Week9_예습과제_팽소원

10 비전 트랜스포머

ViT(Vision Transformer) : 이미지 인식을 위한 딥러닝 모델

- 이미지를 자연어 처리 방식처럼 분류해 보려는 시도에 의해 탄생
- 지역 특징을 추출하는 CNN모델의 합성곱 계층 방법 사용 X, 트랜스포머 모델에서 사용되는 *셀프 어텐션*(전체 이미지 한번에 처리) 적용
- BUT 2차원 구조의 이미지 특성을 온전히 반영 X → 스윈 트랜스포머, CvT 모델들이 제안됨

스윈 트랜스포머(Swin Transformer): 로컬 윈도(Local Window)를 활용해 각 계층의 어텐션이 되는 패치의 크기와 개수를 다양하게 구성해 이미지의 특징을 학습시킴

CvT: 기존 합성곱 연산 과정을 ViT에 적용한 모델로, 저수준 특징과 고수준 특징을 계층적으로 반영함

- 어텐션 과정에서 쿼리, 키 , 값 중 키와 값을 기존 특징 벡터보다 축소해 계산 복잡도를 ↓

ViT

ViT(Vision Transformer) : 트랜스포머 구조 자체를 컴퓨터비전 분야에 적용한 첫 번째 연구

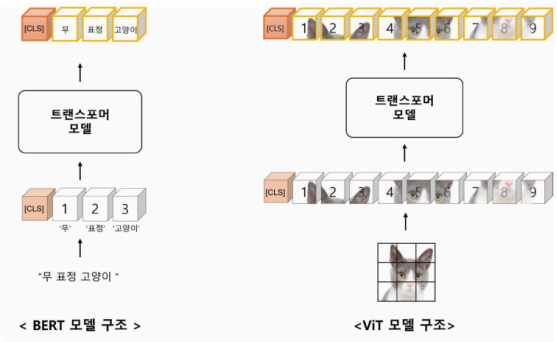

그림 10.2 BERT와 ViT 모델 구조 차이

BERT 모델과 ViT 모델 모두 트랜스포머 모델의 구조를 그래도 사용하지만, 입력 데이터를 만드는 과정이 다르다

BERT-문장보다 작은 단위의 토큰들이 순차적으로 입력

ViT-이미지가 격자로 작은 단위의 이미지 패치로 나뉘어 순차적으로 입력

⇒ ViT 모델에 사용되는 입력 이미지 패치는 왼→오, 위→아래로 표현된 시퀀셜 배열을 가정함

합성곱 모델과 ViT 모델 비교

합성곱 신경망과 트랜스포머의 공통점 : 이미지 특징을 잘 표현하는 임베딩을 만들고자 함

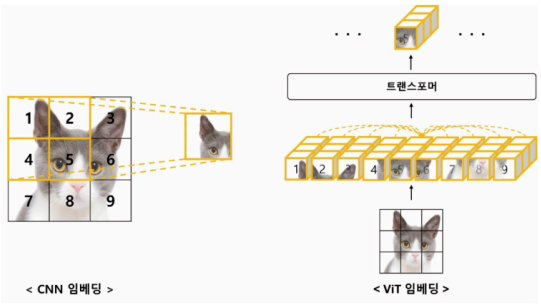

그림 10.3 합성곱 신경망과 ViT 계층의 차이

합성곱 신경망의 임베딩: 이미지 패치 중 일부만 선택하여 학습하며, 이를 통해 이미지 전체의 특징 추출

ViT 임베딩: 이미지를 작은 패치들로 나눠 각 패치 간의 상관관계를 학습, 셀프 어텐션 방법을 사용하 모든 이미지 패치가 서로에게 주는 영향을 고려해 이미지의 전체 특징 추출

⇒ ViT는 모든 이미지 패치가 학습에 관여하여 높은 수준의 이미지 표현 제공

- 좁은 수용 영역을 가진 합성곱 신경망은 전체 이미지 정보를 표현하는데 수많은 계층이 필요하지만, 트랜스포머 모델은 어텐션 거리를 계산하여 오직 한 개의 ViT 레이어로 전체 이미지 정보를 쉽게 표현 O
- ViT는 패치 단위로 이미지를 처리하기 때문에 더 작은 모델로도 높은 성능을 얻을 수 있음
- ViT 모델은 입력 이미지의 크기가 고정되어 있어, 이미지 크기를 맞추는 전처리 과정이 필요
- ViT는 패치간의 상대적인 위치 정보만 고려하기 때문에 이미지 변환에 취약

ViT의 귀납적 편향

귀납적 편향(Inductive Bias) : 일반화 성능 향상을 위한 모델의 가정을 의미

표 10.1 딥러닝 모델의 귀납적 편향

딥러닝 모델	귀납적 편향
합성곱 신경망(CNN)	지역적 편향
순환 신경망(RNN)	순차적 편향
ViT	귀납적 편향이 거의 없음

귀납적 편향은 해당 모델이 가지는 구조와 매개변수들이 데이터에 적합한 가정을 하고 있음을 나타냄

BUT 너무 강한 귀납적 편향은 다른 유형의 데이터나 관계를 표현하는데 어려움을 초래함

ViT 모델은 입력 데이터의 다양한 쿼리, 키, 값의 임베딩 형태로 일반화된 관계를 학습하기 때문에 귀납적 편향이 거의 없고, 대용량 데이터에 대해서 이미지 특징들의 관계를 잘 학습하는 모델로 알려져 있다.

ViT 모델

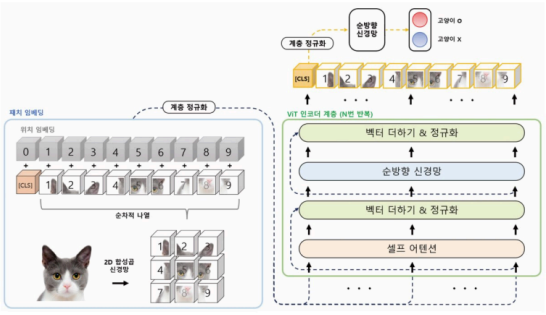

그림 10.4 ViT 모델 구조

ViT 모델 : 입력 이미지를 트랜스포머 구조에 맞게 일정한 크기의 패치로 나눈 다음 각 패치를 벡터 형태로 변환하는 **패치 임베딩**과 각 패치와의 관계를 학습하는 **인코더** 계층으로 구성

패치 임베딩과 인코더 계층을 통해 이미지의 특징을 추출하고 분류나 회귀와 같은 작업에 맞는 출력값으로 변환해 사용

패치 임베딩

패치 임베딩(Patch Embedding) : 입력 이미지를 작은 패치로 분할하는 과정

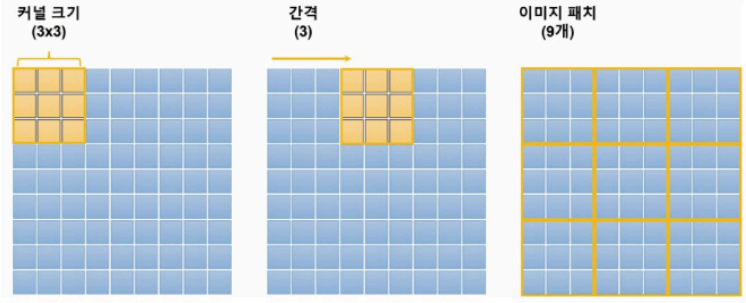

그림 10.5 이미지 패치 생성 과정

1. 이미지 크기를 일정한 크기로 변경
2. 전체 이미지를 패치 크기로 분할하여 시퀀셜 배열 형성(합성곱 신경망 계층 활용)
3. 커널 크기와 stride 설정, 커널 = 패치 크기, stride = 패치가 이동하는 폭
4. 분할된 이미지 패치들을 왼쪽에서 오른쪽, 위에서 아래의 순서로 배열
5. 이 배열 가장 왼쪽에 **분류 토큰**(전체 이미지를 대표하는 벡터로, 득정 문제를 예측하는 데 사용)을 추가
6. 위치 임베딩을 사용하여 인접한 패치 간의 관계 학습
7. 계층 정규화를 적용

인코더 계층

ViT 모델은 이미지를 일정한 크기의 패치로 나눠 각 패치를 벡터로 변환한 후 인코더 계층에 입력으로 전달해 다양한 쿼리, 키, 값 임베딩의 관계를 학습

N개의 인코더 레이어를 반복적으로 적용한 후, 마지막 레이어에서는 분류 토큰이라고 불리는 특별한 패치의 특징 벡터 추출

Swin Transformer

스윈 트랜스포머(Swin Transformer) : 기존 트랜스포머 기반 모델이 가지고 있는 문제를 극복하기 위해 계층적 특징 맵을 구성해 1차 계산 복잡도를 갖는 모델

- 이미지 분류, 객체 감지, 영상 분할과 같은 인식 작업에서 강력한 성능을 내며, 트랜스포머 구조보다 이미지 특성을 더 잘 표현할 수 있음
- 시프트 윈도라는 기술을 이용해 입력 이미지를 일정한 크기의 패치로 분할하고 이 패치들을 쌓아 윈도를 구성, 구성된 윈도 영역만 셀프 어텐션 계산
- 시프트 윈도 방식은 기존 트랜스포커 모델과는 다르게 패치를 겹쳐 더 큰 효율성 제공, 계층마다 어텐션이 수행되는 패치의 크기와 개수를 계층적으로 적용해 처리하므로 높은 해상도와 다양한 크기를 가진 객체를 효율적으로 처리 가능

ViT와 스윈 트랜스포머 차이

표 10.4 ViT와 스윈 트랜스포머 비교

	ViT	스윈 트랜스포머
분류 헤드	[CLS] 토큰 사용	각 토큰의 평균값 사용
로컬 윈도 사용	X	O
상대적 위치 편향 사용	X	O
계층별 패치 크기	동일함	다양함

스윈 트랜스포머 모델 구조

스윈 트랜스포머는 이미지를 패치 파티션으로 구분한 후, 선형 임베딩, 스윈 트랜스포머 블록, 패치 병합 연산이 반복적으로 수행

1. 입력 이미지를 일정한 크기의 패치로 분할
2. 각 패치에 대해 선형 임베딩 수행(각 패치는 고정된 차원의 벡터로 변환)
3. 분할된 패치들을 기반으로 스윈 트랜스포머 블록 구성(어텐션 계층, 계층 정규화, 다층 퍼셉트론 등을 포함하며 패치 간 상호작용 수행)
4. 패치를 병합해 전체 이미지에 대한 분류 수행
5. 분할된 패치들을 순차적으로 합치면서 이미지의 전체적인 정보 추출

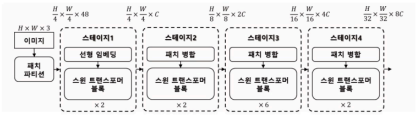

그림 10.7 스윈 트랜스포머 구조

스윈 트랜스포머 모델은 일반적으로 네 개의 스테이지로 구성

각 스테이지는 서로 다른 해상도와 패치 크기를 갖는다.

스테이지 1에서는 패치들이 선형 임베딩된 후 스윈 트랜스포머 블록으로 전달

스테이지 2,3,4에서는 패치 병합 후 각 스윈 트랜스포머 블록에 전달

패치 파티션 : 입력 이미지를 작은 사각형 패치로 분할해 처리하는 방식

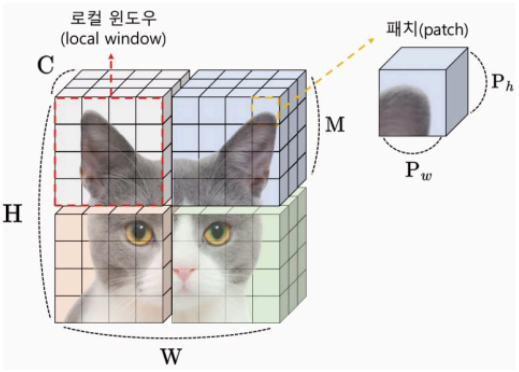

그림 10.8 패치 파티션 구조

패치 병합: 인접한 패치들의 정보를 저차원으로 축소하는 과정

스윈 트랜스포머 블록 : 계층 정규화, W-MSA, MLP, SW-MSA로 구성

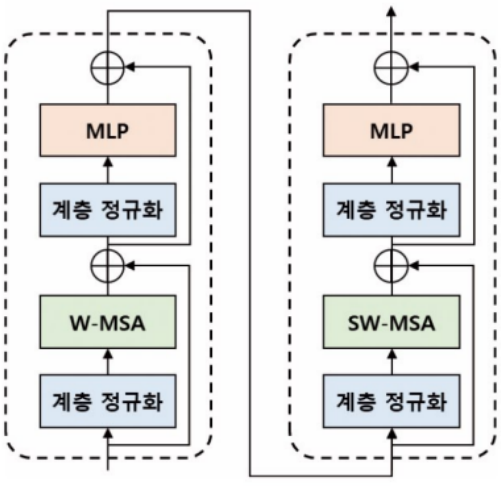

그림 10.9 스윈 트랜스포머 블록 구조

W-MSA : 로컬 윈도를 사용하는 멀티 헤드 셀프 어텐션으로, 입력 특징 맵을 중첩되지 않게 나눈 다음 각 윈도에서 독립적으로 셀프 어텐션 수행

- 각 윈도 내 지역적인 특징 분석 가능
- 전체 입력에 대한 셀프 어텐션 비용을 2차 계산 복잡도에서 1차 계산 복잡도로 줄임
- 효율적인 배치 계산을 통해 연산량 대폭 감소
- 윈도 내부의 이미지 패치 영역에만 셀프 어텐션을 수행한다는 단점 존재

스윈 트랜스포머에서 사용한 셀프 어텐션은 상대적 위치 편향을 고려함

상대적 위치 편향은 로컬 윈도 안에 있는 패치 간의 상대적인 거리를 임베딩하는 목적을 가짐

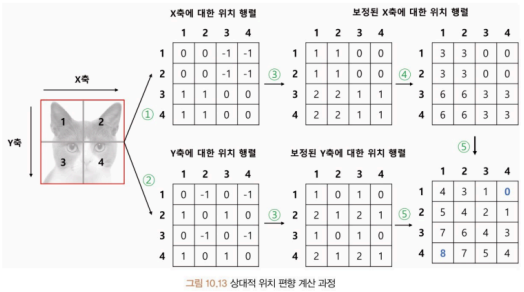

그림 10.13 상대적 위치 편향 계산 과정

X축 방식은 두 패치 간의 가로 방향 거리를 의미하며, Y축 방식은 세로 방향 거리를 의미

```
import torch

window_size = 2
coords_h = torch.arange(window_size)
coords_w = torch.arange(window_size)
coords = torch.stack(torch.meshgrid([coords_h, coords_w], indexing="ij"))
coords_flatten = torch.flatten(coords, 1)
relative_coords = coords_flatten[:, :, None] - coords_flatten[:, None, :]

print(relative_coords) #1,2번 연산 과정
print(relative_coords.shape)
```

window size는 여러 개의 이미지 패치를 포함하는 격자의 크리르 의미하며 윈도 내외부의 패치를 구분해주는 역할

torch.meshgrid 함수는 coords_h 배열 값과 coords_w 배열 값으로 사각형 격자를 만드는데 사용되며, 사각형의 X좌표, Y좌표에 해당하는 배열을 반환한다.

torch.stack을 사용하여 (X,Y) 쌍 배열을 만든 후 torch.flatten으로 각 X, Y축에 대한 위치 인덱스를 생성

마지막으로 각 위치 색인의 차이를 계산하면 X,Y에 대한 위치 행령 1,2를 얻을 수 있다.

```
x_coords = relative_coords[0, :, :]
y_coords = relative_coords[1, :, :]

x_coords += window_size - 1  # X축에 대한 3번 연산 과정
y_coords += window_size - 1  # Y축에 대한 3번 연산 과정
x_coords *= 2 * window_size - 1  # 4번 연산 과정
print(f"X축에 대한 행렬:\n{x_coords}\n")
print(f"Y축에 대한 행렬:\n{y_coords}\n")

relative_position_index = x_coords + y_coords  # 5번 연산 과정
print(f"X, Y축에 대한 위치 행렬:\n{relative_position_index}")
```

위의 코드는 위치 행렬 1,2의 결과를 이용해 상대적 위치 편향 값을 산출하는 과정이다.

relative_coords를 각 X, Y 좌표에 대한 x_coords, y_coords로 분리한 후, 각 x_coords, y_coords에 대해서는 window_size -1만큼 더해준다(3번 연산과정)

그 다음 x_coords에 대해서 2*window_size -1만큼 곱해준다(4번 연산과정)

x_coords, y_coords를 더해주면 X, Y축에 대한 상대적 위치 좌표 행렬을 생성할 수 있다. (5번 연산과정)

```
num_heads = 1
relative_position_bias_table = torch.Tensor(
    torch.zeros((2 * window_size - 1) * (2 * window_size - 1), num_heads)
)

relative_position_bias = relative_position_bias_table[relative_position_index.vie
relative_position_bias = relative_position_bias.view(
    window_size * window_size, window_size * window_size, -1
)
print(relative_position_bias.shape)
```

위의 코드는 X,Y 축에 대한 상대적 위치 좌표를 벡터로 변환하는 과정이다.

CvT

CvT : 합성곱 신경망 구조에서 활용하는 계층형 구조를 ViT에 적용한 모델

- 이미지의 지역 특징과 전역 특징을 모두 활용해 비교적 적은 수의 매개변수로 높은 성능을 보임
- 합성곱 계층의 지역 특징을 추출하고, ViT의 셀프 어텐션 계층으로 전역 측징을 결합해 이미지 처리
- 스윈 트랜스포머보다 더 적은 수의 매개변수로도 비슷한 수준의 성능을 보임

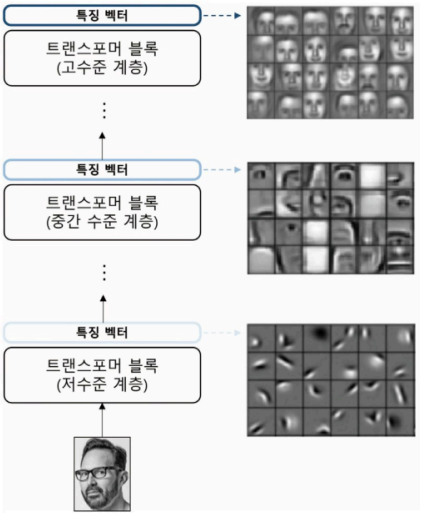

그림 10.14 CvT 모델의 계층적 특징 구조 예시

저수준 계층에서 선 또는 점에 대한 특징 표현

중간 수준에서는 누이나 귀와 같은 특징 표현

고수준 계층에서는 얼굴 구조에 대한 특징 표현

⇒ CvT의 계층형 구조는 물체의 크기나 해상도를 더 잘 표현

표 10.4 ViT와 CvT 모델 비교

모델	위치 임베딩	패치 임베딩	어텐션 임베딩	계층형 구조
ViT	O	겹치지 않는 선형 임베딩	선형 임베딩	X
CvT	X	겹치는 합성곱 임베딩 (overlapping)	합성곱 임베딩	O

합성곱 토큰 임베딩

합성곱 토큰 임베딩 : 2D로 재구성된 토큰 맵에서 중첩 합성곱 연산을 수행하는 임베딩

1. CvT 모델은 이미지의 2D 구조를 보존하면서도 셀프 어텐션을 계산하는 방식 사용
2. 이미지 패치를 2D 합성곱 임베딩을 사용해 쿼리, 키, 값 임베딩으로 변환
3. 변환된 임베딩을 사용해 셀프 어텐션을 계산하고 출력을 생성

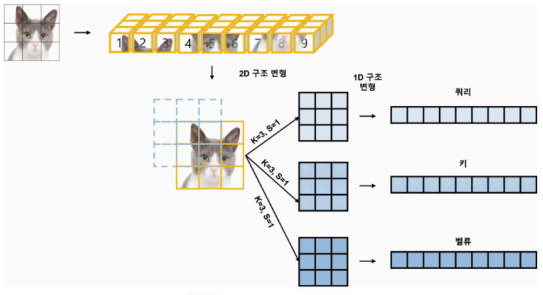

그림 10.15 2D 합성곱 기반의 어텐션 임베딩

이미지에 3×3 커널 크기, 1 간격의 커널을 적용하면 패치 크기는 3×3으로 생성

각각의 패치를 쿼리, 키, 값에 대해 합성곱 연산을 적용하고 1차원 구조로 재배열해 기존 어텐션 방법으로 학습

CvT 모델은 지역 특징을 학습할 수 있고, 시퀀스 길이를 줄이는 동시에 토큰 특징의 차원을 증가

합성곱 모델에서 수행하는 방식처럼 다운샘플링과 특징 맵의 수를 늘리게 됨

어텐션에 대한 합성곱 임베딩

어텐션에 대한 합성곱 임베딩: 2D로 재구성된 토큰맵에서 분리 가능한 합성곱 연산을 적용해 성능 저하 및 계산 복잡도를 낮추는 연산을 의미

일반적으로 쿼리, 키, 값, 임베딩은 차원을 동일하게 설정하지만, CvT 모델은 매개변수의 수를 줄이기 위해 쿼리, 키, 값의 차원을 축소

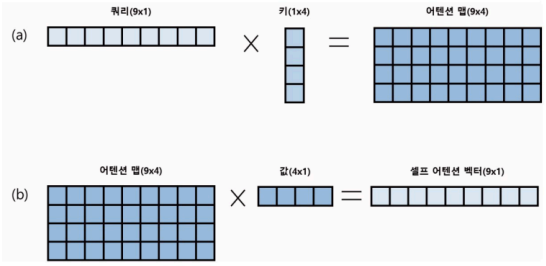

그림 10.17 축소된 셀프 어텐션 임베딩(열 × 행) 예시

9개의 쿼리 패치 임베딩과 4개의 키 패치 임베딩에 대한 중요도를 의미하는 **어텐션 맵** 생성

어텐션 맵이 생성되면 값 패치 임베딩으로 쿼리 어텐션 맵에 대한 가중치 반영

셀프 어텐션 벡터를 확인해 보면 입력 벡터와 출력 벡터의 패치 길이가 동일한 것을 확인할 수 있다.

그러므로 계산해야 하는 이미지 패치 개수가 줄어들어 연산량 ↓

⇒ CvT 모델은 트랜스포머 계층이 깊어질수록 패치 간의 관계를 학습하는 길이를 축소시키는 대신에 벡터의 차원을 증가시켜 모델의 매개변수 수를 대폭 ↓