Tech Interview - Paper Review

Dec 19, 2022

Swin Transformer:
Hierarchical Vision Transformer using Shifted Windows

발표자 : 이주영

목차

1. Transformer 등장 배경

2. Attention For Transformer

3. ViT

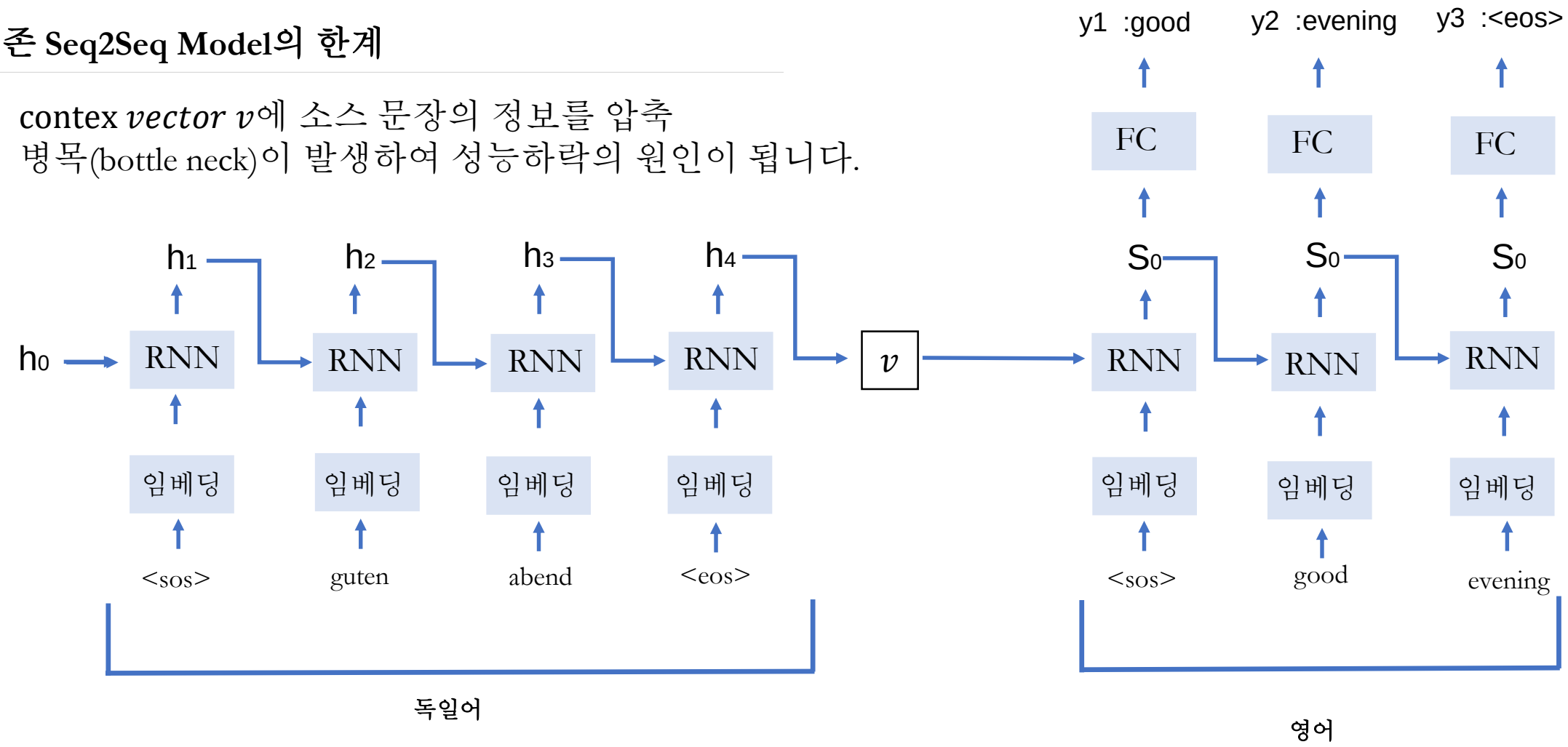
4. Swin Transformer

5. 결론 및 기대 효과

Transformer 등장 배경

기존 Seq2Seq Model의 한계

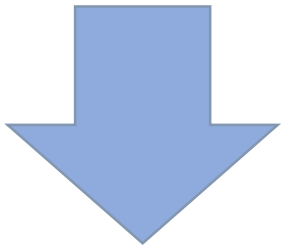
- contex *vector* v에 소스 문장의 정보를 압축
- 병목(bottle neck)이 발생하여 성능하락의 원인이 됩니다.



Transformer 등장 배경

문제 상황

- 하나의 문맥 벡터가 소스 문장의 모든 정보를 가지고 있어야 하므로 성능이 저하됨

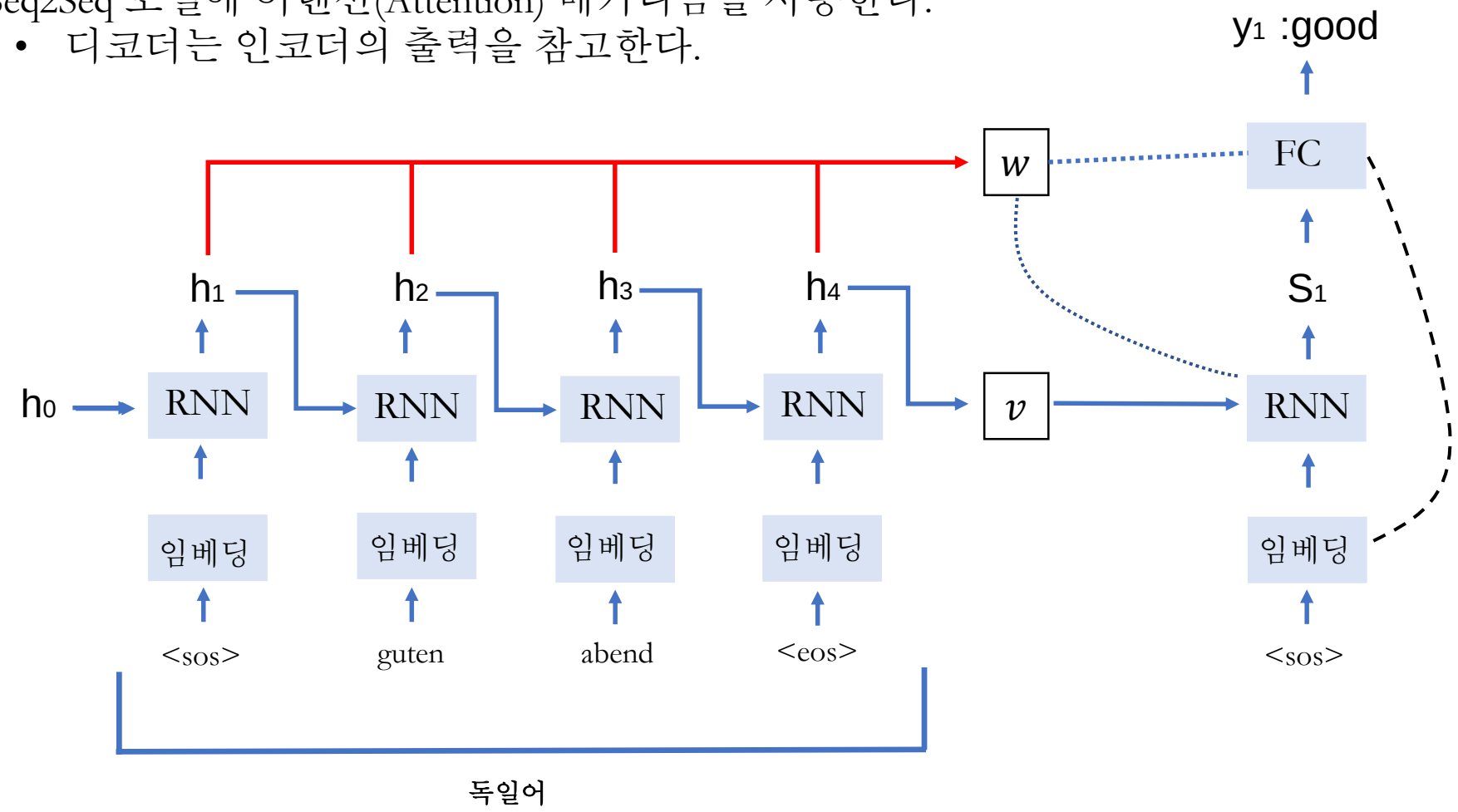


- 그렇다면 매번 소스 문장에서 출력 전부를 입력으로 받으면 어떨까?
  - 최신 하드웨어 Resource는 많은 메모리와 빠른 병렬 처리를 지원한다.

Transformer 등장 배경

Attention with Seq2Seq

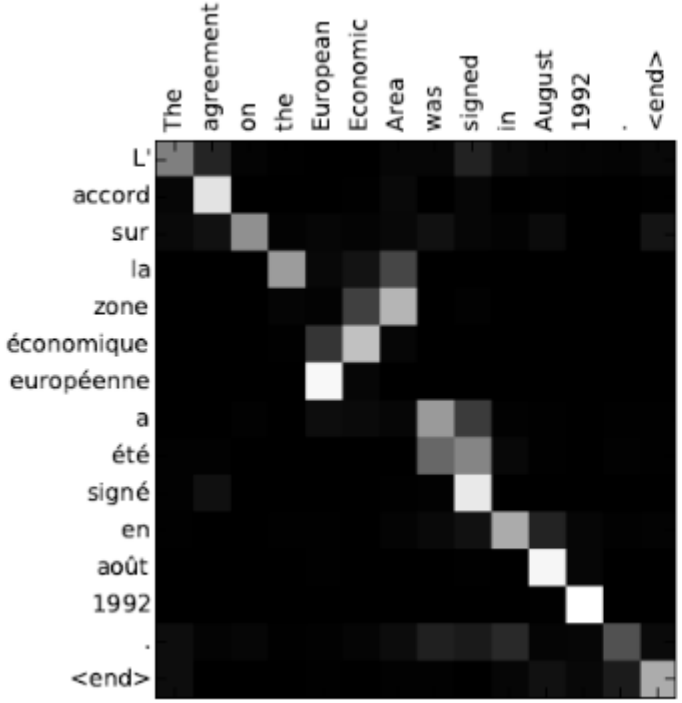
- Seq2Seq 모델에 어텐션(Attention) 매커니즘을 사용한다.
  - 디코더는 인코더의 출력을 참고한다.

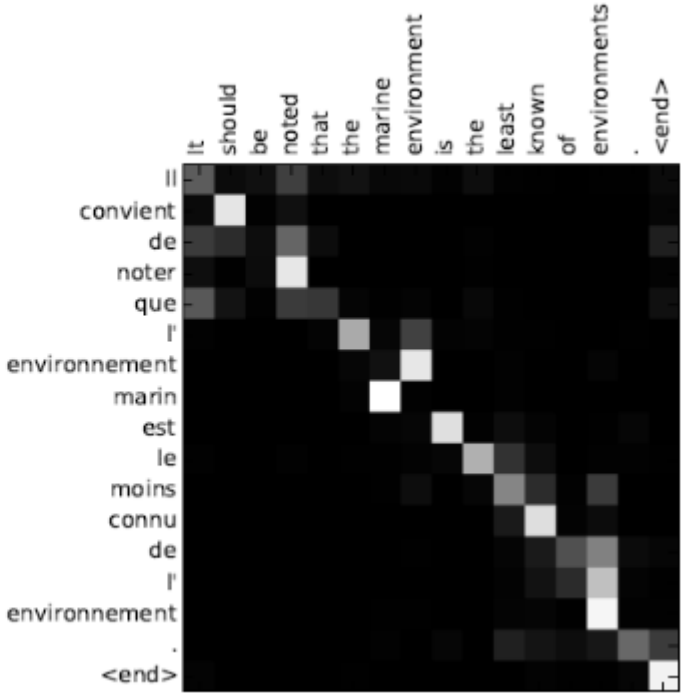


Transformer 등장 배경

문제 상황

- 어텐션(Attention) 가중치를 사용해 각 출력이 어떤 입력 정보를 참고했는지 알 수 있다.
  - French => English





Transformer 특징

Transformer

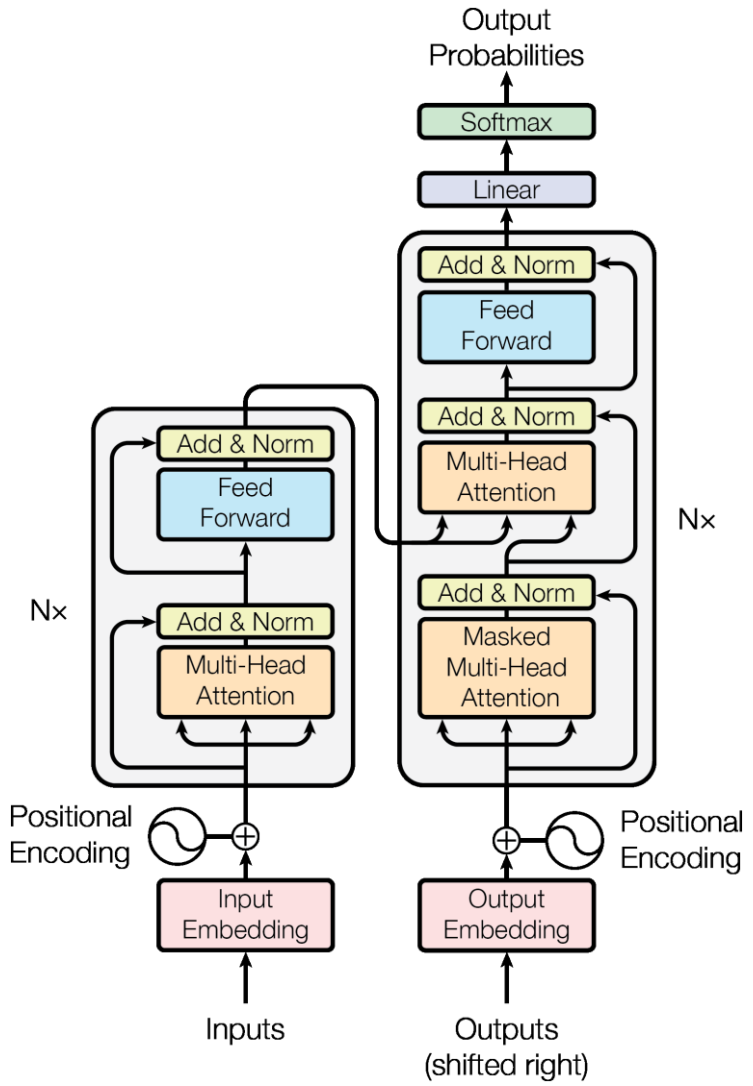
- 2022년 **기준으로 현대의 자연어 처리 네트워크에서 핵심이 되는 모델 구조입니다.**
  - **논문의** 원제목은 Attention Is All You Need이고, Attention Mechanism이 핵심이 됩니다.
  - 또한 RNN이나 CNN을 전혀 사용하지 않는 것이 가장 큰 특징

- GPT, BERT와 같은 향상된 네트워크에서도 사용하고 있다.
- 인코더와 디코더로 구성됩니다.
  - Attention 과정을 여러 레이어에서 반복

Transformer 특징

Transformer 기본 구조

- RNN을 사용하지 않고, 위치정보를 포함하고 있는 임베딩을 사용
  - 이를 위해서 Positional Encoding 사용
- 임베딩이 끝난 이후에 어텐션(Attention)을 진행함.
- 성능 향상을 위해 Residual Learning을 사용
- Attention과 Normalization 과정 반복
  - 각 Layer는 서로 다른 파라미터를 가짐



ViT

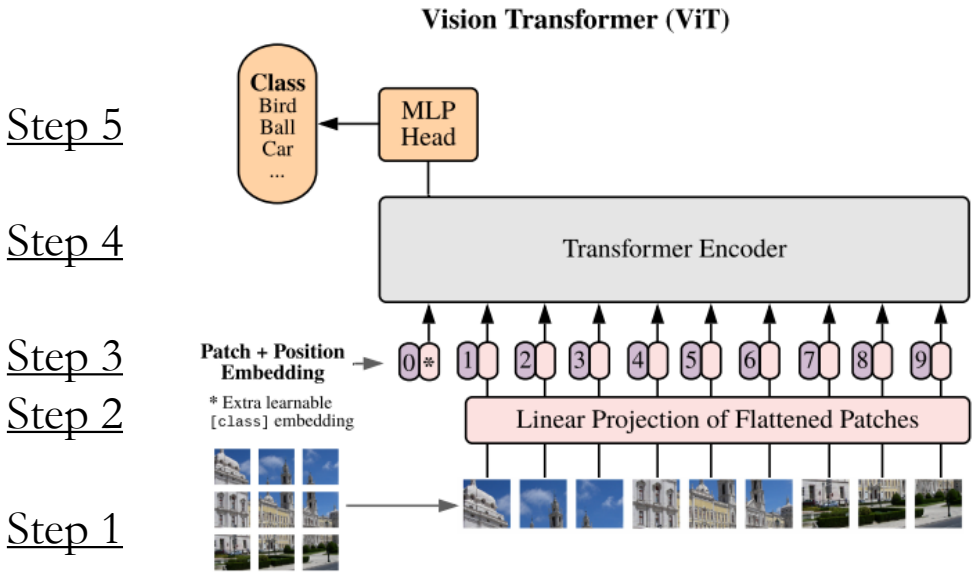
Attention In Vision Task

- Vision Transformer(ViT) 개요
  - 본 연구에서는 NLP에서 사용되는 Standard Transformer를 이미지에 그대로 적용하여 이미지 분류에 좋은 성능을 도출한 Vision Transformer(ViT)를 제안함
  - ViT는 이미지를 패치로 분할한 후, 이를 NLP의 단어(토큰)으로 취급해 각 패치의 Linear embedding을 순서대로 Transformer의 Input으로 넣어 이미지를 분류
  - ViT를 ImageNet-1k에 학습했을 때, 비슷한 크기의 ResNet 보다 낮은 정확도를 도출하는것으로 봤을 때, ViT가 CNN보다 Inductive Bias가 낮은 것을 알 수 있음
  - 반면, ImageNet-21K와 JFT-300M에 pretraining한 ViT를 다른 Image recognition task에 transfer learnin했을 때, ViT가 SOTA성능을 도출하는 것을 통해 Large Scale 학습이 낮은 Inductive bias로 인한 성능 저하를 해소시키는 것을 알 수 있음

ViT

ViT Architecture

- ViT의 작동 과정은 다음과 같음
  - Step 1. 이미지 x∈ R^{HxWxC}가 있을 때, (PxP) 크기의 패치 N(=HxW/P^2)개로 분할하여 패치 sequence $x_p \in R^{Nx(P^2 \cdot C)}$를 구축
  - Step 2. Trainable linear projection을 통해 x_p의 각 패치를 flatten한 벡터를 D차원으로 변환한 후, 이를 패치 임베딩으로 사용

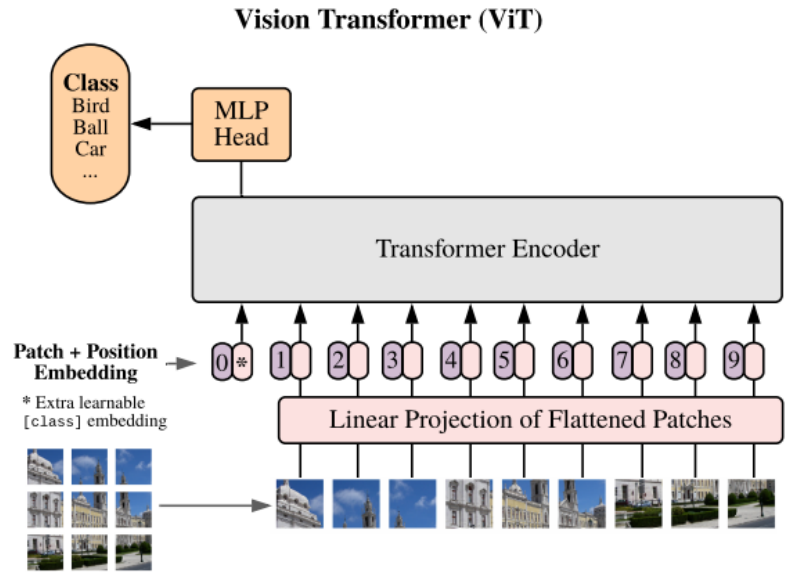


ViT

ViT Architecture

- ViT의 작동 과정은 다음과 같음
  - Step 3. learnable class임베딩과 패치 임베딩에 learnable position 임베딩을 더함
  - Step 4.임베딩을 더한 vanilla Transformer encoder에 input으로 넣어 마지막 layer에서 class embedding에 대한 outpu인 image representation 도출
  - Step 5. MLP에 image representation을 input으로 넣어 이미지의 class를 분류

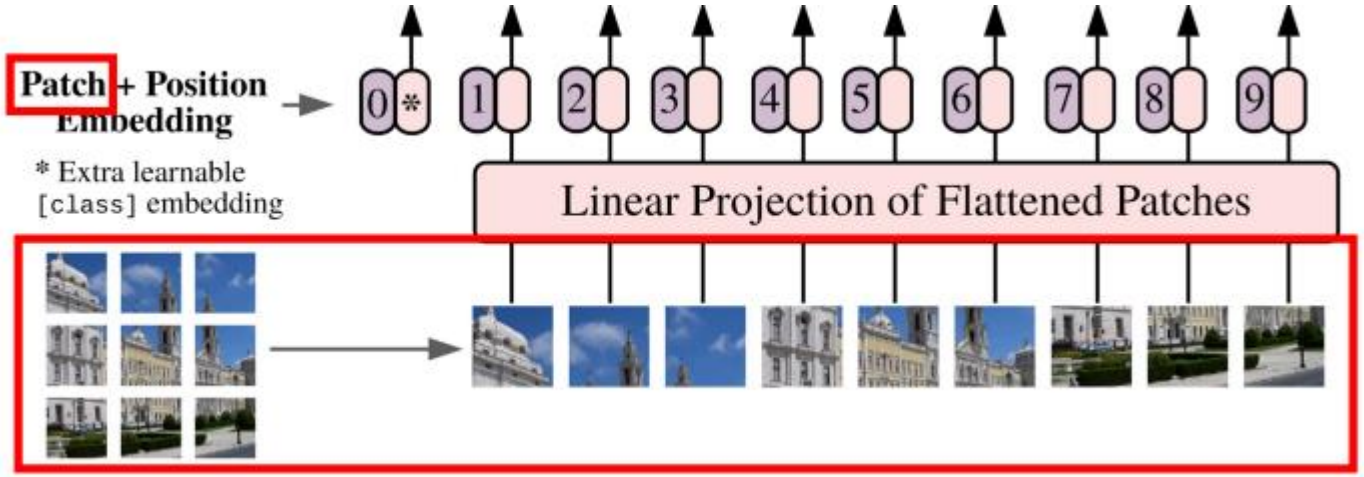
Step 5
Step 4
Step 3
Step 2
Step 1



ViT

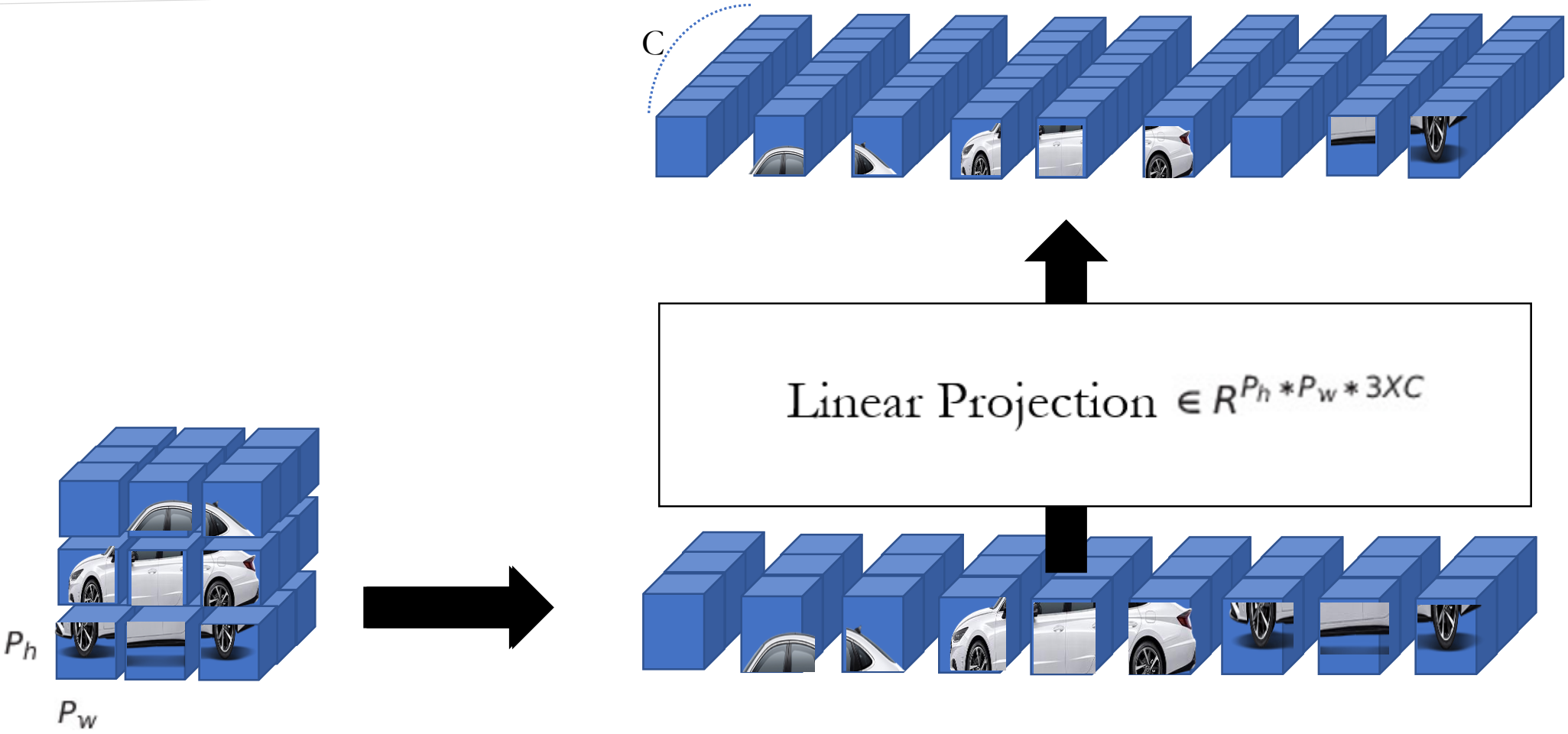
ViT 동작원리

- Flatten된 각 패치들을 linear projection을 통해서 원하는 차원으로 바꿔주는 작업이 존재



ViT

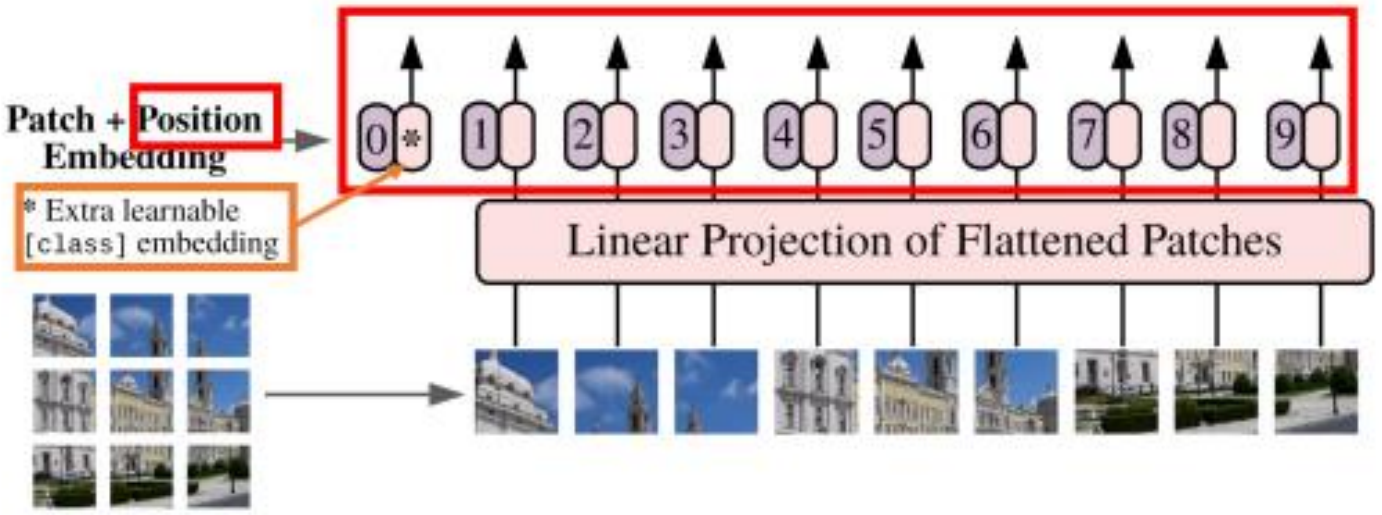
ViT 동작원리



ViT

ViT 동작원리

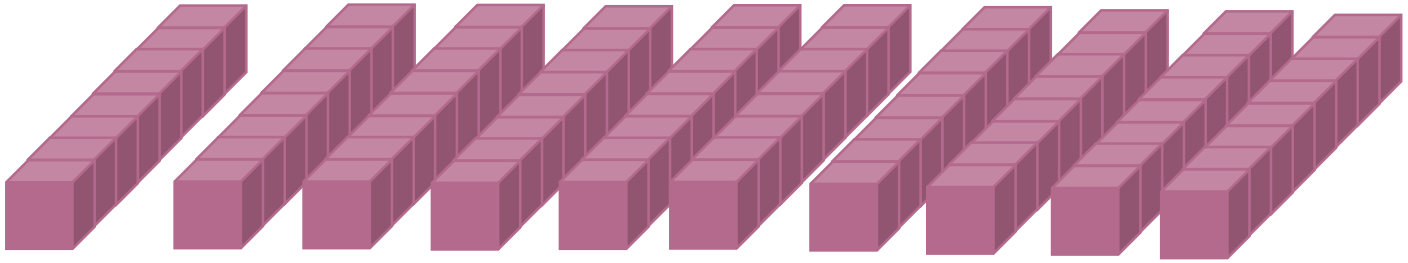
- Position Embedding을 적용하고, Class Embedding 토큰을 만듦



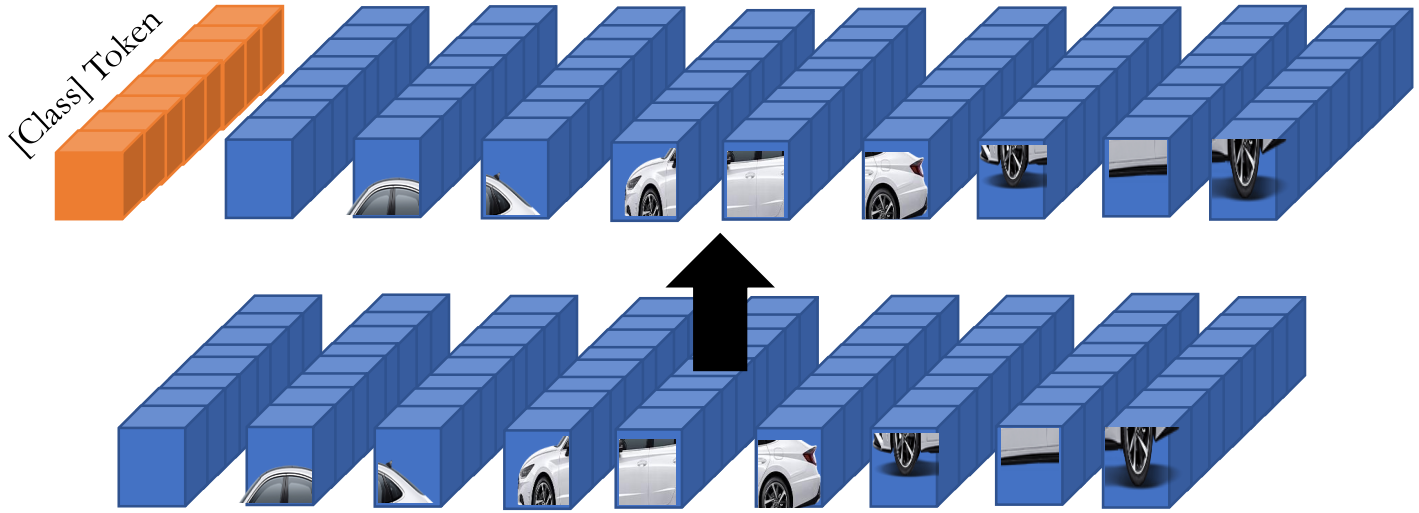
ViT

ViT 동작원리

Position Embedding



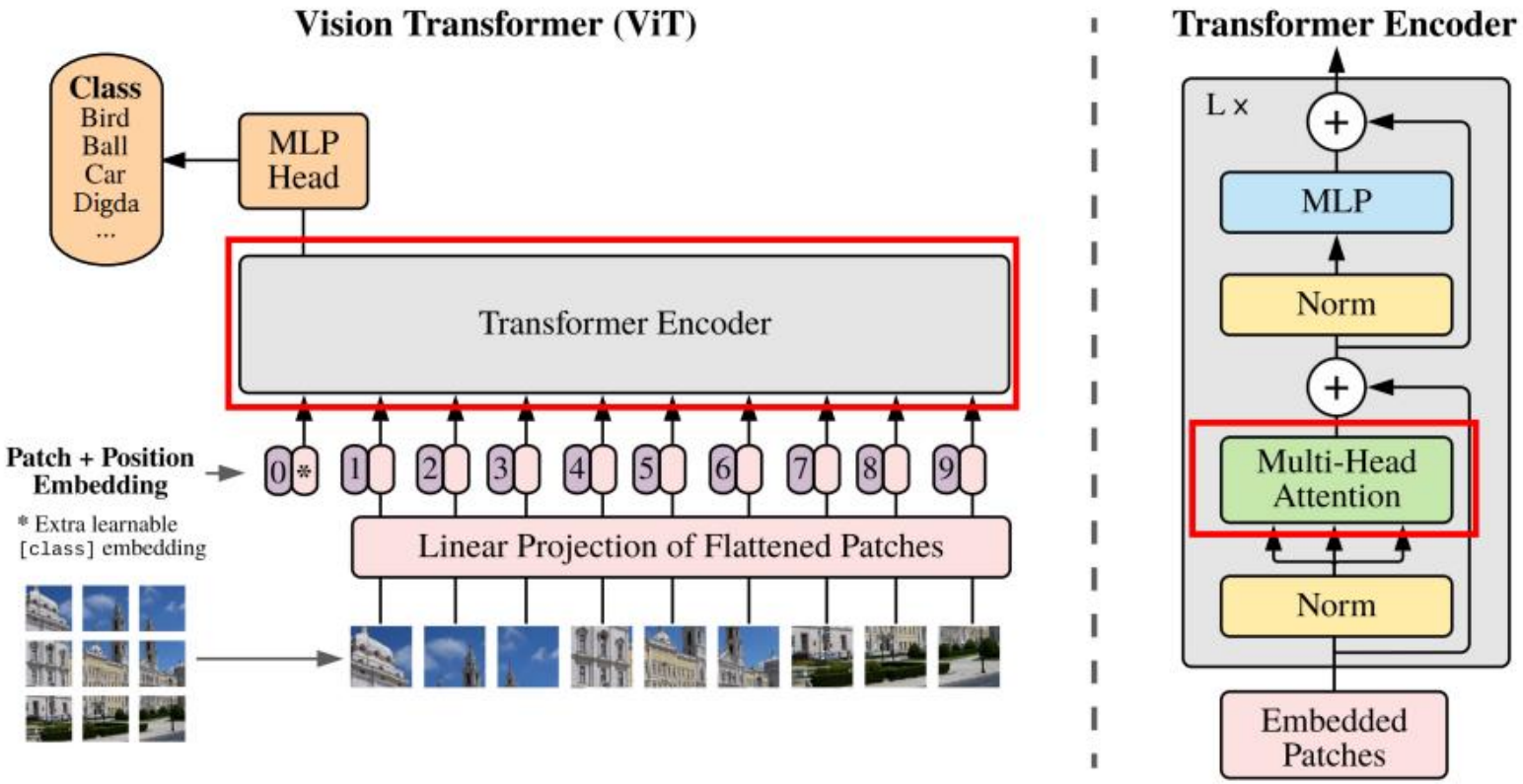
Patch Embedding



ViT

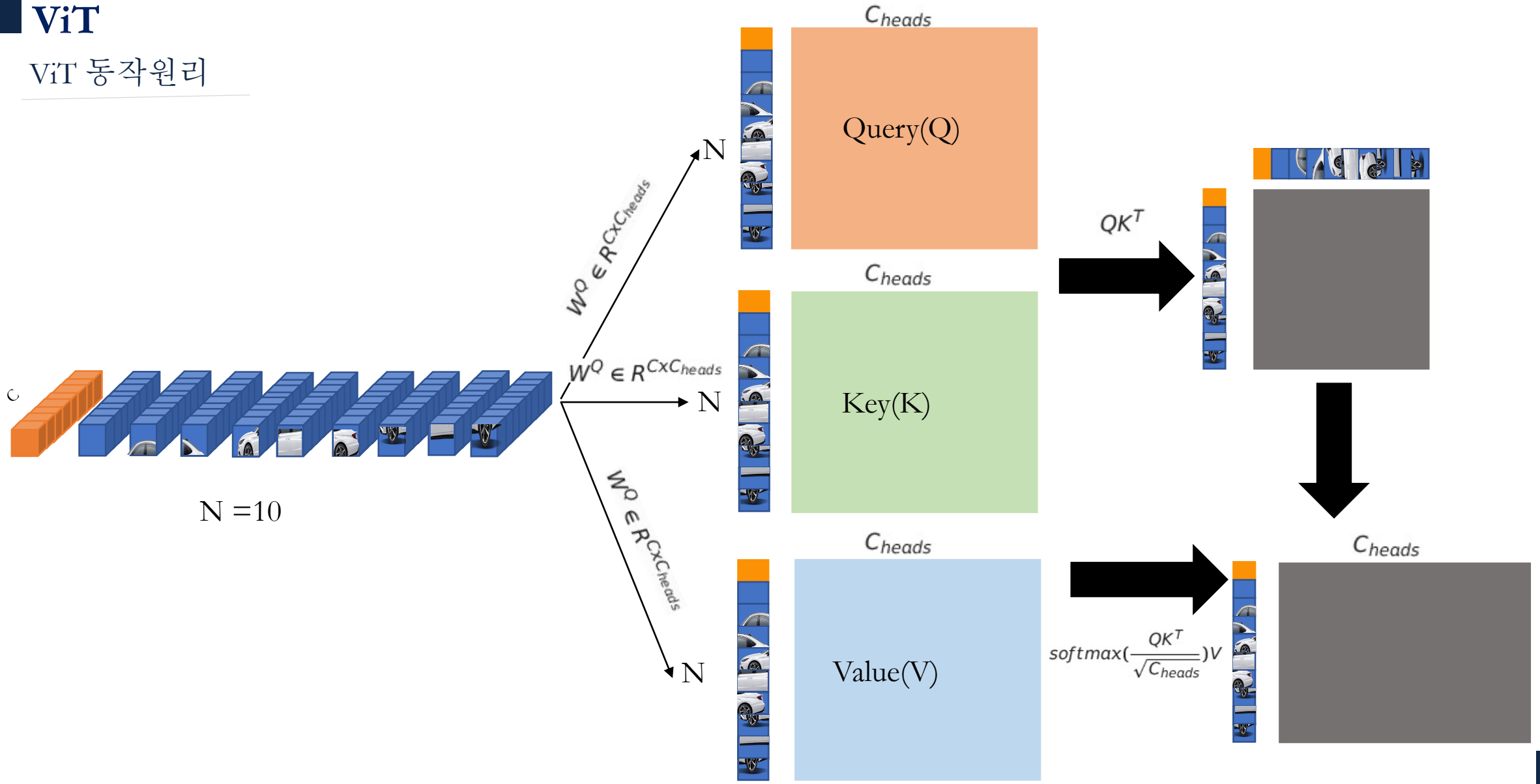
ViT 동작원리

- Transformer Encoder는 첫번째로 free layer normalization, multi-head attention, residual learning, layer normalization, MLP, residual learnin이 n번 반복



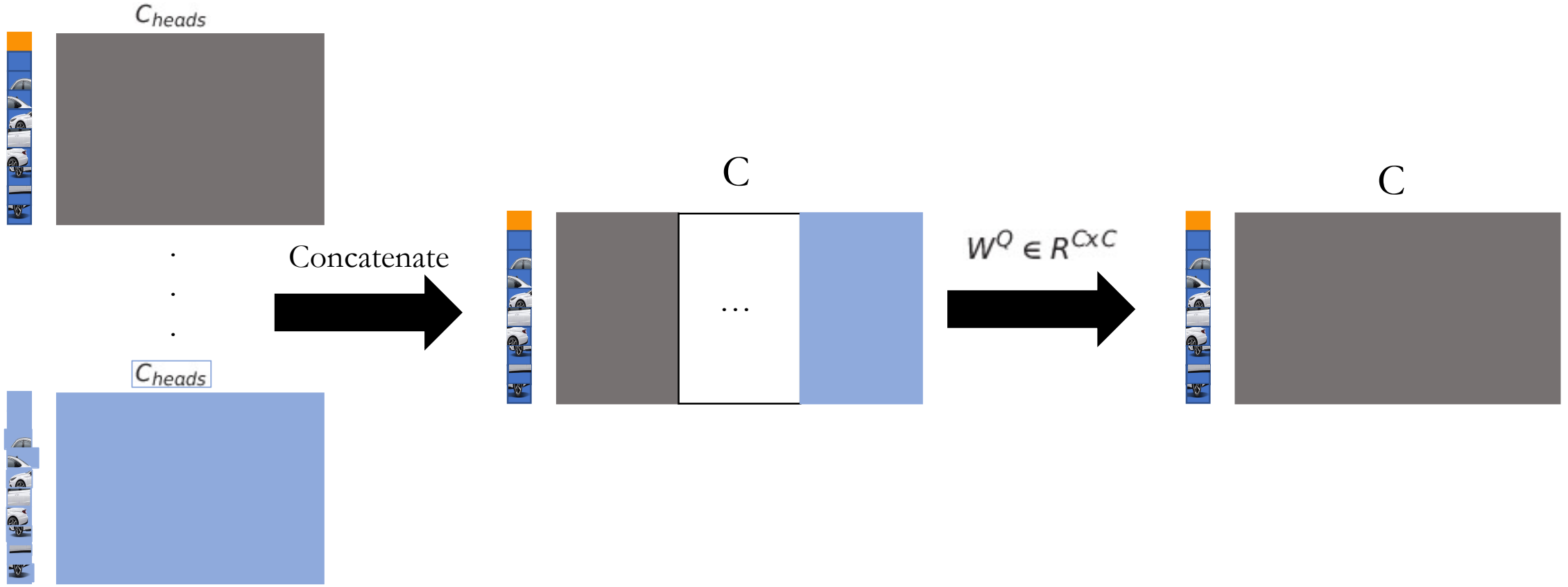
ViT

ViT 동작원리


C
N =10
$W^Q \in R^{CxC_{heads}}$
$W^Q \in R^{CxC_{heads}}$
$W^Q \in R^{CxC_{heads}}$
N
N
N
C_{heads}
Query(Q)
C_{heads}
Key(K)
C_{heads}
Value(V)
QK^T
$softmax(\frac{QK^T}{\sqrt{C_{heads}}})V$
C_{heads}

ViT

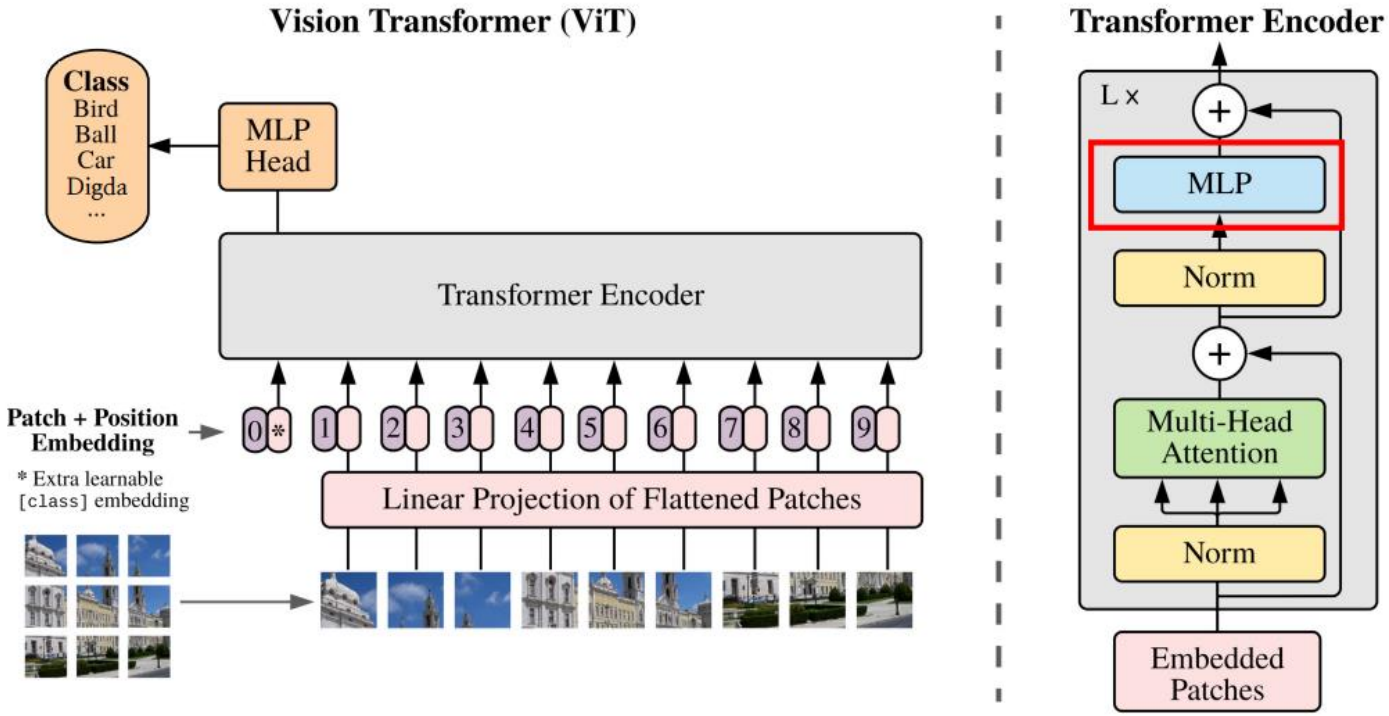
ViT 동작원리



ViT

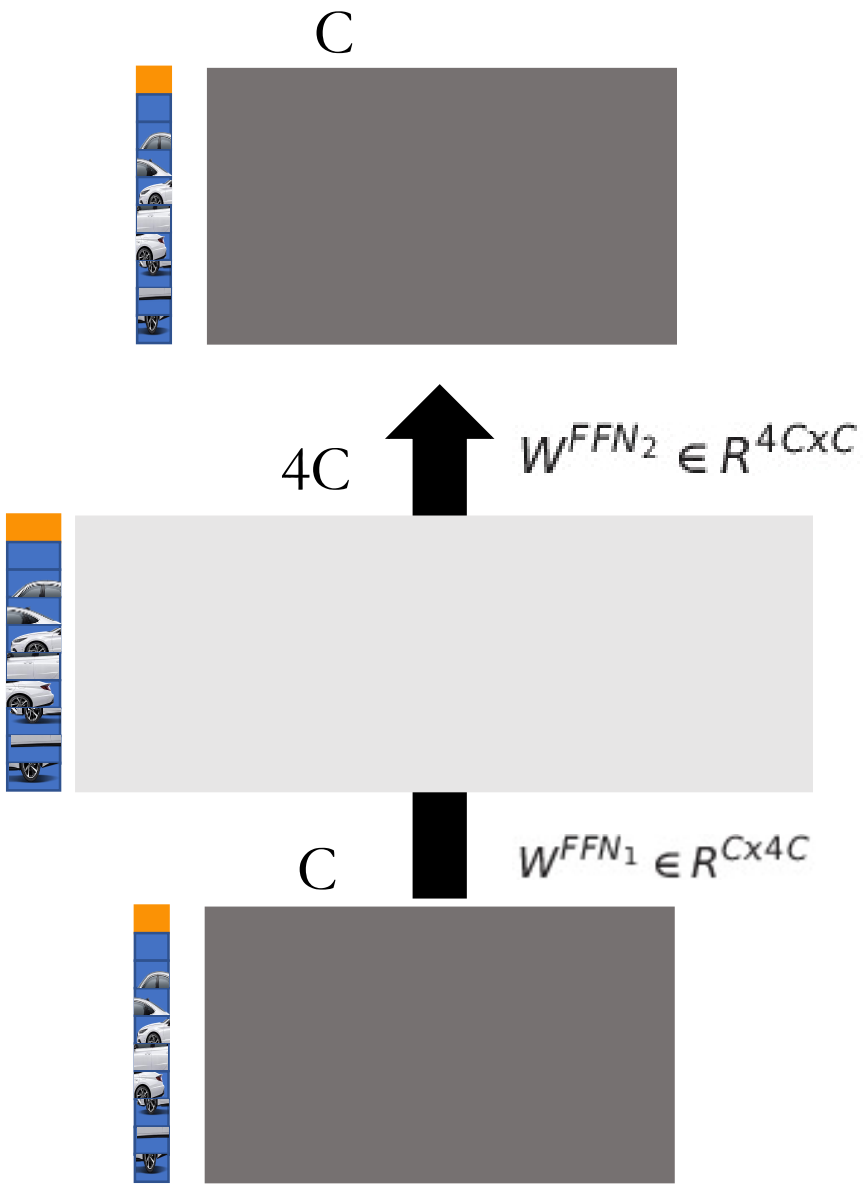
ViT 동작원리

- MLP는 C dimension을 4C dimension으로 변환 후 다시 C dimesion으로 사용하도록 하는 역할



ViT

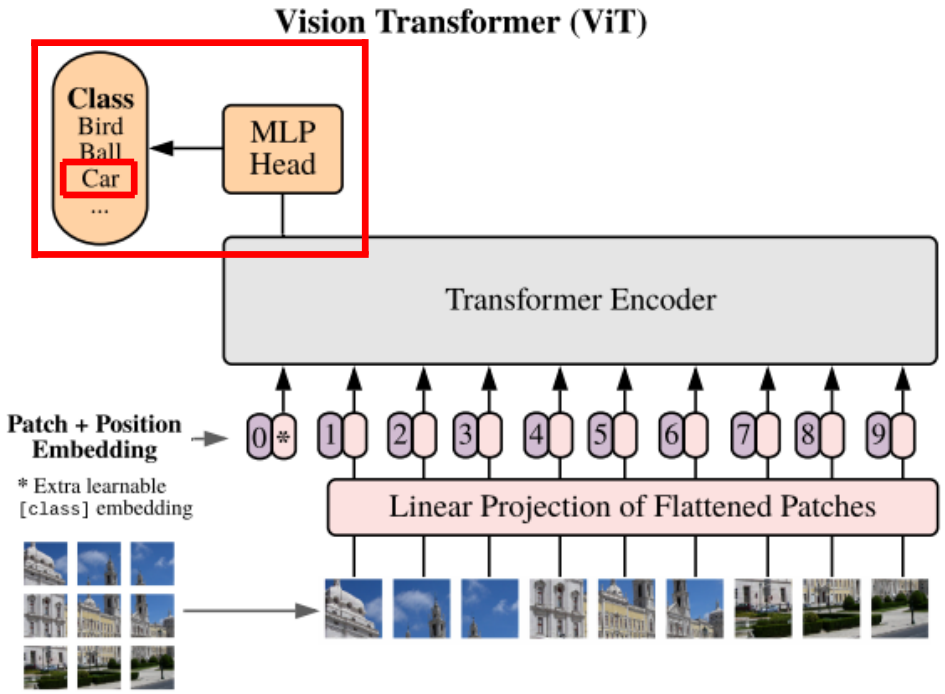
ViT 동작원리



ViT

ViT 동작원리

- 마지막으로 Token의 결과값 중에서 CLS(Special Classification Token)만을 사용해서 MLP Head에 적용해, 우리가 예상하는 정답으로 분류할 수 있도록 모델을 학습



Swin Transformer

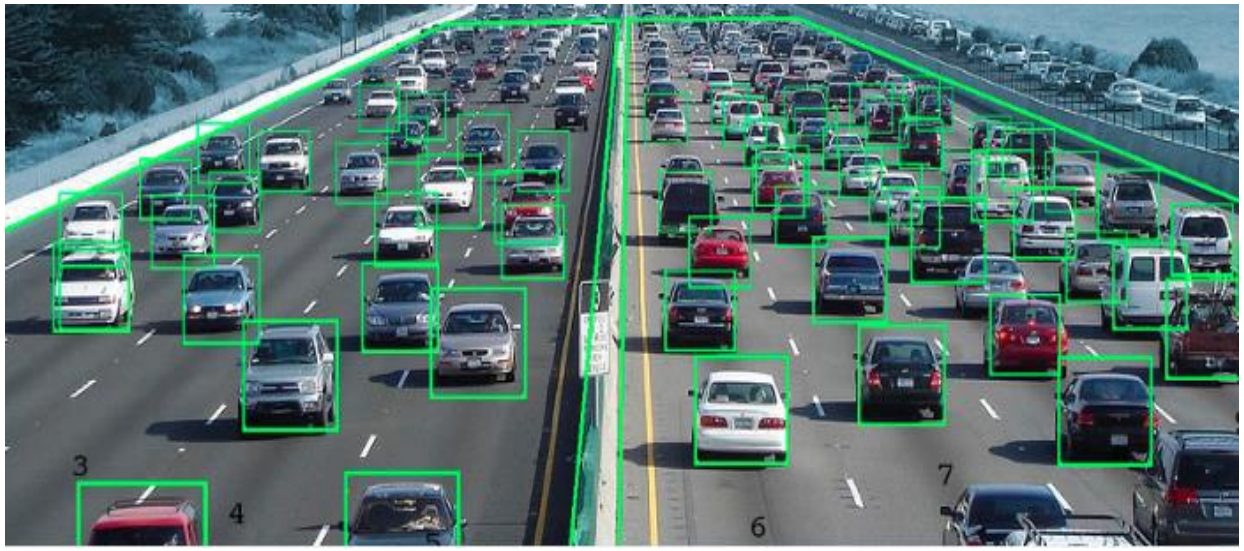
Swin Transformer 등장 배경

- Previous Problem
  - 기존 ViT 모델은 분류 문제를 풀기 위한 모델로 제안됨
  - 텍스트와 달리 이미지를 위한 특성이 ViT에 없음
  - Token의 수가 증가함에 따라 연산량이 Quadratic 하게 증가
- Purpose
  - 다양한 목적에 backbone으로 사용될 수 있는 모델 제안
  - Transformer 구조에 이미지의 특성을 반영할 수 있는 방법 제안
  - 기존 ViT 모델보다 더 적은 연산량을 갖는 방법 제안

Swin Transformer

텍스트와 다른 이미지의 특성 적용

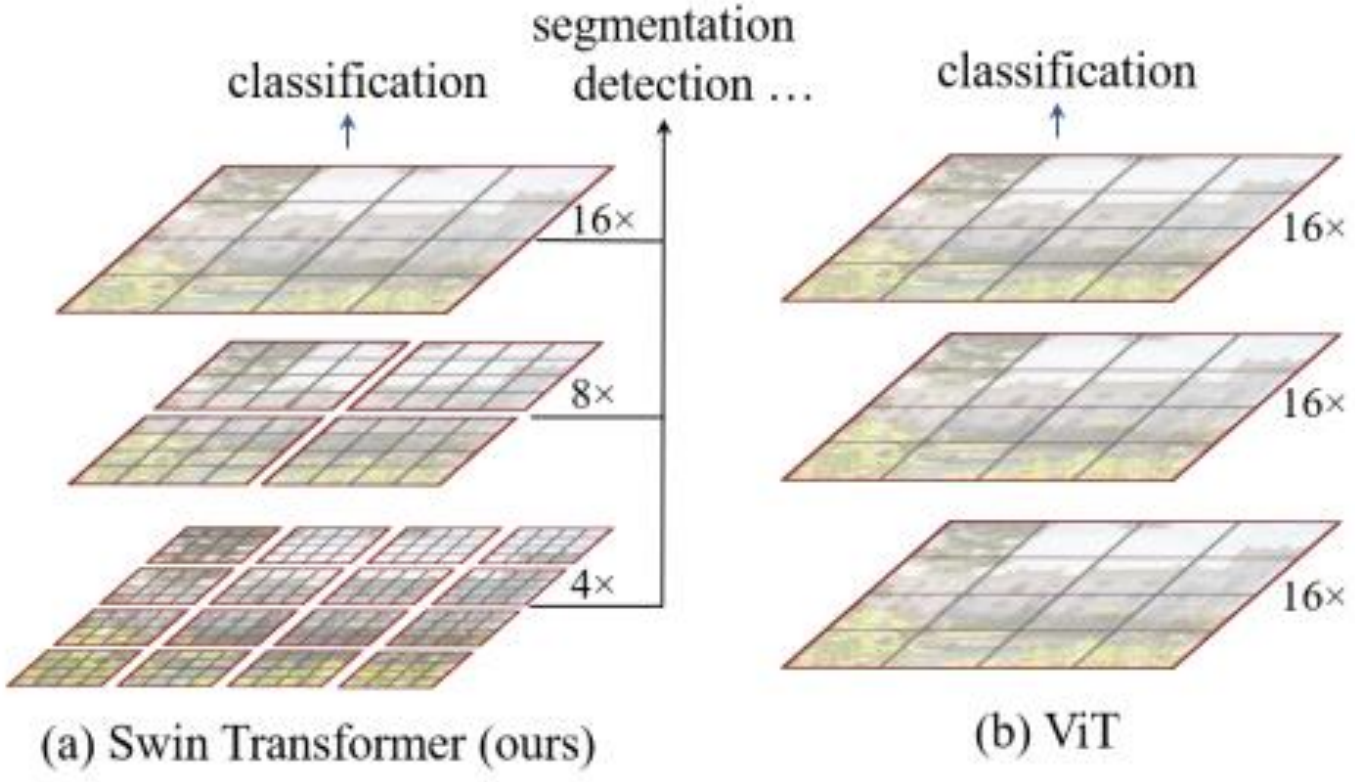
- 해상도 (resolution)
- 물체의 크기 (the scale of visual entities)



Swin Transformer

Solution

- Local Window를 모델에 적용

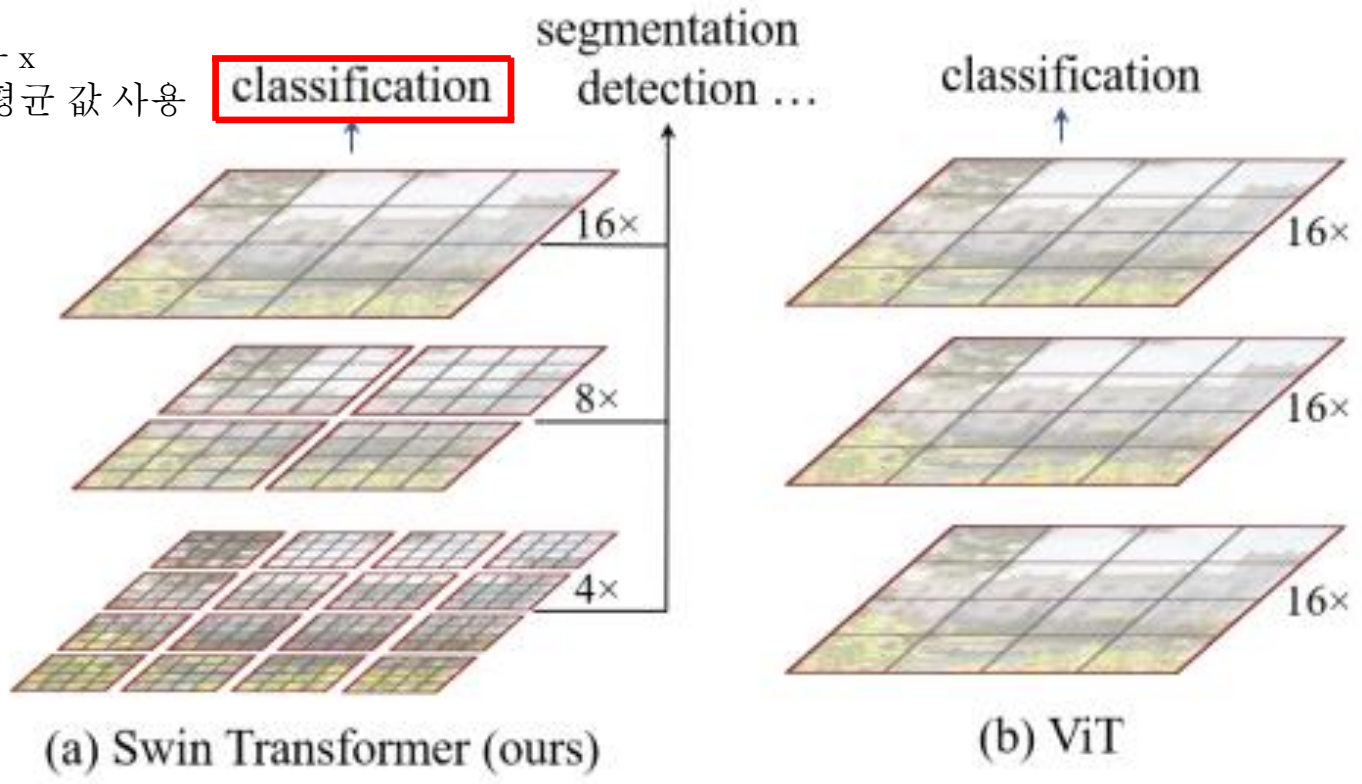


Swin Transformer

Solution

- Local Window를 모델에 적용

[class] token 사용 x
=> Token들의 평균 값 사용



Swin Transformer

Solution

- Local Window를 모델에 적용
- 기존 ViT보다 더 적은 complexity로 학습

$$\Omega(MSA) = 4hwC^2 + 2(hw)^2C$$ → ViT

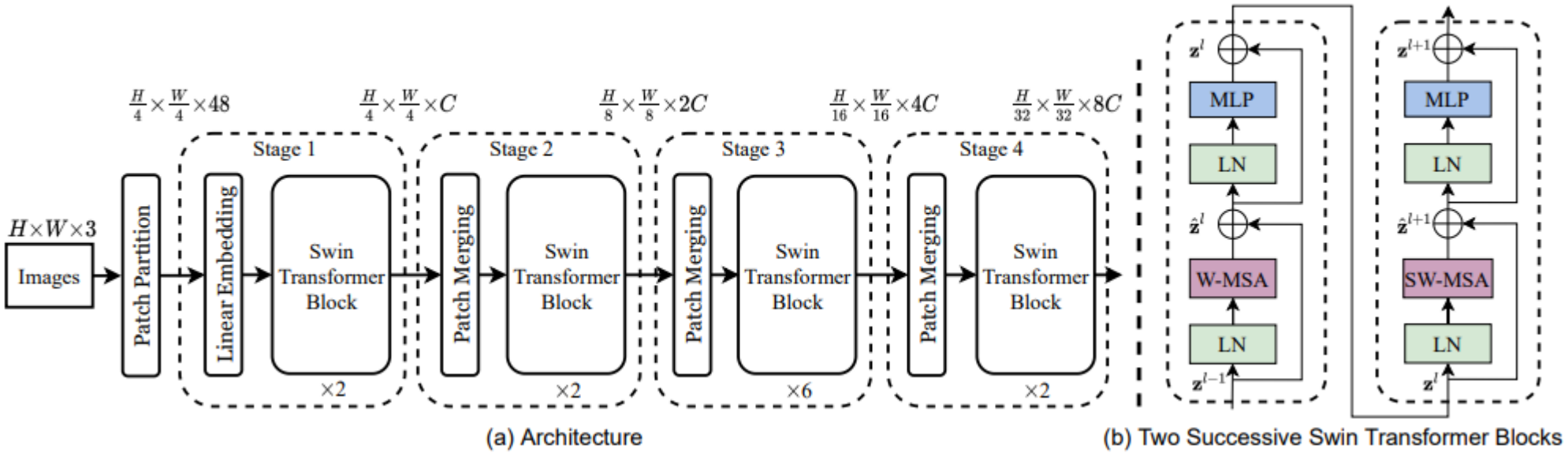
$$\Omega(W-MSA) = 4hwC^2 + 2M^2hwC$$ → Swin Transformer

- h,w : 세로 ,가로 patch 개수
- M : local window 사이즈

Swin Transformer

Model Architecture

- Patch Merging
- Swin Transformer Block

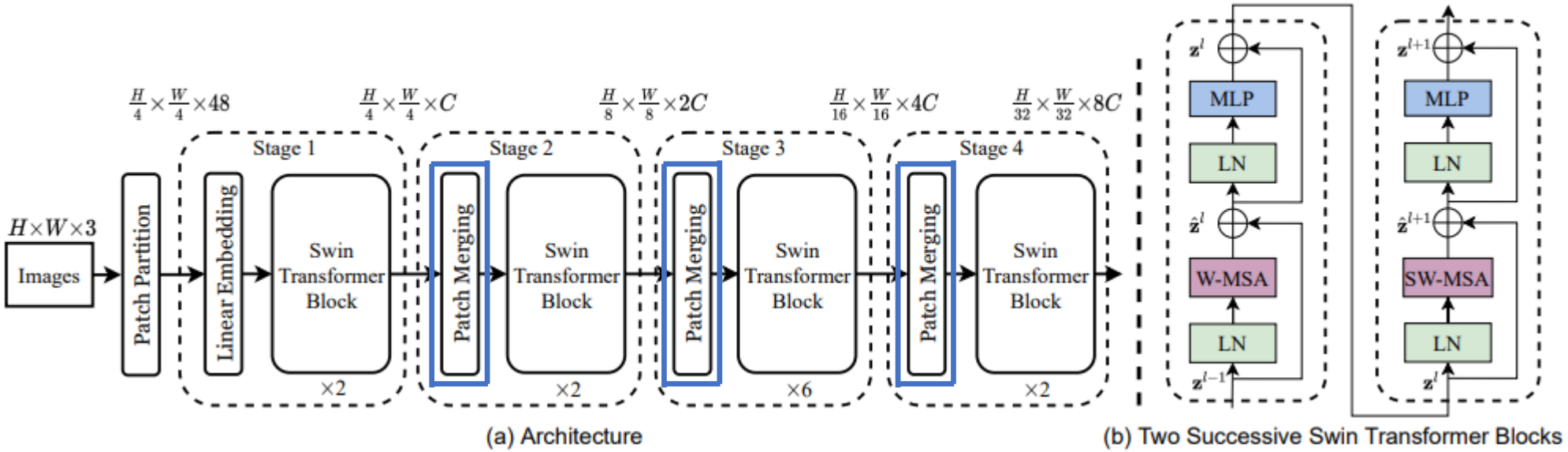


W-MSA (Window Multi-head Self Attention
SW-MSA (Shifted Window Multi-head Self Attention)

Swin Transformer

Model Architecture

- Patch Merging
- Swin Transformer Block

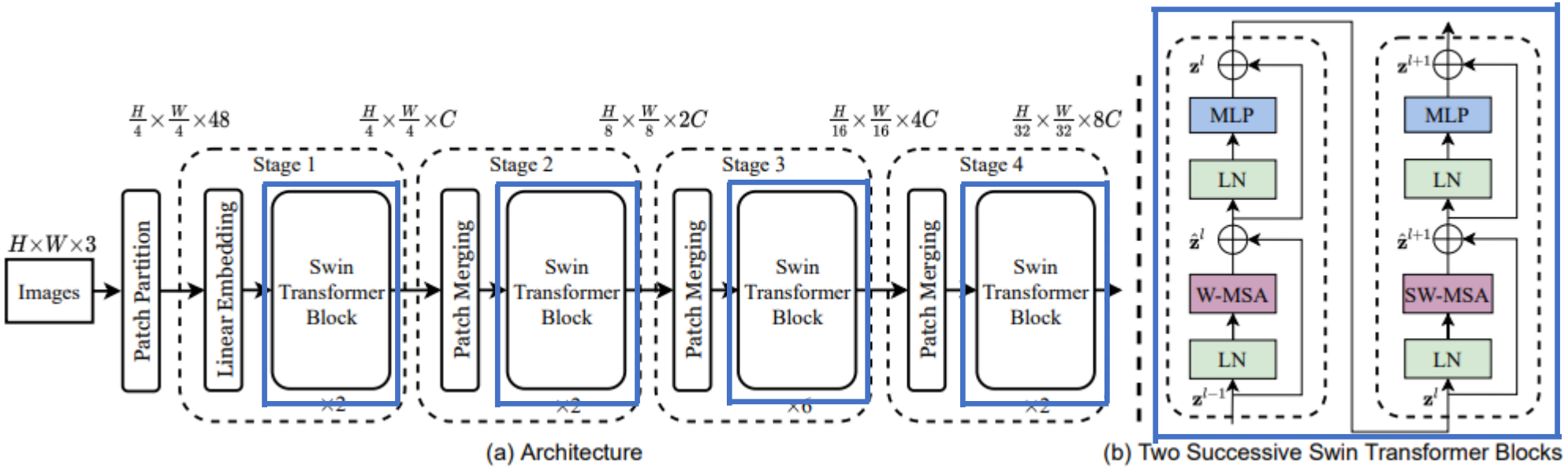


W-MSA (Window Multi-head Self Attention
SW-MSA (Shifted Window Multi-head Self Attention)

Swin Transformer

Model Architecture

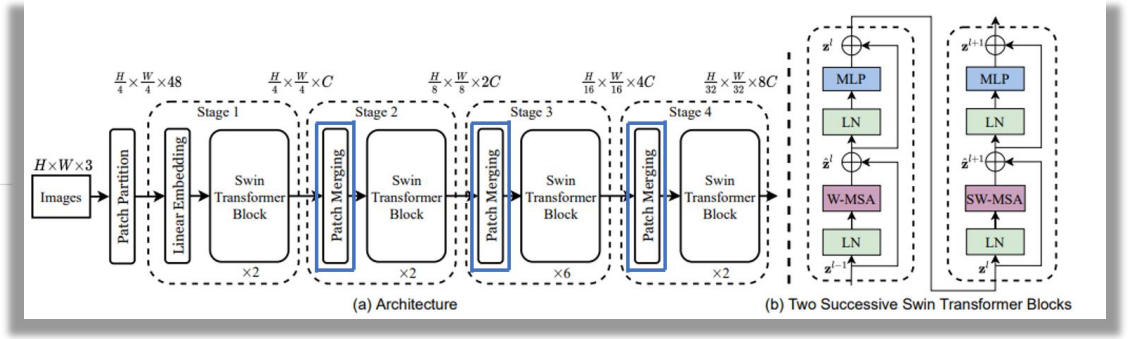
- Patch Merging
- Swin Transformer Block

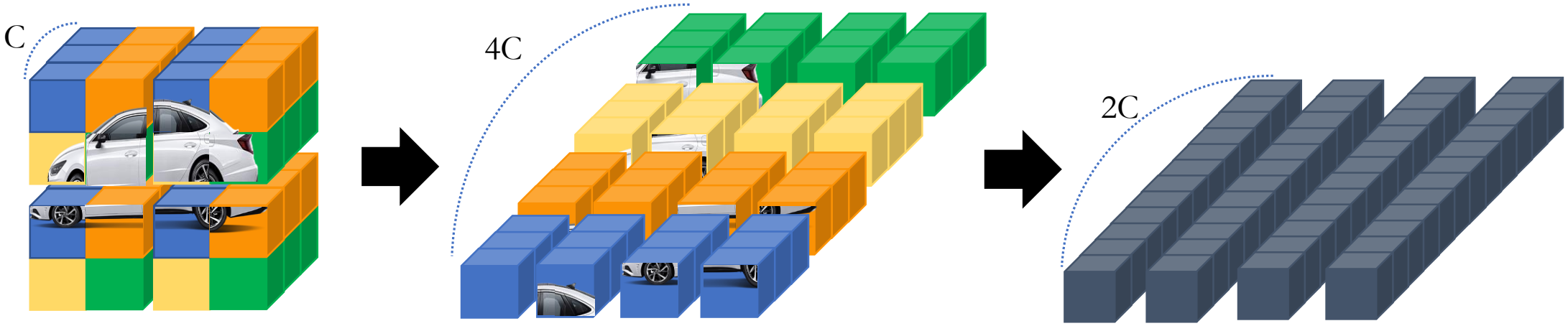


W-MSA (Window Multi-head Self Attention
SW-MSA (Shifted Window Multi-head Self Attention)

Swin Transformer

Patch Merging





Swin Transformer

Swin Transformer Block

- W-MSA : Local Window 안에서 Self Attention
- SW-MSA : Local Window 간의 Self Attention

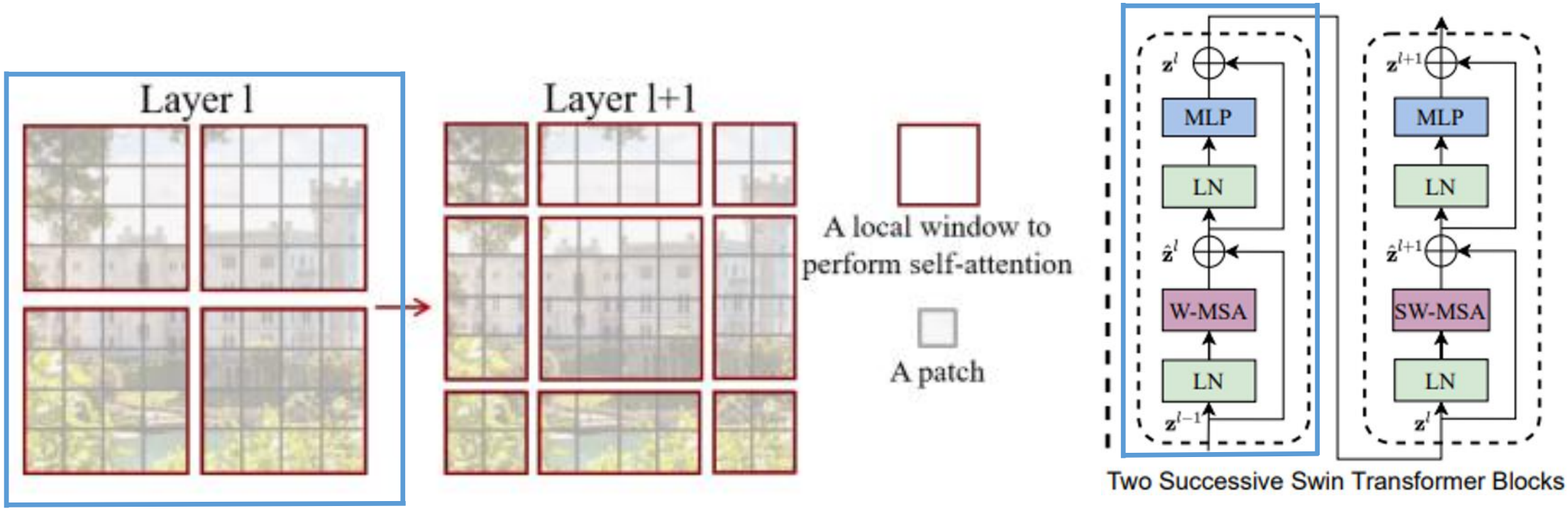
Details

- Efficient batch computation
- Relative Position Bias
- Cyclic Shift & Attention Mask

Swin Transformer

Swin Transformer Block

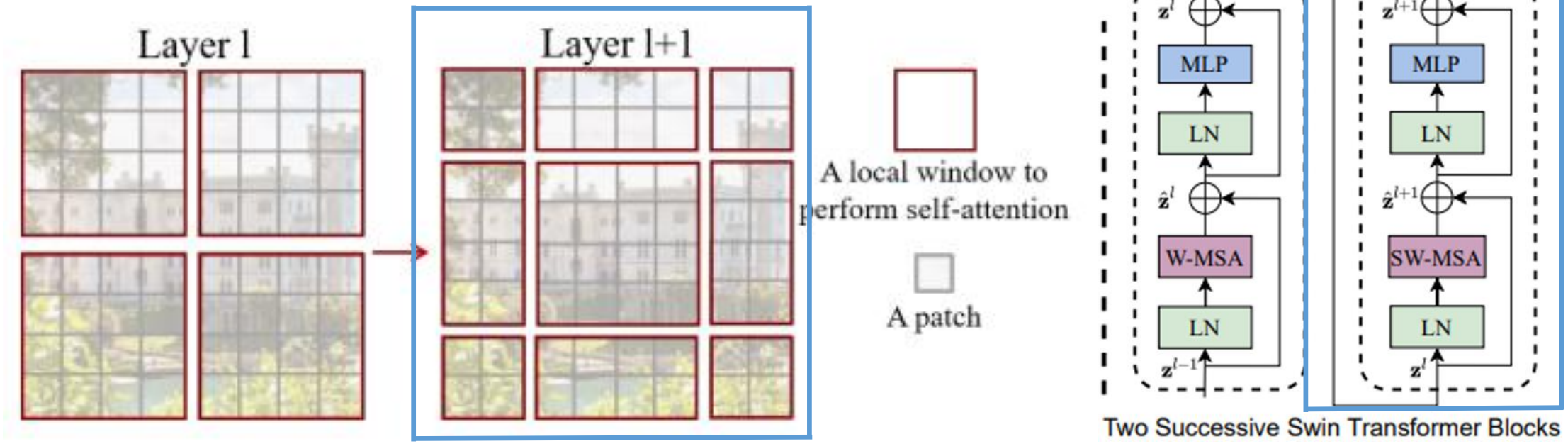
- W-MSA : Local Window 안에서 Self Attention
- SW-MSA : Local Window 간의 Self Attention



Swin Transformer

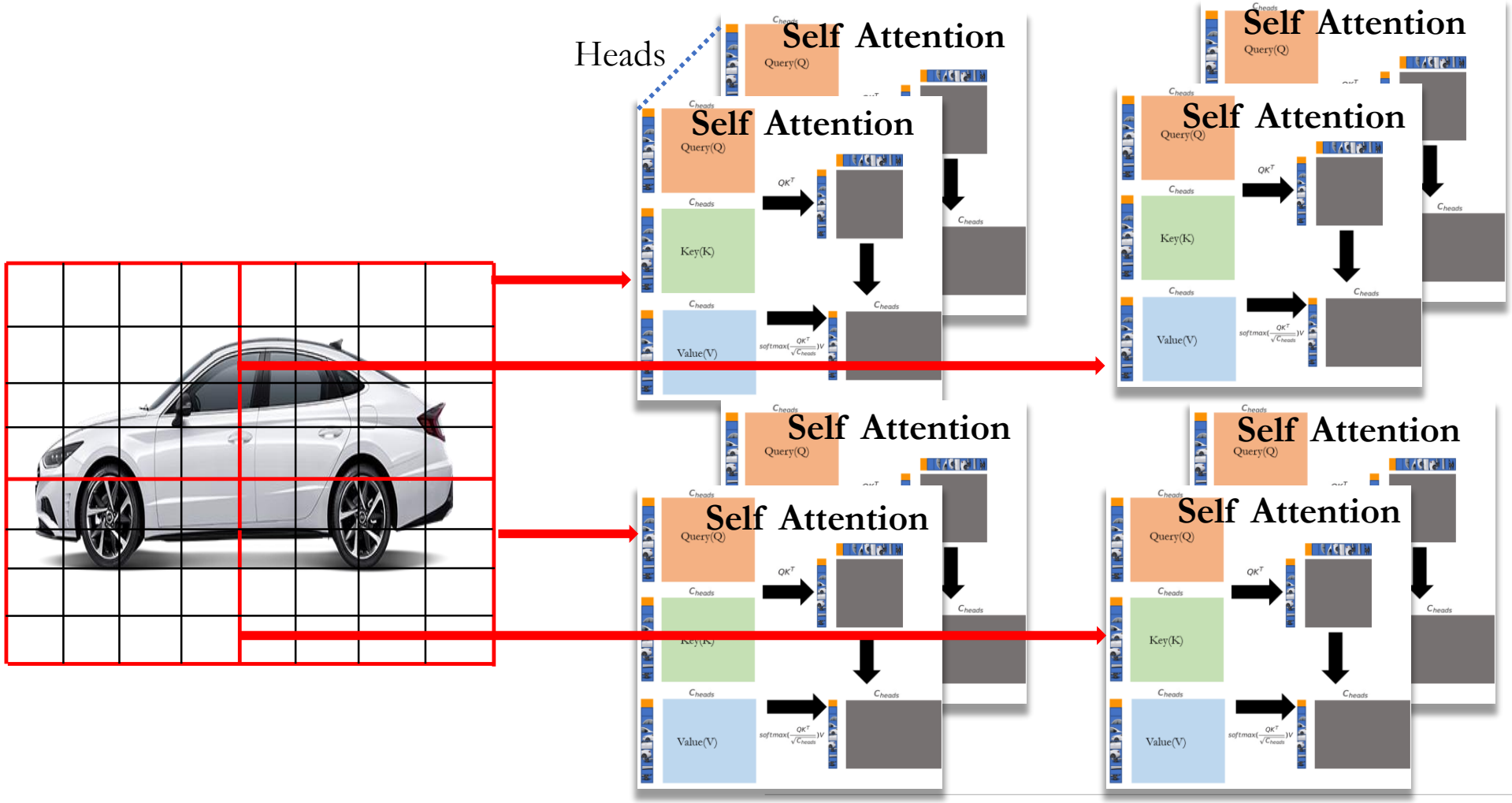
Swin Transformer Block

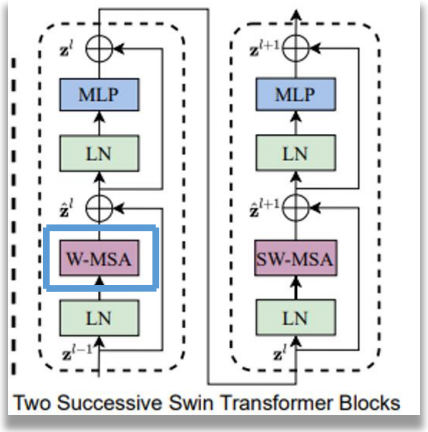
- W-MSA : Local Window 안에서 Self Attention
- SW-MSA : Local Window 간의 Self Attention



Swin Transformer

Swin Transformer Block : W-MSA

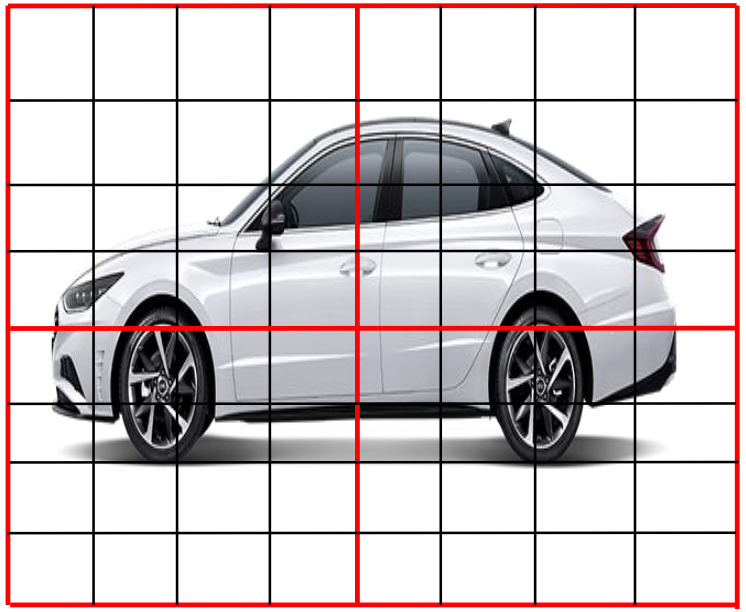




Swin Transformer

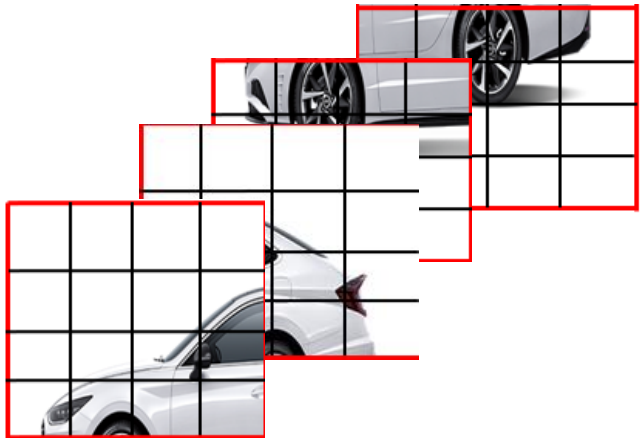
Swin Transformer Block : Efficient Computation

$$x \in R^{BxN_hxN_wxC}$$



$$x \in R^{1x8x8x3}$$

$$x \in R^{nBxMxMxC}, n = \frac{N_h}{M} + \frac{N_w}{M}$$



$$x \in R^{4x4x4x3}, M = 4, n = 4$$

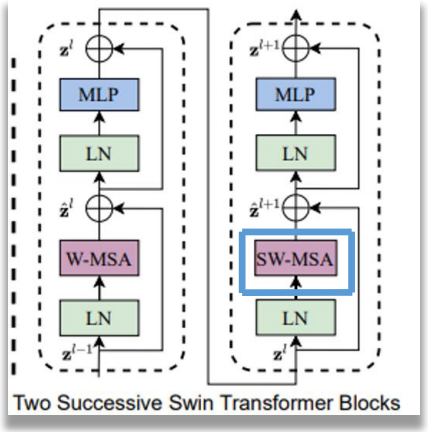
Swin Transformer

Swin Transformer Block : Relative Position Bias

- 두 축 마다 Relative Position의 범위 [-M + 1, M-1]
- Bias Index Matrix $(\hat{B}) = R^{(2M-1)x(2M-1)}$
- B는 B hat의 값을 사용

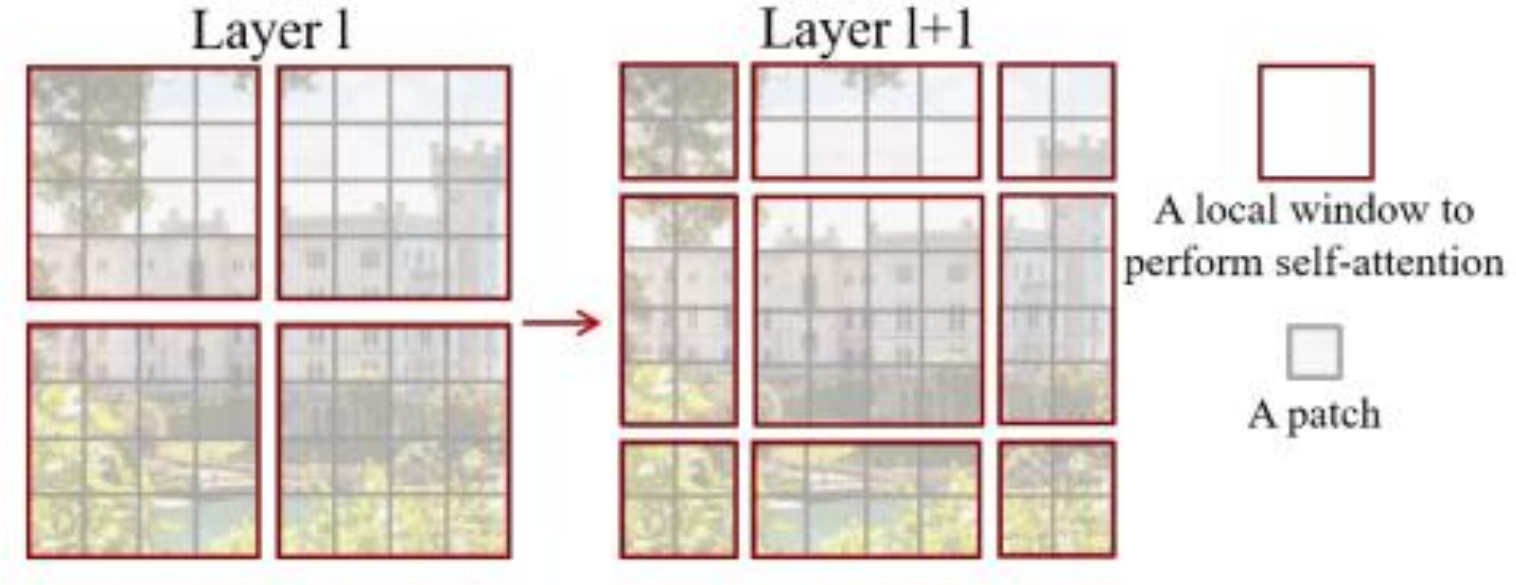
$$Attention(Q, K, V) = SoftMax(QK^T/\sqrt{d} + B)V$$

Swin Transformer

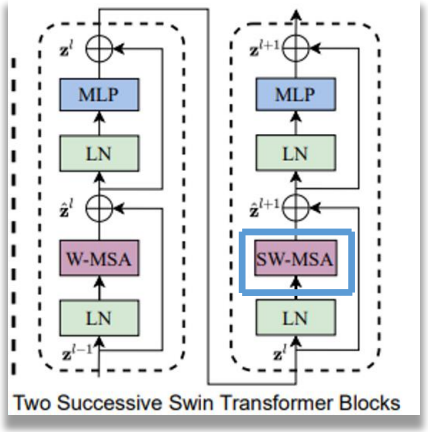


Swin Transformer Block : SW-MSA

- SW-MSA 수행 시 window 개수가 $(\frac{N_h}{M}+1)x(\frac{N_w}{M}+1)$
- Cyclic Shift & Attention Mask를 통해서 W-MSA와 동일한 Window 개수 사용

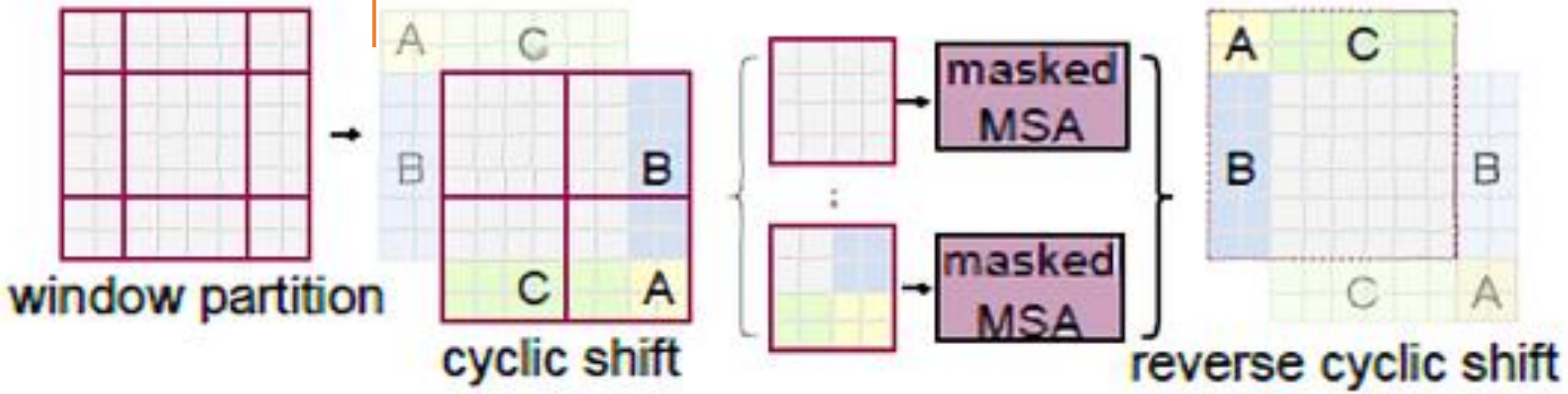


Swin Transformer



Swin Transformer Block : SW-MSA

Shift Size = Window Size(M) //2



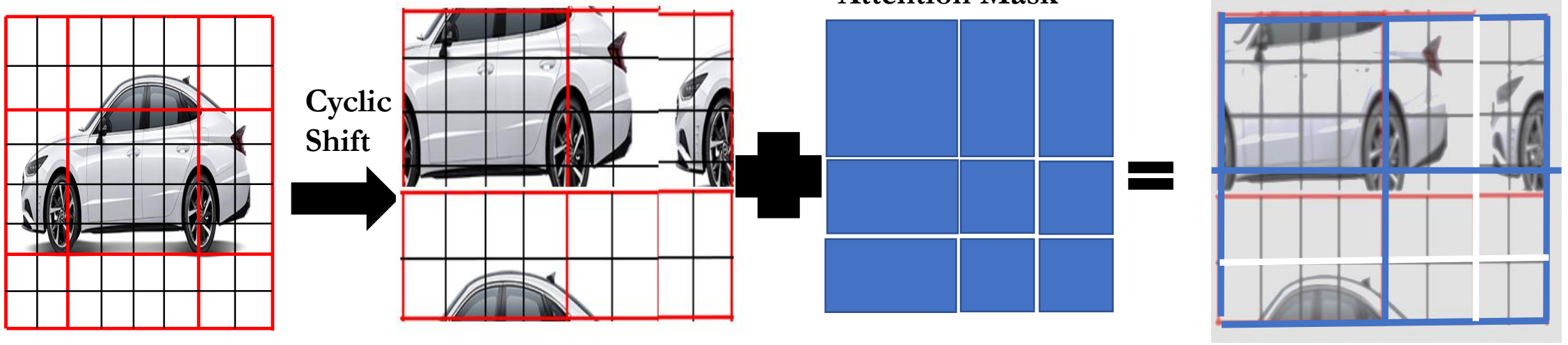
서로 인접한 위치가 아니었기 때문에, Mask를 적용해 Mask 부분에서만 Self Attention을 적용

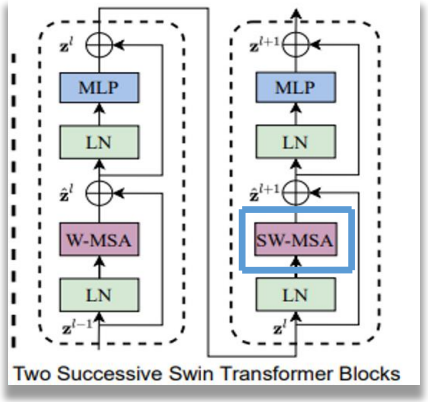
Swin Transformer

Swin Transformer Block : SW-MSA

$$QK^T \in R^{M^2 x M^2}$$

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





Swin Transformer

Experiment

- 모델 버전
  - Image Resolution (Default) : 224 x 224
  - 384 x 384 일 때는 window size 12 사용

	downsp. rate (output size)	Swin-T	Swin-S	Swin-B	Swin-L
stage 1	4× (56×56)	concat 4×4, 96-d, LN	concat 4×4, 96-d, LN	concat 4×4, 128-d, LN	concat 4×4, 192-d, LN
		[win. sz. 7×7, dim 96, head 3] × 2	[win. sz. 7×7, dim 96, head 3] × 2	[win. sz. 7×7, dim 128, head 4] × 2	[win. sz. 7×7, dim 192, head 6] × 2
stage 2	8× (28×28)	concat 2×2, 192-d , LN	concat 2×2, 192-d , LN	concat 2×2, 256-d , LN	concat 2×2, 384-d , LN
		[win. sz. 7×7, dim 192, head 6] × 2	[win. sz. 7×7, dim 192, head 6] × 2	[win. sz. 7×7, dim 256, head 8] × 2	[win. sz. 7×7, dim 384, head 12] × 2
stage 3	16× (14×14)	concat 2×2, 384-d , LN	concat 2×2, 384-d , LN	concat 2×2, 512-d , LN	concat 2×2, 768-d , LN
		[win. sz. 7×7, dim 384, head 12] × 6	[win. sz. 7×7, dim 384, head 12] × 18	[win. sz. 7×7, dim 512, head 16] × 18	[win. sz. 7×7, dim 768, head 24] × 18
stage 4	32× (7×7)	concat 2×2, 768-d , LN	concat 2×2, 768-d , LN	concat 2×2, 1024-d , LN	concat 2×2, 1536-d , LN
		[win. sz. 7×7, dim 768, head 24] × 2	[win. sz. 7×7, dim 768, head 24] × 2	[win. sz. 7×7, dim 1024, head 32] × 2	[win. sz. 7×7, dim 1536, head 48] × 2

Table 7. Detailed architecture specifications.

Swin Transformer

Experiment

- Settings
  - Classification에 ImageNet 1K로 실험
  - ImageNet-22K을 사전 학습에 사용
  - Object Detection에 COCO 2017 사용
  - Semantic Segmentation에 ADE20K 사용

** 다양한 Augmentation과 Regularization 필요
** ViT에서 중요했던 repeated augmentation과 EMA는 필요 X
** AdamW optimizer 사용

Swin Transformer

Experiment

- Vision Task
  - Classification
  - Object Detection
  - Semantic Segmentation

- 그 외 실험
  - Relative Positional Bias
  - Shifted Window
  - Window 적용 방법 및 속도
  - Sliding Window와 Shifted Window

Swin Transformer

Experiment

- Baseline Models
  - CNN Based
    - RegNetY
    - EfficientNet
  - ViT Based
    - ViT
    - DeiT

Swin Transformer

Experiment

- Classification
  - ImageNet 1K
    - DeiT vs Swin
    - CNN Based vs Swin
      - 성능과 학습 속도의 trade off 가 적음
  - ImageNet 22K
    - ViT vs Swin
      - 더 적은 파라미터 , 더 좋은 성능

(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].

Swin Transformer

Experiment

- Classification
  - ImageNet 1K
    - DeiT vs Swin
    - CNN Based vs Swin
      - 성능과 학습 속도의 trade off 가 적음
  - ImageNet 22K
    - ViT vs Swin
      - 더 적은 파라미터 , 더 좋은 성능

(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].

Swin Transformer

Experiment

- Classification
  - ImageNet 1K
    - DeiT vs Swin
    - CNN Based vs Swin
      - 성능과 학습 속도의 trade off 가 적음
  - ImageNet 22K
    - ViT vs Swin
      - 더 적은 파라미터 , 더 좋은 성능

(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].

Swin Transformer

Experiment

- Object Detection
  - ResNet-50 vs Swin
    - +3.4~4.2 box AP
  - Backbone 기준
    - CNN Based Model vs Swin
    - DeiT는 ViT와 거의 동일구조
      - 모든 layer가 동일 해상도
      - Deconvolution layer를 사용해 다른 해상도 추출

(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 AP^{box}	mini-val AP^{mask}	test-dev AP^{box}	test-dev AP^{mask}	#param.	FLOPs
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.

Swin Transformer

Experiment

- Object Detection
  - ResNet-50 vs Swin
    - +3.4~4.2 box AP
  - Backbone 기준
    - CNN Based Model vs Swin
    - DeiT는 ViT와 거의 동일구조
      - 모든 layer가 동일 해상도
      - Deconvolution layer를 사용해 다른 해상도 추출

(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 AP^{box}	mini-val AP^{mask}	test-dev AP^{box}	test-dev AP^{mask}	#param.	FLOPs
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.

Swin Transformer

Experiment

- Segmantic Segmentation
  - SETR + T- Large vs UperNet + Swin –L
  - 적은 파라미터로 더 높은 성능

Method	Backbone	ADE20K val mIoU	ADE20K 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.

Swin Transformer

Experiment

- Relative Positional Bias & Shifted Window 여부
  - Window간 관계를 학습하는 것이 Detection과 Segmentation 성능 향상

	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).

Swin Transformer

Experiment

- Relative Positional Bias & Shifted Window 여부
  - Dense Prediction에서 매우 큰 성능 향상

	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).

Swin Transformer

Experiment

- Window 적용 방법에 따른 속도 비교

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.

Swin Transformer

Experiment

- Sliding Window VS Shifted Window

	Backbone	ImageNet top-1	ImageNet top-5	COCO AP^{box}	COCO AP^{mask}	ADE20k 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.

Swin Transformer

Conclusion

- Architecture 관점
  - Local Window Adation을 통해 Inductive bias 개입
  - Patch Merging을 통해 Hierarchical 구조 형성

- 이는 이미지의 특징을 고려한 구조
- 다양한 Vision Task에 적합
- ViT와 비교했을 때, 적은 Computational Complexity

Swin Transformer

Thank You

발표자 : 이주영