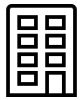
2023 동계 세미나

Self-attention 경량화



Sogang University
Vision & Display Systems Lab, Dept. of Electronic Engineering

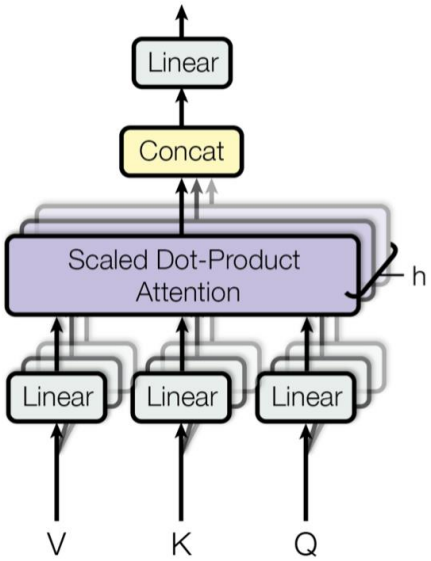
Presented by
유현우

Outline

- Background
  - Multi head self attention
  - Self attention 연산량
- Self attention 경량화
  - Efficient Attention: Attention with Linear Complexities
    - WACV 2021 인용 150
  - Hydra Attention : Efficient Attention with Many Heads
    - ECCV workshop 2022
- Conclusion

Background

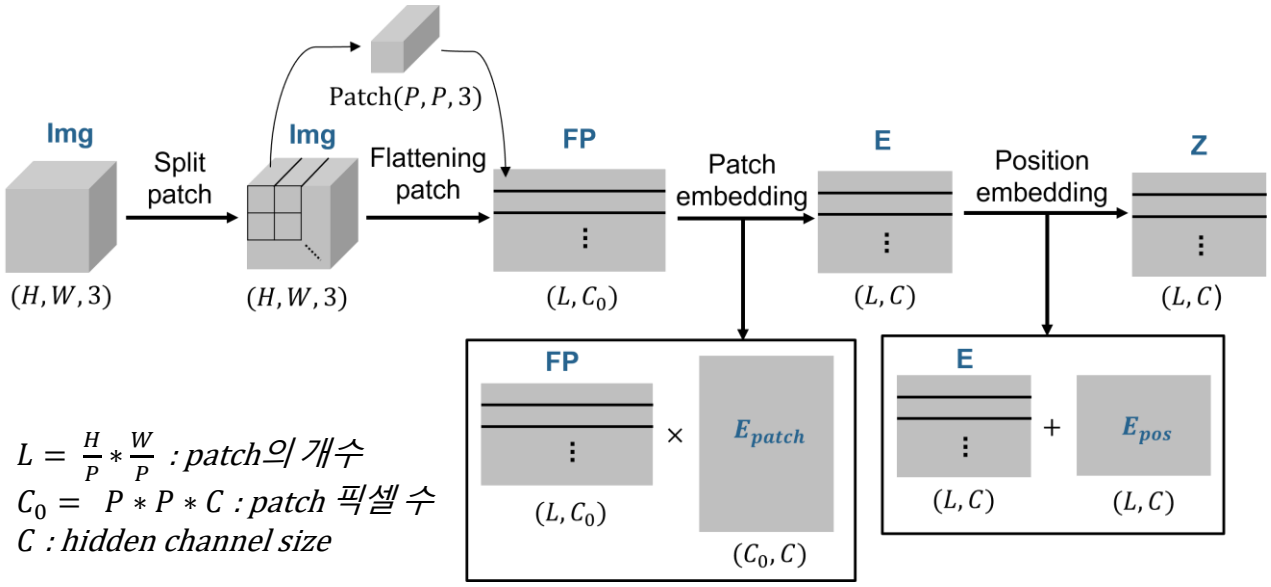
- Transformer
  - Query, key, value 간에 similarity를 기반으로 feature representation을 수행
  - 특정 token과 전역적으로 존재하는 token과의 관계를 고려하므로 global한 특성을 고려하여 feature를 추출할 수 있다는 장점이 있음
    - 자연어 처리 분야에서 처음 제안되었고, vision 분야에서도 다양한 task에서 좋은 성능을 보임



Multi head self attention

Background

- Embedding
  - Input patch를 flatten한 후 linear projection을 이용해 patch embedding 수행
    - Patch를 대표하는 feature를 1d token의 형태로 표현
  - Linear projection 연산은 MLP layer 또는 1x1 convolution으로 수행 가능



Embedding overview

Background

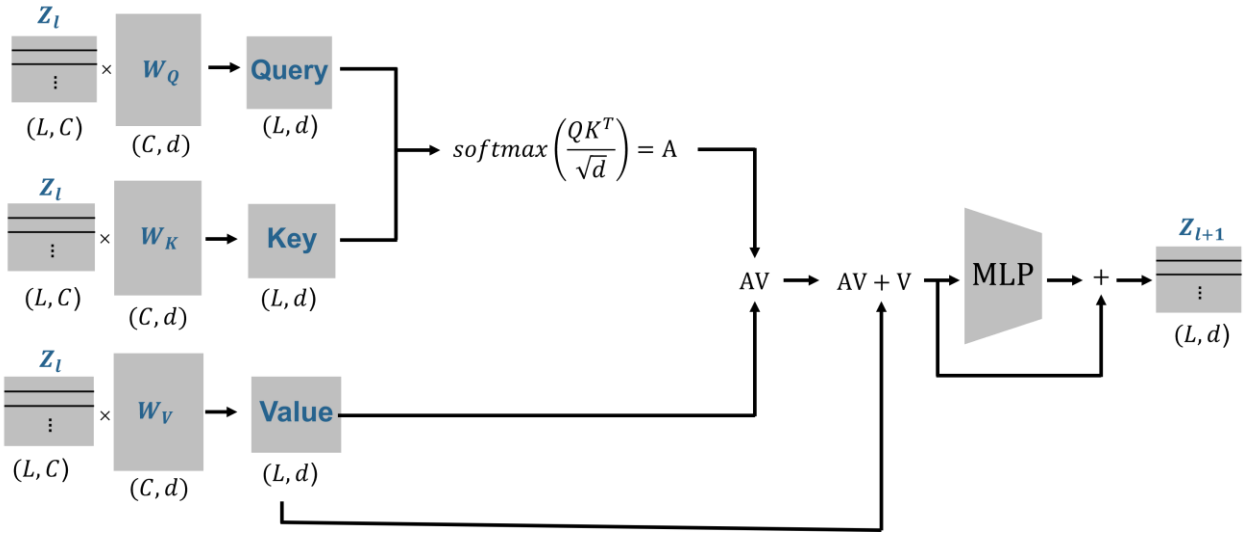
- Self attention
  - Learnable parameter로 linear projection을 수행하여 query, key, value 생성
  - Query, key간의 연산(Key · Query.transpose)을 통해 얻은 가중치를 value에 적용

$$[\mathbf{q}, \mathbf{k}, \mathbf{v}] = \mathbf{z}\mathbf{U}_{qkv} \qquad \mathbf{U}_{qkv} \in \mathbb{R}^{D \times 3D_h},$$

$$A = \text{softmax}\left(\mathbf{q}\mathbf{k}^{\top}/\sqrt{D_h}\right) \qquad A \in \mathbb{R}^{N \times N},$$

$$\text{SA}(\mathbf{z}) = A\mathbf{v}\,.$$

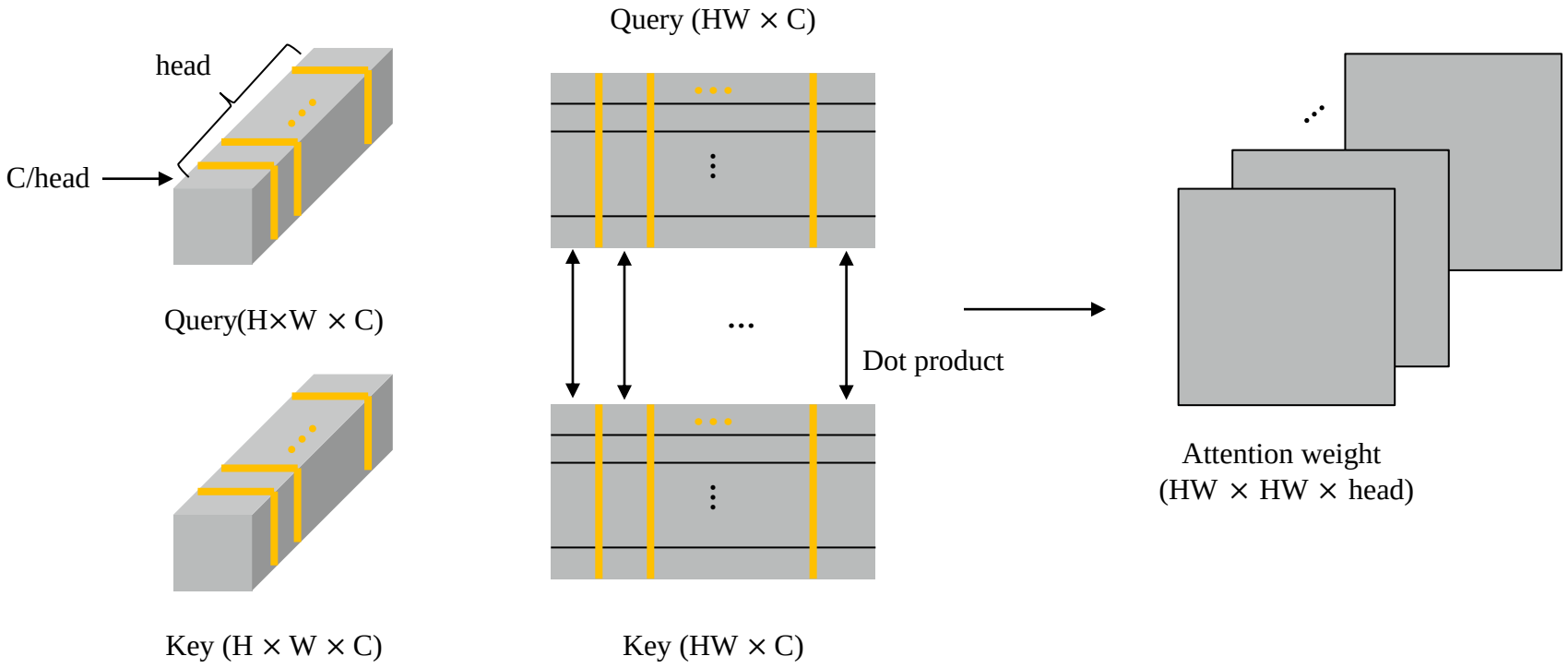
$$\text{MSA}(\mathbf{z}) = [\text{SA}_1(z); \text{SA}_2(z); \cdots; \text{SA}_k(z)]\,\mathbf{U}_{msa} \qquad \mathbf{U}_{msa} \in \mathbb{R}^{k \cdot D_h \times D}$$



Self-attention overview

Background

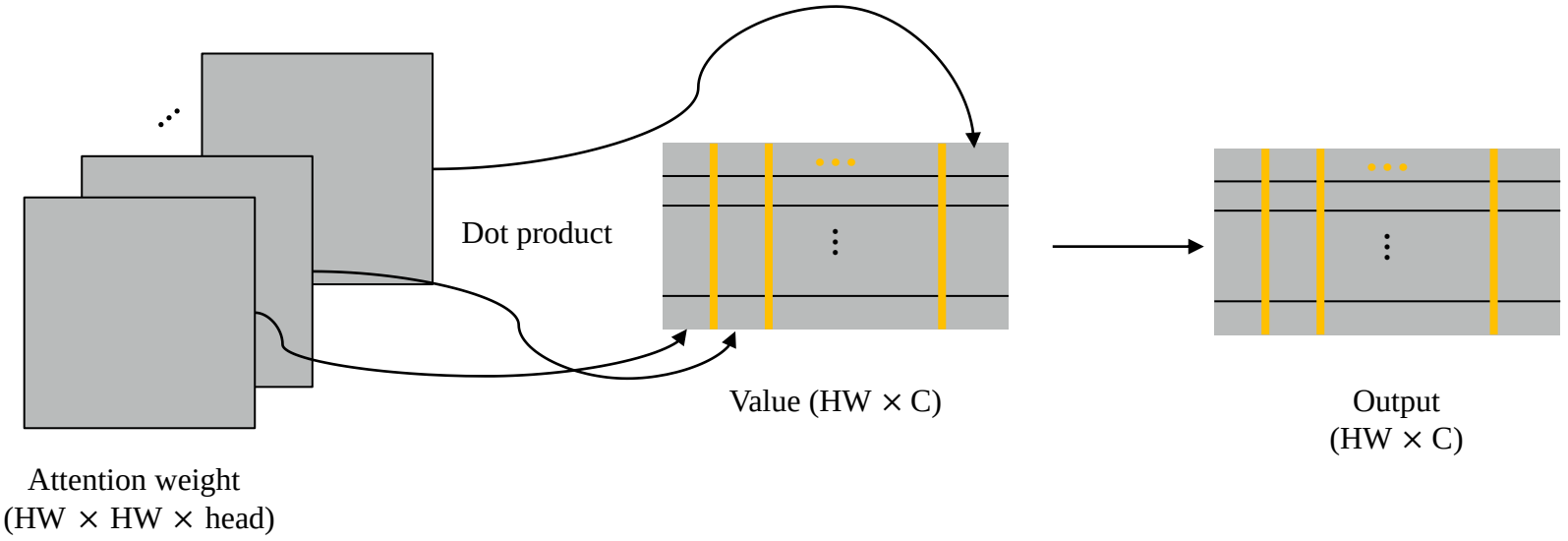
- Multi head self attention



(HW × C/head)·(C/head ×HW)

Background

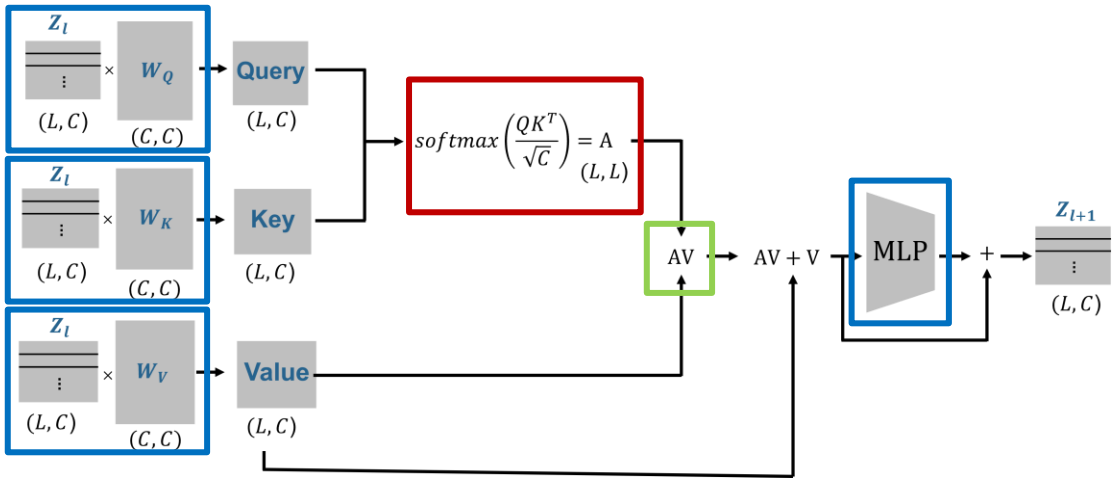
- Multi head self attention



(HW ×HW)(HW × C/head)

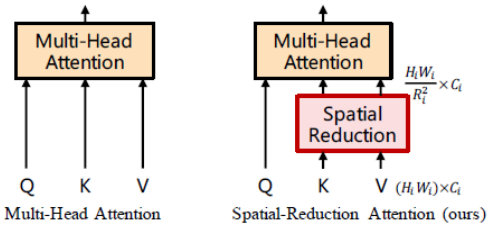
Background

- Vision transformer(ViT) 연산량
  - Self attention computational complexity of ViT : $2(hw)^2C \rightarrow O((hw)^2C)$
    - hw : patch 개수(L), C : hidden dimension
    - Computational complexity 계산
      - $(hw \times C)\cdot(C \times hw)$ 연산 1번 → $(hw)^2C$
      - $(hw \times hw)\cdot(hw \times C)$ 연산 1번 → $(hw)^2C$

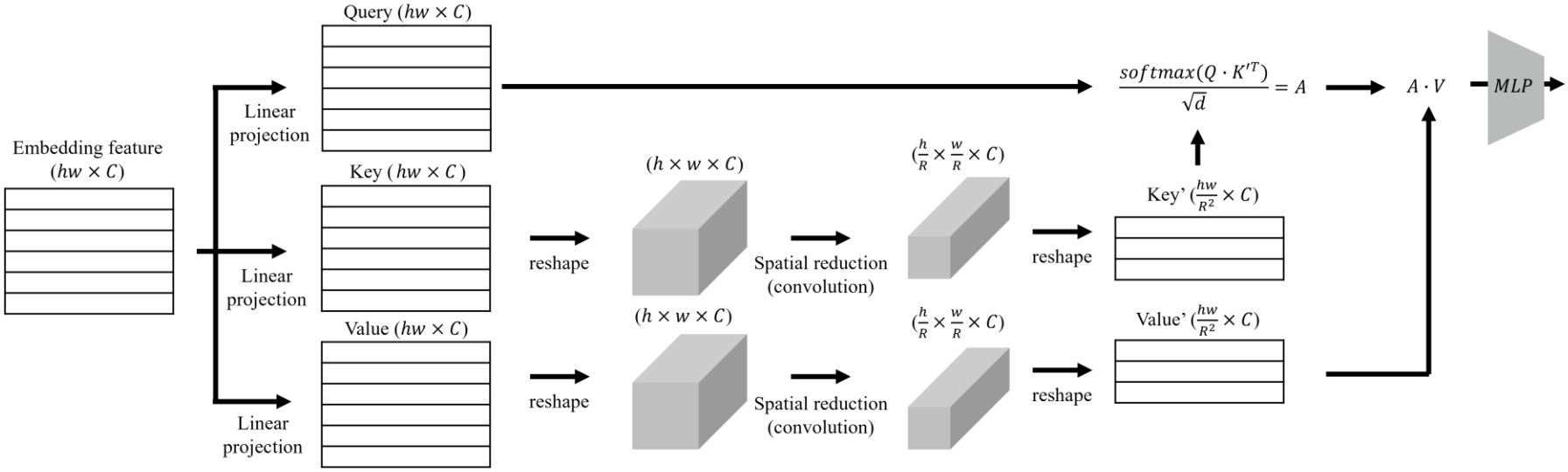


Self-attention layer

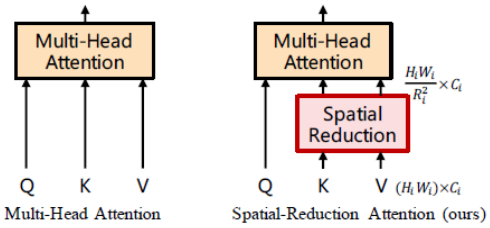
Background



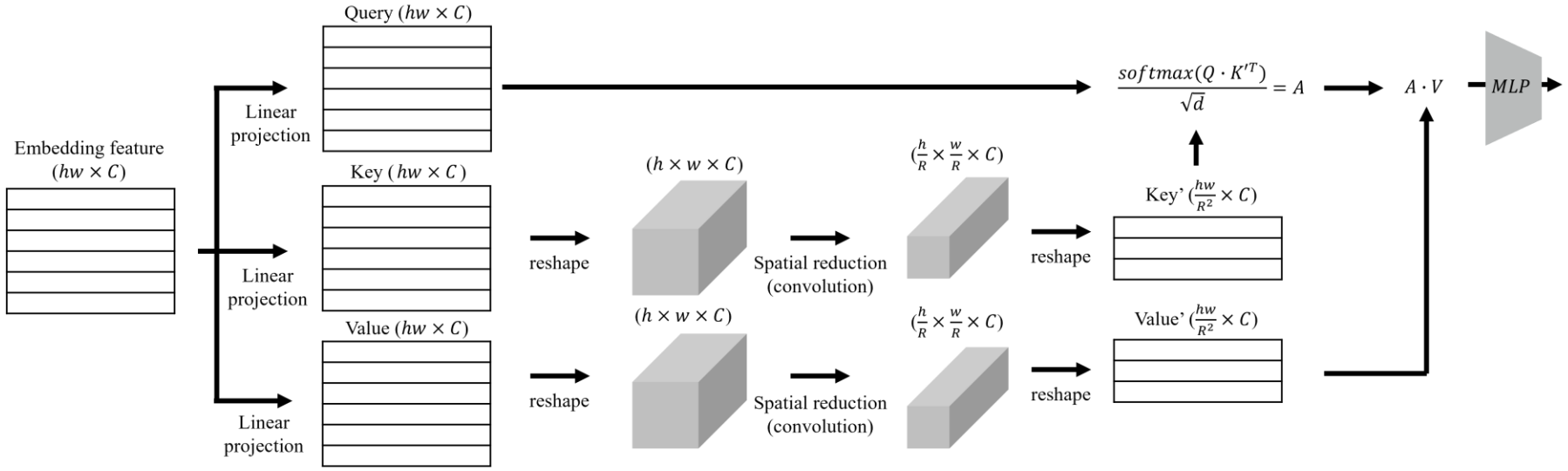
- Pyramid vision transformer(PVT) [1]
  - Key, value의 resolution을 감소시켜 self attention을 수행하는 spatial reduction attention (SRA) 제안



Background

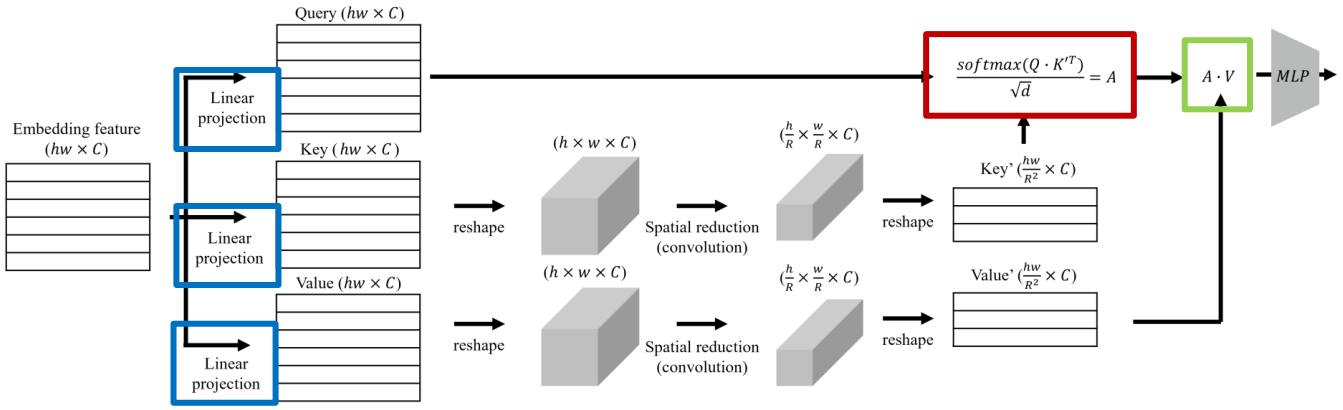


- Pyramid vision transformer(PVT)
  - Query · key.transpose = $A \rightarrow (hw \times C) \cdot (C \times \frac{hw}{R^2}) = (hw \times \frac{hw}{R^2})$
  - A · value = output → $(hw \times \frac{hw}{R^2}) \cdot (\frac{hw}{R^2} \times C) = (hw \times C)$
    - hw : patch 개수(L), C : hidden dimension



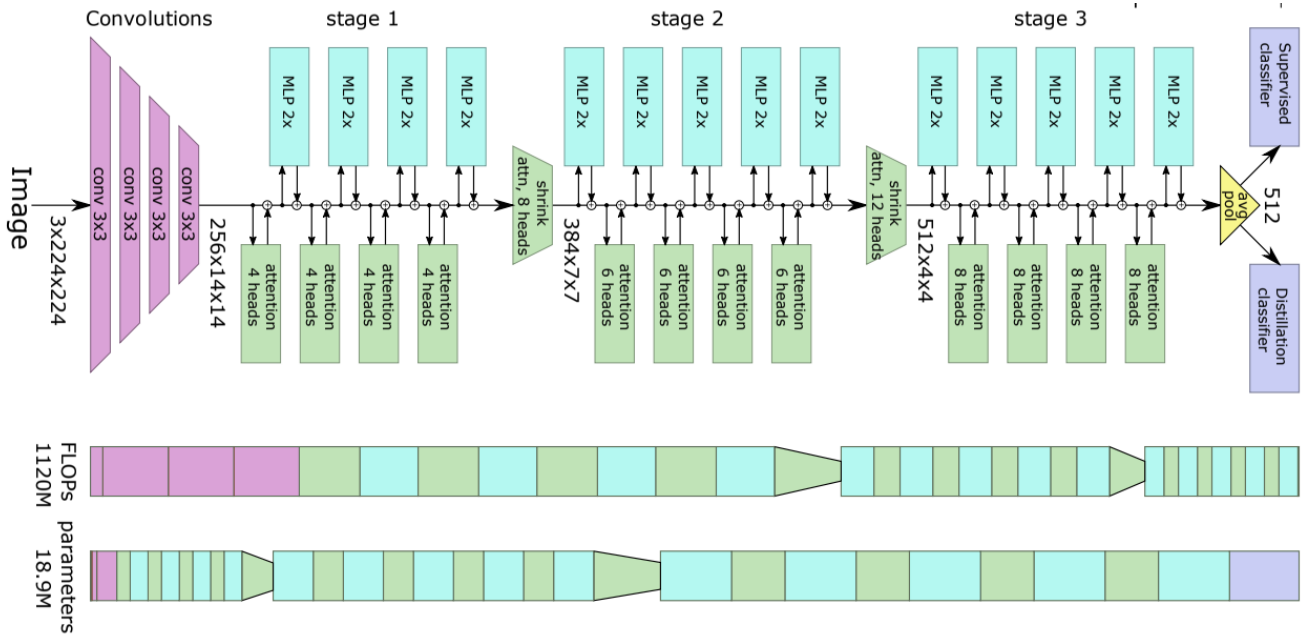
Background

- Pyramid vision transformer(PVT)
  - Self attention computational complexity : $2\ \frac{(hw)^2}{R^2}C \rightarrow \mathrm{O}(hw^2\mathrm{C})$
    - hw : patch 개수(L), C : hidden dimension
    - Computational complexity 계산
      - $(hw \times C)\cdot(\mathrm{C} \times \frac{hw}{R^2})$ 연산 1번 $\rightarrow \frac{(hw)^2}{R^2}C$
      - $(hw \times \frac{hw}{R^2})\cdot(\frac{hw}{R^2} \times C)$ 연산 1번 $\rightarrow \frac{(hw)^2}{R^2}C$



Background

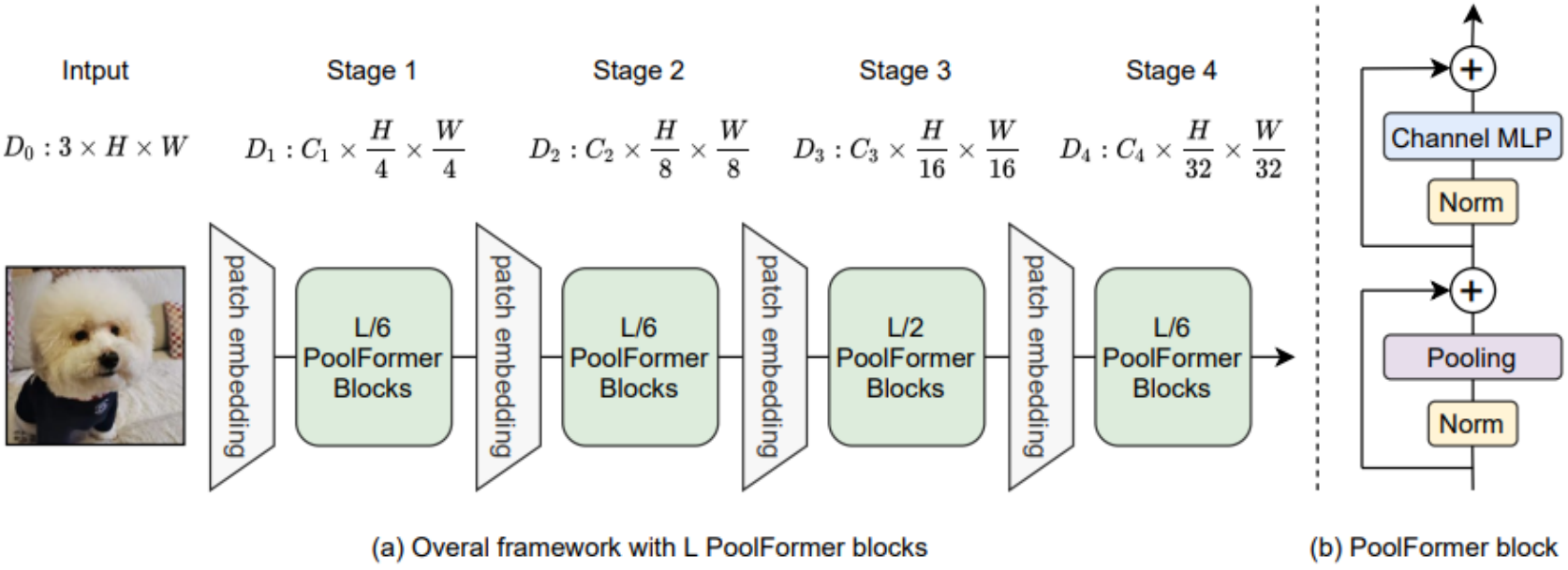
- 이 이후에도 transformer 기반의 다양한 경량화 모델이 등장
  - LeViT[1] : Convolution과 self attention을 적절히 섞어서 사용하는 모델



Levit architecture

Background

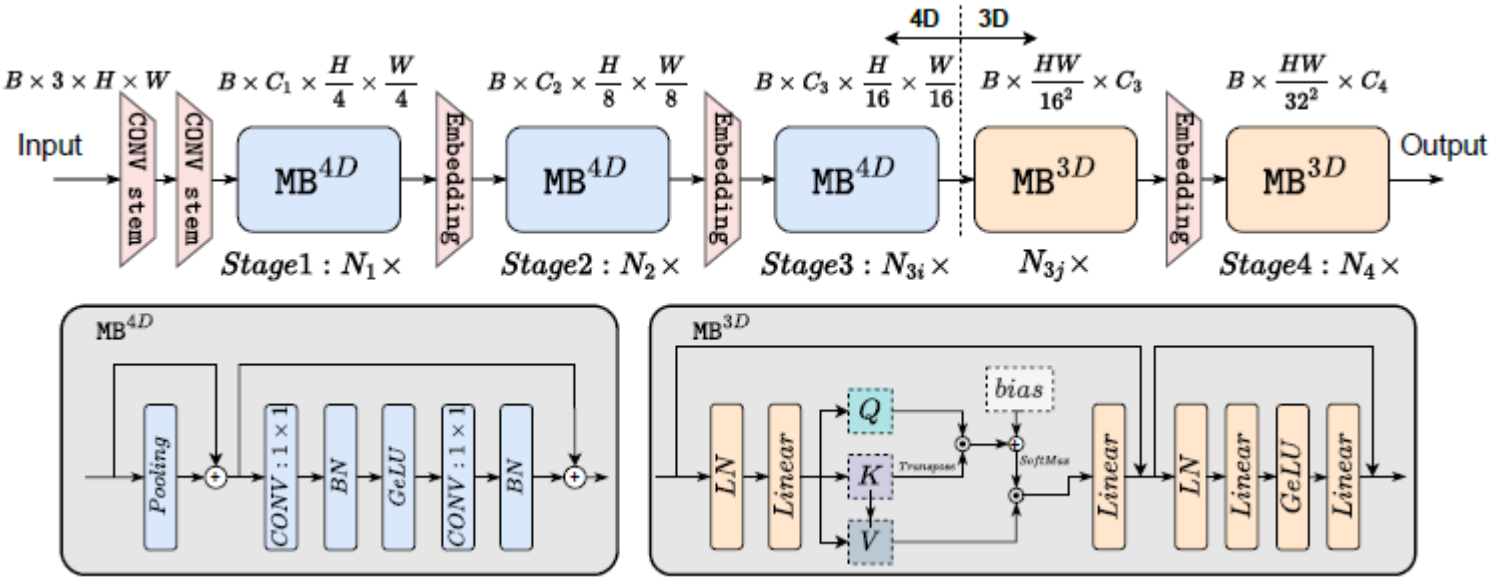
- 이 이후에도 transformer 기반의 다양한 경량화 모델이 등장
  - Poolformer[1] : self attention이 중요한 것이 아니라 transformer가 갖고 있는 기본적인 구조가 중요하다고 주장
    - Self attention의 자리에 가장 기본적인 연산 pooling을 넣어도 잘 된다는 것을 보여줌



Poolformer architecture

Background

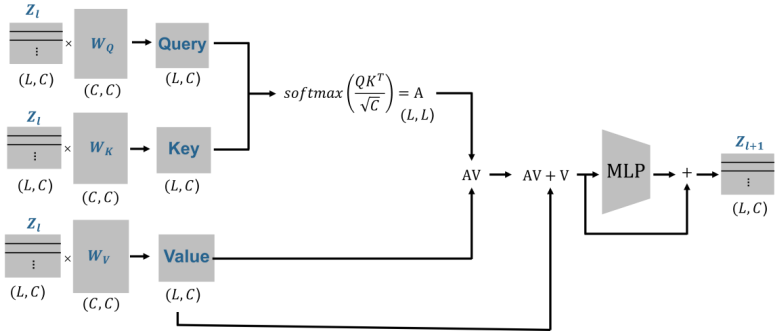
- 이 이후에도 transformer 기반의 다양한 경량화 모델이 등장
  - Efficientformer[1] : Poolformer를 기반으로 convolution과 self attention을 적절히 사용



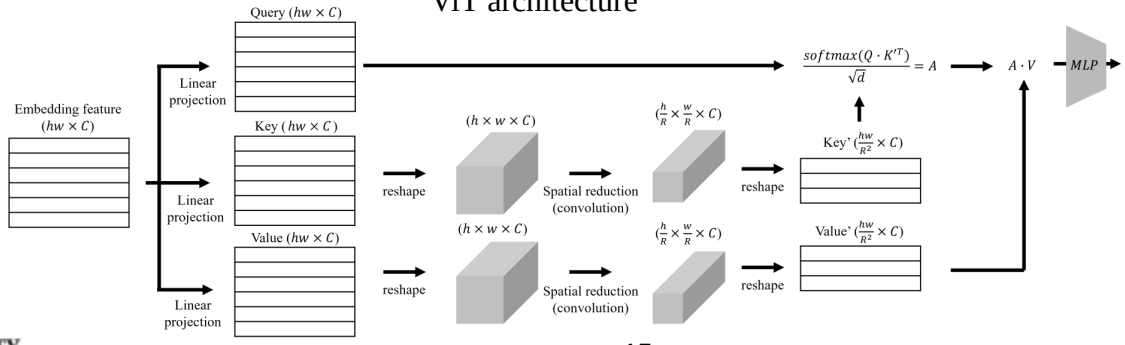
Efficientformer architecture

Efficient attention

- 기존 문제
  - Self attention을 수행할 때 quadratic한 computational cost가 발생
    - Vision transformer(ViT) : $2(hw)^2C \rightarrow \mathrm{O}(hw^2C)$
    - Pyramid vision transformer(PVT) : $2\ \frac{(hw)^2}{R^2}C \rightarrow \mathrm{O}(hw^2C)$
      - 즉, 해상도 (token의 개수) 에 quadratic 하게 computational cost가 발생



ViT architecture



PVT architecture

Efficient attention

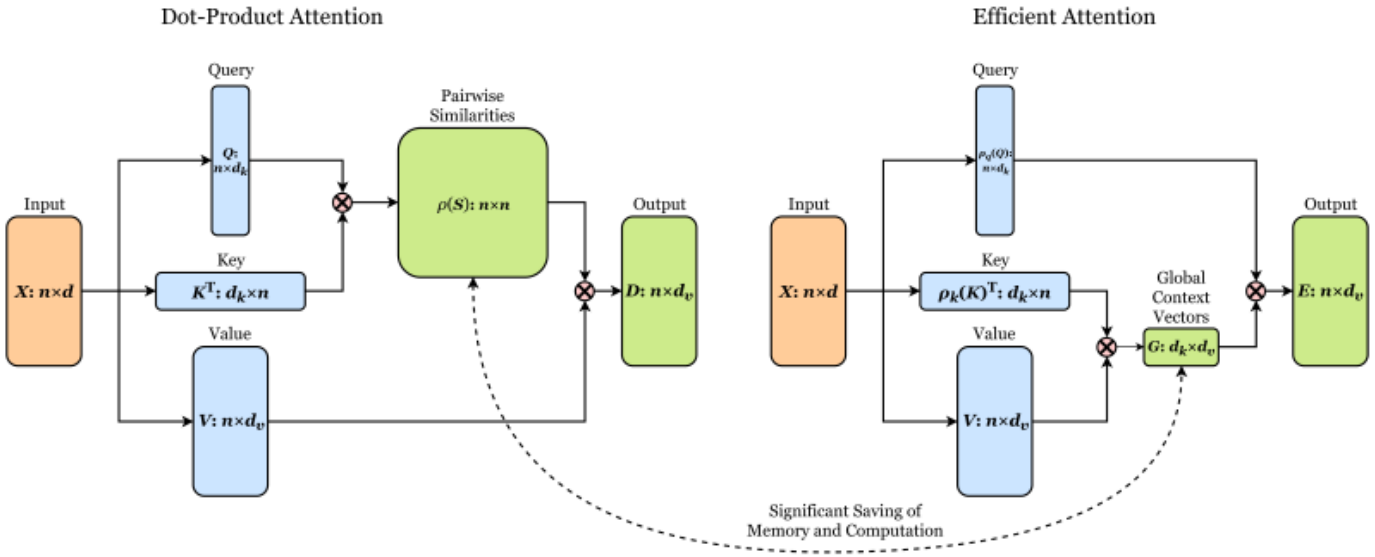
- Attention with linear complexity[1]
  - self attention 연산
    - $A(Q\ ,K\ ,V\ ) = softmax(\frac{QK^T}{\sqrt{D}})V$
    - 행렬곱은 결합법칙 성립
      - (AB)C = A(BC)
    - 근데 이때 $\sqrt{D}$로 나눠주고 softmax를 취하는 과정이 없다고 가정하면
    - $(Q_hK_h^T)V_h$ 을 $Q_h(K_h^TV_h)$ 으로 바꿔줬을 때 똑같은 연산인데 computational complexity가 바뀜
      - Query, key, value의 shape이 모두 $T \times D$ 일 때
      - $(Q_hK_h^T)V_h$ 연산 $\rightarrow O(T^2D)$
        - ✓$(T \times D) \cdot (D \times T) \rightarrow (T \times T)$ 연산량 T^2D
        - ✓$(T \times T) \cdot (T \times D) \rightarrow (T \times D)$ 연산량 T^2D
      - $Q_h(K_h^TV_h)$ 연산 $\rightarrow O(D^2T)$
        - ✓$(D \times T) \cdot (T \times D) \rightarrow (D \times D)$ 연산량 D^2T
        - ✓$(T \times D) \cdot (D \times D) \rightarrow (T \times D)$ 연산량 D^2T

Efficient attention

- Attention with linear complexity
  - Multi head self attention 연산
    - $A(Q\ ,K\ ,V\ ) = softmax(\frac{QK^T}{\sqrt{D}})V$
      - $(Q_h K_h^T)V_h$ 연산 → $O(T^2 D)$
      - $Q_h(K_h^T V_h)$ 연산 → $O(D^2 T)$
    - 그럼 이제 $\sqrt{D}$로 나눠주기 및 softmax로 묶여있는 함수를 decompose 해야함
    - 근데 위의 과정이 결국에 normalization을 수행하는 과정
    - 그래서 이를 $A(Q\ ,K\ ,V\ ) = softmax(\frac{QK^T}{\sqrt{D}})V \approx \frac{QK^T}{T}V$ 로 근사
    - 그런데 $softmax\left(\frac{QK^T}{\sqrt{D}}\right)V \approx \left(\phi(Q)\phi(K^T)\right)V$ 로 근사시키는 방법[1]도 있음
      - ϕ는 learnable parameter로 구성된 linear projection 수행하는 함수

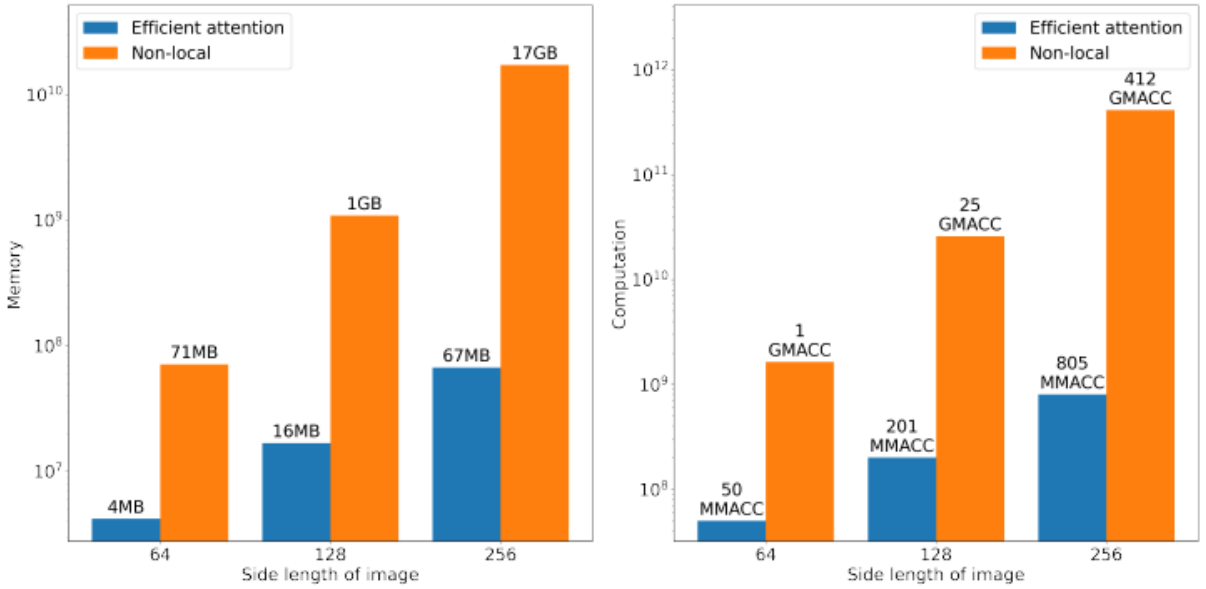
Efficient attention

- Attention with linear complexity
  - Multi head self attention 연산
    - $A(Q\ ,K\ ,V\ ) = softmax(\frac{QK^T}{\sqrt{D}})V$
      - $\frac{Q}{\sqrt{T}}\frac{K^T}{\sqrt{T}}V$로 decompose할 수 있고 $\frac{Q}{\sqrt{T}}(\frac{K^T}{\sqrt{T}}V)$의 연산을 수행 → $O(D^2T)$
      - 이를 통해 input image resolution T에 대해서 linear complexity를 가질 수 있음



Efficient attention

- Attention with linear complexity
  - Multi head self attention 연산
    - $(Q_h K_h^T)V_h$ 연산 → O(T^2D)
    - $Q_h(K_h^T V_h)$ 연산 → O(D^2T)



Resource requirements under different input sizes

Results

	EA module				Non-local module				
Layer(s)	Box	Mask	Mem.	Comp.	Box	Mask	Mem.	Comp.	Input size
None	39.4	35.1	0	0	39.4	35.1	0	0	N/A
res3	40.2	36.0	41.3 M	1.21 G	40.3	35.9	122 M	3.74 G	56×80
res4	40.2	35.9	19.5 M	596 M	40.1	36.0	24.5 M	748 M	28×40
fpn1	39.9	35.8	220 M	5.28 G	OOM	OOM	20.8 G	662 G	224×320
fpn2	39.7	35.7	55.1 M	1.32 G	OOM	OOM	1.34 G	42.3 G	112×160
fpn3	39.7	35.5	13.8 M	330 M	39.8	35.5	94.0 M	2.86 G	56×80
fpn4	39.7	35.4	3.46 M	82.6 M	39.5	35.3	8.46 M	234 M	28×40
fpn5	39.6	35.3	877 K	20.6 M	39.4	35.2	1.17 M	28.4 M	14×20
res3-4+fpn3-5	40.6	36.2	78.9 M	2.24 G	**40.7**	**36.3**	250 M	7.62 G	N/A
res3-4+fpn1-5	**41.2**	**36.7**	354 M	8.85 G	OOM	OOM	22.4 G	712 G	N/A

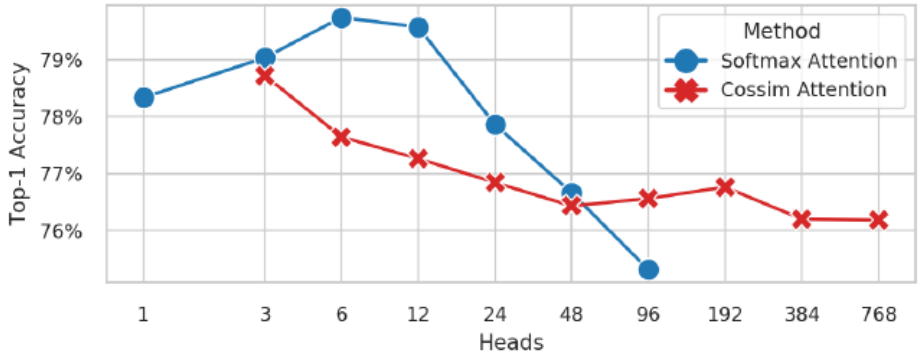
Comparison between the efficient attention and non-local modules on MS-COCO 2017 object detection and instance segmentation.

Hydra Attention[1]

- Multi-head attention
  - $A(Q_h, K_h, V_h) = softmax(\frac{Q_h K_h^T}{\sqrt{D}})V_h \qquad \forall h \in \{1, \dots, H\}$
    - $Q_h, K_h, V_h \in \mathbb{R}^{T \times \frac{D}{H}}$ 일 때
    - $Q_h K_h^T$ 연산 시 $(T \times \frac{D}{H}) \cdot (\frac{D}{H} \times T)$ 연산 H번 $\rightarrow HT^2 \frac{D}{H} \rightarrow T^2 D$
    - 즉, multi head를 적용하기 전과 후가 똑같음
- 하지만 linear attention을 사용한다면 상황이 달라짐
  - $A(Q_h, K_h, V_h) = \left(\phi(Q_h)\phi\left(K_h^T\right)\right)V_h = \phi(Q_h)(\phi\left(K_h^T\right)V_h) \qquad \forall h \in \{1, \dots, H\}$
    - $\phi\left(K_h^T\right)V_h$ 연산 시 $(\frac{D}{H} \times T) \cdot (T \times \frac{D}{H})$ 연산 H번 $\rightarrow HT(\frac{D}{H})^2 = TD^2/H$
    - Head 개수가 연산량에 영향을 미침
      - Head 개수가 증가할수록 complexity가 감소하는 형태

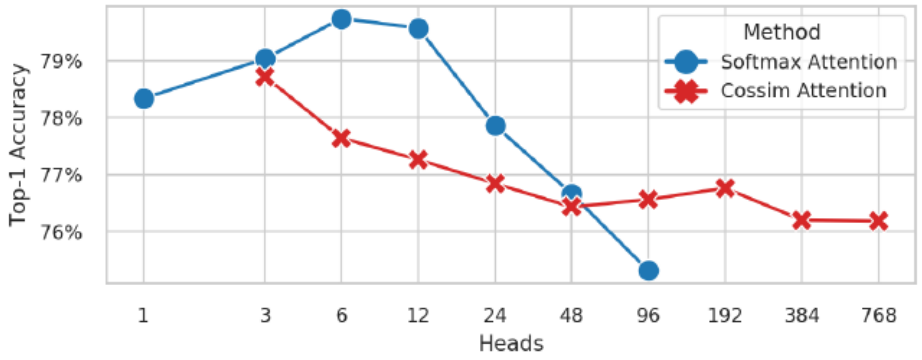
Hydra Attention

- Multi-head linear attention
  - $A(Q_h, K_h, V_h) = \left(\phi(Q_h)\phi\left(K_h^T\right)\right)V_h = \phi(Q_h)\left(\phi\left(K_h^T\right)V_h\right) \qquad \forall h \in \{1, \ldots, H\}$
    - $\phi\left(K_h^T\right)V_h$ 연산 시 $\left(\frac{D}{H} \times T\right) \cdot \left(T \times \frac{D}{H}\right)$ 연산 H번 $\rightarrow HT\left(\frac{D}{H}\right)^2 = TD^2/H$
    - Head 개수가 증가할수록 complexity가 감소하는 형태
    - 그럼 head를 얼마나 증가시킬 수 있을지 최적의 값을 찾아봄
      - 원래 기존 transformer에서 사용되는 head 개수는 보통 6개~16개 정도 가량임
      - 기존 multi head attention (Softmax attention)과 multi head linear attention (Cossim attention)을 head 개수에 따라 분석
        - ✓기존 self attention 은 head 개수가 12개 이상부터 성능이 급격히 감소하는 것에 반면 Linear attention의 경우 H=768까지 상당히 일관되게 유지되는 것을 실험적으로 분석



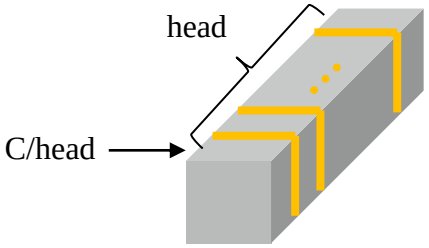
Hydra Attention

- Multi-head linear attention
  - 분석 실험에 의거해 head개수를 channel(token dimension)과 동일하게 만드는 방법 제안
    - $A(Q_h, K_h, V_h) = \left(\phi(Q_h)\phi\left(K_h^T\right)\right)V_h = \phi(Q_h)(\phi\left(K_h^T\right)V_h) \qquad \forall h \in \{1, \dots, H\}$
    - 위의 수식을 바탕으로 head개수를 token dimension과 동일하게 하면 각 head에 할당된 dimension이 1이 됨
      - $Q_h, K_h, V_h \in \mathbb{R}^{T\times 1}$ 즉, vector가 됨
    - $Hydra(Q_h, K_h, V_h) = \phi(Q_h) \odot \sum_{t=1}^{T} \phi(K_h^t) \odot V_h^t$
      - $\odot$: element wise multiplication
      - Key 및 value vector에 대한 내적을 수행한 scalar값이 query에 곱해지는 형태

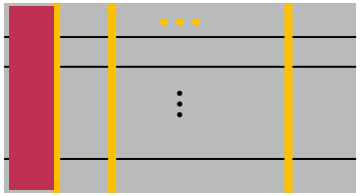


Hydra Attention

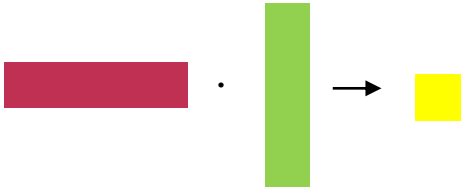
- Multi-head linear attention

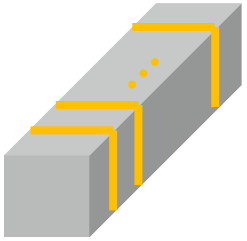


Key(H×W × C)



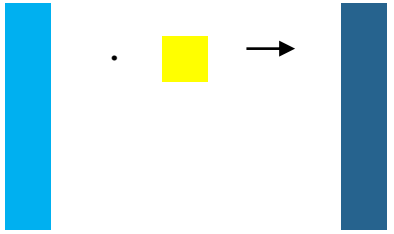
Key (HW × C)





Value (H × W × C)

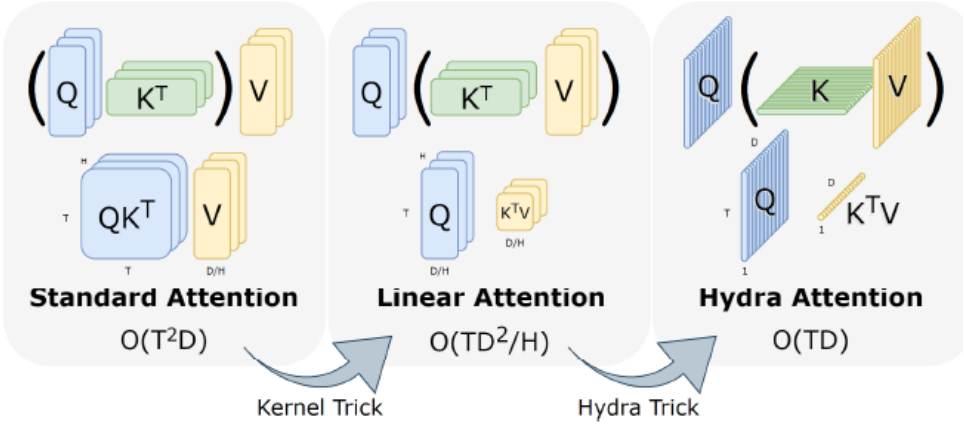
Value (HW × C)



Query (HW × C)

Hydra Attention

- Multi-head linear attention
  - 분석 실험에 의거해 head개수를 channel(token dimension)과 동일하게 만드는 방법 제안
    - $A(Q_h, K_h, V_h) = \left(\phi(Q_h)\phi\left(K_h^T\right)\right)V_h = \phi(Q_h)\left(\phi\left(K_h^T\right)V_h\right) \qquad \forall h \in \{1, \dots, H\}$
    - 위의 수식을 바탕으로 head개수를 token dimension과 동일하게 하면 각 head에 할당된 dimension이 1이 됨
      - $Q_h, K_h, V_h \in \mathbb{R}^{T\times 1}$ 즉, vector가 됨
    - $Hydra(Q_h, K_h, V_h) = \phi(Q_h) \odot \sum_{t=1}^{T} \phi(K_h^t) \odot V_h^t$
      - $\odot$: element wise multiplication
      - Key 및 value vector에 대한 내적을 수행한 scalar값이 query에 곱해지는 형태



Hydra Attention

- Multi-head linear attention
  - 분석 실험에 의거해 head개수를 channel(token dimension)과 동일하게 만드는 방법 제안
    - $Hydra(Q_h, K_h, V_h) = \phi(Q_h) \odot \sum_{t=1}^{T} \phi(K_h^t) \odot V_h^t$
      - $Q_h, K_h, V_h \in \mathbb{R}^{T \times 1}$
      - $\odot$: element wise multiplication
      - Key 및 value vector에 대한 내적을 수행한 scalar값이 query에 곱해지는 형태
    - Hydra attention의 연산량
      - $\sum_{t=1}^{T} \phi(K_h^t) \odot V_h^t$ 연산시 (1×T)·(T ×1)연산이 H번 → TH
      - $\phi(Q_h) \odot$A 연산시 (T×1) · (1×1) 연산이 H번 → TH
        - ✓$\sum_{t=1}^{T} \phi(K_h^t) \odot V_h^t$=A
      - 따라서 총 연산량 2TH
      - 그런데 H = D이므로 2TD → O(TD)

Hydra Attention

- Complexity 비교
  - 기존 self attention complexity $O(T^2D)$ $$A(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V$$
  - Linear attention complexity $O(TD^2)$ $$A(Q,K,V;\phi) = \phi(Q)\left(\phi(K)^T V\right)$$
  - 제안하는 Hydra attention complexity $O(TD)$ $$\text{Hydra}(Q,K,V;\phi) = \phi(Q) \odot \sum_{t=1}^{T} \phi(K)^t \odot V^t$$

Hydra Attention

- Results

Method	Accuracy (%)	FLOPs (G)	Speed (im/s)
Standard Attention [32]	79.57	17.58	314.8
AFT-Simple [37]	74.02 (-5.55)	**16.87** (-4.0%)	346.1 (+9.9%)
PolyNL [2]	73.96 (-5.61)	**16.87** (-4.0%)	353.8 (+12.4%)
Hydra (2 layers)	**80.64** (+1.1)	17.46 (-0.7%)	321.9 (+2.3%)
Hydra (8 layers)	79.45 (-0.12)	17.11 (-2.7%)	334.8 (+6.3%)
Hydra (12 layers)	76.40 (-3.17)	**16.87** (-4.0%)	346.8 (+10.2%)

Results for different attention methods in a DeiT-B

Method	Accuracy (%)	FLOPs (G)	Speed (im/s)
Standard Attention [32]	81.33	55.54	92.5
Hydra (2 layers)	**81.92** (+0.59)	54.52 (-1.8%)	96.3 (+4.1%)
Hydra (7 layers)	81.26 (-0.07)	51.96 (-6.4%)	106.8 (+15.4%)
Hydra (12 layers)	77.85 (-3.48)	**49.40** (-11.0%)	117.6 (+27.1%)

384px Fine-Tuning

Conclusion

- Self attention의 complexity 관점의 경량화 기법 조사

- 현재 self attention 및 cross attention이 다양한 task에 적용되어 연구되고 있으므로 경량화 관점을 도입한다면 응용할 수 있음