Vision Transformer (ViT)

Sang Yup Lee

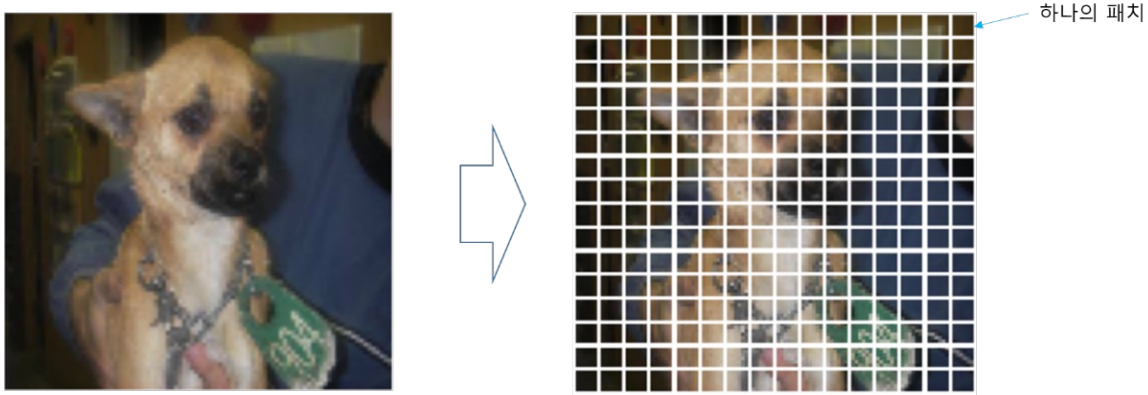
ViT

- ViT의 주 목적
  - Transformer를 이미지 분류에 적용
- Motivations
  - CV 분야에서는 여전히 CNN 기반 방법들이 주
  - CNN과 self-attention 결합 시도 (Wang et al., 2018; Carion et al., 2020 등) => 하지만, 그렇게 성공적이지 못함
  - CNN 구조의 특성 (inductive bias라고 표현)
    - Locality
    - Parameter sharing (translation equivariance)
    - 하지만, 이런 것이 한계로 작용할 수 있다.
  - Transformer는 이러한 inductive bias가 덜하다/부족하다.
- Related works
  - 어떠한 시도들이 있었는지를 알고자 하는 사람들은 해당 논문 참고
  - 연구 아이디어를 얻는데 도움

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.

ViT

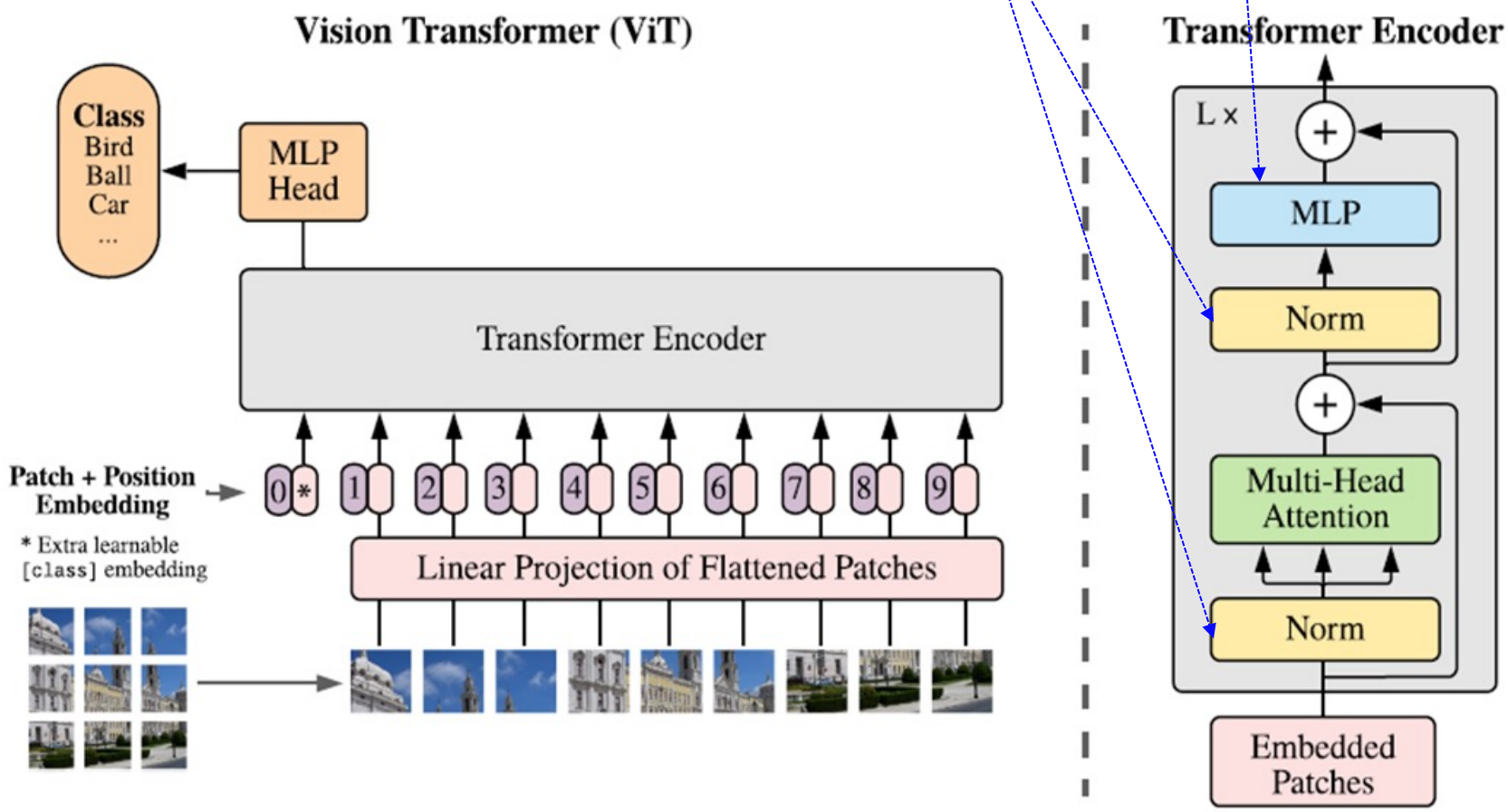
- 주요 내용
  - 원래의 트랜스포머를 가능한 한 변화를 주지 않고 이미지 데이터에 적용하여 이미지 분류 작업을 수행
  - 이를 위해 ViT 논문에서는 하나의 이미지를 여러 개의 패치(patch) 단위로 분할, 각 패치는 원 트랜스포머가 적용된 시퀀스 데이터에서의 토큰으로 간주
  - 각 패치에 대한 임베딩 벡터를 생성하고, 그렇게 생성된 임베딩 벡터를 트랜스포머 인코더 블록의 입력값으로 입력
  - 나머지 부분은 기존 트랜스포머 혹은 문서 분류에 적용된 BERT가 작동하는 방식과 거의 동일 (ViT도 트랜스포머의 인코더 부분만 사용)



ViT의 구조

원래의 모형에 비해서 Layer Norm 적용 순서 차이

GELU 활성화 함수를 사용

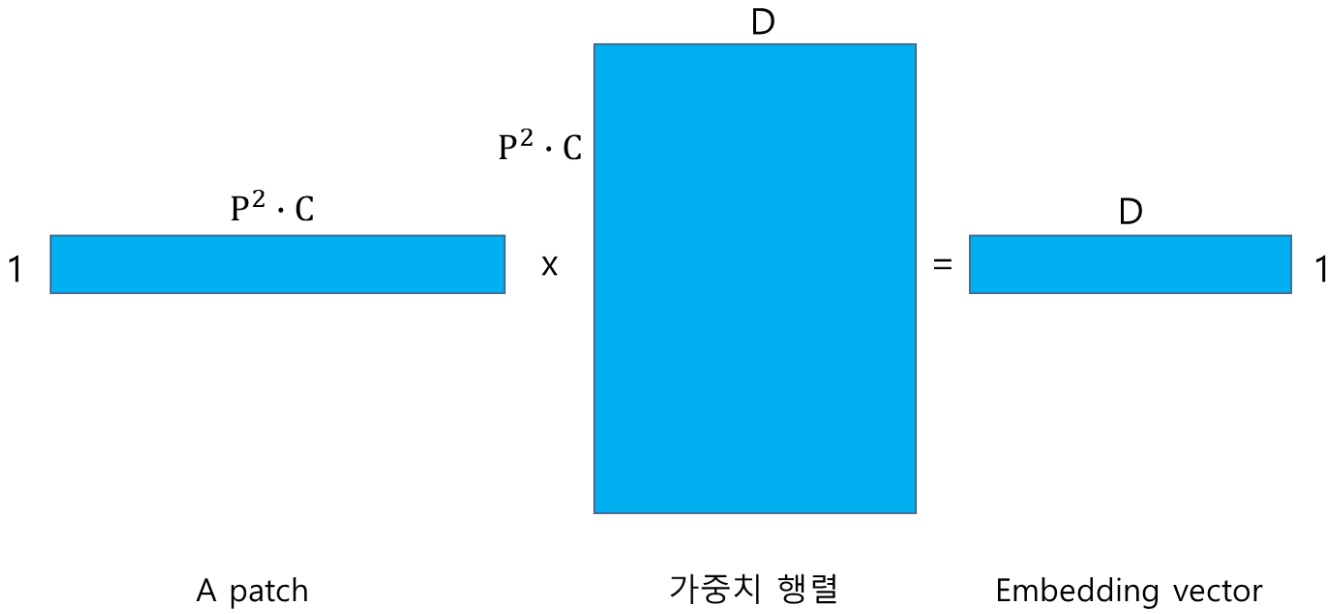


ViT

- 주요 작동 과정
  - 이미지를 여러 개의 겹치지 않는 정사각형의 패치 (patch)로 분할
  - 각 패치가 하나의 토큰으로 간주
  - 하나의 픽셀을 토큰으로 간주할수도 있지만, 그렇게 되면 연산량이 너무 많아짐
  - HxW(xC)의 이미지를 PxP(xC) 형태의 패치로 분할 (중복 없이)
    - # of patches = $HW/P^2 = N$
    - N개의 패치가 존재하고 각 패치가 갖는 색상 정보의 수는 PxPxC
  - ViT에서는 PxPxC 형태 (즉, 3D array 형태)의 패치를 평탄화 하여 1D 형태 (즉, $1xP^2 \cdot C$)로 변환한 다음 선형 변환을 통해서 D 차원의 임베딩 벡터를 생성
  - 생성된 임베딩 벡터에 위치 임베딩 정보를 더한 정보가 인코더 블록의 입력값으로 입력
  - BERT와 마찬가지로 이미지 분류를 위해 [class] 토큰을 사용

ViT

- 주요 작동 원리
  - linear projection



ViT

- 주요 작동원리
  - Linear projection => 임베딩 벡터 (아래에서 식(1))
  - E가 가중치 행렬
  - $\mathbf{x}_p^i$는 각 패치의 벡터를 의미
  - $\mathbf{x}_{\text{class}}$는 [CLS] 토큰 벡터를 의미

위치 임베딩 벡터 사용

MLP (이는 트랜스포머에서 위치기반 FFN를 의미)에서는 GELU 활성화함수를 사용하는 두 개의 층이 사용

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \cdots; \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \tag{1}$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \qquad \ell = 1 \ldots L \tag{2}$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \qquad \ell = 1 \ldots L \tag{3}$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \tag{4}$$

Layer L 즉, 마지막 인코더 블락에서 출력하는 [class] 토큰의 은닉 상태 벡터 (위에서는 $\mathbf{z}_L^0$이라고 표현되었음)는 이미지를 대표하는 값이 되고 이를 이용해서 종속변수의 값을 예측합니다 (식 (4)).

ViT

- 모형의 성능

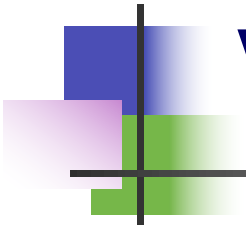
Dataset	ViT-Huge	ViT-Large	ResNet baselines
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	87.54 ± 0.02
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	90.54
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.37 ± 0.06
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.51 ± 0.08

- ViT-Large와 Vit-Huge의 차이

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

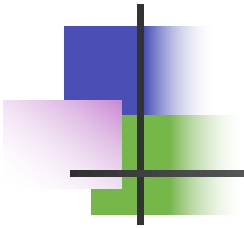
ViT

- Hybrid Architecture
  - CNN을 적용해서 도출되는 feature map을 이용해서 패치를 추출하고, 거기에 임베딩 프로젝션을 적용해서 임베딩 벡터를 만들 수 있다.
  - special case: 패치의 크기 = 1x1



ViT

- Python coding
  - Training from scratch
    - 1) ViT_image_classification_example.ipynb 참고
    - 2) ViT_image_classification_example_cats_dogs.ipynb 참고
    - pip install -U tensorflow-addons # addons 설치
    - Tensorflow 2.9.0 권장
    - 학습 데이터의 양이 많지 않은 경우, 모형의 성능이 좋지 않다. 주된 이유는 inductive bias가 부족하기 때문.
  - 사전 학습을 이용한 fine-tuning
    - https://huggingface.co/blog/fine-tune-vit
  - PyTorch
    - https://www.akshaymakes.com/blogs/vision-transformer
    - https://lightning.ai/docs/pytorch/stable/notebooks/course_UvA-DL/11-vision-transformer.html



Q & A