Swin-Transformer

Ze Liu / 17 AUG 2021

논문 구현

CloseAI팀 이상헌 김유철 박준혁 이정훈

목차

- ViT
- Swin-Transformer
- 구현결과
- QnA

Vision-Transformer

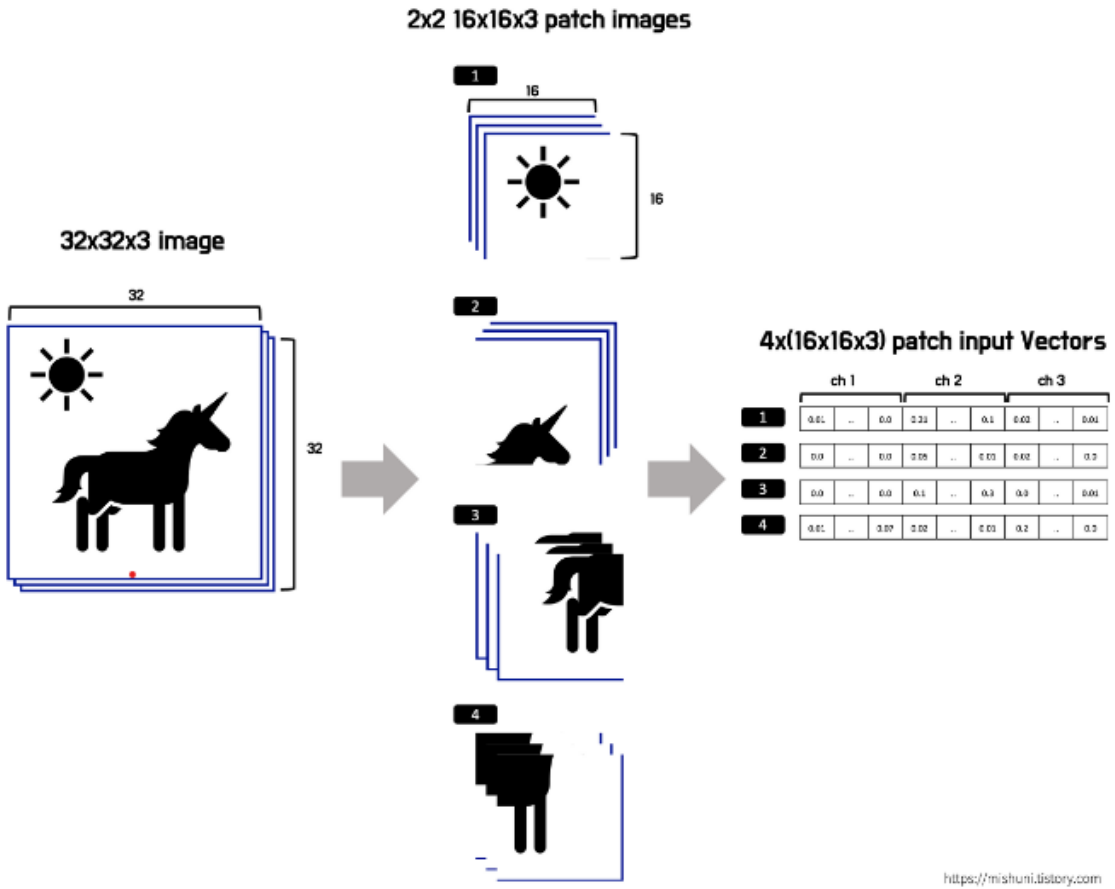
기존 computer vision 모델

- **Parmar** : Attention을 전역적으로 적용하는 대신 각 쿼리 픽셀에 대해 local neighborhood에만 적용

- **Sparse_transformer, weissenborn** : attention 범위를 scailing하는 방식으로 self-attention적용

- **Cordonier :** 이미지를 2x2패치로 쪼갠 후 self-attention 적용

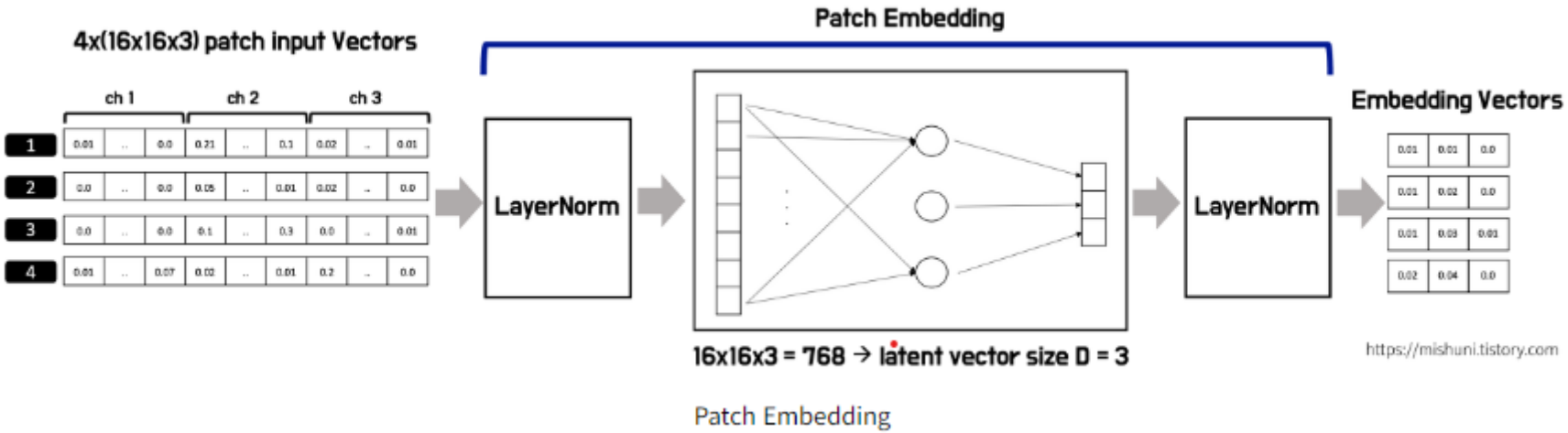
ViT(Vision-Transformer)



from image to patches and vectors

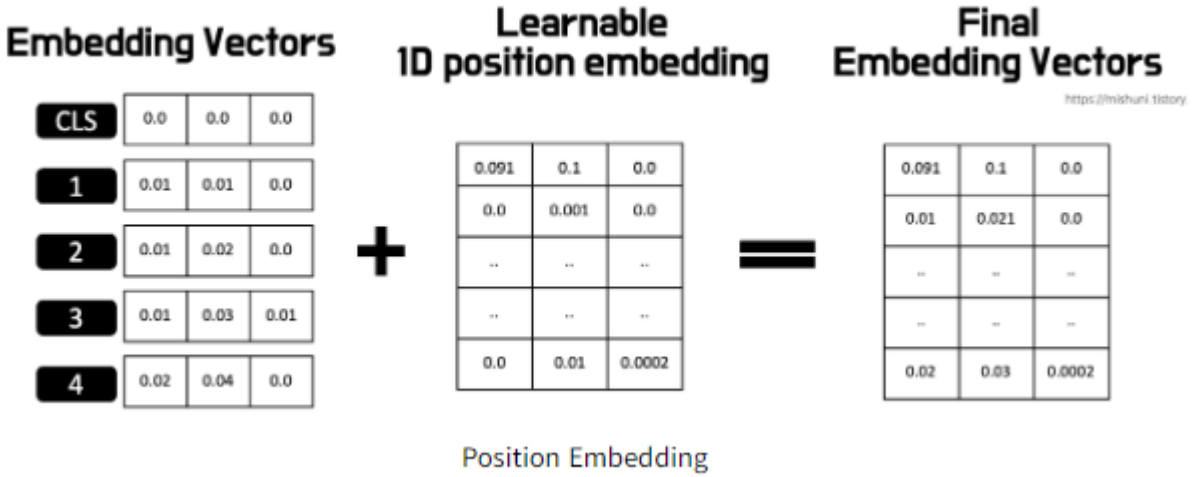
ViT(Vision-Transformer)

패치 임베딩 과정

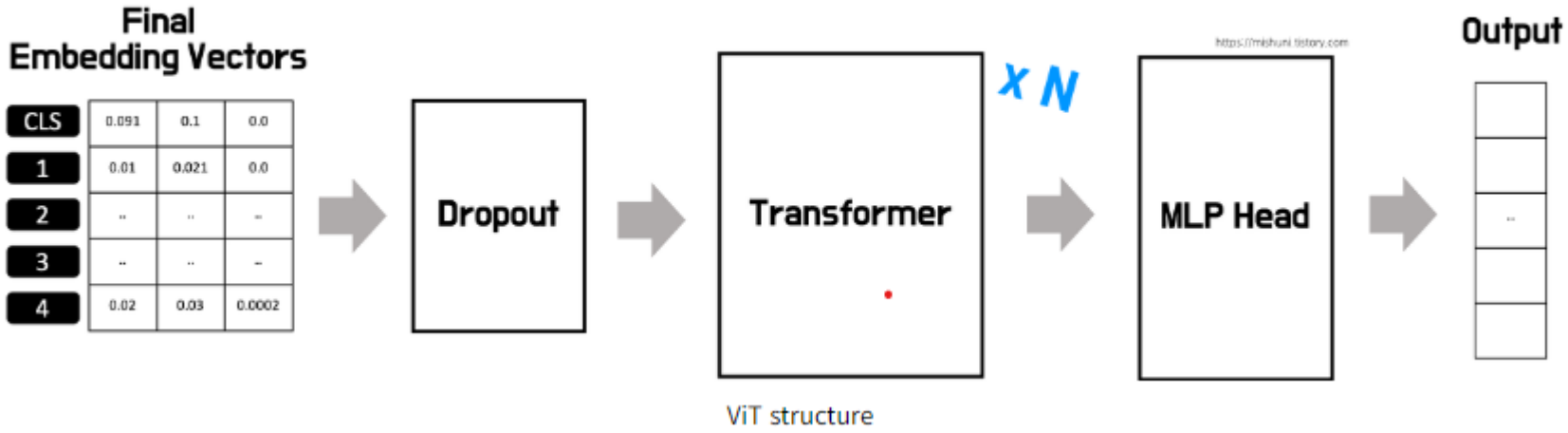


Patch Embedding

ViT(Vision-Transformer)

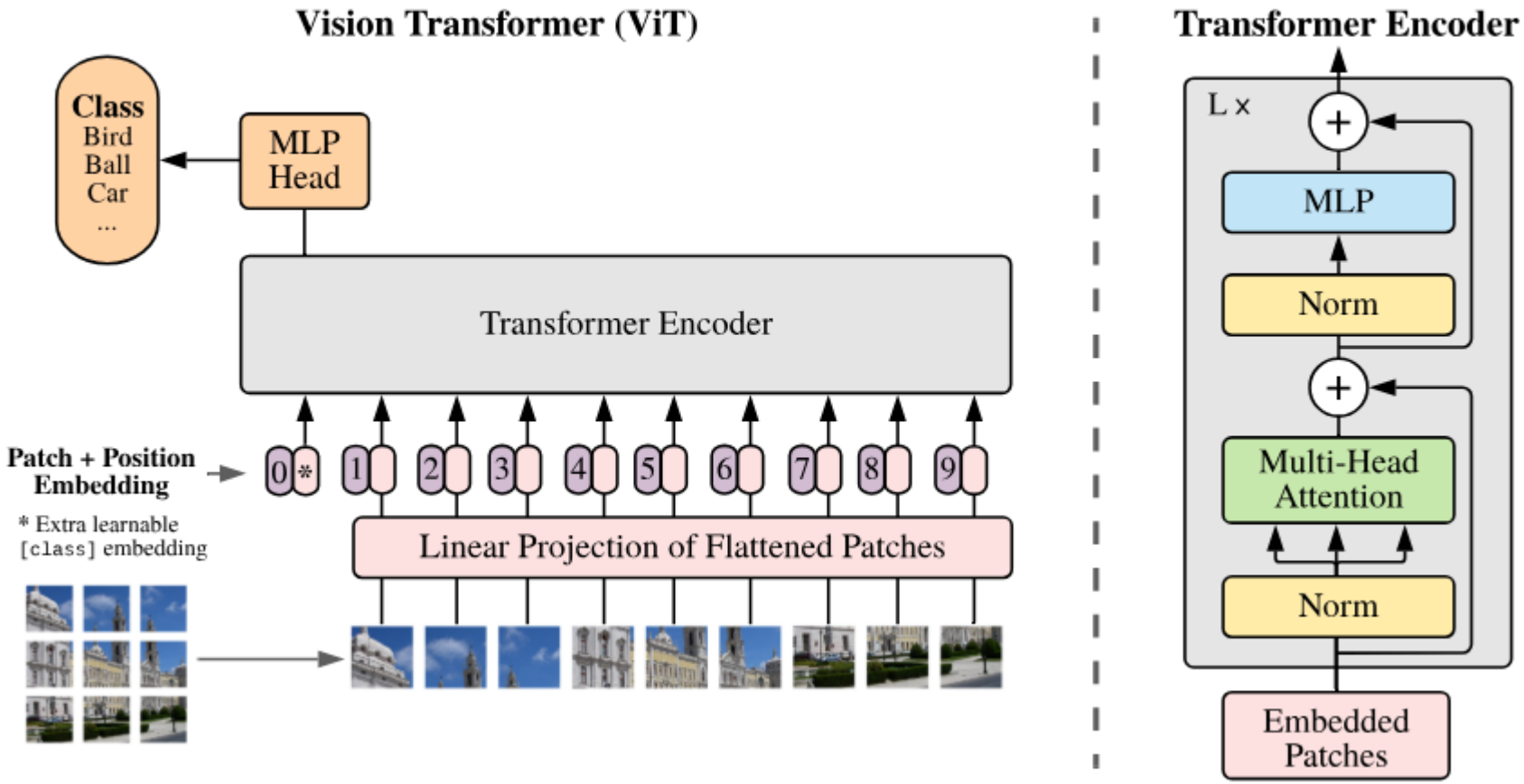


->Positional Encoding 과정

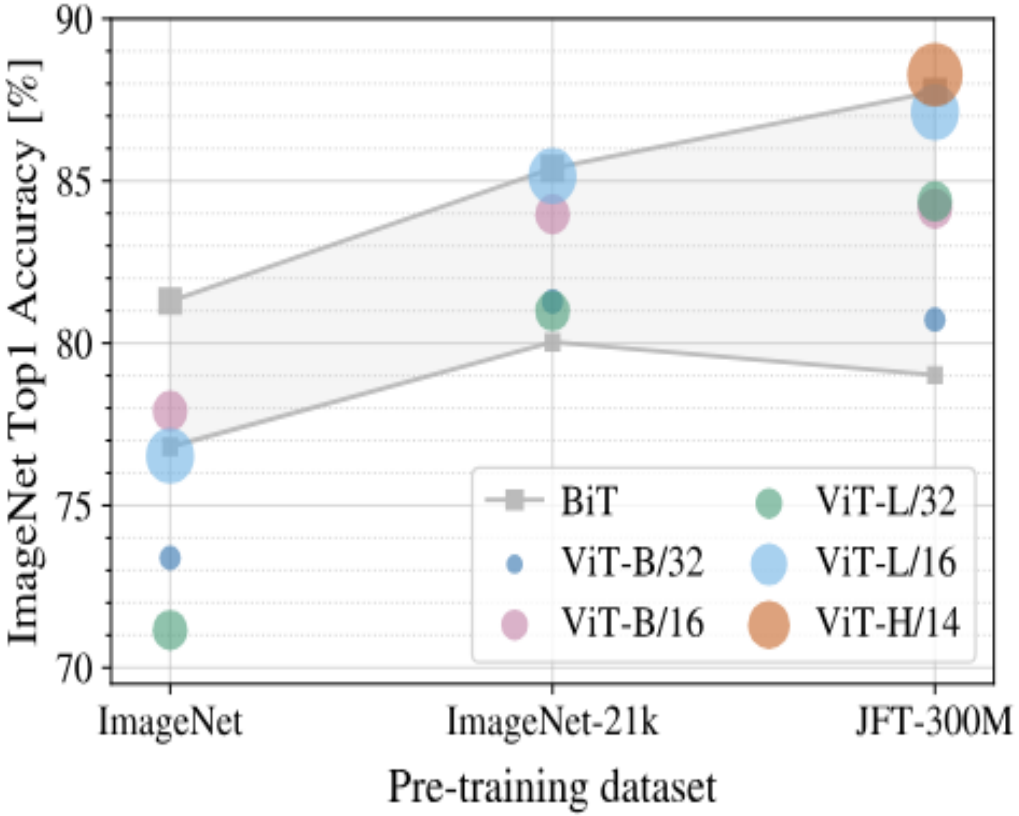


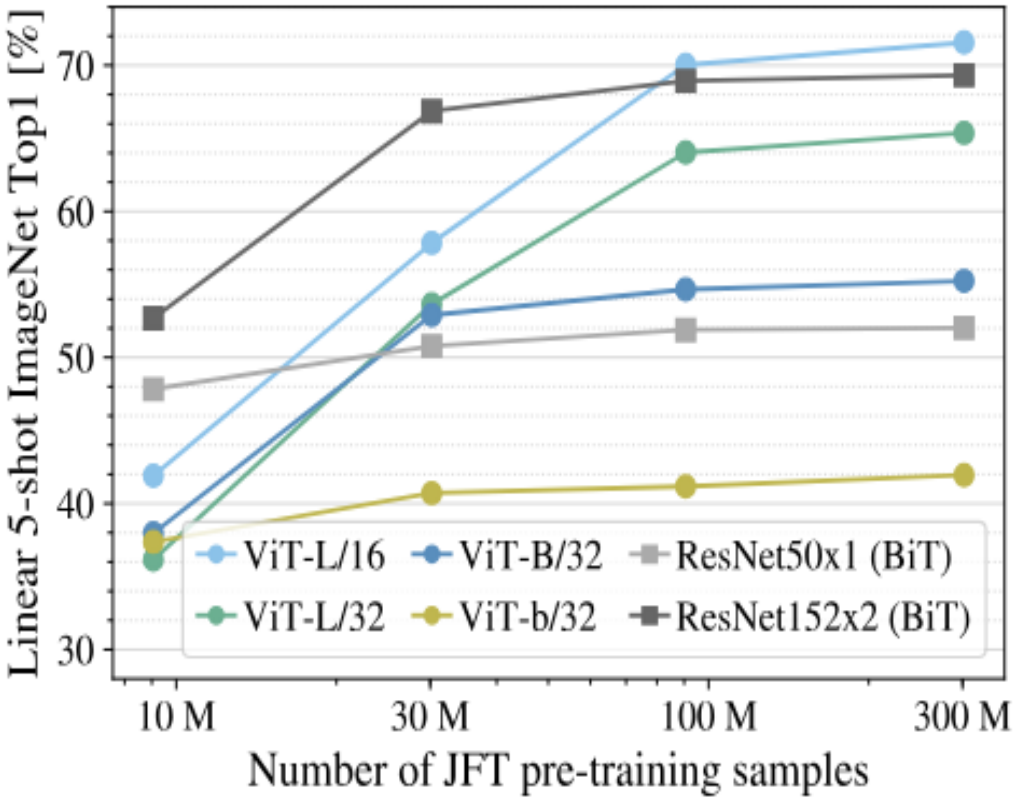
-> MLP HEAD로 들어가는 과정

ViT 아키텍처


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0 * 1 2 3 4 5 6 7 8 9
Linear Projection of Flattened Patches
Transformer Encoder
L x
MLP
Norm
Multi-Head Attention
Norm
Embedded Patches

ViT 데이터셋


90
85
80
75
70
ImageNet Top1 Accuracy [%]
BiT
ViT-B/32
ViT-B/16
ViT-L/32
ViT-L/16
ViT-H/14
ImageNet
ImageNet-21k
JFT-300M
Pre-training dataset


70
60
50
40
30
Linear 5-shot ImageNet Top1 [%]
ViT-L/16
ViT-B/32
ResNet50x1 (BiT)
ViT-L/32
ViT-b/32
ResNet152x2 (BiT)
10 M
30 M
100 M
300 M
Number of JFT pre-training samples

ViT 장점

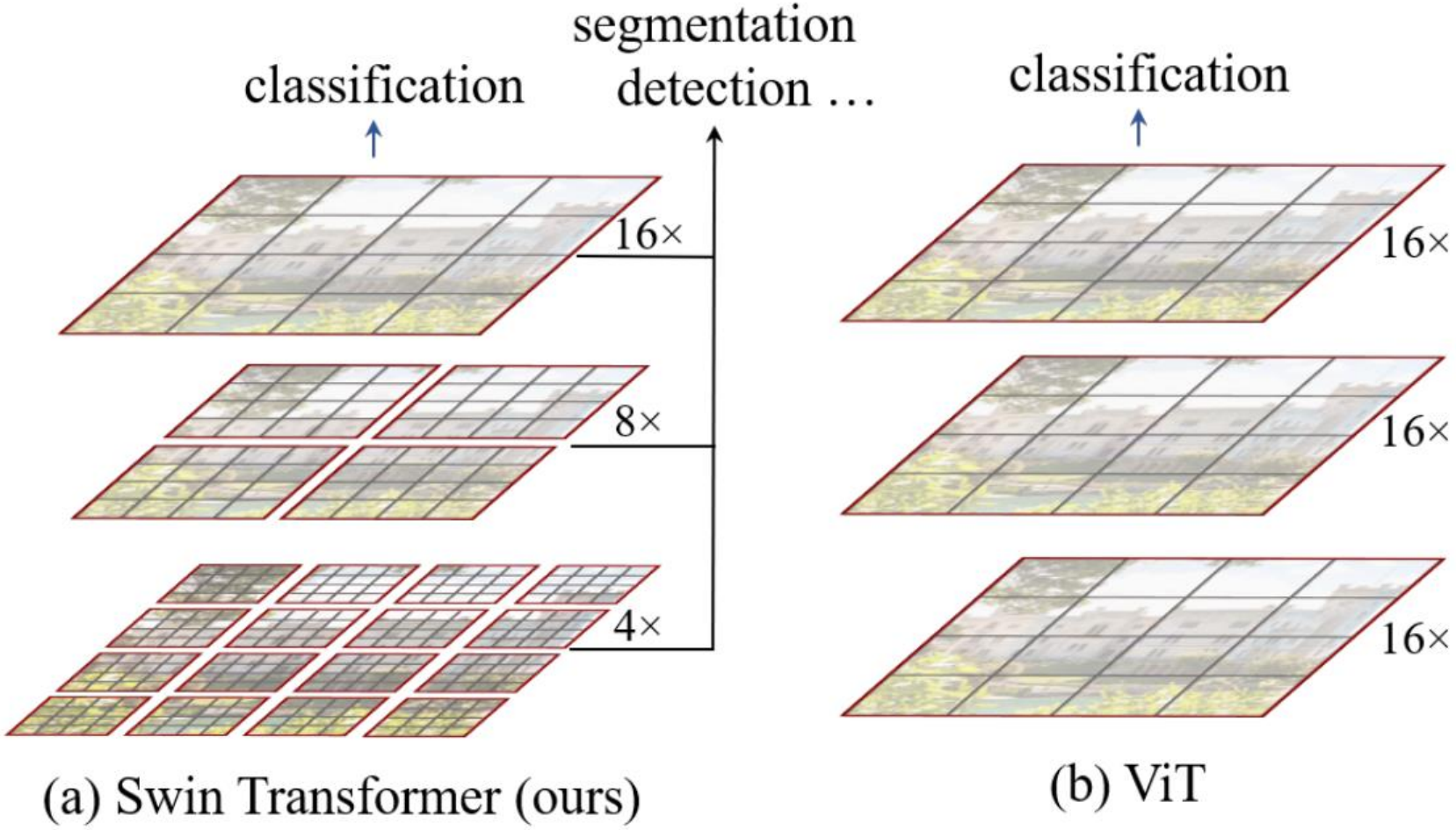
- **뛰어난 전이 학습 성능 -** 대규모 데이터셋에서 사전 학습된 후, 다양한 컴퓨터 비전 작업에 전이 학습할 때 뛰어난 성능을 발휘
- **단순한 구조 -** 합성곱 연산을 사용하지 않고, 단순한 트랜스포머 블록으로 구성
- **대규모 데이터셋에서의 성능 -** 트랜스포머의 Self-Attention메커니즘이 많은 양의 데이터를 효과적으로 학습 가능

ViT 단점

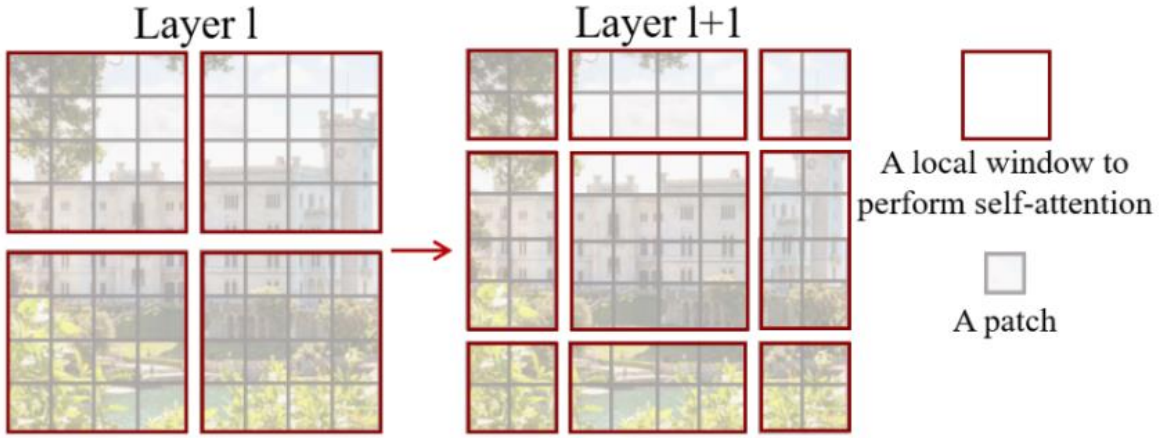
- 입력 이미지의 모든 패치 간의 관계를 전역적으로 계산하기 때문에, 계산 비용이 매우 큽니다
- 큰 패치 크기에서 전역적인 정보를 놓칠 수 있다
- 불충분한 데이터 양으로 학습을 하게 되면 Inductive bias 의 부족으로 인해 일반화 성능이 떨어지게 되어 대규모의 데이터가 요구된다

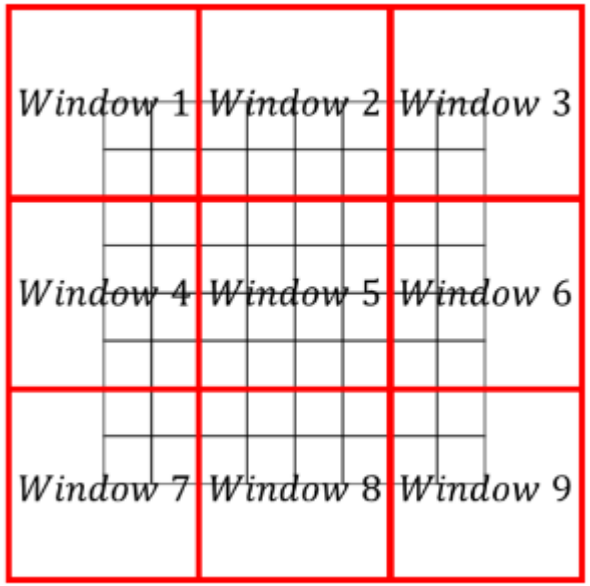
Swin-Transformer

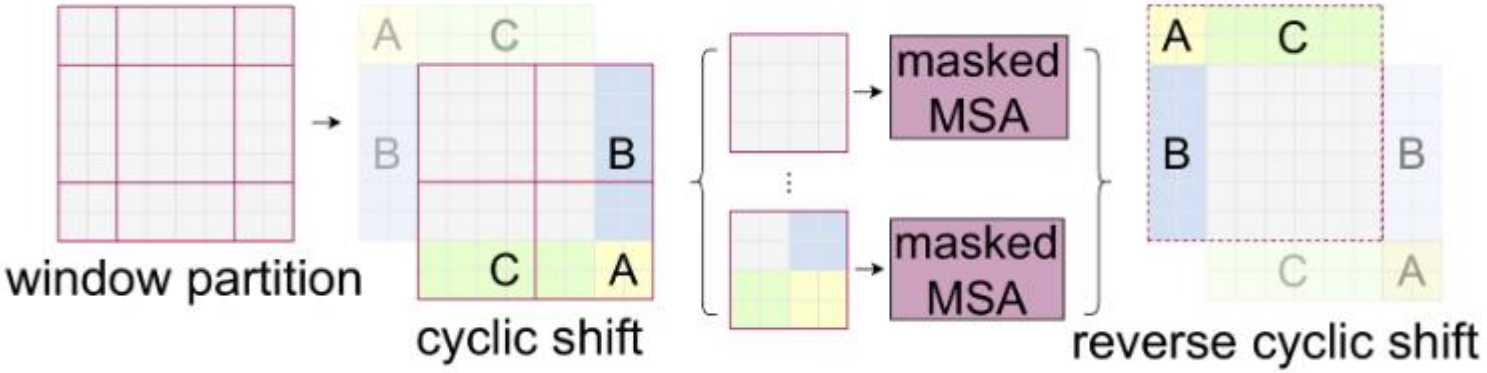
Swin-Transformer와 ViT 차이



Shifted Window







Relative position bias

기존 트랜스포머 모델에 들어가는 positional encoding을 사용하지 않는다

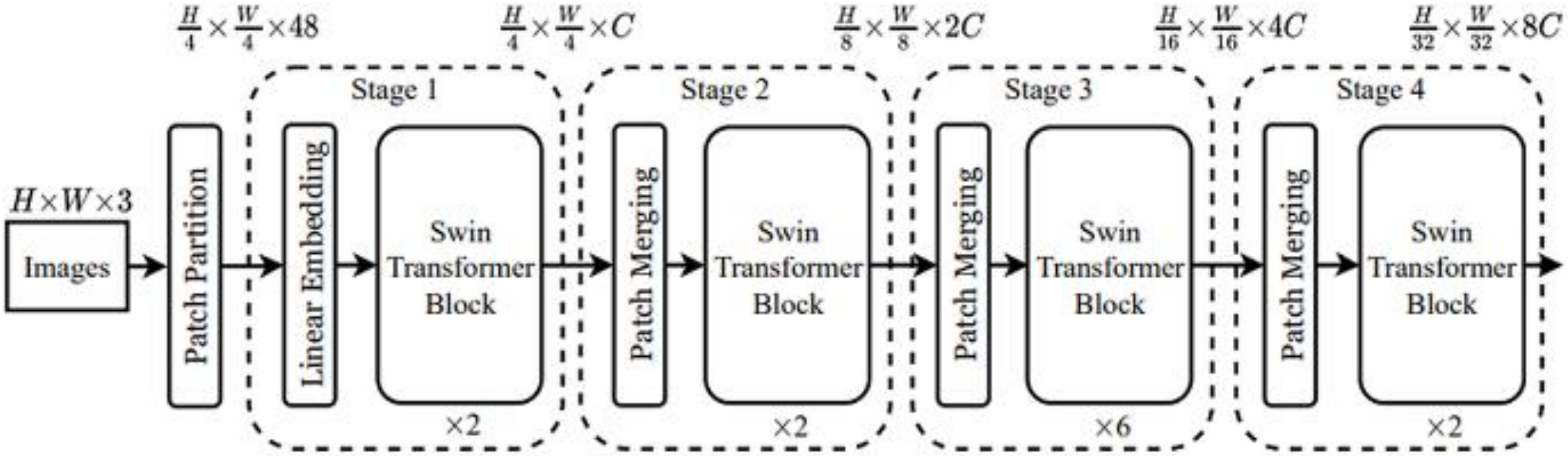
Self-attention 계산에서 각 head에 대한 상대적 위치 바이어스 사용

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

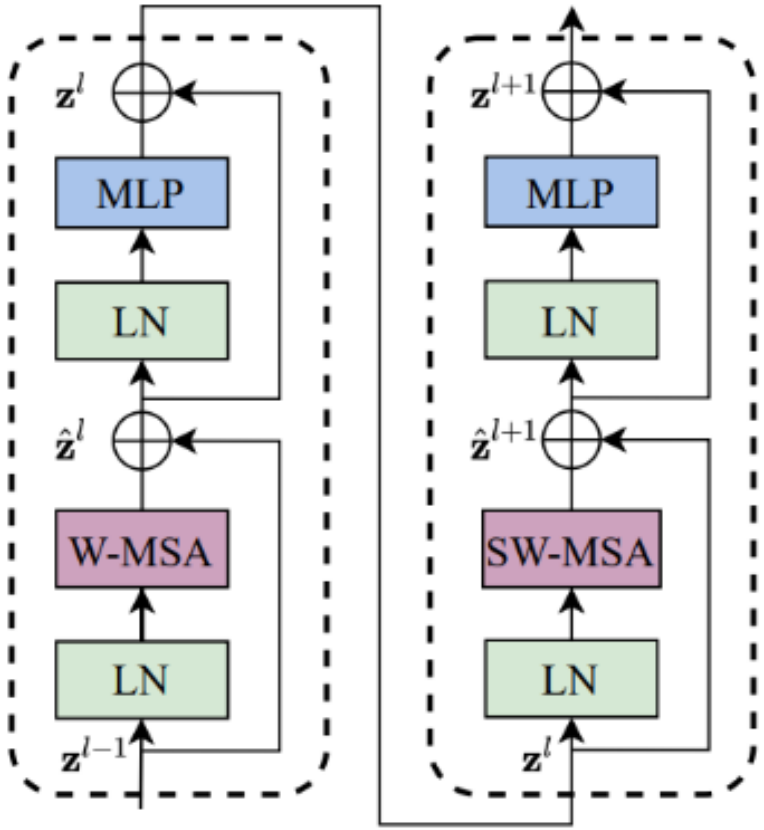
	ImageNet		COCO		ADE20k
	top-1	top-5	AP^{box}	AP^{mask}	mIoU
w/o shifting	80.2	95.1	47.7	41.5	43.3
shifted windows	**81.3**	**95.6**	**50.5**	**43.7**	**46.1**
no pos.	80.1	94.9	49.2	42.6	43.8
abs. pos.	80.5	95.2	49.0	42.4	43.2
abs.+rel. pos.	81.3	95.6	50.2	43.4	44.0
rel. pos. w/o app.	79.3	94.7	48.2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

Score

Swin-transformer 아키텍처(1)



Swin-transformer 아키텍처(2)



- 기존 MSA(Multi-Head self Attention)
->W-MSA(윈도우 기반 Multi-Head self Attention)
- LN(Layer Normalization)
- MLP(2개의 linear와 GELU함수 포함)
- Gelu(Gaussian Error Linear Unit)

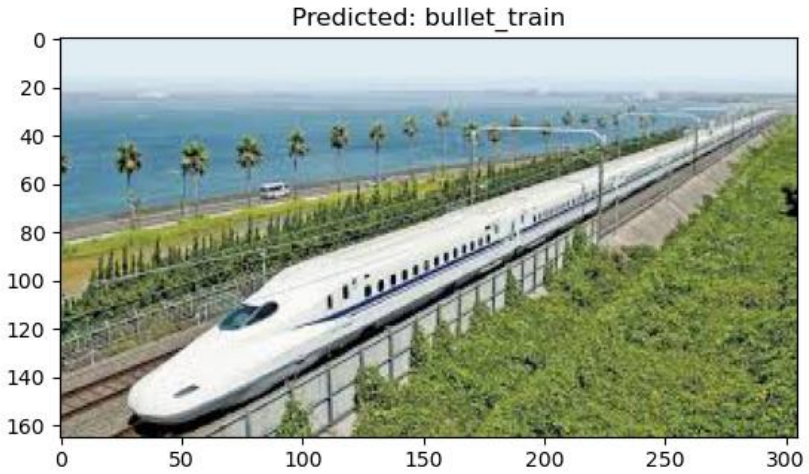
구현 결과

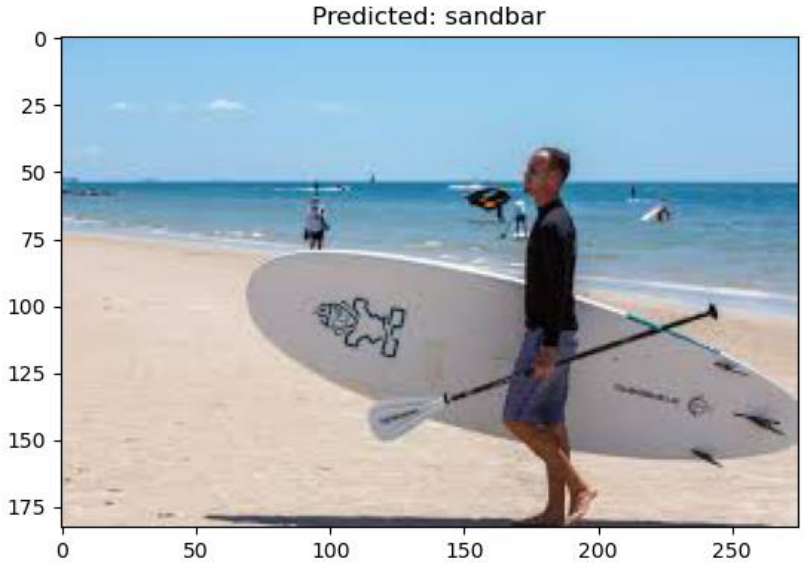
핵심코드

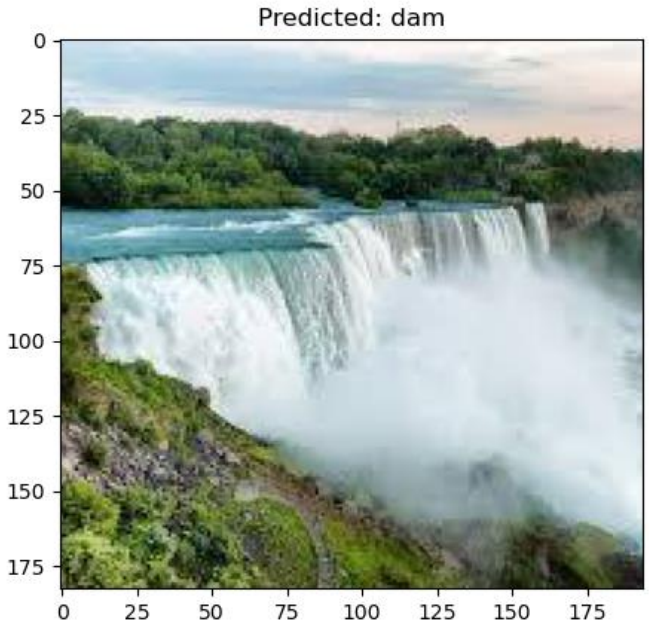

```python
class SwinTransformerBlock(nn.Module):
    def __init__(self, dim, input_resolution, num_heads, window_size=7, shift_size=0, mlp_ratio=4.):
        super().__init__()
        # 입력 차원, 입력 해상도,  > 비율 초기화
        self.dim = dim  # 특성 차
        self.input_resolution = input_resolution  # 입력 이미지의 해상도 (높이, 너비)
        self.num_heads = num_heads  # 멀티 헤드 어텐션의 헤드 수
        self.window_size = window_size  # 윈도우의 크기
        self.shift_size = shift_size  # 윈도우 시프트 크기
        self.mlp_ratio = mlp_ratio  # MLP에서 확장될 차원의 배수

        # 어텐션 메커니즘 초기화
        # 시프트 크기가 0보다 크면 시프트된 윈도우 어텐션을 사용
        if self.shift_size > 0:
            self.attn = WindowAttention(dim, window_size=to_2tuple(self.window_size),
                                        num_heads=num_heads, shift_size=to_2tuple(self.shift_size))
        else:
            self.attn = WindowAttention(dim, window_size=to_2tuple(self.window_size),
                                        num_heads=num_heads)
        # to_2tuple 함수는 일반적으로 하나의 값을 두 요소를 갖는 튜플 형태로 변환하는 함수입니다.
        # 이 함수는 입력된 값을 (value, value) 형태의 튜플로 바꾸어 주어, 코드 내에서 이차원 처리를 할 때 일관성을 유지할 수 있도록 돕습니다.
        # 예를 들어, 함수에 숫자 4가 입력되면 (4, 4)를 반환하게 됩니다. 이는 이미지 처리나 신경망에서 차원을 통일하여 다루기 위해 사용됩니다.

        # MLP 구성
        self.mlp = nn.Sequential(
            nn.Linear(dim, int(dim * mlp_ratio)),  # 첫 번째 선형 변환
            nn.GELU(),  # GELU 활성화 함수
            nn.Linear(int(dim * mlp_ratio), dim),  # 두 번째 선형 변환
        )

    def forward(self, x):
        # 입력 텐서에서 배치 크기, 길이, 차원 추출
        H, W = self.input_resolution
        B, L, C = x.shape
        assert L == H * W, "input feature has wrong size"  # 입력 크기 확인

        # 단축 연결을 위한 복사본 저장
        shortcut = x
        x = self.attn(x)  # 어텐션 적용
        x = shortcut + x  # 단축 연결

        # MLP 적용
        x = x + self.mlp(x)  # 결과에 MLP 적용 후 원래 값과 더함

        return x
```

(parameter) input_resolution: Any

결과


Predicted: bullet_train


Predicted: sandbar


Predicted: dam


Predicted: library

Reference

Reference

- **ViT(AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE)**

  Alexey Dosovitskiy/ 3 JUN 2021

- **Swin Transformer: Hierarchical Vision Transformer using Shifted Windows**

  Ze Liu†* Yutong Lin/ 17 AUG 2021

Q&A