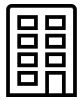
2023 하계 세미나

Efficient Vision Transformer



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

Presented by
유현우

Outline

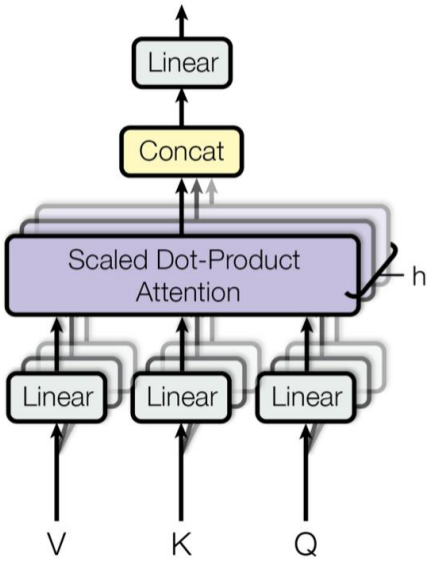
- Background
  - Transformer
  - Self attention
  - Vision Transformer architecture

- Token merging 기반의 efficient vision transformer
  - Token Pooling in Vision Transformers for Image Classification
    - WACV 2023 (23회 인용)
  - TOKEN MERGING: YOUR ViT BUT FASTER
    - ICLR 2023 (34회 인용)

- Conclusion

Background

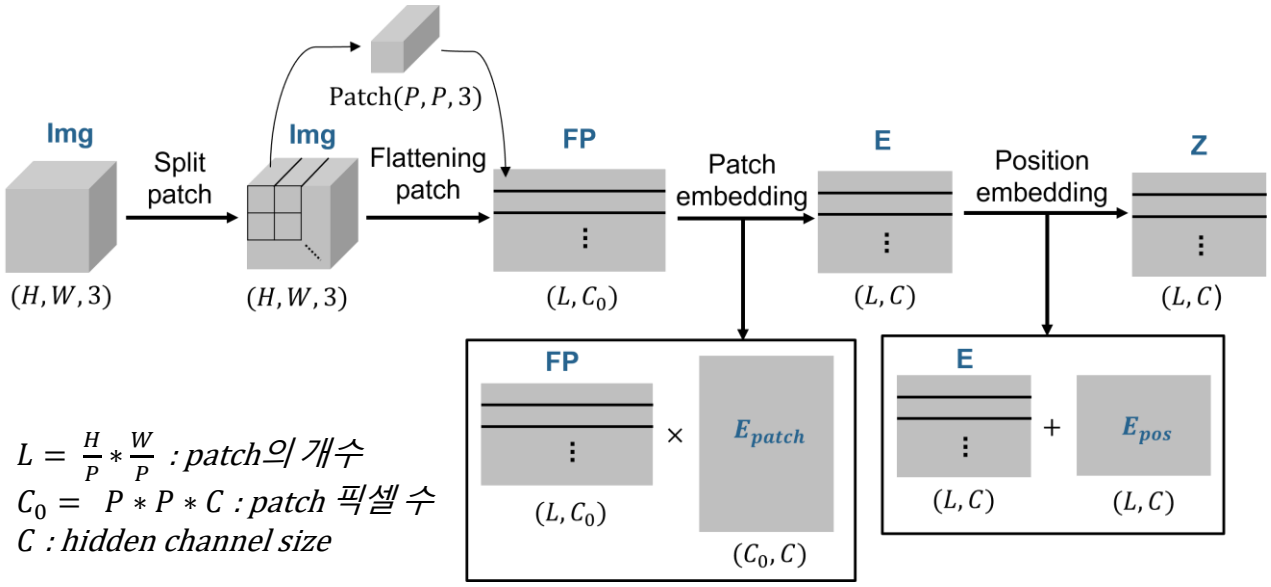
- Transformer[1]
  - Query, key, value 간에 similarity를 기반으로 feature representation을 수행
  - 특정 token과 전역적으로 존재하는 token과의 관계를 고려하므로 global한 특성을 고려하여 feature를 추출할 수 있다는 장점이 있음
    - 자연어 처리 분야에서 처음 제안되었고, vision 분야에서도 다양한 task에서 좋은 성능을 보임



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

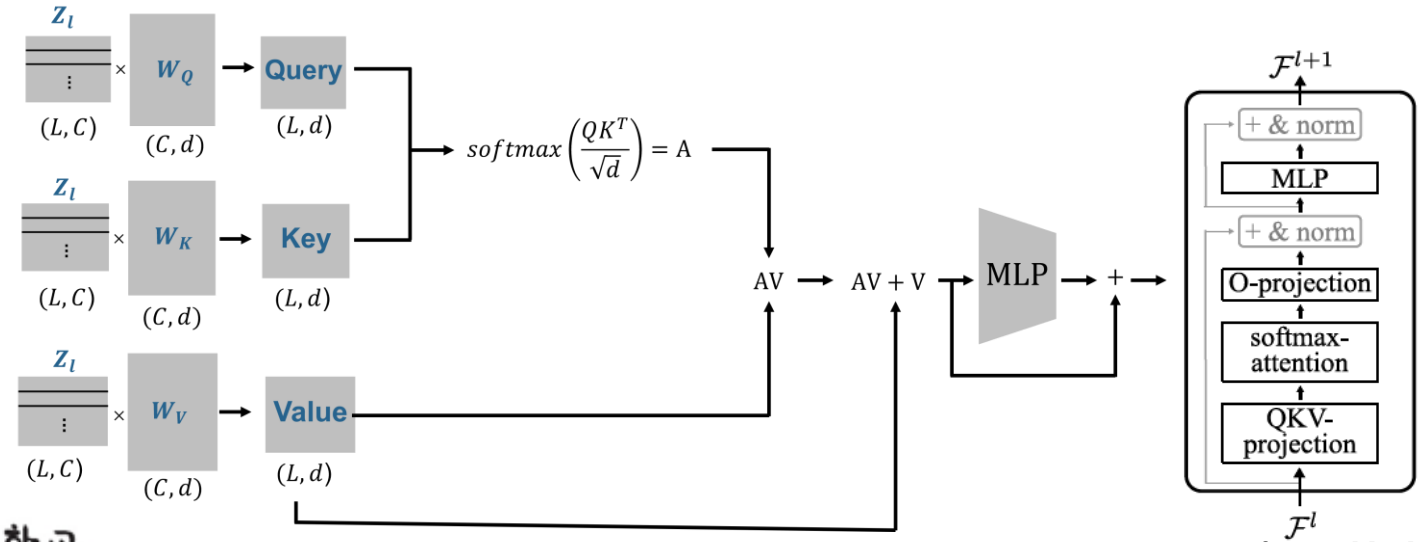
- Self attention
  - Learnable parameter로 linear projection을 수행하여 query, key, value 생성
  - Query, key간의 연산(Key · Query.transpose)을 통해 얻은 가중치를 value에 적용

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

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

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

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

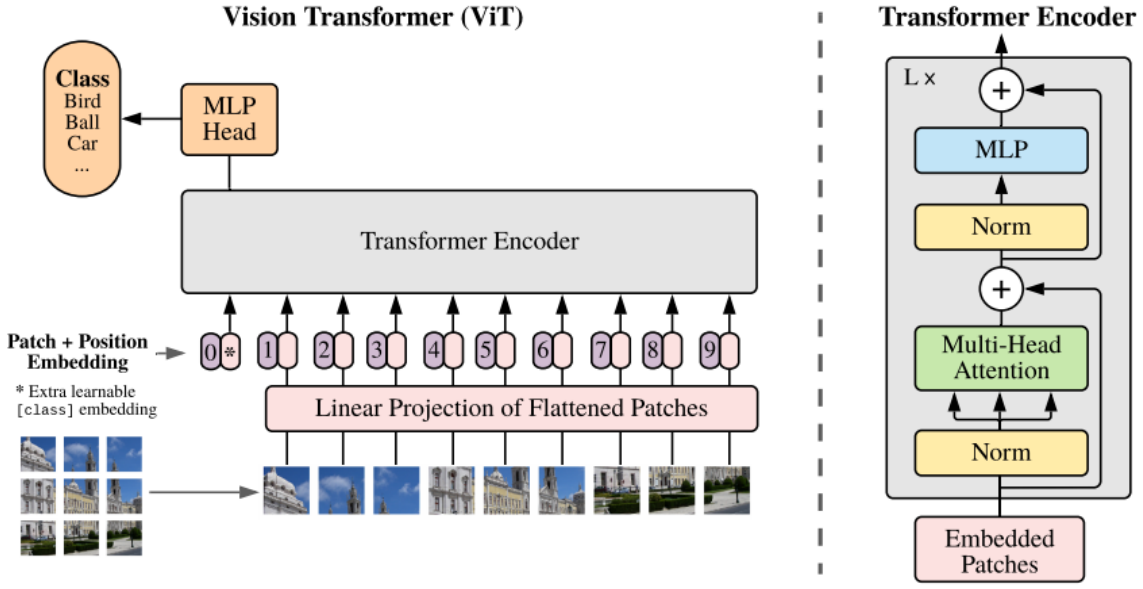


Self-attention overview

Transformer block

Background

- Vision Transformer[1] architecture 세부 구조
  - Patch size를 16x16으로 설정하여, patch embedding 시 16배 down-sampling 수행
  - Transformer encoder block이 L번 반복되는 구조
    - Transformer encoder block은 self-attention 및 feed-forward layer로 구성되어 있음

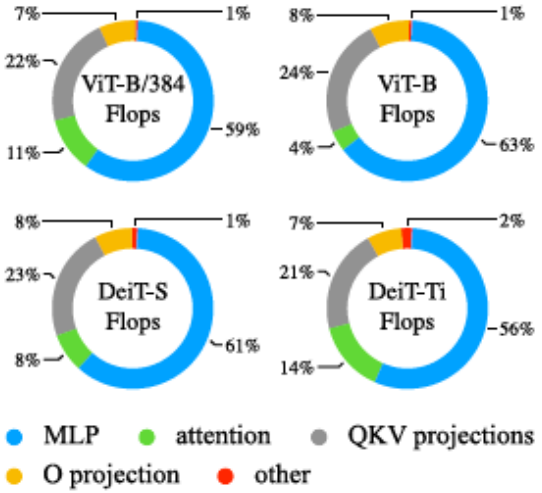


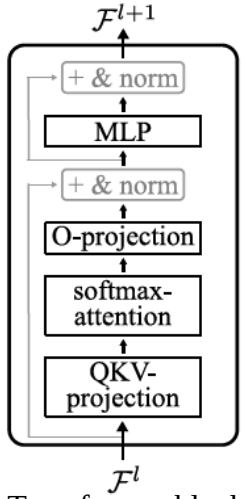
Vision transformer architecture

Token pooling[1]

- 수많은 연구가 self-attention 에서의 quadratic complexity를 개선하기 위해 연구되어 오고 있음
- 하지만 실제 computational cost의 비중을 살펴보면 MLP의 비중이 가장 큼
  - 이는 classification task에서 ViT 기반 모델을 사용할 때 한정되는 조건
    - Resolution에 대해 quadratic 한다고 하더라도 classification task에서는 특별한 조건이 아닌 이상 대부분 224x224 resolution 으로 정해 놓고 train과 inference를 진행
- 즉, 이를 통해 근본적으로 사용되는 token의 개수를 줄이는 것이 경량화에 더 효과적일 수 있음을 보여줌

Layer	Complexity	Computation (10^9 Flops)			
		ViT-B/384 (N=577)	ViT-B (N=197)	DeiT-S (N=197)	DeiT-Ti (N=197)
softmax-attn.	$\mathcal{O}(LN^2M)$	6.18	0.72	0.36	0.18
QKV proj.	$\mathcal{O}(LNM^2)$	12.25	4.18	1.05	0.26
O proj.	$\mathcal{O}(LNM^2)$	4.08	1.39	0.35	0.09
MLP	$\mathcal{O}(LNM^2)$	32.67	11.15	2.79	0.70
Total	$\mathcal{O}(LNM(M+N))$	55.5	17.6	4.6	1.3





Transformer block

ViT 모델의 내부 연산별 computation 비중

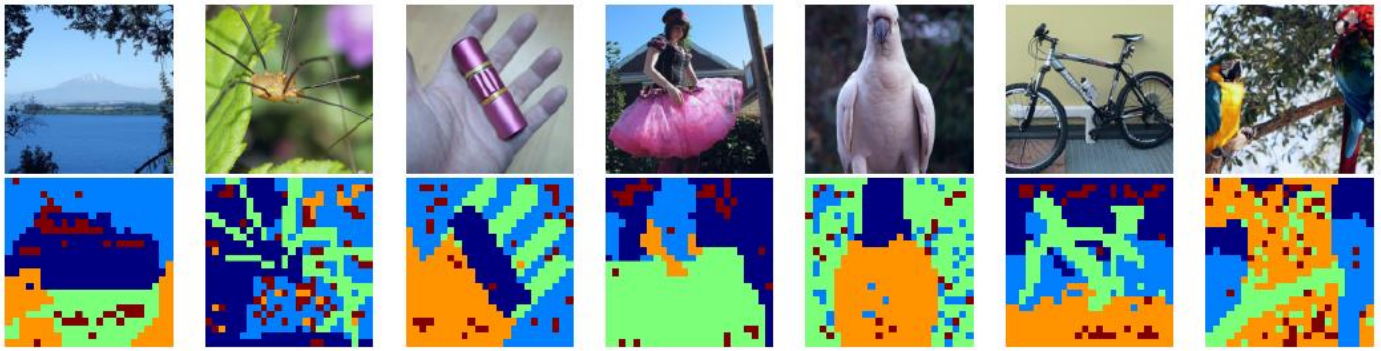
Token pooling

- 이미지에는 edge가 적고 smooth한 표면이 많이 포함되어 있음
  - 그에 따라 feature 내에 redundancy 한 token을 포함하게 됨
- 또한 attention 연산은 low-pass filter의 역할과 유사
  - 따라서 attention의 output은 redundant information을 가진 similar한 token을 포함

$$\boldsymbol{O}_h = \boldsymbol{A}_h \boldsymbol{V}_h, \text{ such that}$$
$$\boldsymbol{A}_h = \text{softmax}(\boldsymbol{Q}_h \boldsymbol{K}_h^\top / \sqrt{d}) \in \mathbb{R}^{N \times N}$$

$$\longrightarrow \quad \boldsymbol{o}(\boldsymbol{q}) = \frac{1}{z(\boldsymbol{q})} \sum_{i=1}^{N} \exp(\alpha\, \boldsymbol{q} \cdot \boldsymbol{k}_i)\, \boldsymbol{v}_i$$
$$z(\boldsymbol{q}) = \textstyle\sum_{i=1}^{N} \exp(\alpha\, \boldsymbol{q} \cdot \boldsymbol{k}_i)$$

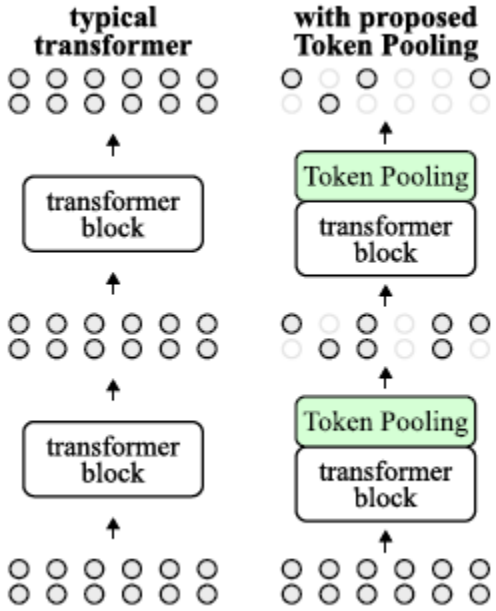
$$\boldsymbol{o}(\boldsymbol{q}) = \frac{1}{z'(\boldsymbol{q})} \sum_{i=1}^{N} \exp\left(-\frac{\alpha}{2}\|\boldsymbol{q} - \boldsymbol{k}_i\|^2\right) \boldsymbol{v}_i$$
$$= \frac{1}{z'(\boldsymbol{q})} \int \exp\left(-\frac{\alpha}{2}\|\boldsymbol{q} - \boldsymbol{k}\|^2\right) \left(\sum_{i=1}^{N} \delta(\boldsymbol{k} - \boldsymbol{k}_i) \boldsymbol{v}_i\right) \mathrm{d}\boldsymbol{k}$$
$$= \frac{1}{z'(\boldsymbol{q})}\, G\left(\boldsymbol{q}; \frac{1}{\alpha}\right) * S(\boldsymbol{q}; \mathcal{K}, \mathcal{V}), \qquad (7)$$



Token pooling via cluster analysis of token representations

Token pooling

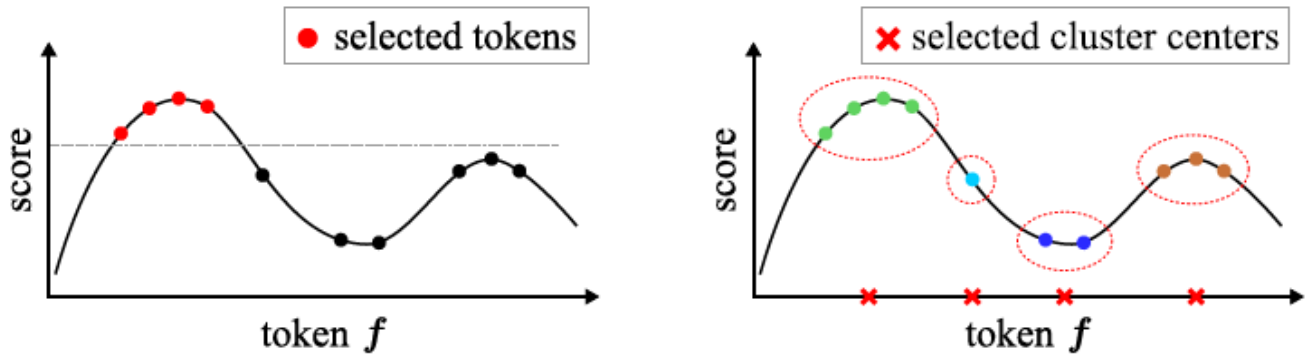
- 그래서 transformer block 뒤에 제안하는 token pooling 방법을 적용하는 구조를 제안
  - 기존 efficient vision transformer 에서 self-attention에 주목하는 것과는 다른 관점
  - 이때 redundant한 token에 대해서 token을 줄이는 것이 핵심



기존 transformer와 token pooling 방법

Token pooling

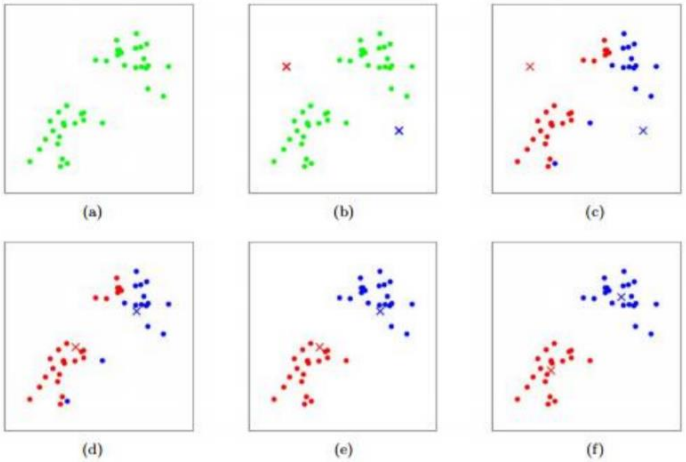
- Down-sampling method
  - Grid-downsampling
    - 기존 CNN 등에서 사용되는 특정 gird 단위로 convolution 및 pooling을 적용해서 downsampling을 수행하는 방법
  - Score-based token downsampling [1]
    - Attention이 수행된 feature를 대상으로 높은 score의 token만을 남겨둠
    - 각 layer 별로 K개의 token을 남겨두고 token pruning
  - Proposed token pooling
    - Feature를 대상으로 clustering을 적용하여 각 token의 군집별로 pooling을 적용



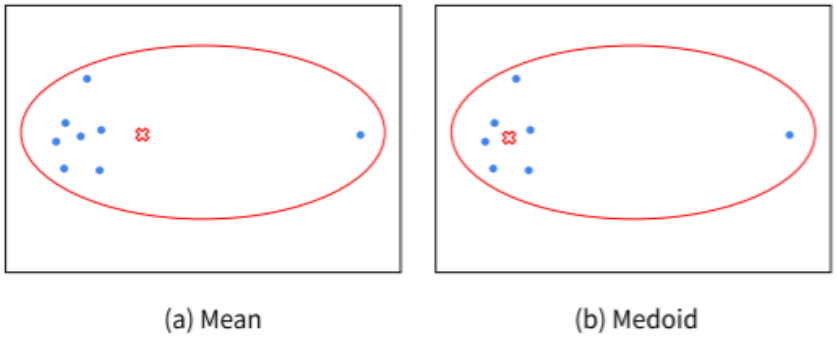
Score-based token downsampling 방법과 제안하는 token pooling 방법

Token pooling

- Clustering
  - Clustering은 cluster 개수 k와 중심점의 초기값을 입력하면 iteration에 따라 각 토큰의 그룹을 할당함
  - K-means clustering : 중심점에 해당하는 군집을 할당한 후, 할당된 군집의 좌표 평균을 새로운 중심점으로 설정하여 이를 반복
  - K-medoids : K-means 방법은 노이즈 및 아웃라이어에 민감하다는 점을 보완하여 평균 대신 중간점을 사용
  - Weighted clustering : 군집을 이루는 token에 다른 가중치를 주어 means 및 medoids 방법을 사용



Iteration에 따른 clustering 과정



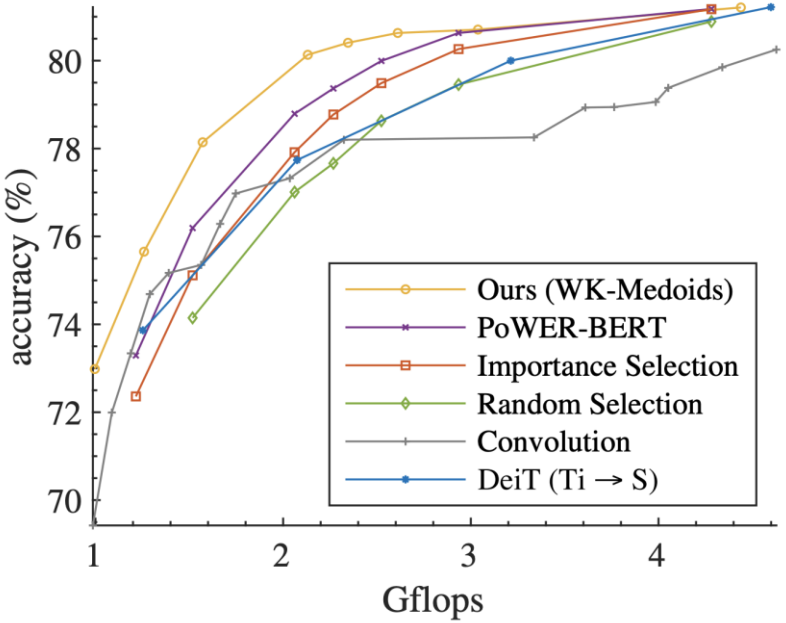
K-means와 k-medoid clustering 차이

Token pooling

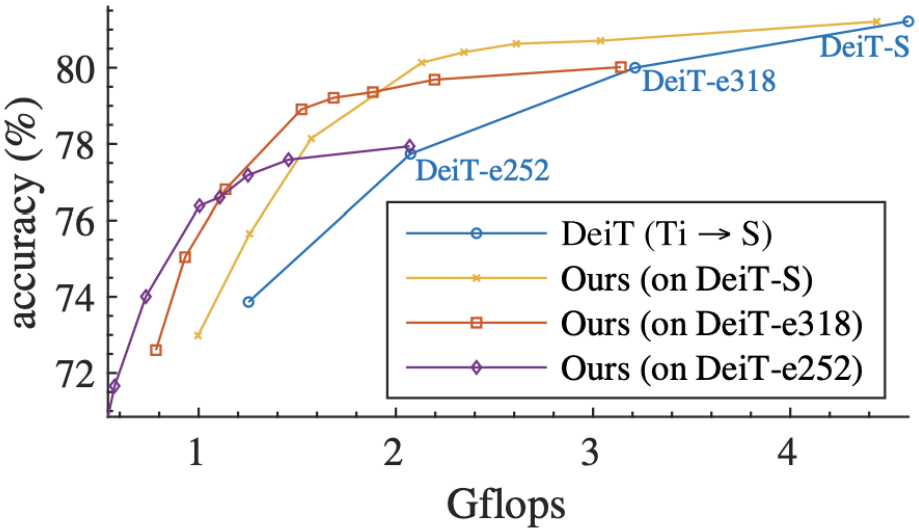
- Token pooling 과정
  - 기존 DeiT 모델을 이용하여 training 수행
  - 선택할 token의 개수 k를 결정하기 위해, 이를 구할 수 있는 layer를 삽입한 후 finetuning 진행
  - K에 맞추어 clustering을 하고, 이를 이용해 downsampling을 수행
  - 줄어든 모델에 대해 다시 finetuning 진행

Token pooling

- Experimental results
  - Downsampling 방법 비교
    - 다양한 k selection 기반 방법 중 기존에 좋은 성능을 보이는 score-based downsampling (PoWER-BERT)와 비교하여 우수한 성능을 보임



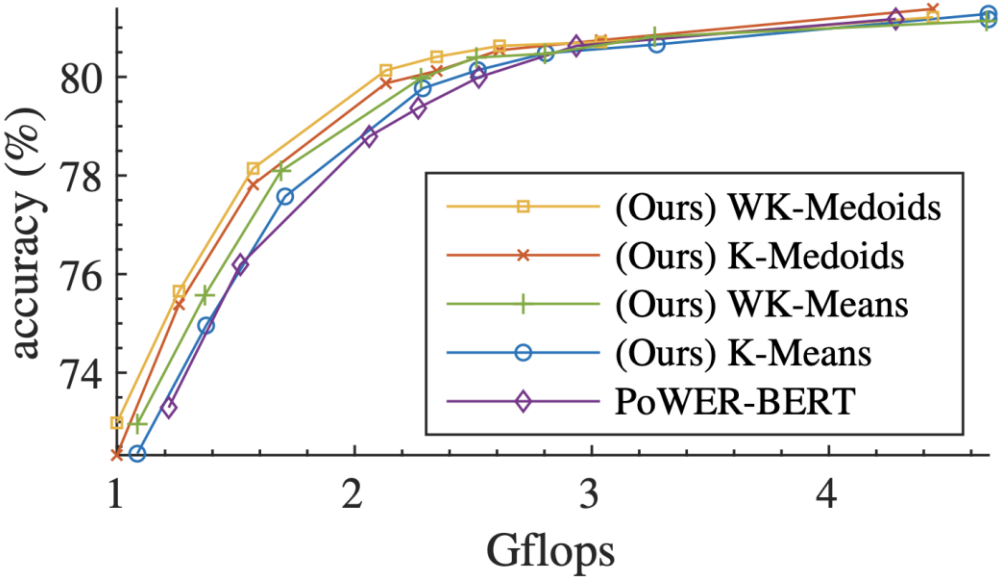
Downsampling 방법에 따른 ImageNet-1k top 1 accuracy



모델 크기별 token pooling 적용 결과

Token pooling

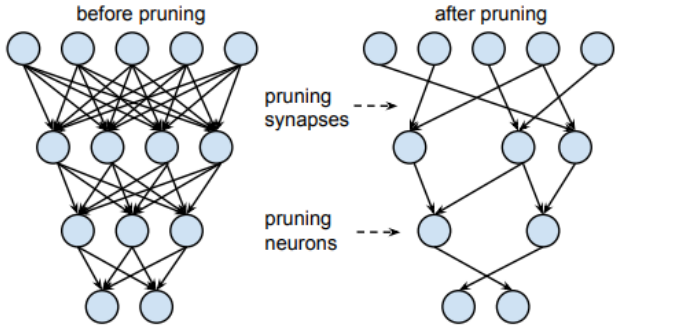
- Experimental results
  - 제안하는 방법의 clustering 기법 간의 성능 비교
    - 모든 cluster 기반 방법이 score 기반 방법인 PoWER-BERT 보다 좋은 성능을 보임
    - Clustering 기법 간에는 weighted k medoids clustering 기법이 가장 성능이 좋음



제안하는 방법에서 clustering 기법에 따른 성능 비교

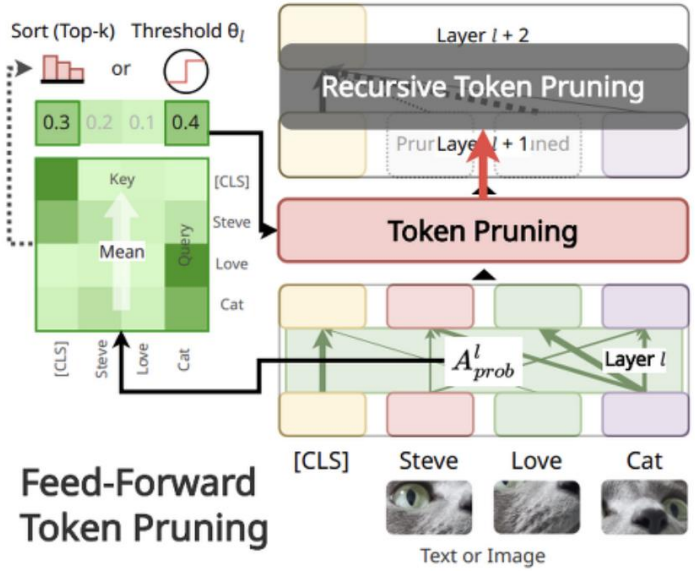
Token Merging[1]

- 기존 token reduction 방법
  - Pruning
    - Model의 weight들 중 중요도가 낮은 weight의 연결을 제거하여 경량화 하는 기법
  - Token pruning
    - Token 단위로 데이터를 다루는 transformer에서 사용되는 기법
    - 모델의 입력으로 사용되는 token들 중에서 중요하지 않거나 불필요한 토큰들을 제거



Pruning 개념도

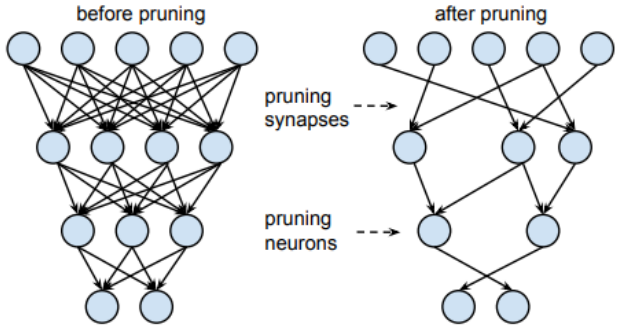
Pruning 과정



Token pruning 예시[2]

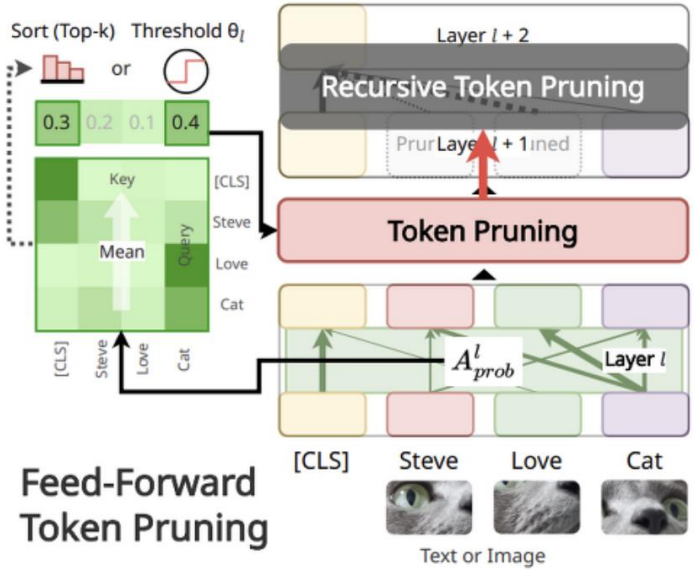
Token Merging

- 기존 token reduction 방법
  - Pruning 의 단점
    - Pruning으로 인한 정보 손실이 큼 (성능 저하가 큼)
    - 모델을 재학습시키는 과정이 요구됨 (몇몇은 hyper parameter를 변경)
    - 대부분 훈련 속도를 높이는데 적용할 수 없음
    - Input data에 대해서 다른 개수를 pruning 하므로 일괄적인 추론을 어렵게 함



Pruning 개념도

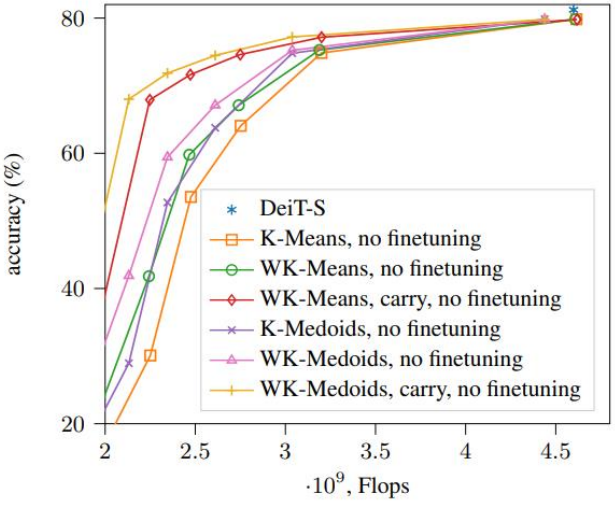
Pruning 과정

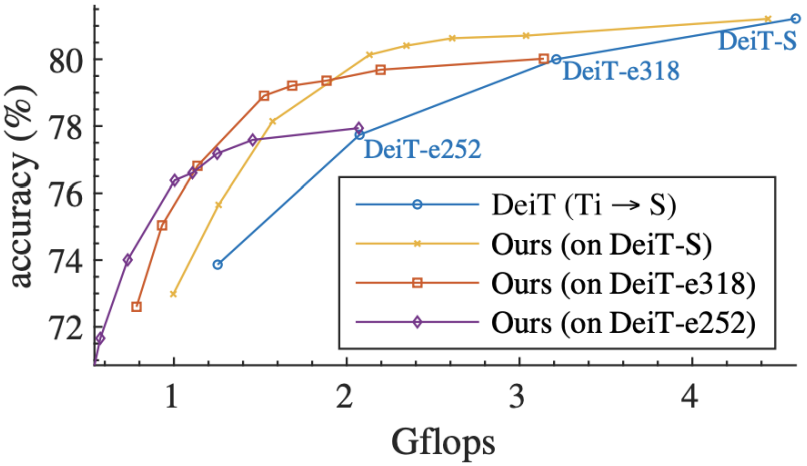


Token pruning 예시

Token Merging

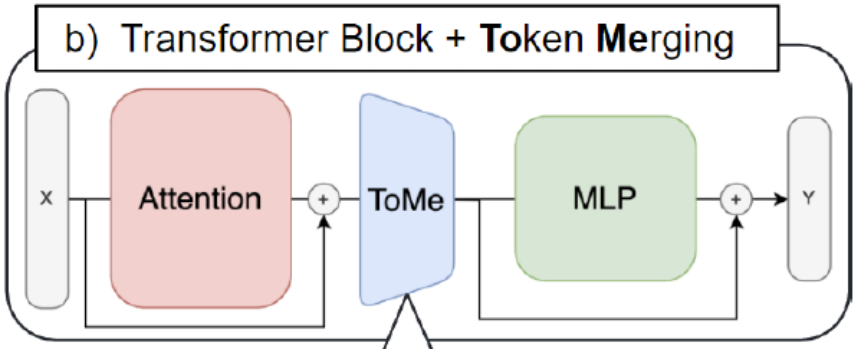
- Combining token
  - Token reduction을 하는 연구는 많지만, combining 하는 방법은 적음
  - Token pooling이 본 논문의 방법과 가장 유사
  - Token pooling의 단점
    - K-means 기반 방식은 iteration에 따라 수행되므로 느림
    - Pretrained model만을 이용해서 제대로 동작하지 못함
      - 꼭 finetuning을 필요로 하는 것은 아니지만, finetuning을 적용하지 않았을 경우 매우 큰 폭의 accuracy가 감소



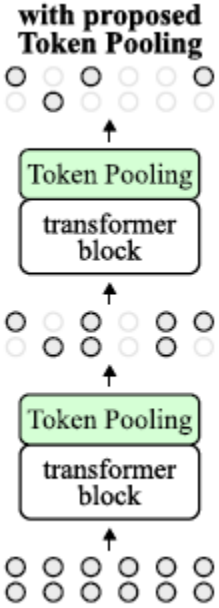


Token Merging

- Token merging 방법
  - 기존의 token pooling 방법과 다르게 attention과 MLP layer 사이에서 token merging을 적용
    - 이는 attention 내에 있는 feature를 사용해서 merging할 token을 결정하기 때문임
      - Attention 내의 정보를 활용하여 정확도를 향상
    - 만약 training 시에 ToMe를 적용한다면, 병합될 토큰을 기반으로 정보를 propagation할 수 있는 구조



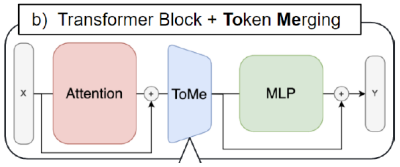
Token merging



Token pooling

Token Merging

- Token 간의 similarity를 어떻게 구할 것인지?
  - Transformer의 중간 feature는 각 token의 특성을 완벽히 encode할 수가 있음
    - 각 패치 영역의 특징을 완벽히 encoding
    - 이는 similarity를 구하는데 중요하지 않은 token 정보가 포함할 수 있음
  - Self-attention으로 해결
    - Attention 내부의 key feature는 query간의 similarity를 구하도록 학습되어, 각 token에 포함된 정보를 summarize 함
    - 따라서 제안하는 방법은 key feature를 이용해서 similarity를 구함
      - 이를 통해 유사한 정보를 포함하는 token을 결정



feature	acc	im/s
X_{pre}	83.02	**186.8**
X	83.70	182.8
K	**84.25**	182.9
Q	84.04	182.8
V	83.80	182.9

Feature choice

function	acc	im/s
eucl	**84.26**	182.5
cosine	**84.25**	**182.9**
dot	82.78	**183.0**
softmax	82.00	**183.0**

Distance function

aggregate	acc	im/s
concat	**84.32**	180.3
mean	84.25	**182.9**

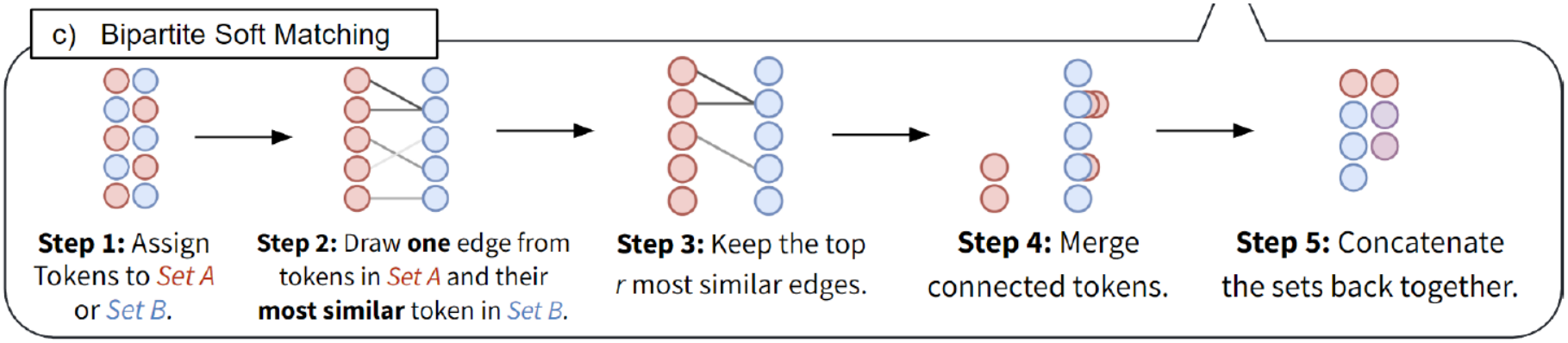
Head aggregation

Token Merging

method	acc	im/s
keep one	81.01	**185.4**
max pool	83.50	184.6
avg pool	83.57	183.8
weighted avg	**84.25**	182.9

Combine method

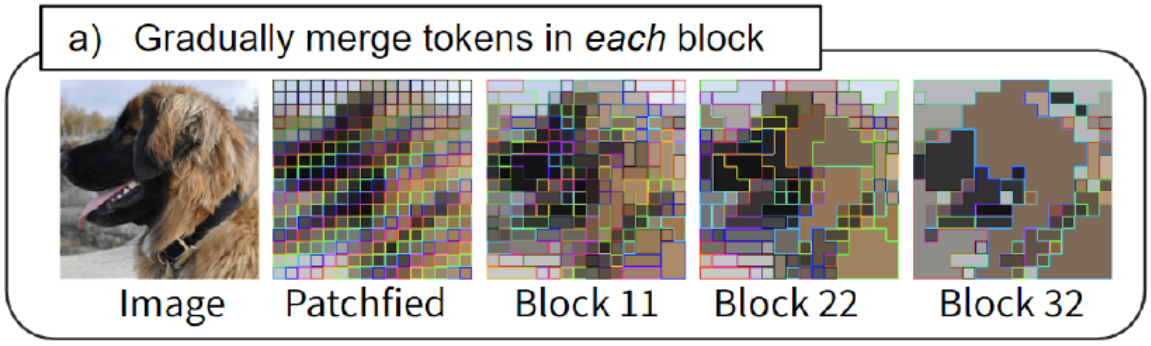
- Bipartite soft matching
  - Token을 spatial 축으로 한 픽셀 위치씩 번갈아가면서 이분함
  - Similarity를 기반으로 가장 유사한 토큰과 edge를 이어줌
  - Edge 중에 similarity score를 기반으로 top r 개를 골라냄
    - 여기서 r은 reducing 하고 싶은 token의 개수
  - Matching이 된 token 간에 combine을 진행
    - Weighted average pooling 방법이 가장 좋은 성능을 보임



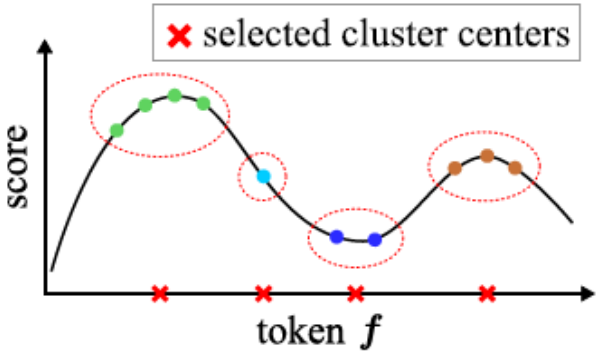
Bipartite soft matching 방법 과정

Token Merging

- Token merging의 적용 조건 및 목적
  - 주된 적용 네트워크로 ViT-Large/16 모델을 사용
    - Merging할 token 수 r=8로 두어 24개의 layer에서 98%의 토큰을 점진적으로 reduce
  - 병렬화할 수 없는 반복적인 작업을 피함
  - Token merging이 점진적으로 진행
    - 제안하는 방법은 reduce할 token의 개수 r에 대한 token merging을 모든 레이어(L)에서 점진적으로 진행
    - 반면 token pooling은 각 layer에서 reduce할 token의 개수를 제한하지 않음
      - 이는 pre-trained network에 악역향을 줄 수 있음



Transformer block에 따라 token merging visualization



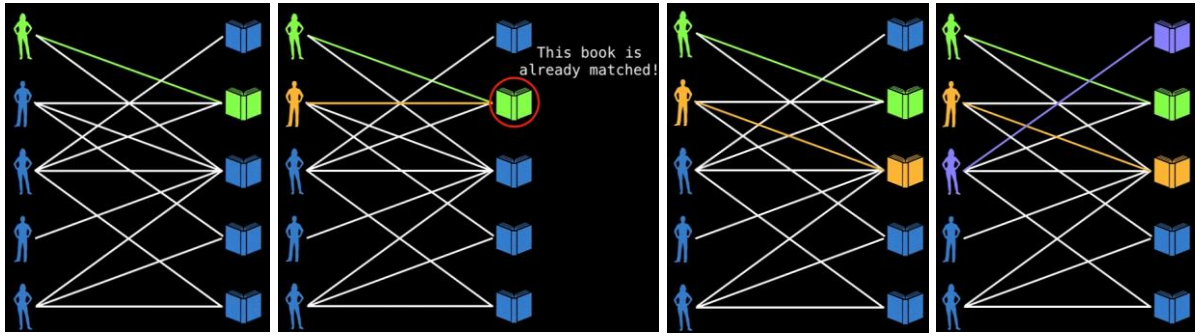
Token pooling에서 사용한 cluster 방법

Token Merging

- Experimental results
  - ViT-Large/16의 accuracy는 85.86, image/sec : 93.3 에 대해서 진행
    - masked autoencoder (MAE) 방법으로 학습
  - Token pruning 방법은 빠르지만, 성능 감소 폭이 큼
    - 중요한 정보를 갖고 있는 token을 지워버리기 때문
  - Token merging 방법 중 clustering 기반의 방법은 pruning 보다 약간 나은 정확도를 보이지만 처리 속도가 느려짐
    - 위의 방법은 finetuning 을 필요로 함
  - 제안하는 방법은 greedy matching과 bipartite matching에 대해서 실험
    - Greedy matching이 bipartite matching 보다 약간 정확하지만 순차적으로 진행되므로 처리속도가 느려짐

style	algorithm	acc	im/s
prune	random	79.22	**184.4**
prune	attn-based	**79.48**	183.8
merge	kmeans (2 iter)	80.19	169.7
merge	kmeans (5 iter)	80.29	147.5
merge	greedy matching	**84.36**	179.4
merge	bipartite matching	84.25	**182.9**

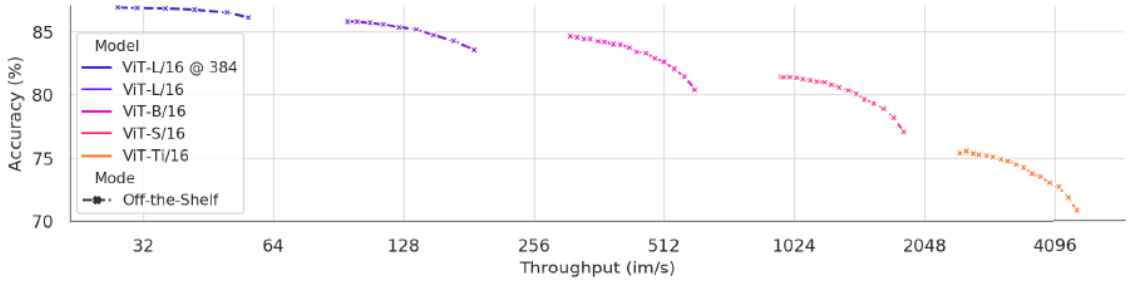
Matching 방법에 따른 성능



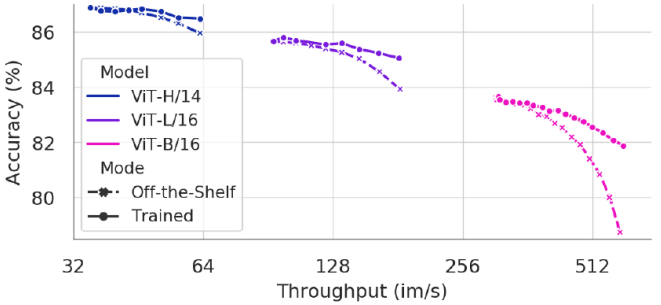
Greedy matching

Token Merging

- Experimental results
  - Token merging 기법을 적용하여 ViT 모델 크기와 관련 없이 모두 2배 가량의 throughput을 개선 시 정확도 분석
    - 작은 모델 (ViT-B, S, Ti)에서는 4~5 % 정확도 저하
    - 가장 큰 모델에서는 0.7 정확도 저하
    - 이는 큰 모델이 더 깊어서 feature 의 점진적인 변경을 allow하여 merging의 영향을 줄이기 때문
  - Finetuning 시 경량화에 따른 성능저하를 개선할 수 있음



ViT 모델(AugReg) 크기별 ToMe 적용 성능 분석



ViT 모델(MAE) 크기별 ToMe 적용 성능 분석 및 Finetuning 결과

Token Merging

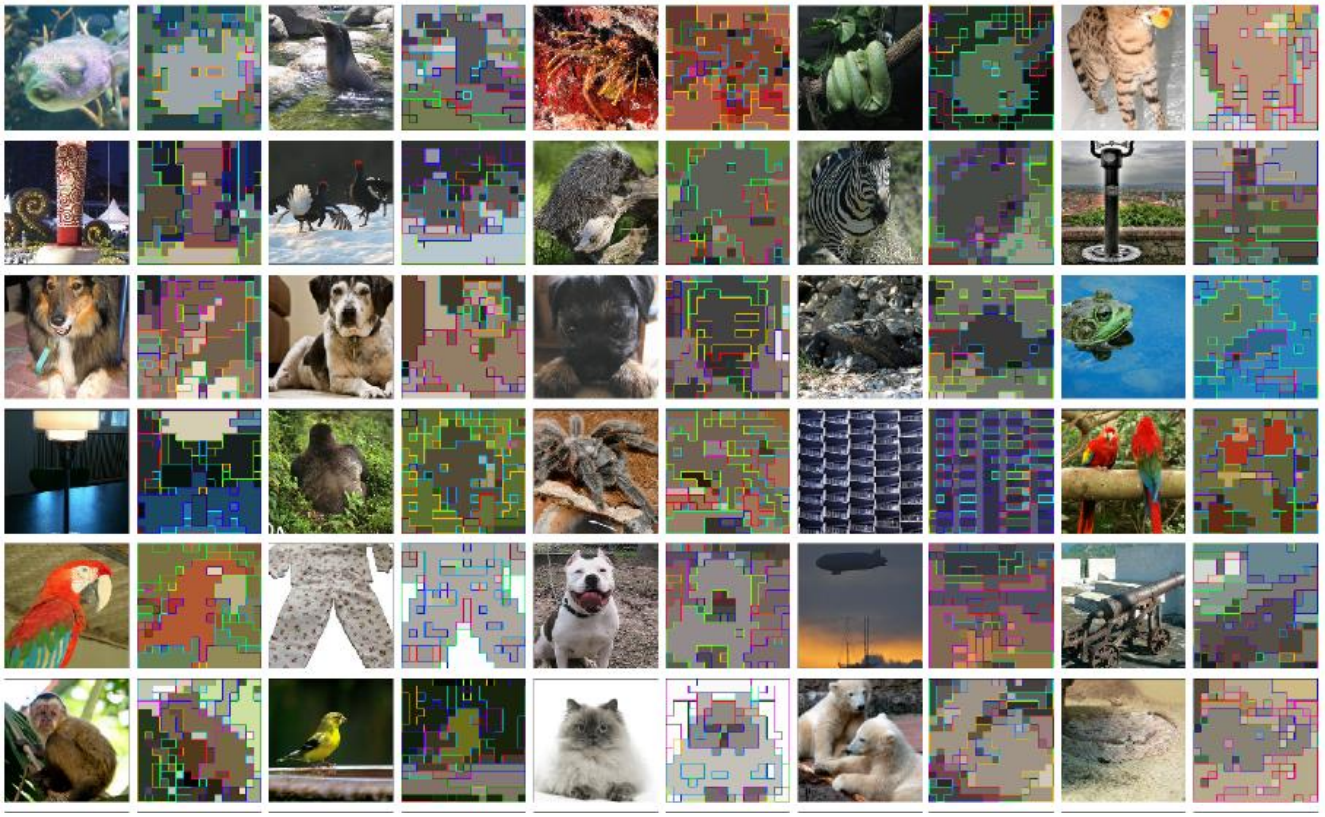
- Experimental results
  - 기존 SOTA 모델들과 비교했을 때, 경쟁력 있는 처리속도와 정확도를 보여줌

model	input	acc	gflops	im/s
Eff-B5	456	83.6	9.9	169‡
ViT-B MAE	224	83.6	17.6	309
Swin-B*	224	84.0	15.4	278‡
CSWin-B	224	84.2	15.0	250‡
MViTv2-B	224	84.4	10.2	253‡
ToMe				
ViT-L MAE $r8\searrow$	224	84.2	22.3	241
ViT-L MAE $r8\rightarrow$	224	**85.1**	31.0	183
ViT-L MAE	224	85.7†	61.6	93
Eff-B6	528	84.0	19.0	96‡
MViTv2-L	224	85.3	42.1	81
ToMe				
ViT-H MAE $r7\searrow$	224	86.1	72.6	81
ViT-H MAE $r7\rightarrow$	224	**86.5**	92.9	63
ViT-H MAE	224	86.9†	167.4	35
SwinV2-H*	224	85.7	118.1	49

기존 SOTA 모델과 비교 (ImageNet-1k)

Token Merging

- Experimental results
  - Token merging visualization 결과를 확인해보면 이미지의 유사한 부분을 merging한 결과를 확인할 수 있음



Conclusion

- Token을 줄이려고 할 때 prune할 것인지, merging할 것인지에 대한 연구

- Token을 merging한다면 어떤 방법을 사용할 것인지
  - Inference시에만 적용 가능한지 또는 재학습이 필요한지 대한 아이디어 및 실험