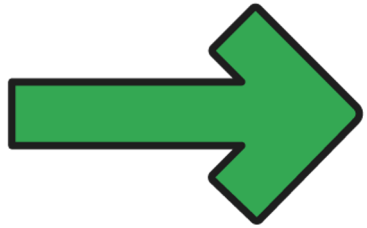
Google Developer Group
Incheon National Univ

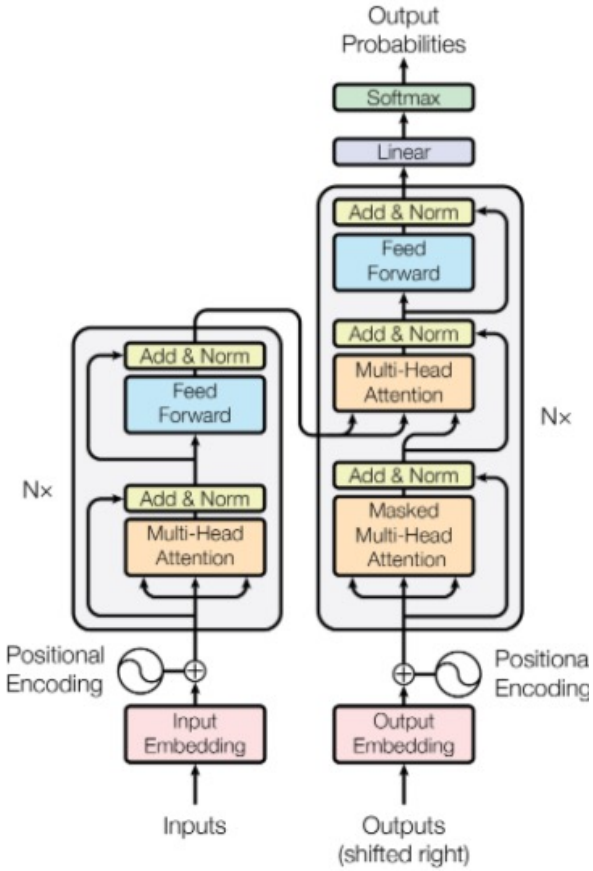
AN IMAGE IS WORTH 16X16 WORDS (ViT)

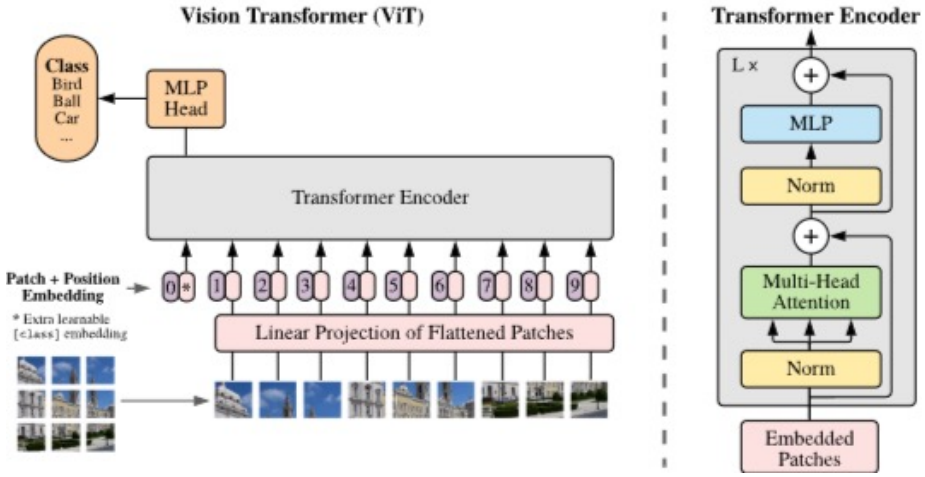
GDGoC INU AI Part Paper Seminar
김준수 이도형



Introduction

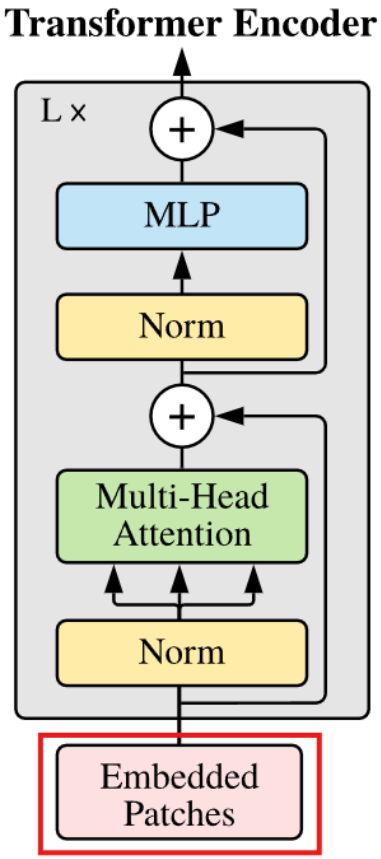
기존 자연어 처리 분야에서 뛰어난 성능을 보인 트랜스포머를 이용해서 이미지 처리 분야에서도 활용하기 위해서 ViT가 탄생되었습니다.



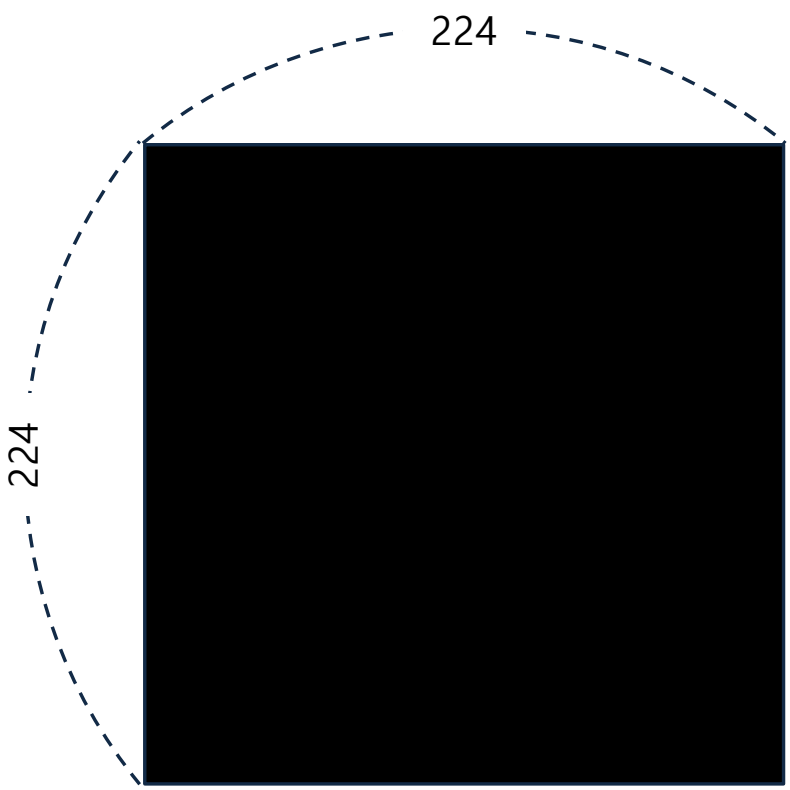


Method

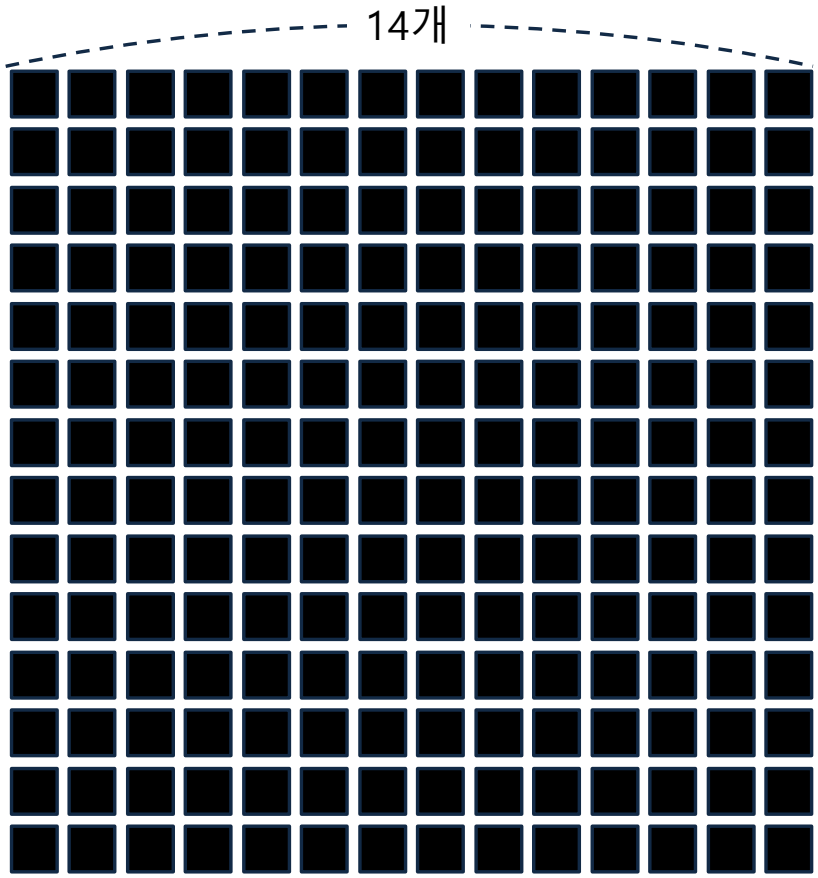
1. Embedded Patches



1단계: 이미지(224x224)를 16x16크기의 패치로 분할 - 14x14의 패치 생성

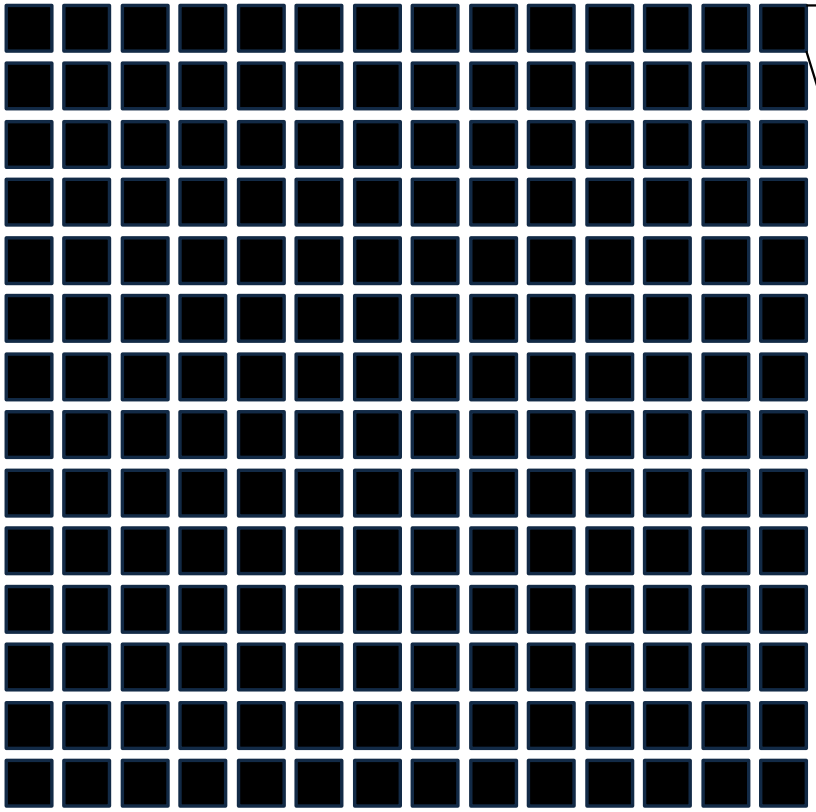


224x244 크기의 이미지

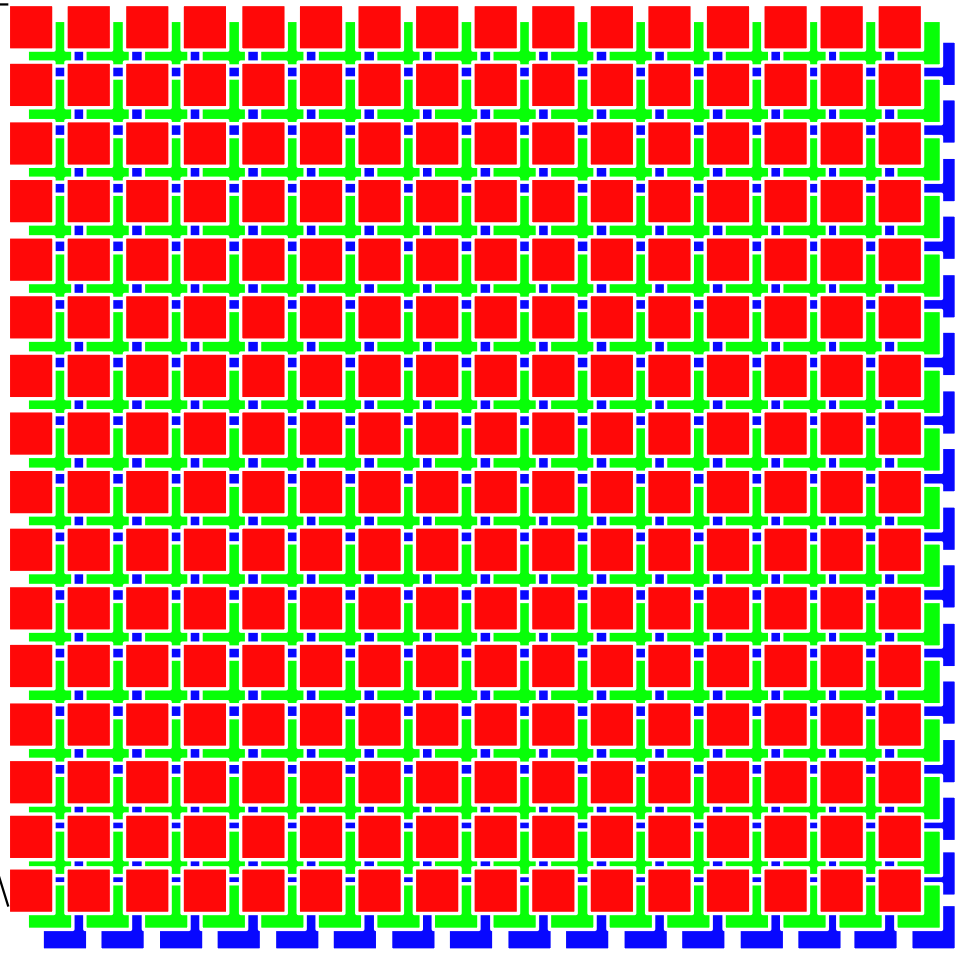


16x16크기의 패치로 나눈 이미지
224를 16으로 나눠 196개의 패치가 생성

1단계: 이미지(224x224)를 16x16크기의 패치로 분할 - 한개의 패치는 16x16x3(채널 수)이다. 즉, 768개

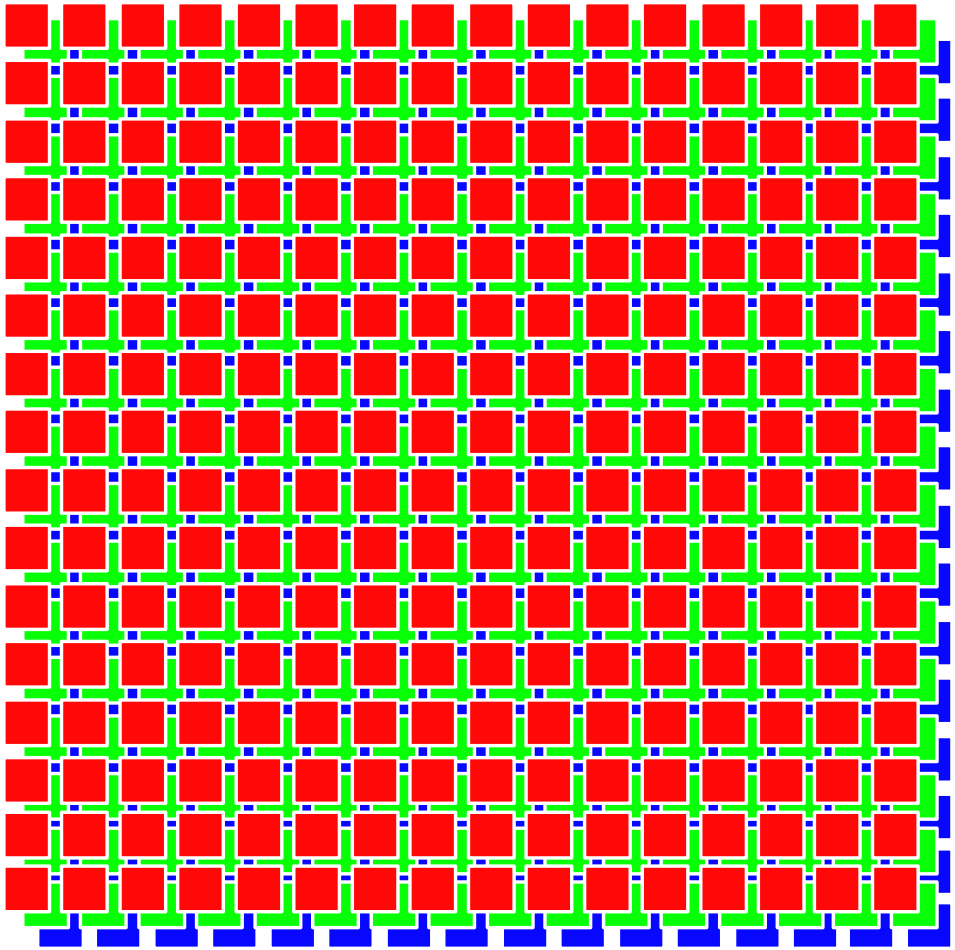


16x16크기의 패치로 나눈 이미지

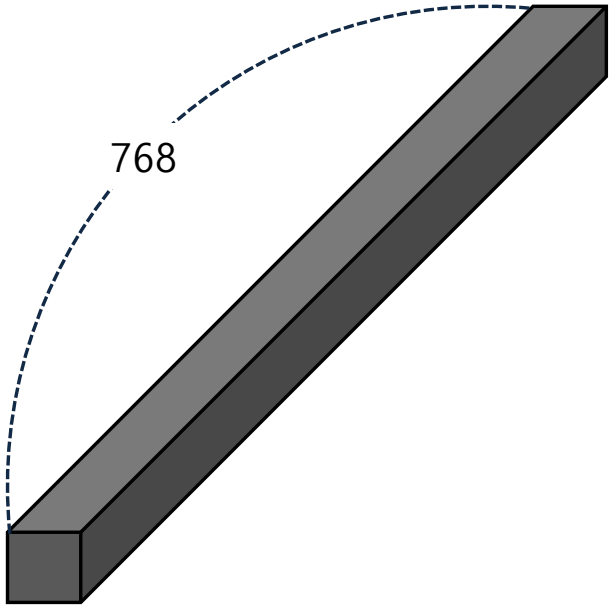


패치 하나의 크기는 16x16x3 = 768개의 픽셀

1단계: 이미지(224x224)를 16x16크기의 패치로 분할 - 768개 픽셀인 패치를 Flatten

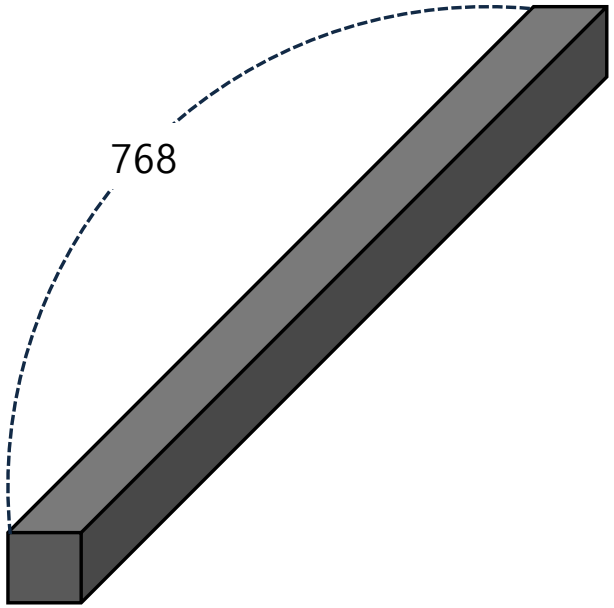


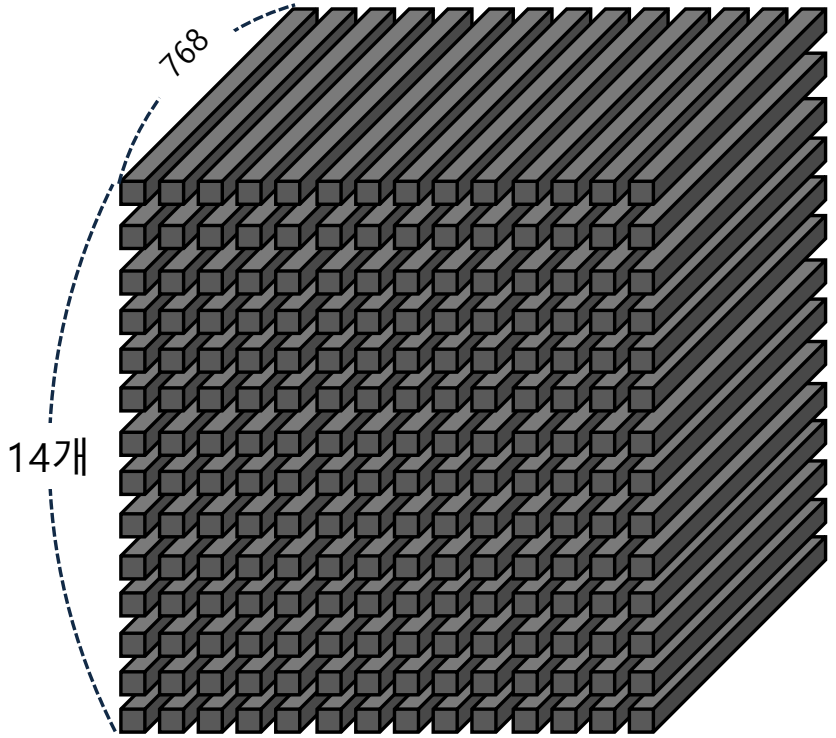
한개의 패치 : 크기는 16x16x3 = 768개의 픽셀



한 개의 패치(Flatten)
➡️ 크기가 768인 1차원 벡터

2단계: 패치 임베딩 - 전체 패치

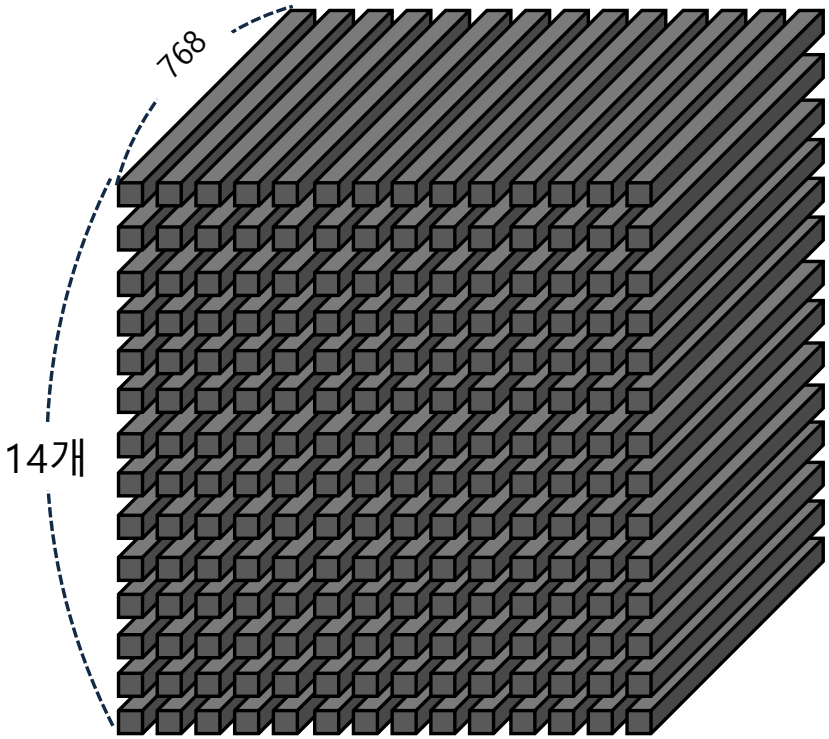




한 개의 패치(Flatten)
➡️ 크기가 768인 1차원 벡터

196개의 패치(Flatten)
➡️ 크기가 768인 1차원 벡터가 196개

2단계: 패치 임베딩(Linear Projection) - 패치 하나하나를 768차원에서 D(모델 차원)차원으로 변환
여기서는 D를 768차원이라고 가정



196개의 패치(Flatten)
➡️ 크기가 768인 1차원 벡터가 196개

$$W \in \mathbb{R}^{768 \times D}$$

$$b \in \mathbb{R}^{D}$$

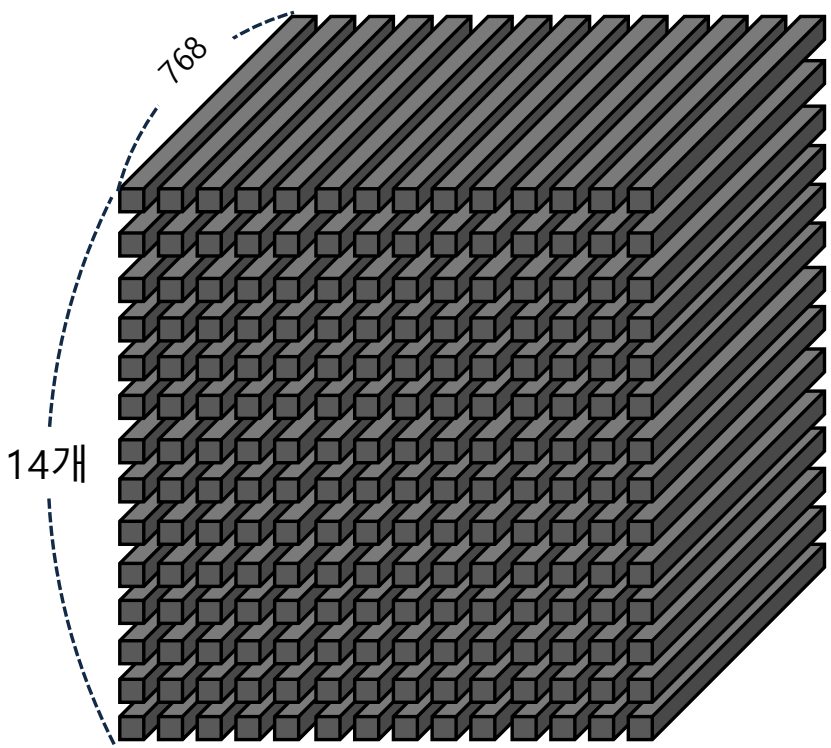
$$\cdot\ W + b = 196 \times 768(D)$$

196개의 패치 토큰(임베딩 벡터)

🔍 **왜 Linear Projection을 진행할까?**

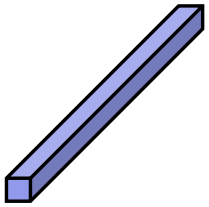
각 패치 벡터의 차원을 Transformer 모델에서 사용하는 "모델 차원 D"에 맞춰주기 위해서

3단계: [CLS] 토큰 추가



196개의 패치 토큰(임베딩 벡터)
➡️ 크기가 768인 1차원 벡터가 196개

패치 토큰에 추가

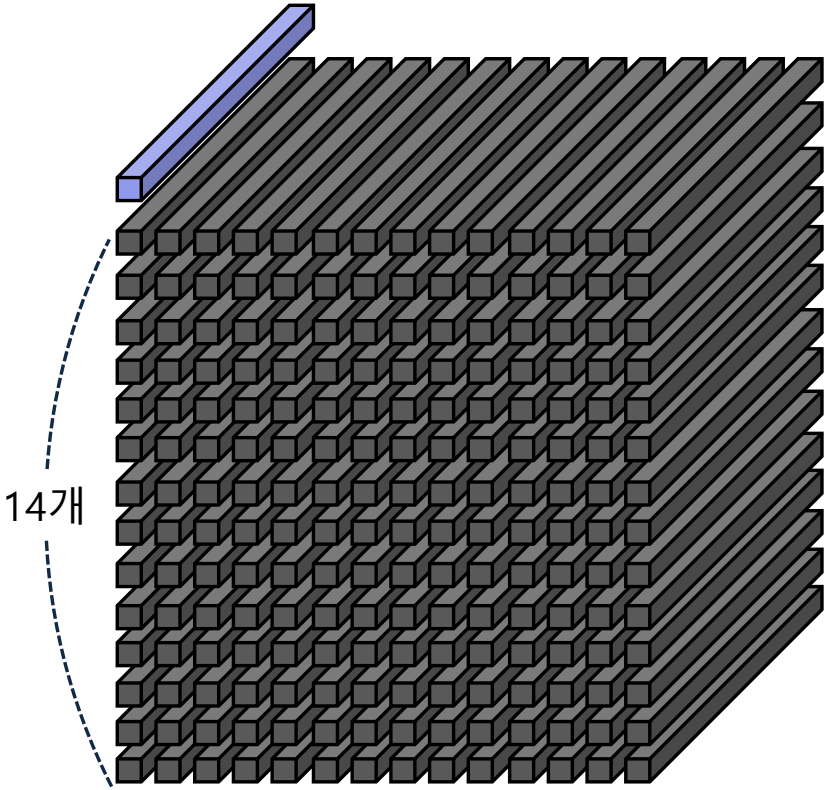


1개의 [CLS] 토큰
➡️ 크기가 768인 1차원 벡터

🔍 **[CLS] 토큰이란?**

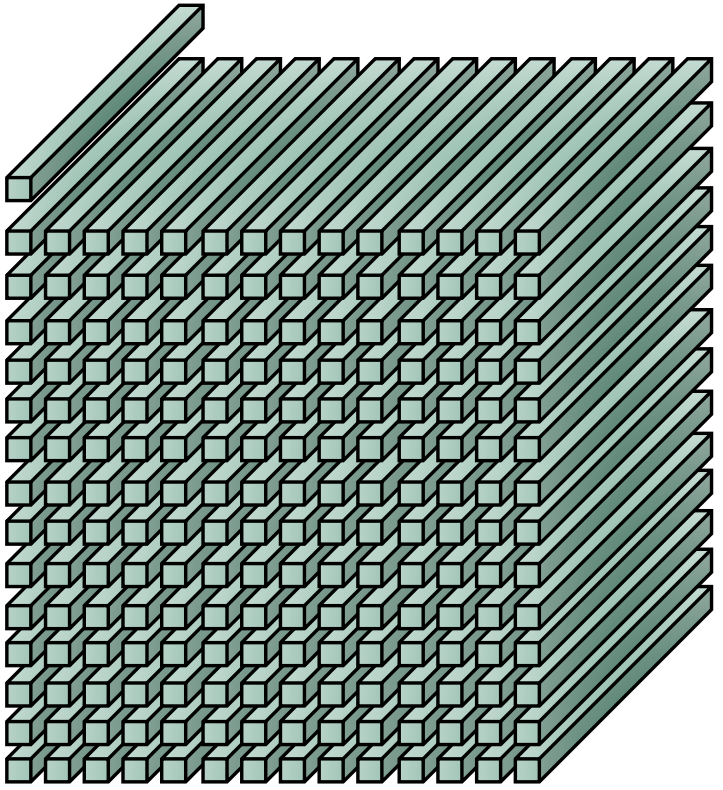
"시퀀스 전체를 대표하는" 특별한 학습 벡터이며, 패치 토큰과 같은 차원(D)을 가진다.

4단계: 포지셔널 임베딩 추가 - 요소 합을 통해서 위치 정보 추가



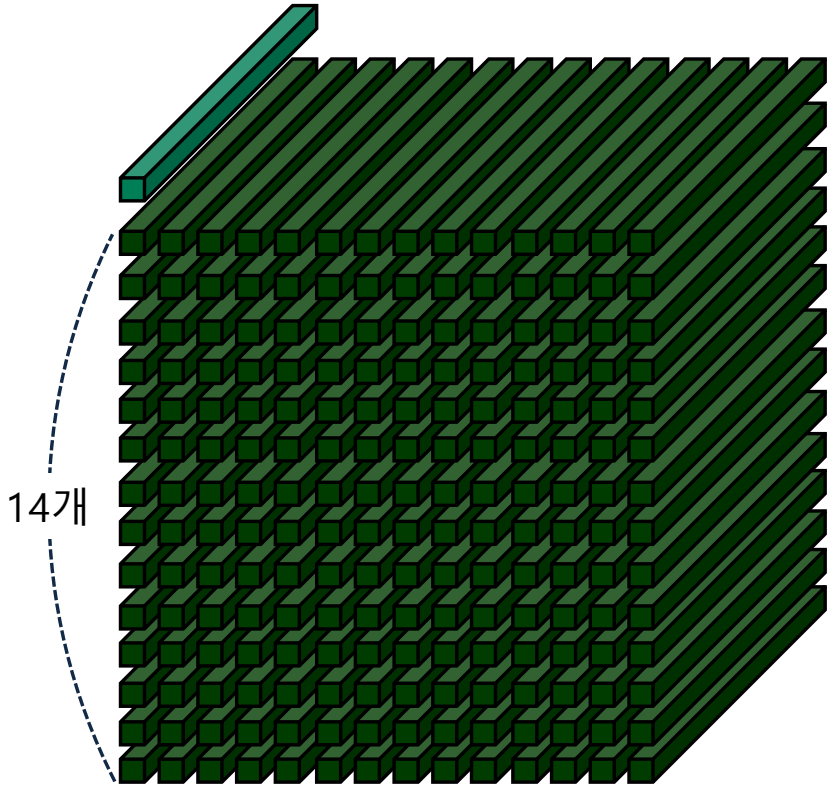
197개의 패치 토큰(임베딩 벡터)
➡️ 크기가 768인 1차원 벡터가 197개

+



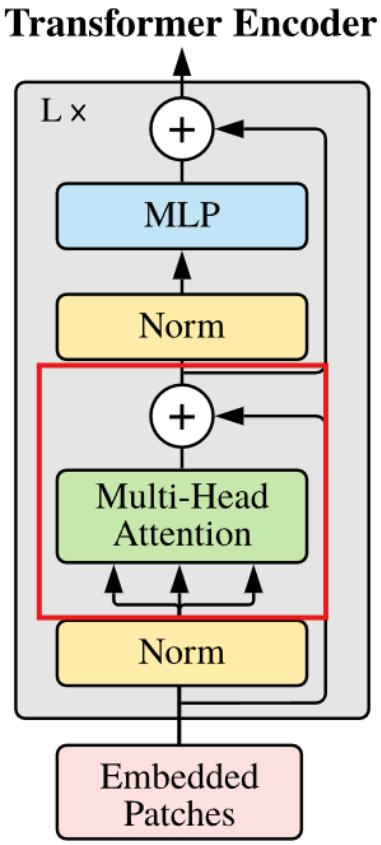
197개의 포지셔널 벡터
➡️ 크기가 768인 1차원 벡터가 197개

4단계: 포지셔널 임베딩 추가 - 포지셔널 벡터를 추가한 패치 토큰

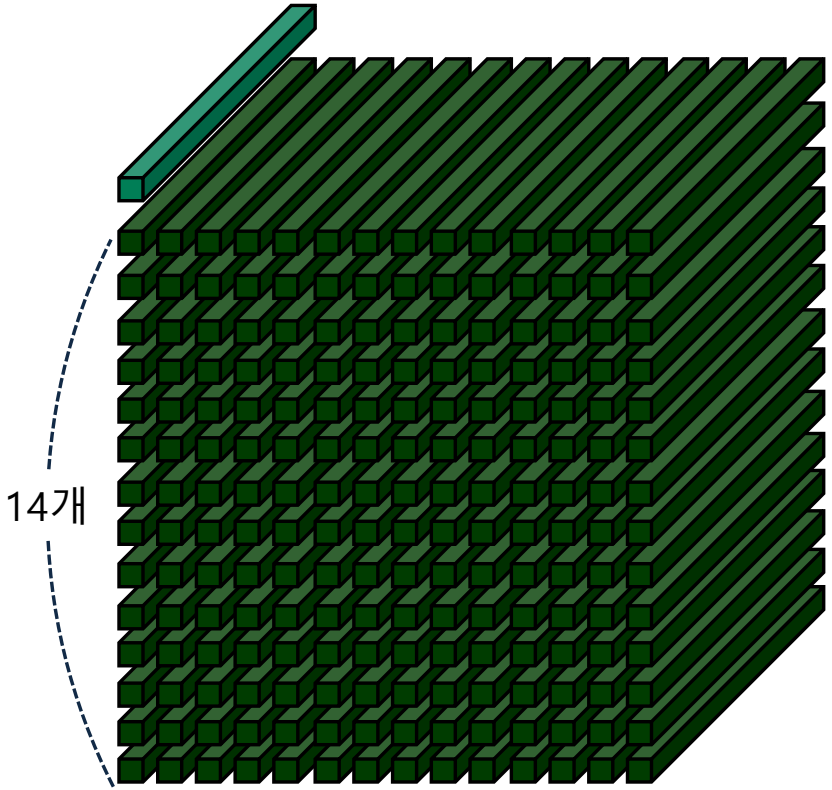


197개의 패치 토큰(임베딩 벡터)
➡️ 크기가 768인 1차원 벡터가 197개

2. Multi-Head Attention



5단계: Transformer Encoder Layer - Q, K, V 생성

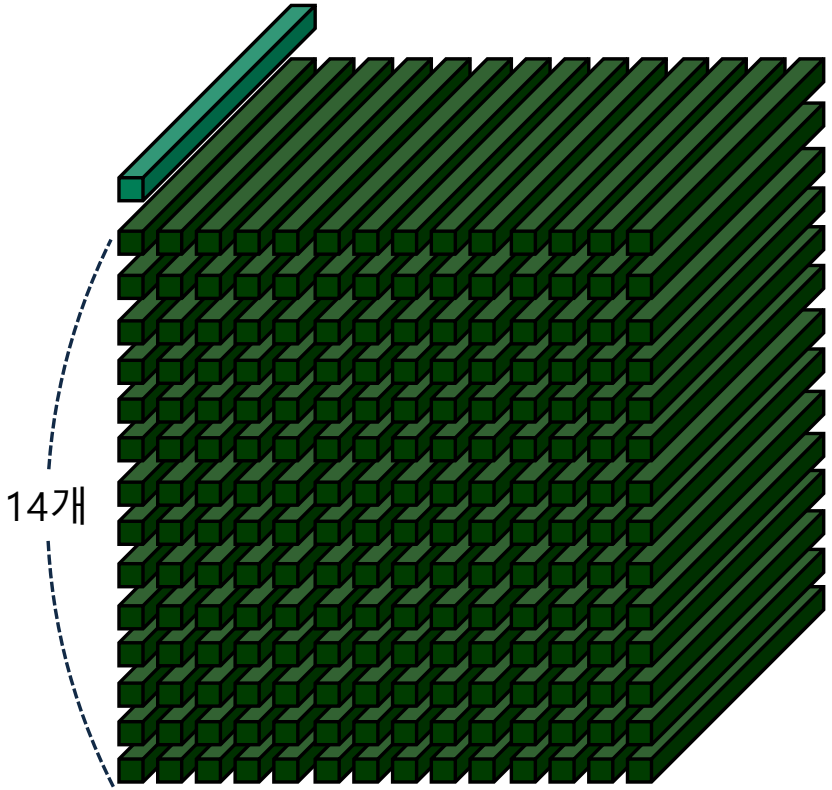


197개의 패치 토큰(임베딩 벡터)
➡️ 크기가 768인 1차원 벡터가 197개

$$\cdot W_q + b_q = Q(197x768)$$

$$W_Q, W_K, W_V \in \mathbb{R}^{768\times768}, \quad b_Q, b_K, b_V \in \mathbb{R}^{768}$$

5단계: Transformer Encoder Layer - Q, K, V 생성

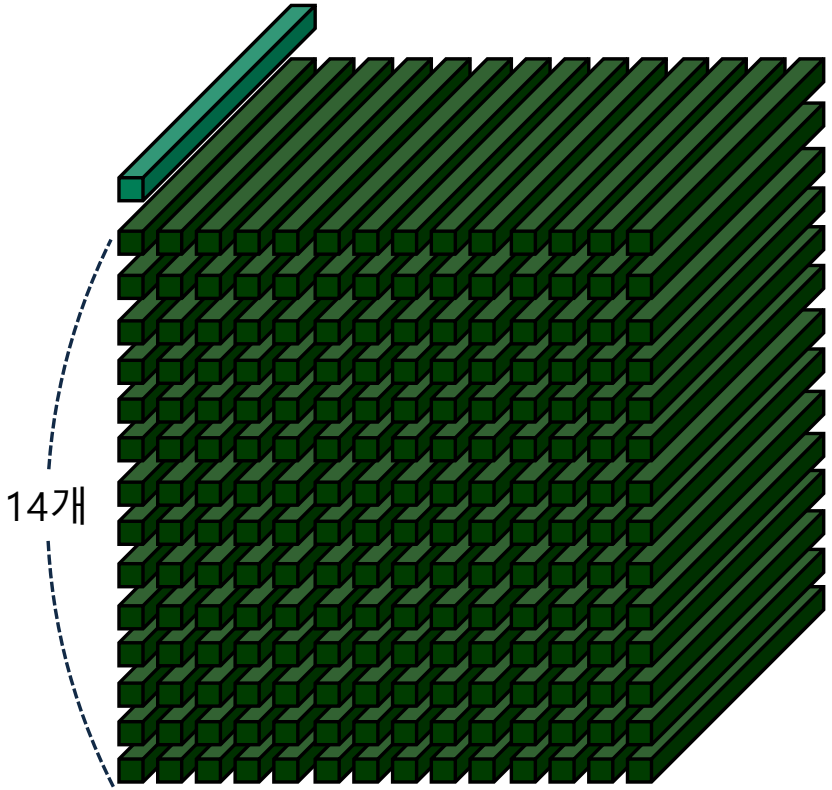


197개의 패치 토큰(임베딩 벡터)
➡️ 크기가 768인 1차원 벡터가 197개

$$\cdot W_k + b_k = K(197x768)$$

$$W_Q, W_K, W_V \in \mathbb{R}^{768\times768}, \quad b_Q, b_K, b_V \in \mathbb{R}^{768}$$

5단계: Transformer Encoder Layer - Q, K, V 생성

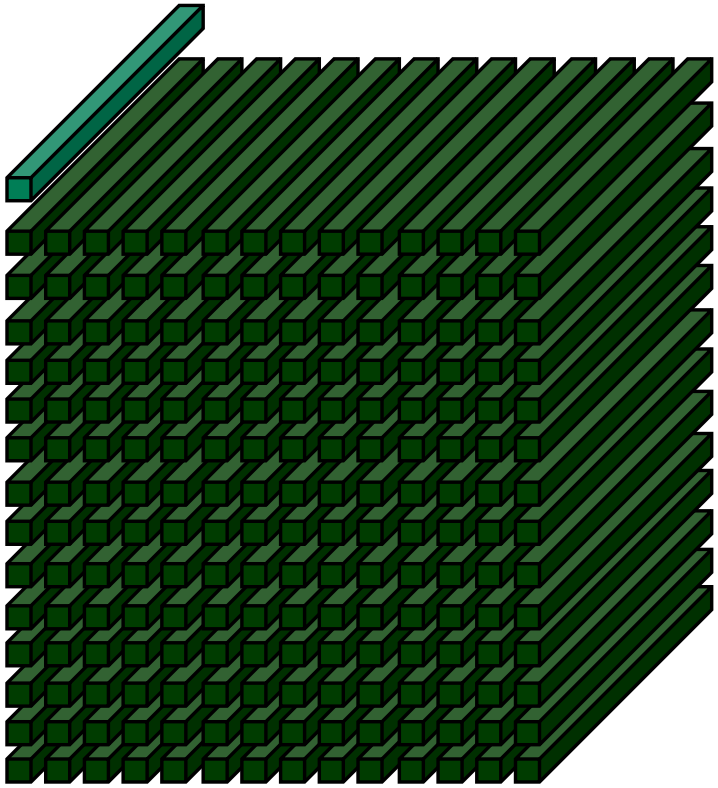


197개의 패치 토큰(임베딩 벡터)
➡️ 크기가 768인 1차원 벡터가 197개

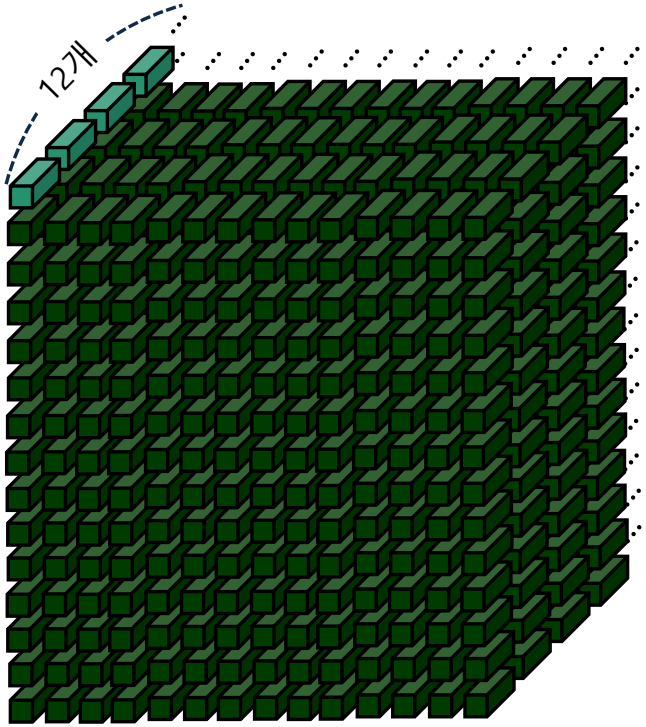
$$\cdot W_v + b_v = V(197x768)$$

$$W_Q, W_K, W_V \in \mathbb{R}^{768\times768}, \quad b_Q, b_K, b_V \in \mathbb{R}^{768}$$

5단계: 헤드 분할 - 헤드가 12개이기 때문에 768차원을 64차원으로 분할

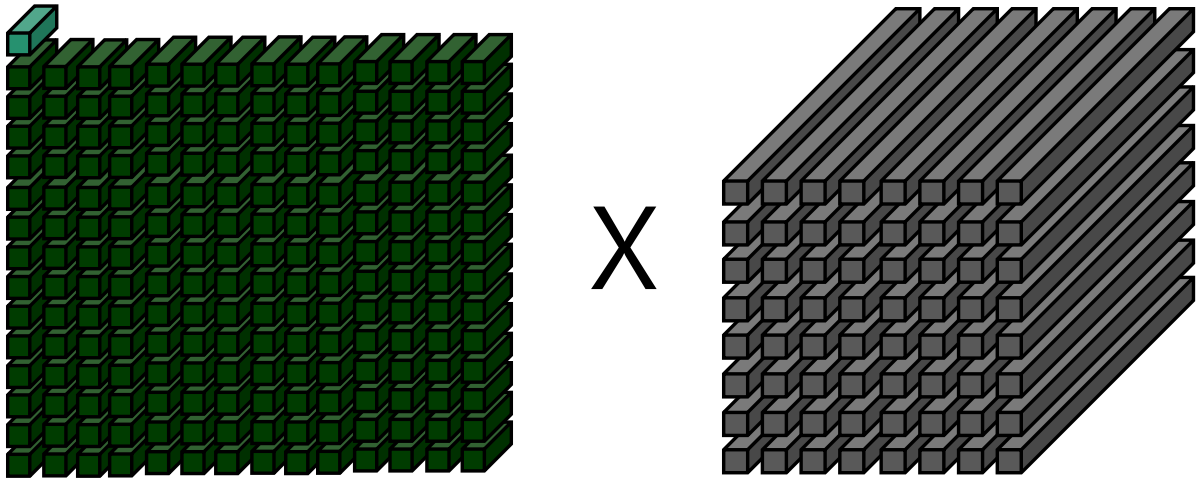


197개의 Q, K, V
➡️ 크기가 768인 1차원 벡터가 197개



64차원으로 만든 197개의 Q, K, V
➡️ 크기가 64인 1차원 벡터가 197개 x 12

5단계: Attention Score 계산 - 각 헤드별로 계산 $S^{(i)} = \frac{Q^{(i)} \left(K^{(i)}\right)^\top}{\sqrt{d_k}} \in \mathbb{R}^{197 \times 197}$



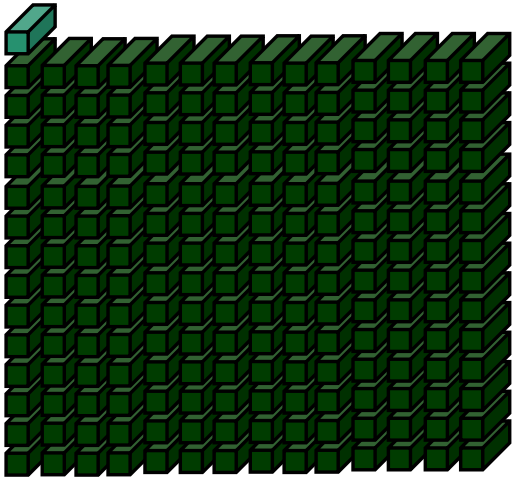
Q(197 x 64) X **K^T(64 x 197)**

√모델 차원(64)

= **Attention Score**

197x197

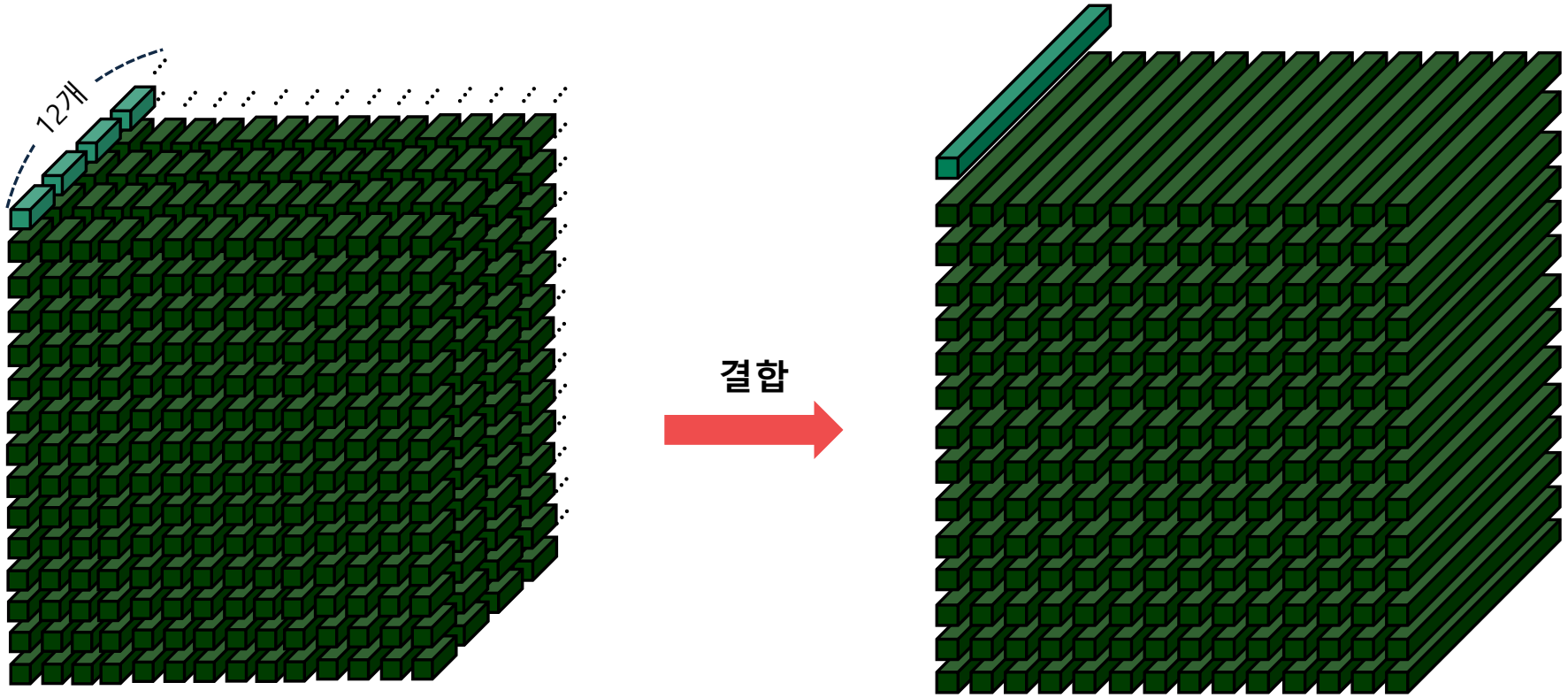
5단계: Softmax(Attention Score) 계산 후 V 곱 - 각 헤드별로 계산 $\text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Softmax(Attention Score) x  = **Attention Output** 97x64

197 x 197

V(197 x 64)

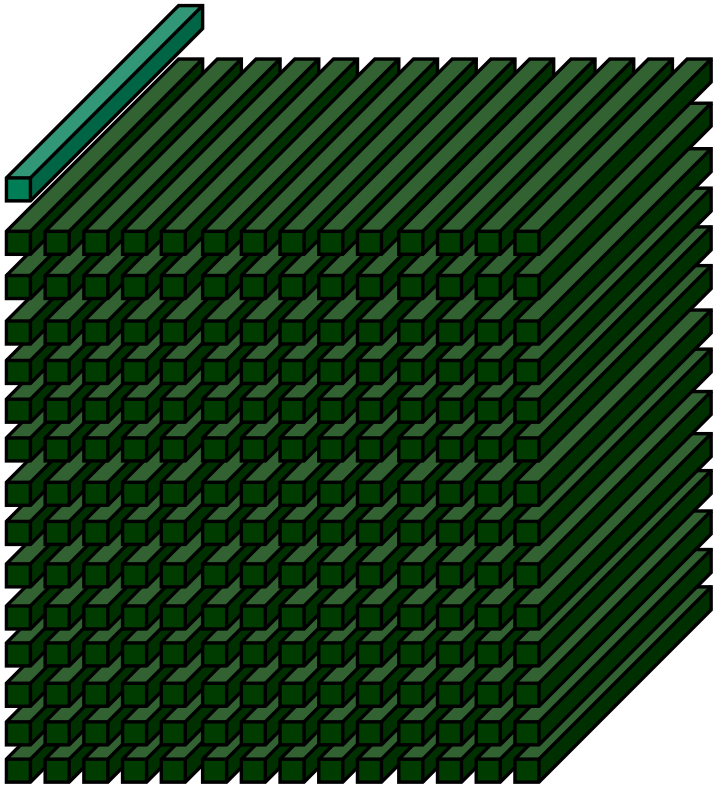
5단계: 헤드 결합



Attention Output 12개
➡️ 크기가 64인 1차원 벡터가 197개 x 12

결합된 Attention Output
➡️ 크기가 768인 1차원 벡터가 197개

5단계: 최종 선형 투영

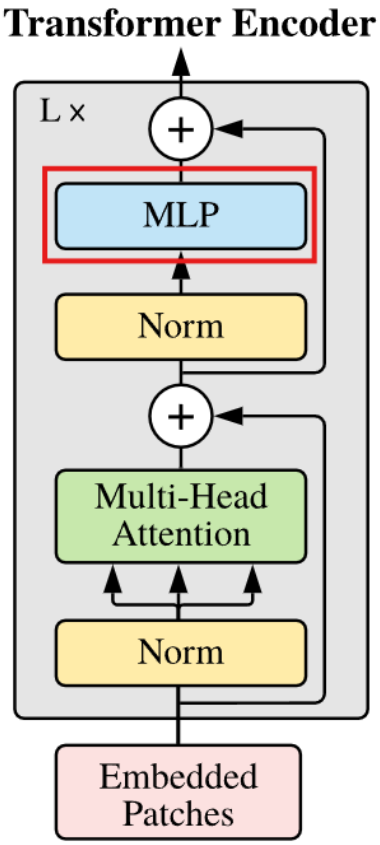


결합된 Attention Output
➡️ 크기가 768인 1차원 벡터가 197개 x 12

$\cdot W_o + b_o =$ **Attention Ouput**

$$W_O \in \mathbb{R}^{768\times768}, \text{ bias } b_O \in \mathbb{R}^{768}$$

3. FFN

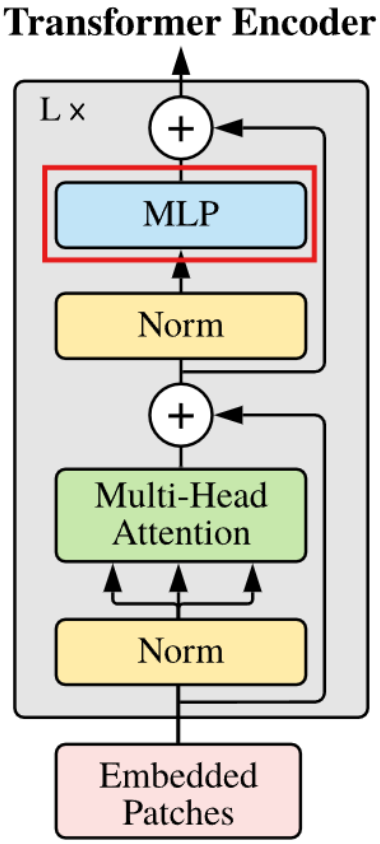

Transformer Encoder
L x
MLP
Norm
Multi-Head Attention
Norm
Embedded Patches

7단계: FFN

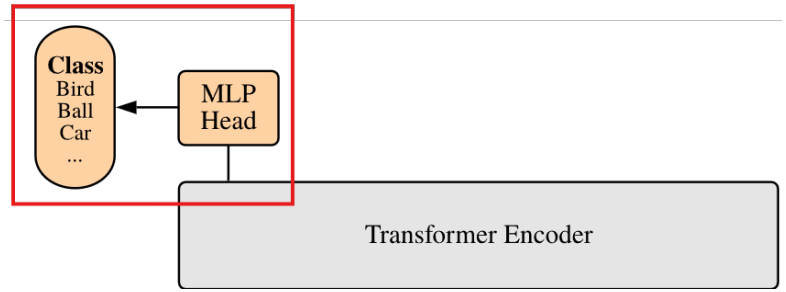
1. 첫 번째 선형변환: $u = x \cdot W_1 + b_1$ $W_1 \in \mathbb{R}^{\{768\times3072\}}$, $b_1 \in \mathbb{R}^{\{3072\}}$

2. 비선형 활성화: $u = GELU(u)$

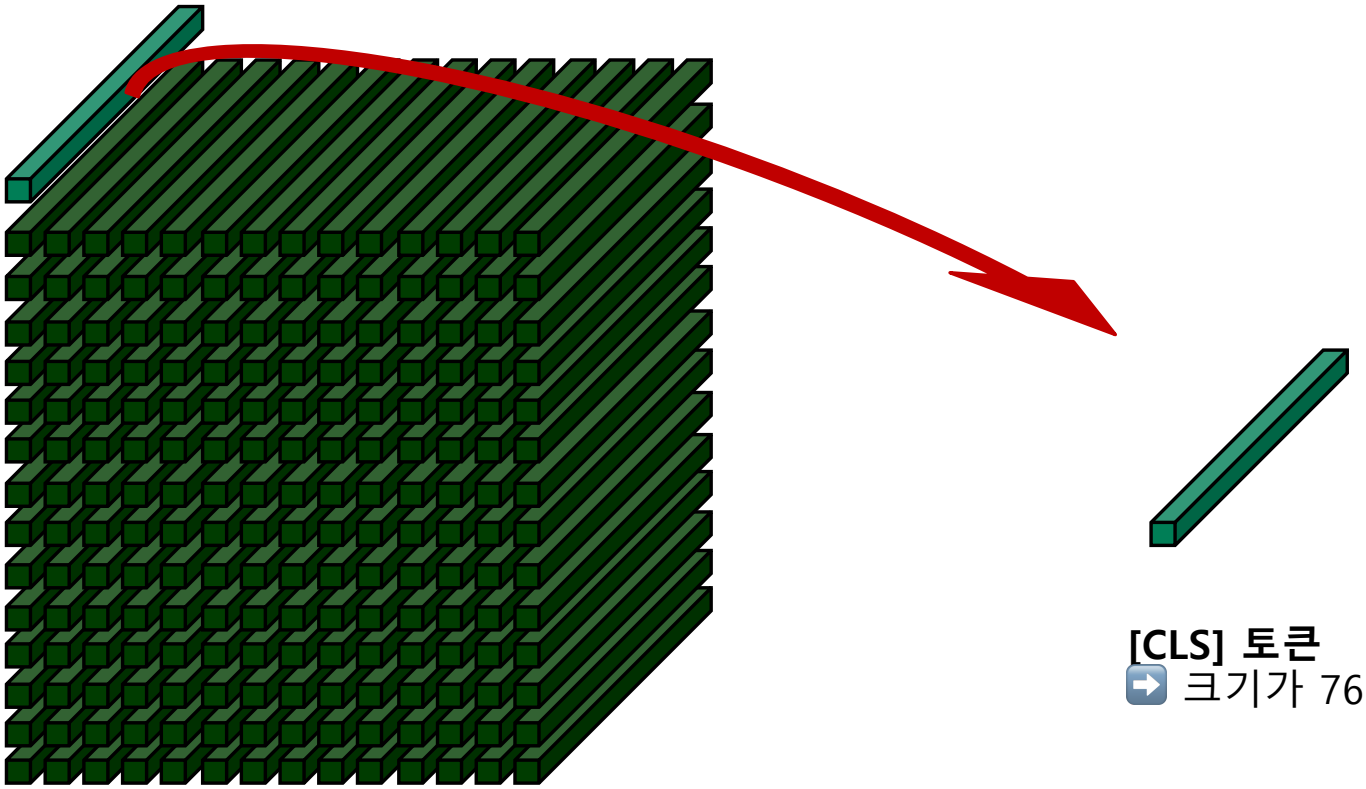
3. 두 번째 선형 변환: $v = u \cdot W_2 + b_2$ $W_2 \in \mathbb{R}^{\{3072x768\}}$, $b_2 \in \mathbb{R}^{\{768\}}$



4. Classification



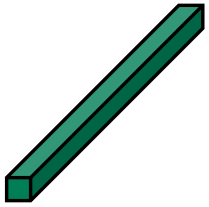
8단계: [CLS] 토큰 추출



Attention Ouput
➡️ 크기가 768인 1차원 벡터가 197개

[CLS] 토큰
➡️ 크기가 768인 1차원 벡터

8단계: [CLS] 토큰을 이용해서 로짓 계산 - 클래스는 총 1000개



$$\cdot\ W_{cls} + b_{cls} = z(1x1000)$$

$W_{cls} \in \mathbb{R}^{\{768\times1000\}}$, $b_{cls} \in \mathbb{R}^{\{768\times1000\}}$

[CLS] 토큰

➡️ 크기가 768인 1차원 벡터

8단계: 계산된 로짓 결과를 Softmax를 통해서 클래스 확률 도출

Softmax(z(1x1000)**) -> p**

p는 총 1000개의 확률 값으로 이루어져있고,
이 확률 중 가장 높은 확률을 가진 클래스가 이미지의 예측 클래스로 결정된다.

Experiments

- 하이퍼파라미터

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Layers: 각 모델의 층 수

Hidden size D : 각 층의 숨겨진 상태 크기

MLP size: 다층 퍼셉트론의 크기

Heads: 멀티헤드 어텐션의 수

Params: 총 파라미터 수

- 확장성(Scalability) 평가

ViT는 CNN과 달리 매우 단순한 구조이지만, 크기를 키우면 성능이 훨씬 좋아진다는 점을 강조하기 위해

Experiments

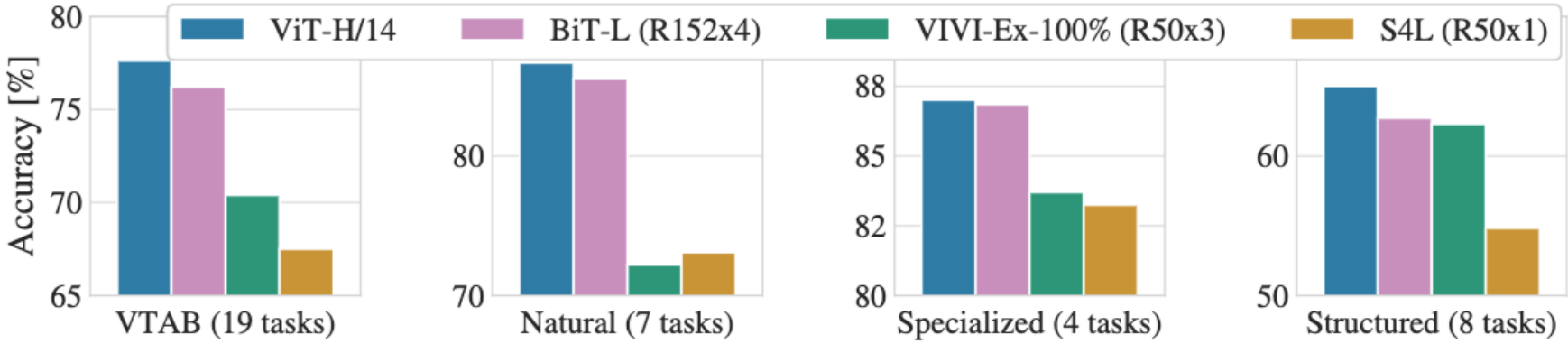
	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 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

- JFT : Google이 내부적으로 구축한 초대규모 이미지·레이블 데이터셋 이름
- Big Transfer (BiT) : large ResNet을 이용해 supervised transfer learning 수행
- Noisy Student : large EfficientNet을 이용해 semi-supervised learning 수행

(ImageNet과 라벨이 지워진 JFT-300M 데이터셋)

=> ResNet보다도 좋은 성능, 훈련 시간 또한 적음

Experiments

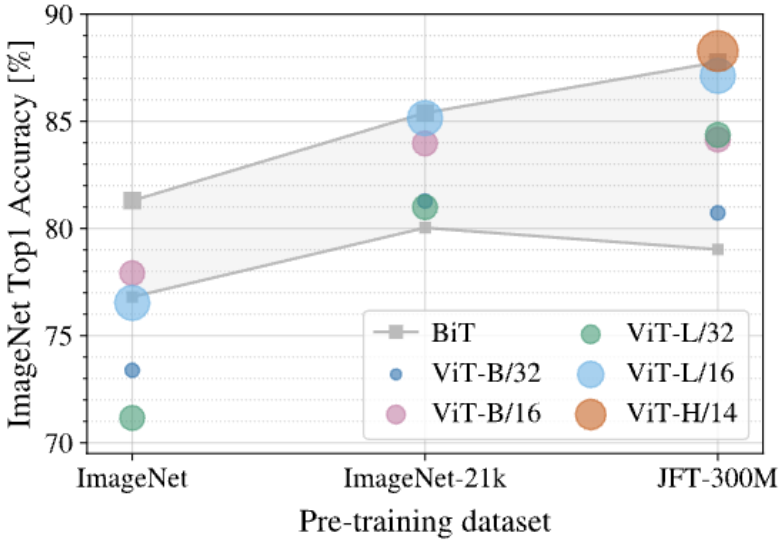


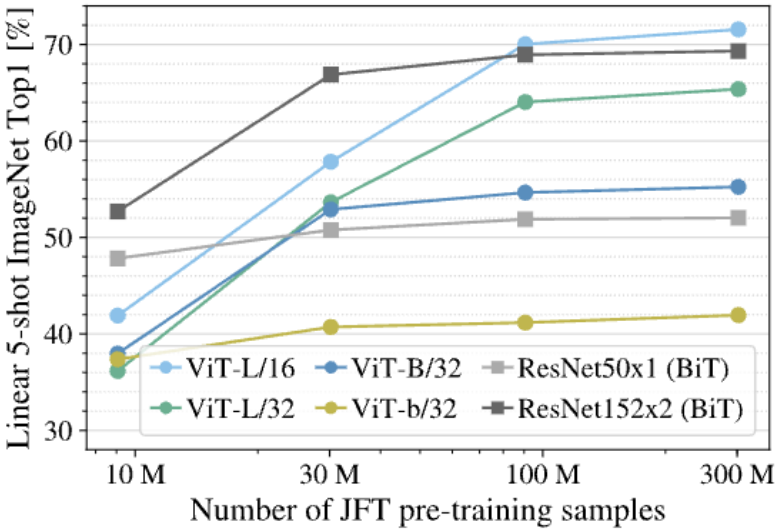
Natural: 일반적인 이미지 분류 (예: Pets, CIFAR)

Specialized: 의료/위성 이미지

Structured: 기하학적 이해 필요 (예: 위치 예측)

Experiments

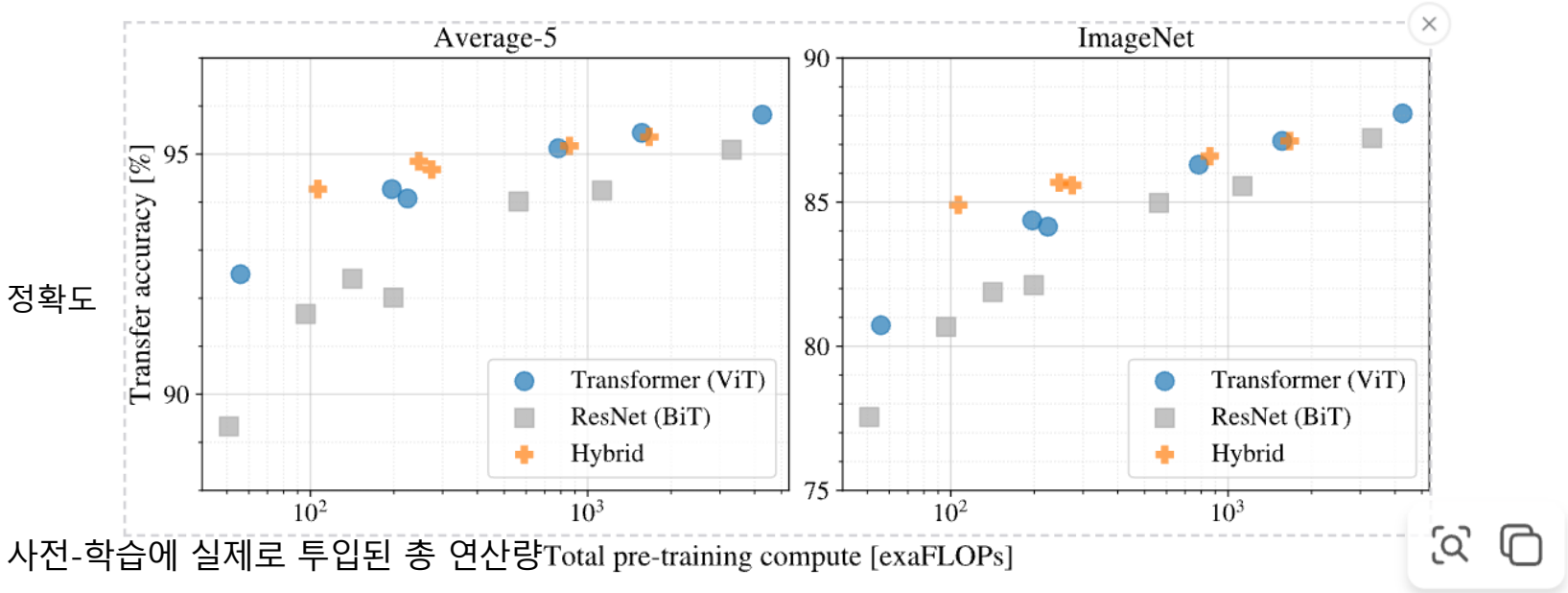




크기가 큰 데이터셋으로 pre-training 하는 경우
BiT보다 ViT가 더 높은 성능을 띄고, 반대의 경우는
반대의 성능을 띈다

작은 데이터셋에서는 확실히 CNN 계열의 BiT가
높은 성능을 보이나,
큰 데이터셋으로 갈수록 ViT 성능이 더 좋음
(일반화 성능)

Experiments



ViT > Hybrid > ResNet 순으로 효율이 좋음

ViT는 연산을 더 투입할수록 성능이 꾸준히 오름

Experiments

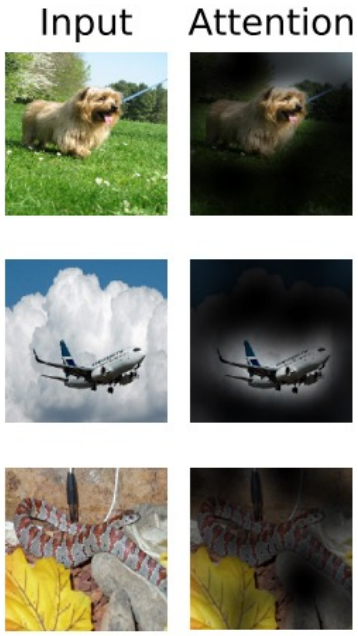
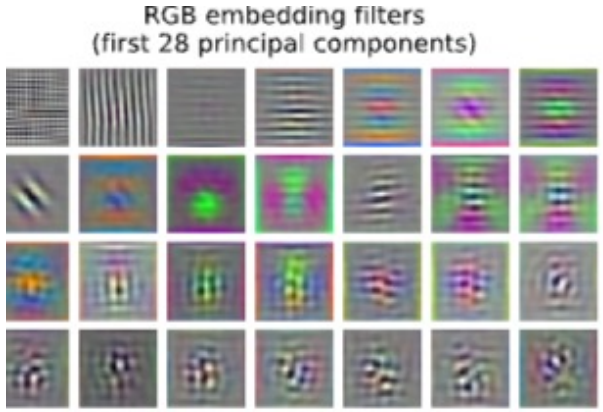
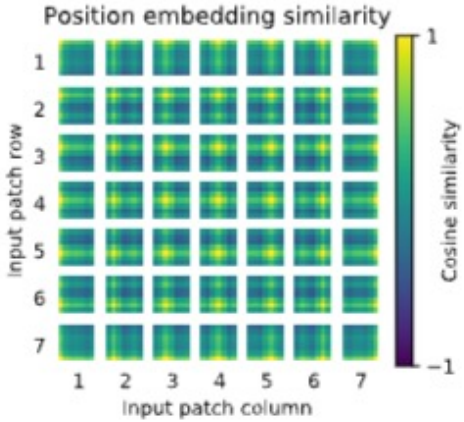


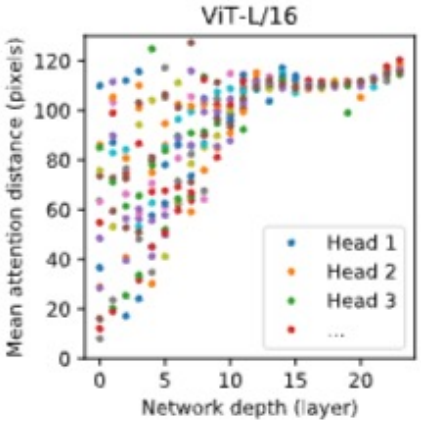
Figure 6: Representative examples of attention from the output token to the input space. See Appendix D.7 for details.

(ViT가 실제로 주목하는 부분)

Experiments







- L : ViT의 입력 임베딩은 CNN의 Conv 필터처럼 동작한다는 것을 보여줌
- Center : 가까운 위치는 유사도 ↑, 멀어지면 유사도 ↓ → 공간 구조(위치) 인식 가능
- R : 초기 레이어 -> 다양한 시야 범위를 가진 head들이 공존
- R : 최종 레이어 -> 높은 수준의 추상적 개념 (예: 객체 전체 형태, 의미) 이해 가능

Conclusion

- **CNN**은 **멀리 떨어진 부분 간의 관계를** 직접 연결하기 **어려움**
- **ViT**는 Self-Attention을 통해 이미지 **전체의 패치들이 서로 영향을** 주고받을 수 있습니다.

  → 예: 한쪽 끝의 고양이 귀와 다른 쪽 끝의 꼬리를 직접 연결해서 해석 가능
- ViT는 기존 CNN 없이 **Transformer만 사용해서 이미지 분류가** 가능하다는 걸 보여준 모델
- 이미지를 단순히 패치로 나눈 뒤, **자연어에서처럼 토큰으로 처리하는 아이디어**
- 충분히 큰 데이터셋과 연산 자원만 있다면 **CNN보다 더 좋은 성능**
- 데이터에 많이 의존하고, 학습 **비용도 크다는 한계점**

Q&A

Google Developer Group
Incheon National Univ