ViT(Vision Transformer)

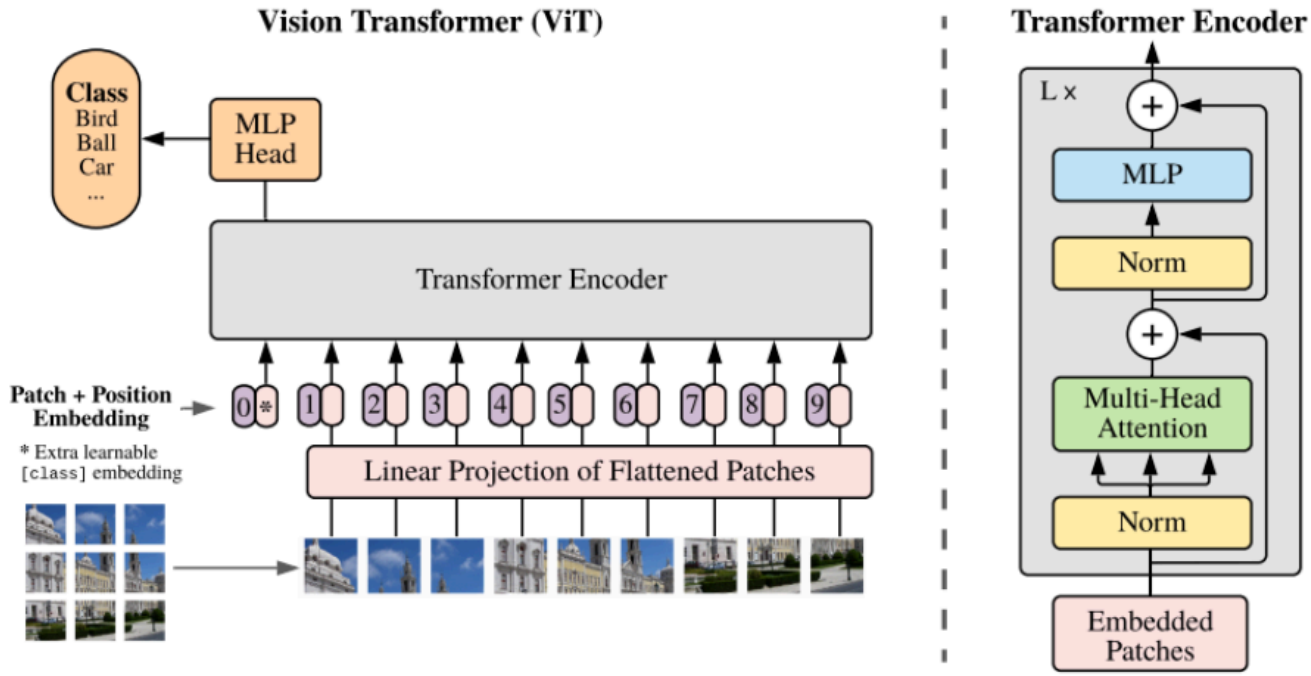
Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In International Conference on Learning Representations (ICLR). https://arxiv.org/abs/2010.11929

Transformer는 NLP 분야에서 사실상 표준이 되었으나, Computer Vision 분야에서는 여전히 CNN이 지배적이다.

기존 연구들은 CNN에 self-attention을 결합하거나 일부 모듈을 대체하는 방식이었다.

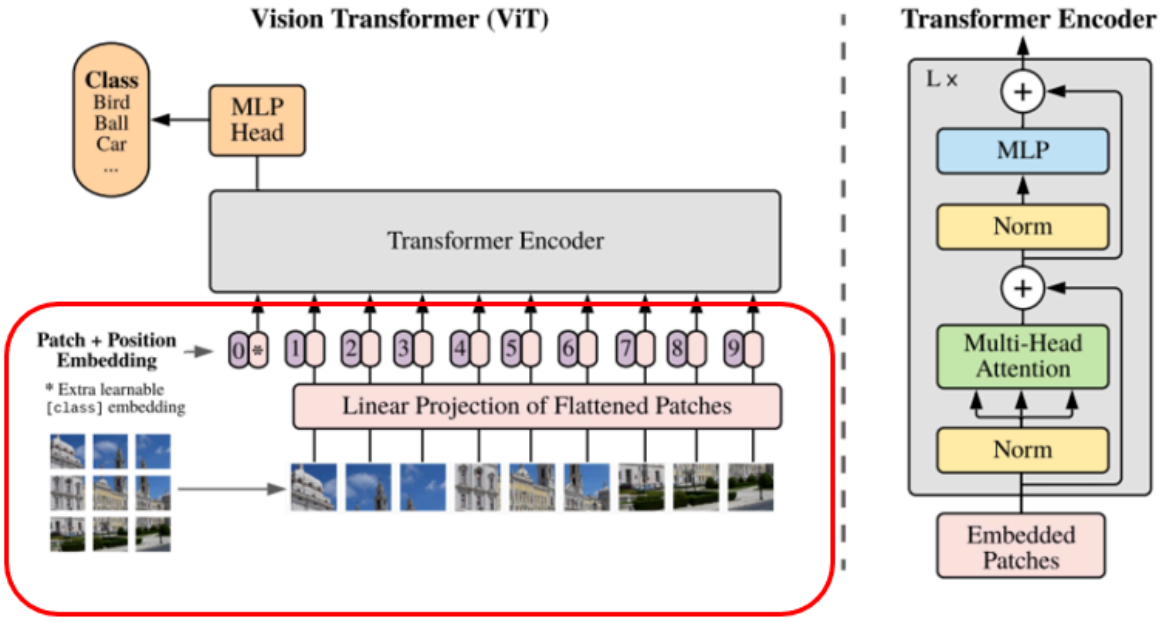
위 논문은 CNN에 의존하지 않고 순수 Transformer 구조만으로 이미지 인식이 가능한지를 실험적으로 검증한다.

구조 및 동작 방식



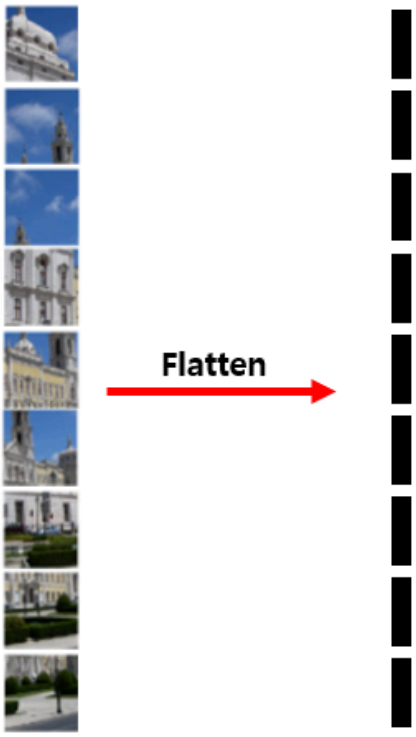
ViT의 핵심 아이디어는 **이미지를 NLP에서의 문장처럼 "토큰" 단위로 변환**하여 Transformer Encoder에 입력한다는 것이다.

전처리



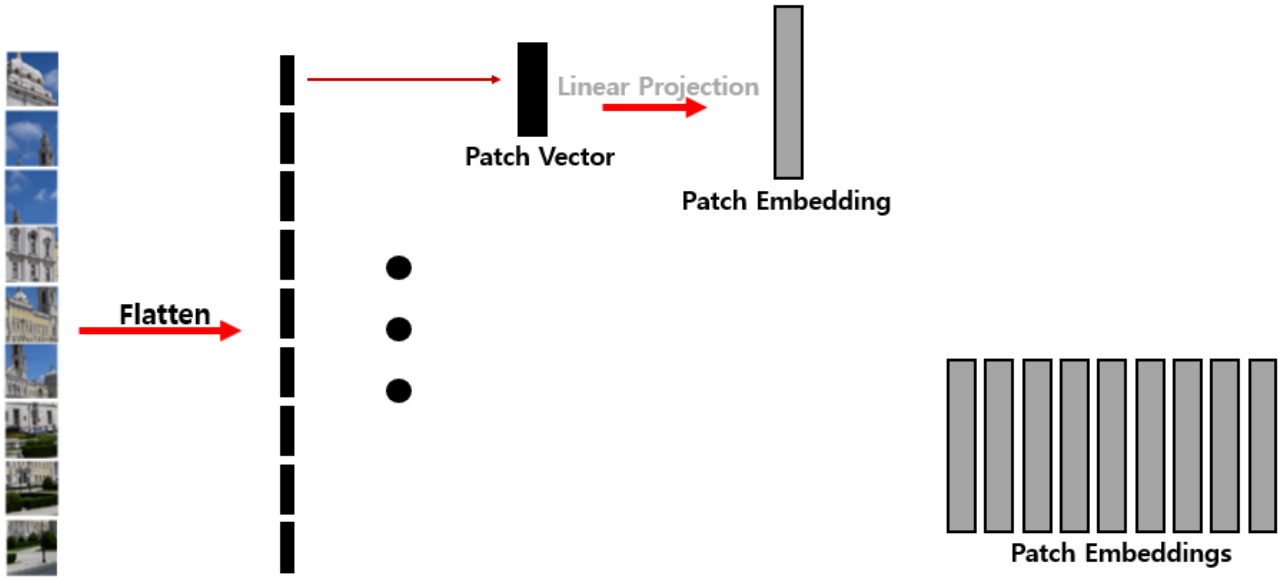
먼저, 전체 이미지를 동일한 크기의 패치(pxp) 이미지로 나누어준다.

이후, 패치들을 $p' = p^2 \times c$ 크기의 벡터로 평탄화(flatten)한다. 이때 p는 패치의 한 변의 크기이고, c는 채널 수이다.



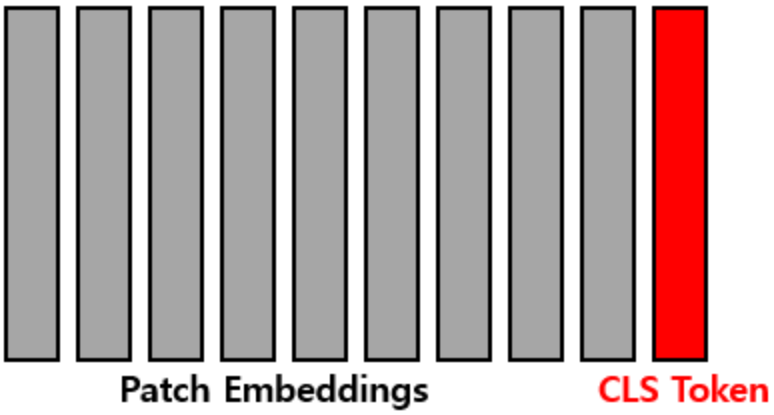
Transformer는 입력 시퀀스가 **고정 차원** d 이어야 하므로, 이미지 패치로부터 만든 벡터($p^2 \cdot C$) 들을 선형 변환을 통해 인코딩한다.

$$[x_p^1 E; x_p^2 E; \cdots; x_p^N E] \in \mathbb{R}^{n \times d}$$



이미지 패치들을 모두 고정된 크기의 벡터로 임베딩하게 되면 $n \times d$ 크기의 배열을 얻게된다. 여기서 n 은 이미지 패치의 개수이고, d는 하나의 패치가 임베딩된 크기이다.

이후, 모델을 효과적으로 학습하기 위해, 패치 임베딩에 추가로 분류 토큰인 **CLS Token** 벡터를 추가한다.



CLS Token은 BERT에서 가져온 아이디어로, 입력 시퀀스 전체를 대표하는 **global한 표현**을 학습하기 위한 특별한 벡터이다.

초기에는 무작위(random)로 초기화되며, 학습을 통해 모델이 의미 있는 표현을 학습하게 된다.

하나의 CLS 토큰만 존재하며, 모든 입력 이미지에 동일하게 사용된다. 입력 시퀀스 수식에, CLS 토큰 x_{class} 를 맨 앞에 붙이면 아래와 같은 식이 된다.

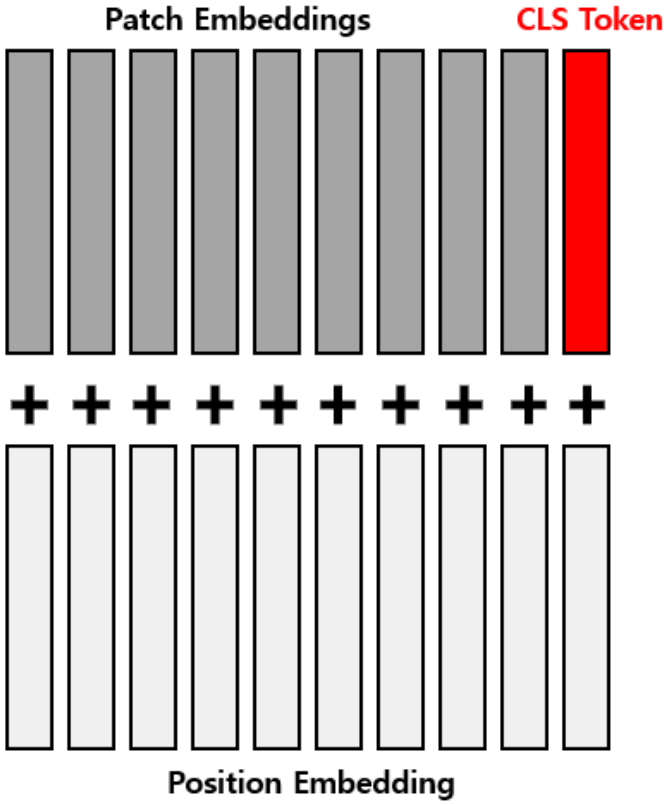
$$[x_{\text{class}}, x_p^1 E; x_p^2 E; \cdots; x_p^N E] \in \mathbb{R}^{(n+1)\times d}$$

즉, 패치 백터 n개 + CLS 1개 $\rightarrow$ 총 $n+1$ 개의 벡터이고, 각 벡터의 크기는 d이므로 최종 입력 시퀀스 크기는 $(n+1) \times d$가 된다.

Transformer는 각 토큰 간 self-attention을 수행하는데, 이때 CLS 토큰은 모든 패치와 상호작용하면서 **이미지 전체 정보를 집약**한다. 최종 Transformer 블록 출력에서 CLS 토큰 벡터만 뽑아내서 **분류기 (MLP Head)** 에 넣어 결과를 예측한다.

현재까지의 패치 임베딩은 단순히 픽셀 값을 펼쳐서 선형 투영한 벡터일 뿐, 어느 패치가 이미지의 어느 위치에 있는지 정보가 없다.

이를 해결하기 위해 Positional Embedding, 즉 각 패치 임베딩에 위치벡터를 더해준다.



각 패치 임베딩에 위치 벡터를 더해준 수식은 아래와 가탇.

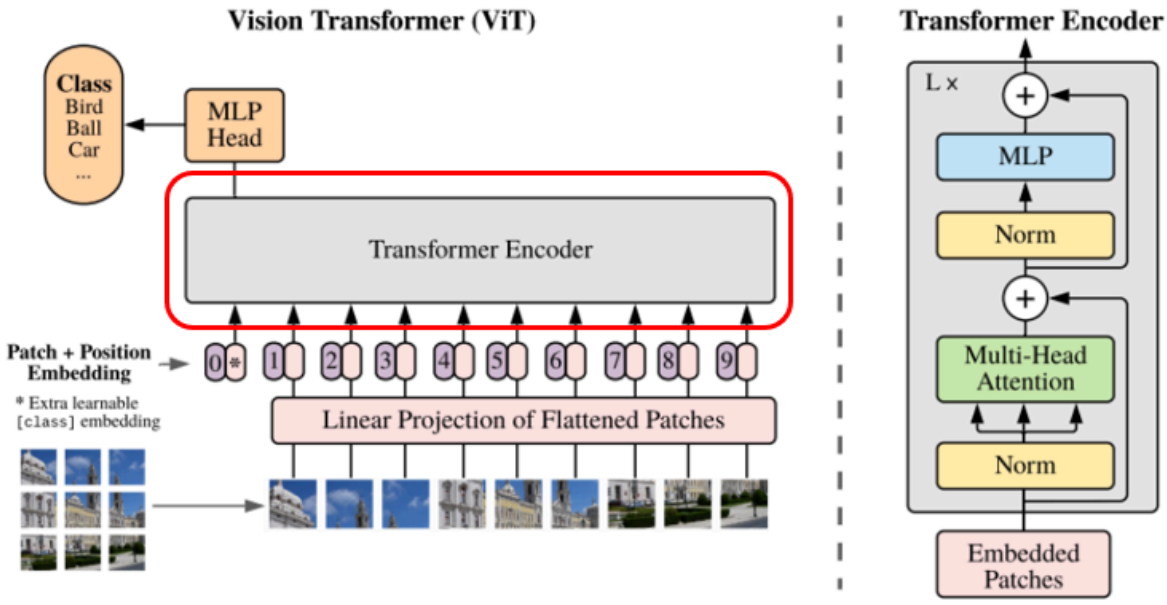
$$z^0 = [x_{\text{class}}; x_p^1 E; x_p^2 E; \cdots; x_p^N E] + E_{\text{pos}}$$

각 위치마다 고유한 학습 가능한 벡터를 갖는다. 이때 CLS 토큰도 하나의 위치로 간주되어, 별도의 Positional Embedding을 받는다.(동일하게 처리됨)

이때, Addition만 수행하기 때문에 임베딩 벡터의 차원 d는 변하지 않는다. 즉, 입력 시퀀스 전체의 shape는 여전히 $(n+1) \times d$이다.

이렇게 위치 임베딩 벡터가 추가된, $(n+1) \times d$ 크기의 배열을 Transformer에 입력한다.

Transformer Encoder



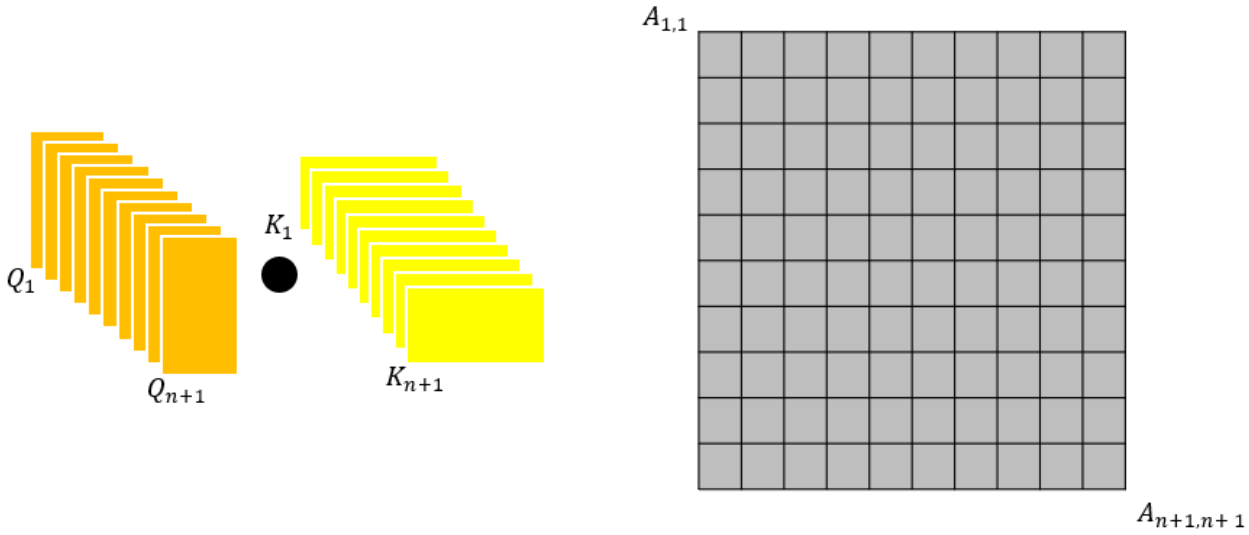
Transformer 입력 패치 임베딩 벡터는 Self-Attention을 하기 위해, 각 토큰 벡터를 **세 개의 다른 공간**으로 mapping한다.

$$Q = z^0 W_Q, \quad K = z^0 W_K, \quad V = z^0 W_V$$

즉, 입력의 각 벡터가 Query, Key, Value 세 가지 벡터로 변환된다. 이 세 가지 모든 벡터들을 n+1개씩 얻게 된다.

이후, Attention Score를 계산해준다. 계산된 Attention Score의 행렬 A의 모든 행의 합이 1이 되도록 모든 행에 softmax 함수를 적용한다.

$$\text{Softmax}\left(\frac{QK^T}{\sqrt{d_h}}\right)$$



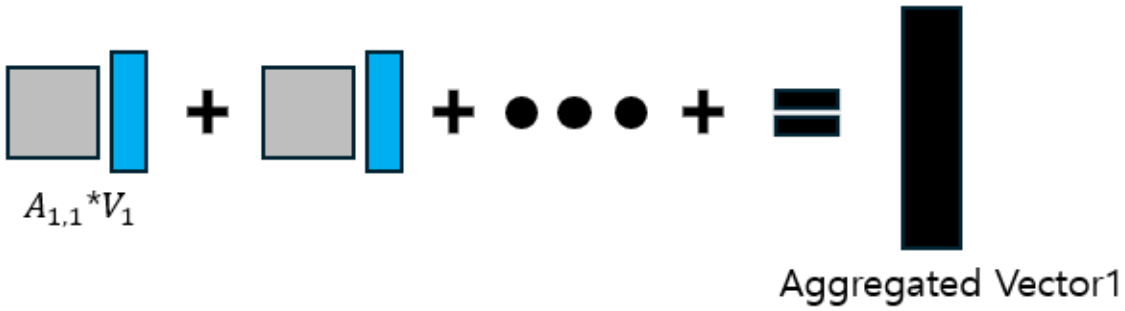
A의 첫 번째 행 A_1 를 보면, 첫 번째 패치 임베딩이 나머지 모든 토큰(자기 자신 포함)과의 관계를 담고 있다.

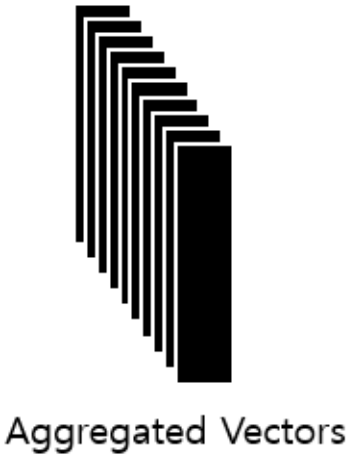
$$A_{1,:} = [\alpha_{11}, \alpha_{12}, \cdots, \alpha_{1(N+1)}]$$

여기서 α_{1j} 는 첫 번째 패치가 j번째 토큰을 얼마나 참고할지를 의미한다.

이제 Value 벡터들의 가중치로 합친다.

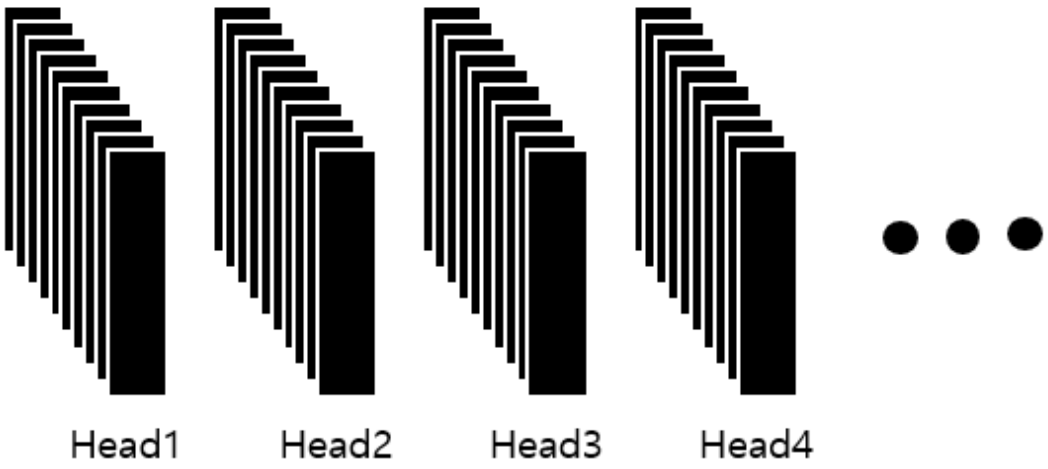
$$z_1 = \sum_{j=1}^{N+1} \alpha_{1j} V_j$$





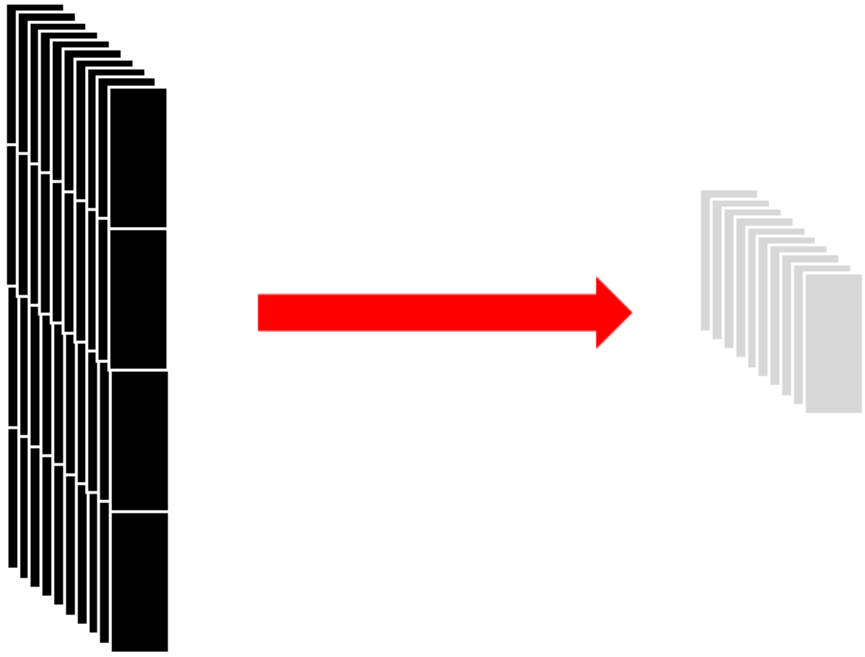
결과 z_1은 첫 번째 패치 임베딩에 대한 집계된 문맥 정보 벡터가 된다. Attention Score 행렬의 다른 행들에 대해서도 위 과정을 반복하여 N+1개의 집계된 문맥 정보 벡터를 구한다. 즉, 모든 패치마다 하나씩(N개) + 분류 토큰(CLS)에 대해서 하나, 총 N+1개이다. 여기까지해서 첫번째 어텐션 헤드(Attention Head)를 구한다.

Transformer는 Multi-head Attention 구조 이므로, 다른 QKV들에 대해, 위 과정을 반복한다.
위 과정을 반복하면 Head 수만큼의 Aggregated Vectors가 생성된다.

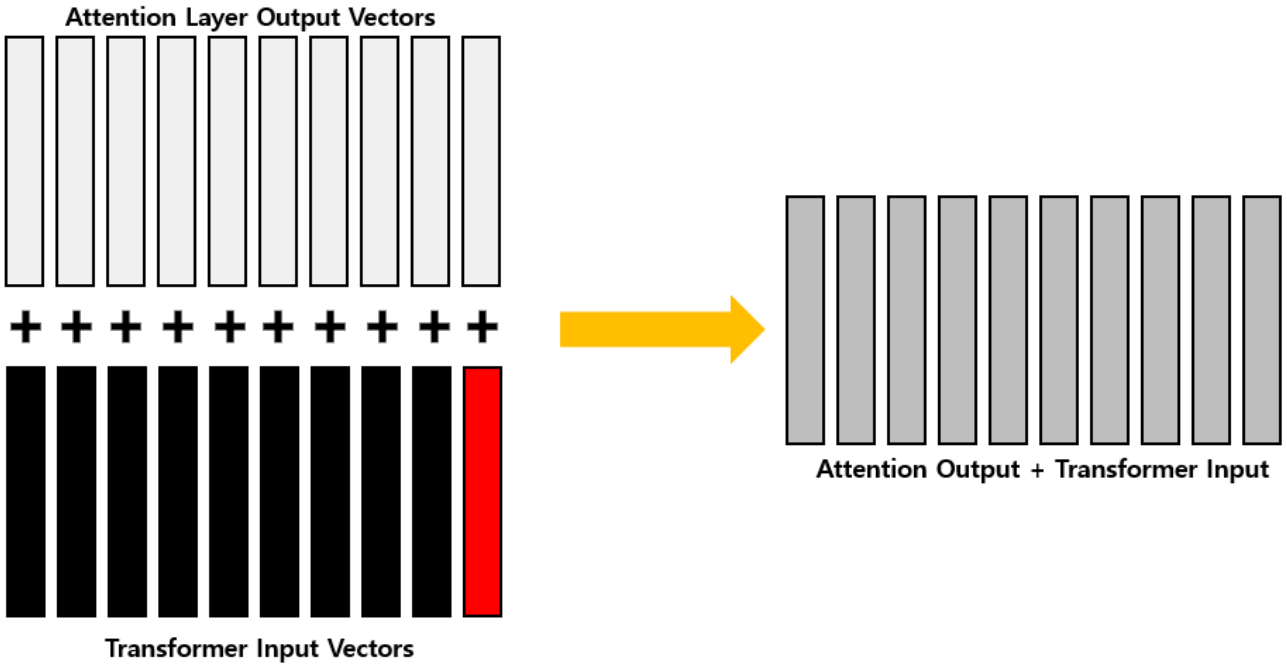


이렇게 생성된 여러 헤드를 쌓은 뒤, 패치 임베딩의 크기와 같은 d 크기의 벡터로 매핑시킨다.

$$\mathrm{MSA}(Z) = \mathrm{Concat}(\mathrm{head}_1, \cdots, \mathrm{head}_h)W_O$$

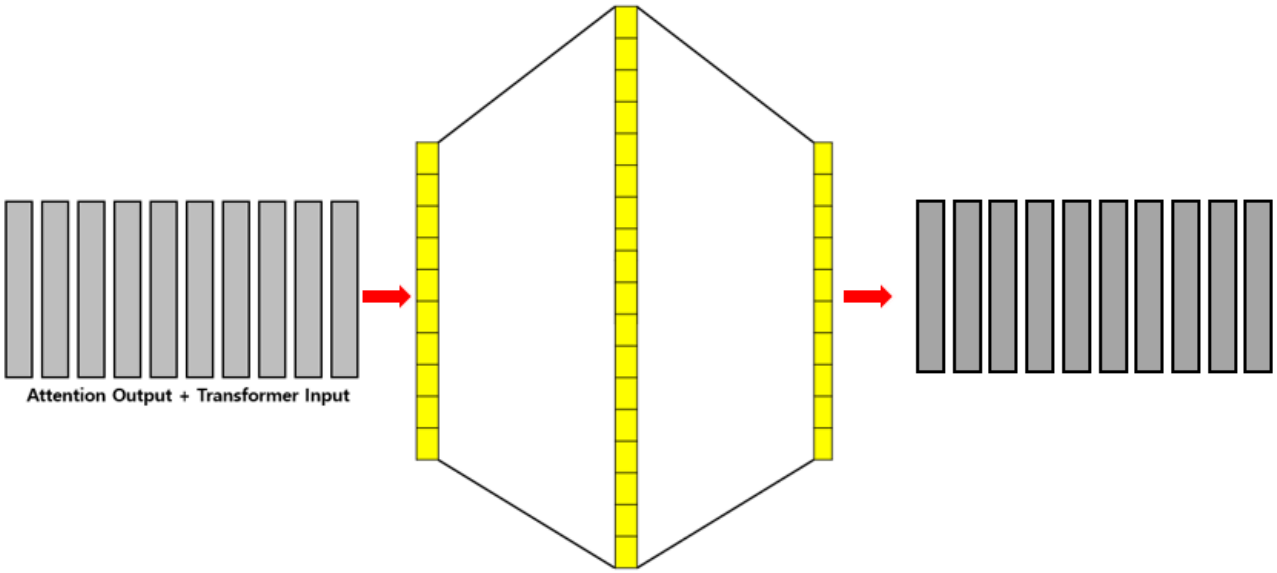


각 헤드는 서로 다른 관점(Representation Subspace)에서 Attention을 수행한다. 이를 합쳐서 다시 원래 차원 d 로 매핑해줌으로써, 입력과 출력이 동일한 차원 d 를 가지게 되어 Residual Connection을 바로 적용할 수 있다.

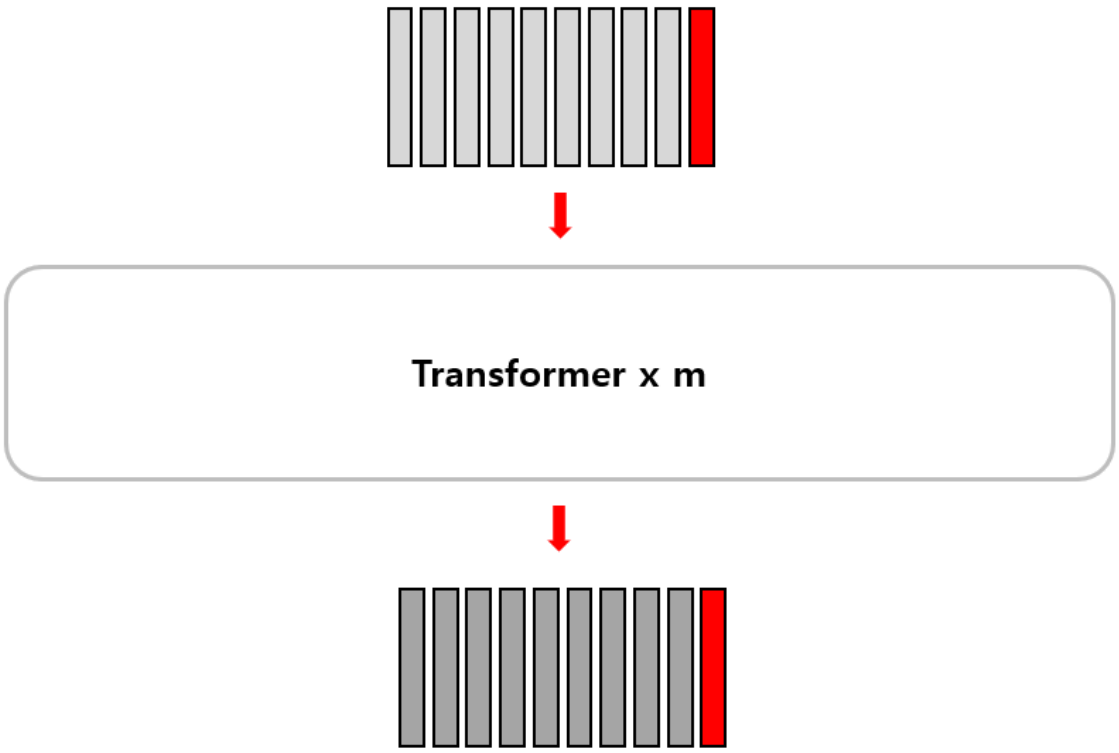


Residual Connection을 통해(동일 크기 d 벡터끼리 더함) 같은 크기의 벡터가 생성된다.

이후의 결과를, 비선형 활성함수를 갖는 Feed Foward Network에 통과시킨다.



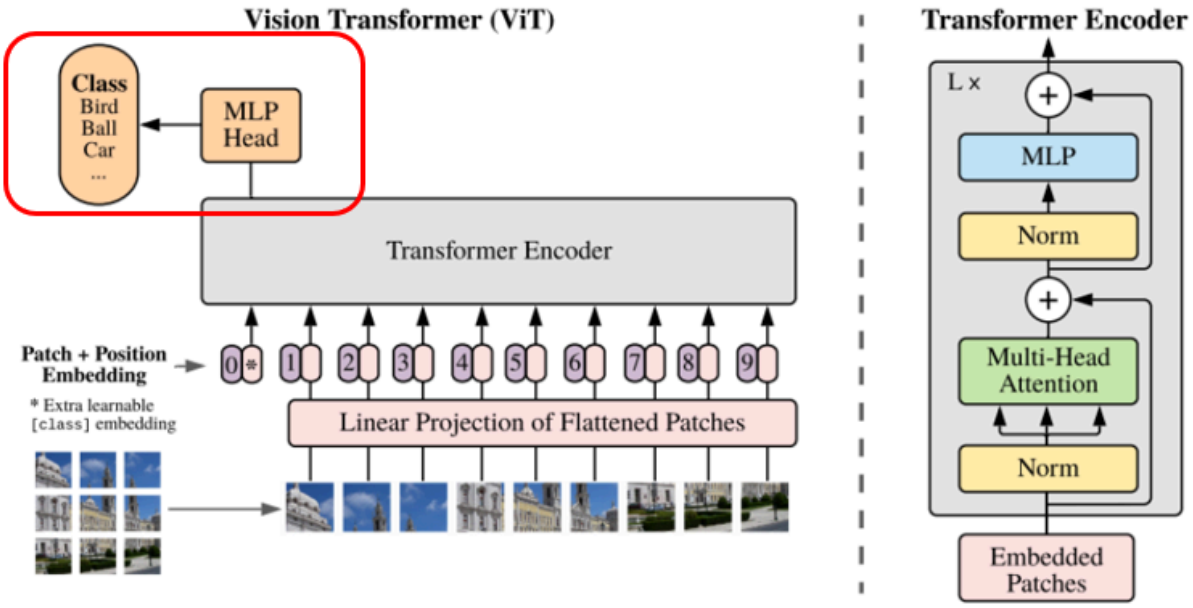
이렇게까지가 Transformer 레이어 연산이고, 최종적으로는 Transformer는 입력 크기와 같은 출력을 생성한다.



위에서 진행한 Transformer 연산을 수차례 반복한 후, (m번) 마지막 단계는 분류 토큰 (CLS Token) 출력을 확인하는 것이다.

CLS 토큰은 입력 이미지 전체의 정보를 종합적으로 담고 있으며, 모델이 학습 과정에서 이미지의 전역적인 특징을 추출하여 표현한 결과물이다.

이 벡터는 단순한 중간 표현이 아닌, 최종적으로 이미지를 어떤 클래스에 속할지 결정하기 위한 핵심 정보로 사용된다.



CLS 벡터는 완전 연결된 신경망 층(분류기Head)에 통과시킨다. 분류기는 CLS 벡터를 각 클래스의 logit 값으로 변환하며, 이어서 softmax 함수를 적용해 각 클래스에 대한 확률 분포를 출력한다.
(CLS 토큰 > 완전 연결 신경망 > Softmax)

실험 결과

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

- ViT-H/14 모델이 ResNet 기반 BiT, EfficientNet 기반 Noisy Student와 동등하거나 더 높은 성능을 달성.
- 특히 **CIFAR-100, VTAB** 등 다양한 다운스트림 데이터셋에서 CNN을 능가.
- **연산량(Compute)** 측면에서도 CNN 대비 훨씬 효율적 (예: ViT-L/16은 BiT-L 대비 15배 적은 compute로 유사/더 좋은 성능).

사용 시 유의점

1. 데이터 규모

- ViT는 CNN보다 Inductive bias가 적다.(지역성, Translation equivariance 같은 특성을 구조적으로 갖지 않음) 따라서 소규모 데이터셋에서 성능이 떨어질 수 있다.(ImageNet-1k 단독 학습 시 ResNet보다 성능 낮음)
- 대규모 Pre-training $\rightarrow$ Downstream Fine-tuning이 사실상 필수이다.

2. 연산 효율성
   - self-attention은 토큰 길이 기준으로 $O(N^2)$ 연산량을 갖는다.
   - 작은 패치 사용 시 토큰 수가 급격히 증가해 연산량이 폭증할 수 있다.
   - 일반적으로 16x16 패치가 균형점으로 많이 사용된다.
3. Positional Embedding 처리
   - ViT는 학습 가능한 위치 임베딩을 사용한다. 해상도가 바뀌면 위치 임베딩을 2D Interpolation으로 맞춰줘야 한다.
   - Fine-Tunning 시 해상도를 크게 할 경우 반드시 position embedding 보정이 필요하다.
4. 모델 크기
   - ViT는 규모가 클수록 성능이 더 좋아진다. 하지만 작은 데이터셋에서는 큰 모델이 오히려 오버피팅된다.
5. 학습 안정화(Regularization)
   - CNN보다 inductive bias가 약하므로 Overfittin에 취약하다.
   - ViT 논문에서는 해결하기 위해 다음을 활용했다.
     - 강한 Weight Decay(0.1)
     - Dropout, Label Smoothing
     - 큰 Batch size(ex. 4096)
     - 작은 데이터셋에서는 Data Augmentation, Mixup, CutMix 같은 기법이 필요하다.