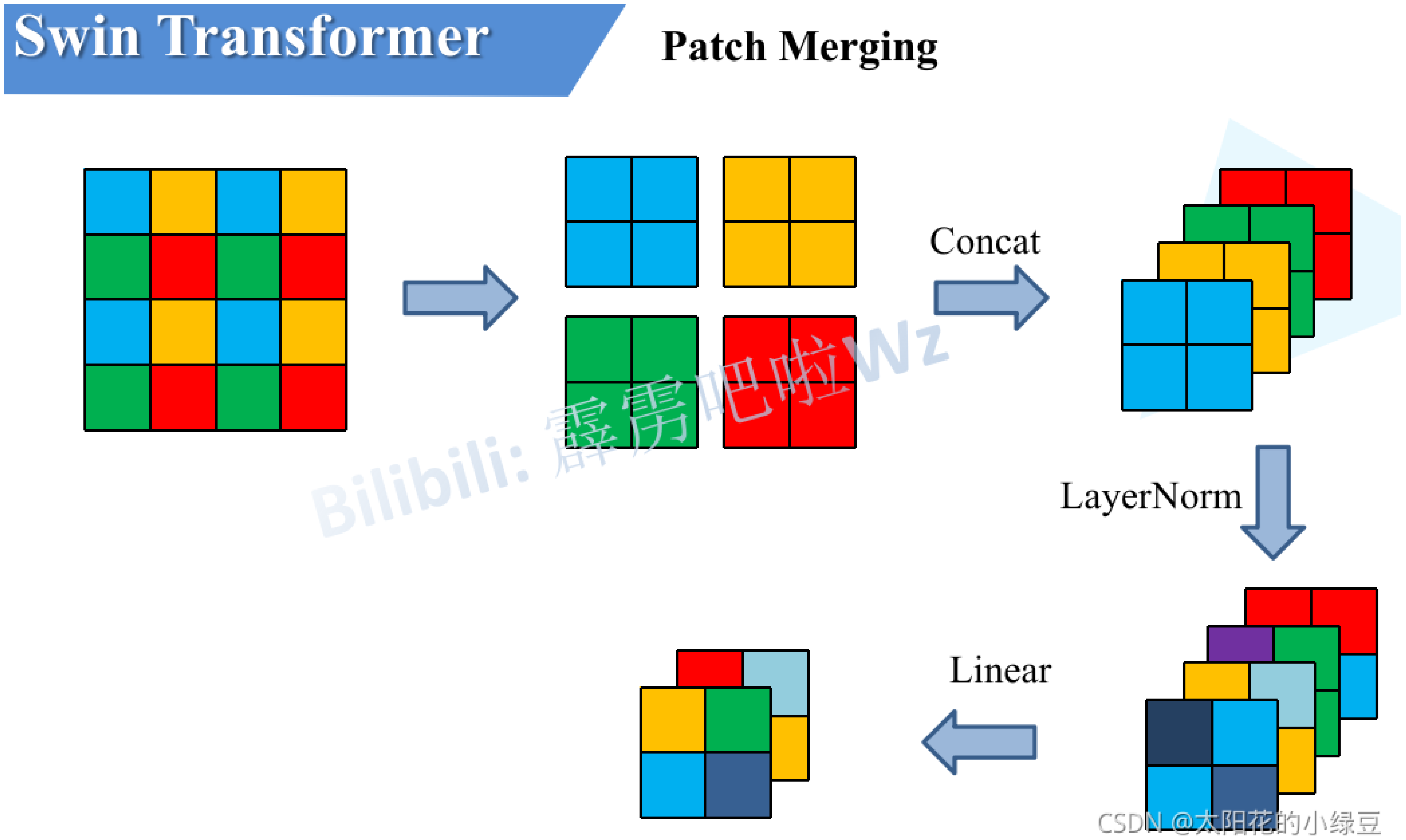
❖ Patch Merging





- ✓ The first patch merging layer concatenates the features of each group of 2 × 2 neighboring patches and applies a linear layer on the 4C-dimensional concatenated features.
- ✓ This reduces the number of tokens by a multiple of 2×2 = 4 (2× downsampling of resolution), and the output dimension is set to 2C.
- ✓ To produce a hierarchical representation, the number of tokens is reduced by patch merging layers as the network gets deeper.

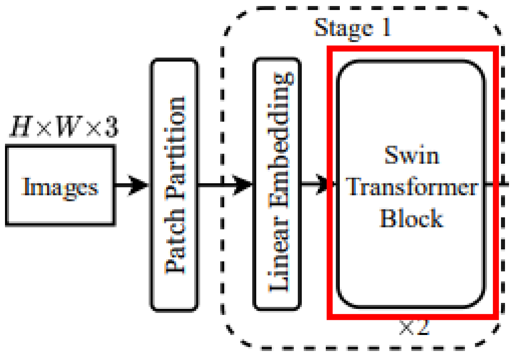
3. Method

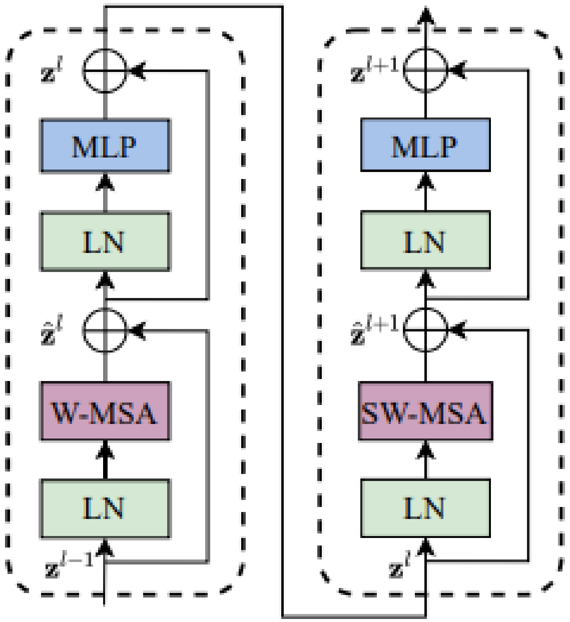


이미지 출처: https://heeya-stupidbutstudying.tistory.com/

3. Method

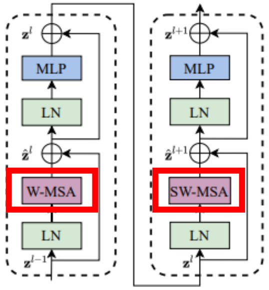
❖ Swin Transformer Block





$$\hat{\mathbf{z}}^{l} = \text{W-MSA}\left(\text{LN}\left(\mathbf{z}^{l-1}\right)\right) + \mathbf{z}^{l-1},$$
$$\mathbf{z}^{l} = \text{MLP}\left(\text{LN}\left(\hat{\mathbf{z}}^{l}\right)\right) + \hat{\mathbf{z}}^{l},$$
$$\hat{\mathbf{z}}^{l+1} = \text{SW-MSA}\left(\text{LN}\left(\mathbf{z}^{l}\right)\right) + \mathbf{z}^{l},$$
$$\mathbf{z}^{l+1} = \text{MLP}\left(\text{LN}\left(\hat{\mathbf{z}}^{l+1}\right)\right) + \hat{\mathbf{z}}^{l+1},$$

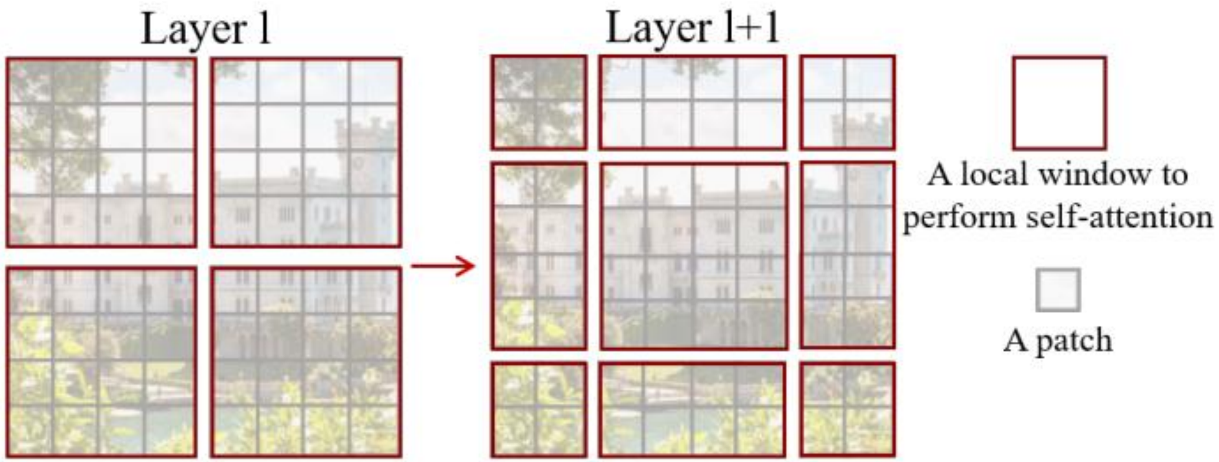
3. Method



❖ W-MSA & SW-MSA

✓ W-MSA: Local Window 안에서 self attention

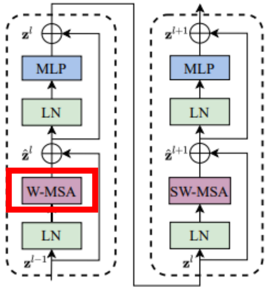
✓ SW-MSA: Local Window 간의 self attention



W-MSA SW-MSA

서울여자대학교 소프트웨어융합학과 김세희

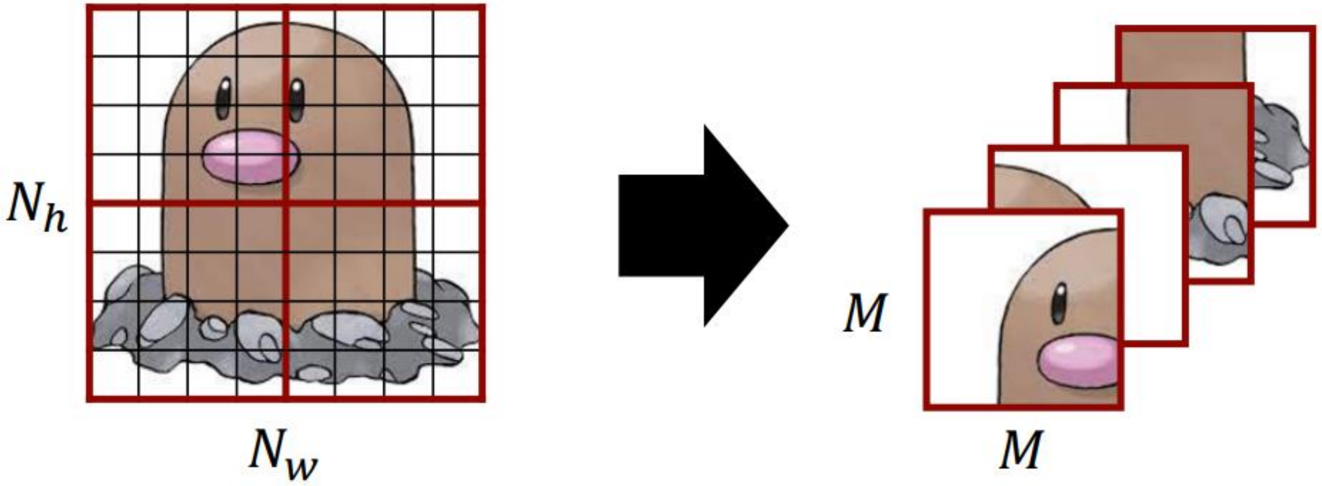
3. Method



❖ Efficient batch computation

$x \in \mathbb{R}^{B \times N_h \times N_w \times C}$

$x \in \mathbb{R}^{nB \times M \times M \times C}, n = \frac{N_h}{M} + \frac{N_w}{M}$



$x \in \mathbb{R}^{1 \times 8 \times 8 \times 3}$

$(batch * patch_h * patch_w * channel)$

$x \in \mathbb{R}^{4 \times 4 \times 4 \times 3}$

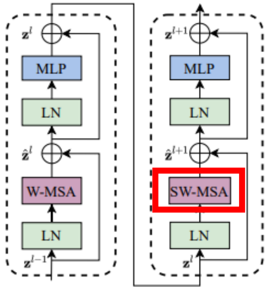
$M = 4, n = 4$

$$\left(batch' * \frac{patch_h}{M} * \frac{patch_w}{M} * channel\right)$$

$$batch' = batch * \frac{patch_h}{m} * \frac{patch_w}{M}$$

이미지 출처: https://velog.io/@201820894/posts
https://youtu.be/2IZvuU_lIMA?si=1ea0MWvUK4IzXQZO

3. Method



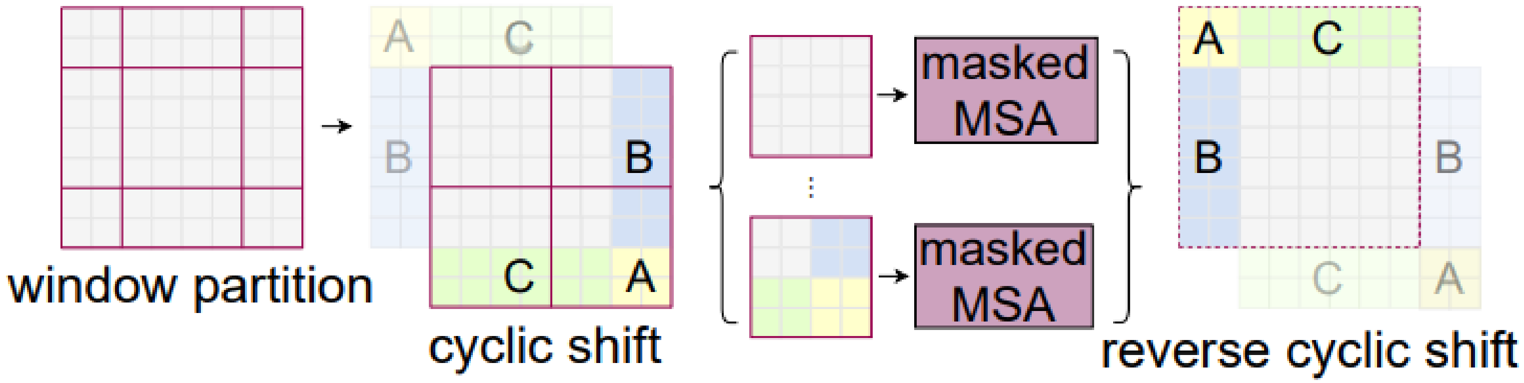
❖ SW-MSA

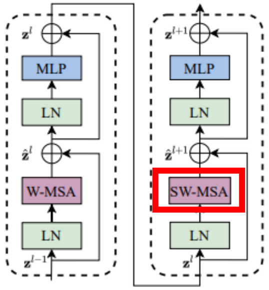
- ✓ Local Window 간의 self attention
- ✓ SW-MSA 수행 시에 window 개수가 H, W 별로 1개 씩 늘어나게 됨
    - ➢ 더 많은 window 사용으로 인한 비효율적인 연산 수행

❖ Cyclic Shift + Attention Mask를 통한 W-MSA와 동일한 window 개수 사용

3. Method

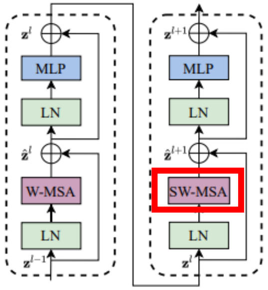
❖ Cyclic shift & Attention Mask



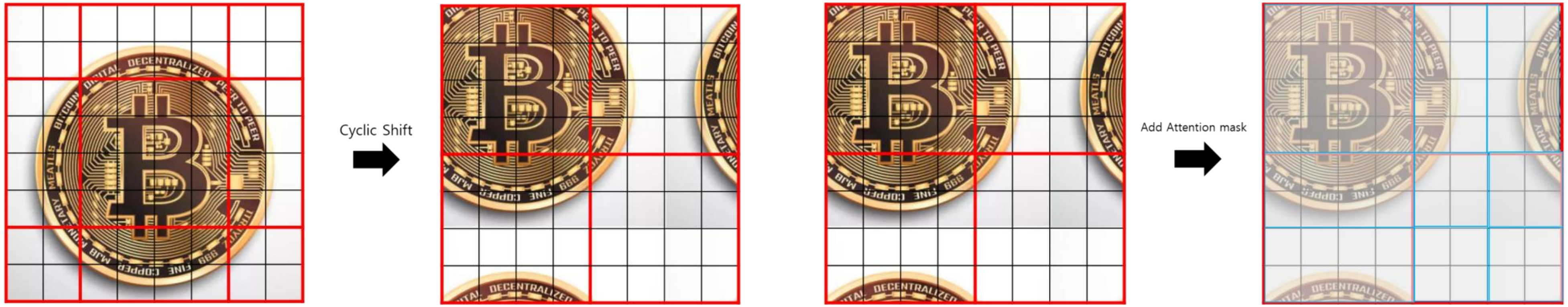


- ✓ SW-MSA를 수행하여 나뉘어진 window로 구성되어야 했던 것을 일정 크기 만큼 shift한다.
  - ➢ $Shift\ size\ =\ Window\ size(M)\ //\ 2$
- ✓ 색이 다른 부분 모두 각각 attention이 적용되어야 하기 때문에 mask를 적용한다.
  - ✓ Why? 실제로 색이 다른 부분은 이미지 상에서 인접한 부분이 아니기 때문이다.
- ✓ W-MSA와 동일하게 4개의 window만 사용하여 local window간에 attention을 계산한다.
- ✓ 계산 결과에 다시 shift를 적용하여 결과를 복원한다.

3. Method



❖ Cyclic shift & Attention Mask



<정리>

✓ Shift 후 W-MSA처럼 4개의 Window만 사용하여 Window 간의 Self Attention 수행

✓ 이후 Attention Mask를 씌워 Mask에서 따로 Attention을 적용한다.

➢ Relative bias 적용 후 Attention Mask가 적용됨

3. Method

❖ Relative position bias

$$\text{Attention}(Q,K,V) = \text{SoftMax}(QK^T/\sqrt{d} \boxed{+ B})V,$$

- ✓ In computing self-attention, ,we apply a relative position bias to each head in computing similarity
  - ➢ $Q, K, V \in R^{M^2 \times d}$: *query, key* and *value* matrices
  - ➢ d = query/key dimension
  - ➢ M^2: the number of patches in a window

3. Method

❖ Relative position bias

$$\mathrm{Attention}(Q,K,V) = \mathrm{SoftMax}(QK^T/\sqrt{d}\boxed{+B})V,$$

- ✓ 두 축 마다 Relative position의 범위: $[-M+1,\ M-1]$
- ✓ Bias Index Matrix: $\hat{B} \in R^{(2M-1)\times(2M-1)}$
- ✓ B는 $\hat{B}$의 값을 사용

3. Method

❖ Relative position bias

x axis

1	2
3	4

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

y axis

1	2
3	4

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

- ✓ 축 별로 상대적인 거리를 계산한다.
  - ➢ 1과 같은 축에 있는 값들은 0, 1칸 차이나면 1 or -1, 2칸 차이나면 – or -2
- ✓ 이렇게 구한 각 Matrix에 (window size – 1)값을 더해 준다.
  - ➢ 실제로 index로 나타내기 위해 범위가 0부터 시작되도록 변환하기 위함

3. Method

❖ Relative position bias

Relative Position Index

	1	2	3	4	5	6	7	8	9
1	12	11	10	7	6	5	2	1	0
2	13	12	11	8	7	6	3	2	1
3	14	13	12	9	8	7	4	3	2
4	17	16	15	12	11	10	7	6	5
5	18	17	16	13	12	11	8	7	6
6	19	18	17	14	13	12	9	8	7
7	22	21	20	17	16	15	12	11	10
8	23	22	21	18	17	16	13	12	11
9	24	23	22	19	18	17	14	13	12

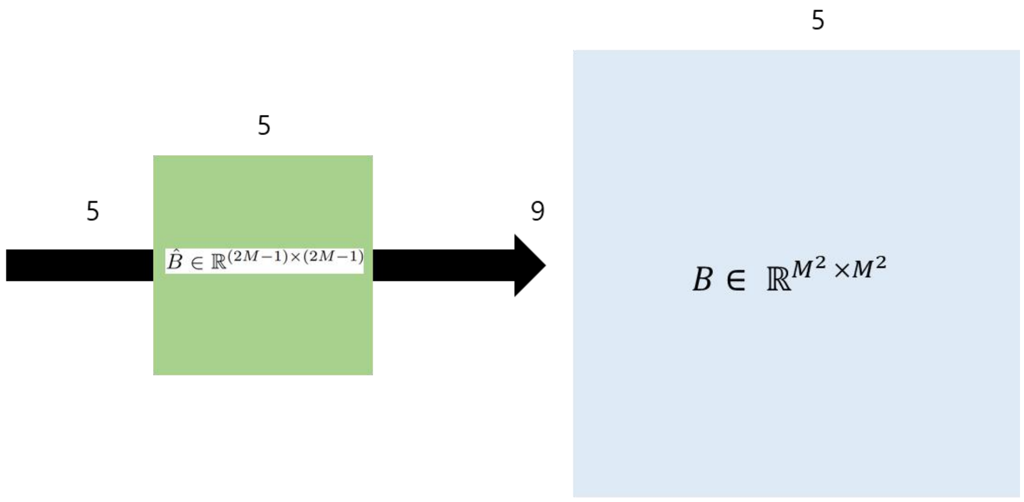
- ✓ $x\_axis\_matrix \mathrel{*}= 2 * window\_size - 1$
- ✓ $relative\_position\_matrix = x\_axis\_matrix + y\_axis\_matrix$
- ✓ 이 행렬의 크기는 $(2M - 1) * (2M - 1)$

3. Method

❖ Relative position bias

Relative Position Index

	1	2	3	4	5	6	7	8	9
1	12	11	10	7	6	5	2	1	0
2	13	12	11	8	7	6	3	2	1
3	14	13	12	9	8	7	4	3	2
4	17	16	15	12	11	10	7	6	5
5	18	17	16	13	12	11	8	7	6
6	19	18	17	14	13	12	9	8	7
7	22	21	20	17	16	15	12	11	10
8	23	22	21	18	17	16	13	12	11
9	24	23	22	19	18	17	14	13	12



✓ 이렇게 만든 Relative position index를 $\hat{B}$에서 값을 조회하여 B라는 행렬을 구성하게 된다.

✓ B를 Attention 수식에 적용한다.

4. Experiments

(a) Regular ImageNet-1K trained models

method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
RegNetY-4G [48]	224^2	21M	4.0G	1156.7	80.0
RegNetY-8G [48]	224^2	39M	8.0G	591.6	81.7
RegNetY-16G [48]	224^2	84M	16.0G	334.7	82.9
EffNet-B3 [58]	300^2	12M	1.8G	732.1	81.6
EffNet-B4 [58]	380^2	19M	4.2G	349.4	82.9
EffNet-B5 [58]	456^2	30M	9.9G	169.1	83.6
EffNet-B6 [58]	528^2	43M	19.0G	96.9	84.0
EffNet-B7 [58]	600^2	66M	37.0G	55.1	84.3
ViT-B/16 [20]	384^2	86M	55.4G	85.9	77.9
ViT-L/16 [20]	384^2	307M	190.7G	27.3	76.5
DeiT-S [63]	224^2	22M	4.6G	940.4	79.8
DeiT-B [63]	224^2	86M	17.5G	292.3	81.8
DeiT-B [63]	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 [38]	384^2	388M	204.6G	-	84.4
R-152x4 [38]	480^2	937M	840.5G	-	85.4
ViT-B/16 [20]	384^2	86M	55.4G	85.9	84.0
ViT-L/16 [20]	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

Table 1. Comparison of different backbones on ImageNet-1K classification. Throughput is measured using the GitHub repository of [68] and a V100 GPU, following [63].

4. Experiments

(a) Various frameworks

Method	Backbone	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	#param.	FLOPs	FPS
Cascade Mask R-CNN	R-50	46.3	64.3	50.5	82M	739G	18.0
	Swin-T	50.5	69.3	54.9	86M	745G	15.3
ATSS	R-50	43.5	61.9	47.0	32M	205G	28.3
	Swin-T	47.2	66.5	51.3	36M	215G	22.3
RepPointsV2	R-50	46.5	64.6	50.3	42M	274G	13.6
	Swin-T	50.0	68.5	54.2	45M	283G	12.0
Sparse R-CNN	R-50	44.5	63.4	48.2	106M	166G	21.0
	Swin-T	47.9	67.3	52.3	110M	172G	18.4

(b) Various backbones w. Cascade Mask R-CNN

	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}	param	FLOPs	FPS
DeiT-S†	48.0	67.2	51.7	41.4	64.2	44.3	80M	889G	10.4
R50	46.3	64.3	50.5	40.1	61.7	43.4	82M	739G	18.0
Swin-T	50.5	69.3	54.9	43.7	66.6	47.1	86M	745G	15.3
X101-32	48.1	66.5	52.4	41.6	63.9	45.2	101M	819G	12.8
Swin-S	51.8	70.4	56.3	44.7	67.9	48.5	107M	838G	12.0
X101-64	48.3	66.4	52.3	41.7	64.0	45.1	140M	972G	10.4
Swin-B	51.9	70.9	56.5	45.0	68.4	48.7	145M	982G	11.6

(c) System-level Comparison

Method	mini-val		test-dev		#param.	FLOPs
	AP^{box}	AP^{mask}	AP^{box}	AP^{mask}		
RepPointsV2* [12]	-	-	52.1	-	-	-
GCNet* [7]	51.8	44.7	52.3	45.4	-	1041G
RelationNet++* [13]	-	-	52.7	-	-	-
SpineNet-190 [21]	52.6	-	52.8	-	164M	1885G
ResNeSt-200* [78]	52.5	-	53.3	47.1	-	-
EfficientDet-D7 [59]	54.4	-	55.1	-	77M	410G
DetectoRS* [46]	-	-	55.7	48.5	-	-
YOLOv4 P7* [4]	-	-	55.8	-	-	-
Copy-paste [26]	55.9	47.2	56.0	47.4	185M	1440G
X101-64 (HTC++)	52.3	46.0	-	-	155M	1033G
Swin-B (HTC++)	56.4	49.1	-	-	160M	1043G
Swin-L (HTC++)	57.1	49.5	57.7	50.2	284M	1470G
Swin-L (HTC++)*	**58.0**	**50.4**	**58.7**	**51.1**	284M	-

Table 2. Results on COCO object detection and instance segmentation. †denotes that additional decovolution layers are used to produce hierarchical feature maps. * indicates multi-scale testing.

4. Experiments

ADE20K Method	Backbone	val mIoU	test score	#param.	FLOPs	FPS
DANet [23]	ResNet-101	45.2	-	69M	1119G	15.2
DLab.v3+ [11]	ResNet-101	44.1	-	63M	1021G	16.0
ACNet [24]	ResNet-101	45.9	38.5	-		
DNL [71]	ResNet-101	46.0	56.2	69M	1249G	14.8
OCRNet [73]	ResNet-101	45.3	56.0	56M	923G	19.3
UperNet [69]	ResNet-101	44.9	-	86M	1029G	20.1
OCRNet [73]	HRNet-w48	45.7	-	71M	664G	12.5
DLab.v3+ [11]	ResNeSt-101	46.9	55.1	66M	1051G	11.9
DLab.v3+ [11]	ResNeSt-200	48.4	-	88M	1381G	8.1
SETR [81]	T-Large‡	50.3	61.7	308M	-	-
UperNet	DeiT-S†	44.0	-	52M	1099G	16.2
UperNet	Swin-T	46.1	-	60M	945G	18.5
UperNet	Swin-S	49.3	-	81M	1038G	15.2
UperNet	Swin-B‡	51.6	-	121M	1841G	8.7
UperNet	Swin-L‡	**53.5**	**62.8**	234M	3230G	6.2

Table 3. Results of semantic segmentation on the ADE20K val and test set. † indicates additional deconvolution layers are used to produce hierarchical feature maps. ‡ indicates that the model is pre-trained on ImageNet-22K.

5. Ablation Study

	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).

5. Ablation Study

method	MSA in a stage (ms)				Arch. (FPS)		
	S1	S2	S3	S4	T	S	B
sliding window (naive)	122.5	38.3	12.1	7.6	183	109	77
sliding window (kernel)	7.6	4.7	2.7	1.8	488	283	187
Performer [14]	4.8	2.8	1.8	1.5	638	370	241
window (w/o shifting)	2.8	1.7	1.2	0.9	770	444	280
shifted window (padding)	3.3	2.3	1.9	2.2	670	371	236
shifted window (cyclic)	3.0	1.9	1.3	1.0	755	437	278

Table 5. Real speed of different self-attention computation methods and implementations on a V100 GPU.

	Backbone	ImageNet		COCO		ADE20k
		top-1	top-5	AP^{box}	AP^{mask}	mIoU
sliding window	Swin-T	81.4	95.6	50.2	43.5	45.8
Performer [14]	Swin-T	79.0	94.2	-	-	-
shifted window	Swin-T	81.3	95.6	50.5	43.7	46.1

Table 6. Accuracy of Swin Transformer using different methods for self-attention computation on three benchmarks.

5. Conclusion

- ✓ This paper presents Swin Transformer, a new vision Transformer which produces a hierarchical feature representation.
- ✓ Swin Transformer achieves the state-of-the-art performance on COCO object detection and ADE20K semantic segmentation
- ✓ As a key element of Swin Transformer, the shifted window based self-attention is shown to be effective and efficient on vision problems, and we look forward to investigating its use in natural language processing as well.