And the bit goes down: Revisiting the Quantization of Neural Network

UST-ETRI
석사2학기 김형민
2021.06.04

Published as a conference paper at ICLR 2020

AND THE BIT GOES DOWN: REVISITING THE QUANTIZATION OF NEURAL NETWORKS

Pierre Stock[1,2], Armand Joulin[1], Rémi Gribonval[2], Benjamin Graham[1], Hervé Jégou[1]
[1]Facebook AI Research, [2]Univ Rennes, Inria, CNRS, IRISA

ICLR 2020
기관 : Facebook AI Research, University Rennes
인용 : 43회

Contents

- Model Compression
- Quantization
- Method
- Experimental Results
- Conclusion

Model Compression

실제 서비스에서는 다양한 환경에서 ML이 돌아가고 있다.

실제 제품화시 사용되는
임베디드 보드/FPGA
(AI 로봇청소기, AI 스피커 등)

현재 가장 사용자에게 친숙한
모바일 디바이스

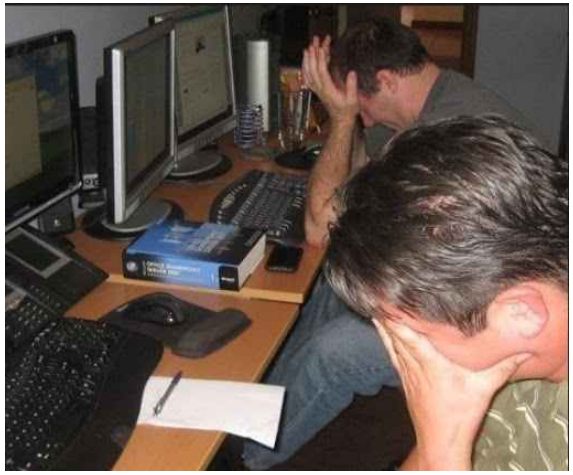
Edge / Mobile 디바이스의 낮은 성능

1. 모바일 디바이스의 성능이 좋아지고 있지만 PC의 GPU에 비할 수 없다.
2. 단가절감을 위해 FPGA/임베디드 보드 선정 및 설계 시 넉넉한 성능상 마진을 줄 수 없다.
3. NN연산은 전력을 많이 소모하는데, 모바일 디바이스에서는 전력 사용양도 문제다.
4. 큰 모델 사이즈도 문제(Vision Transformer, BERT, ERNIE 등)

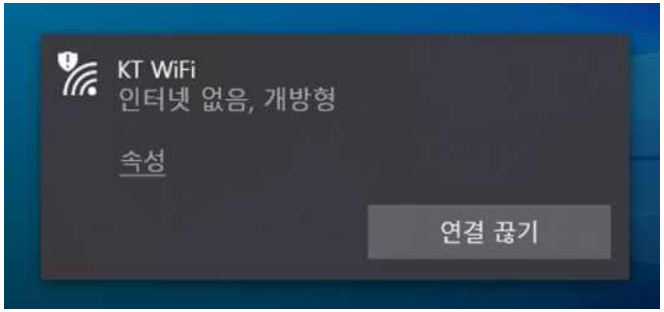
GPU 서버?

GPU 서버에서 API를 제공해주는 방법이 있지만...

개인정보 유출문제



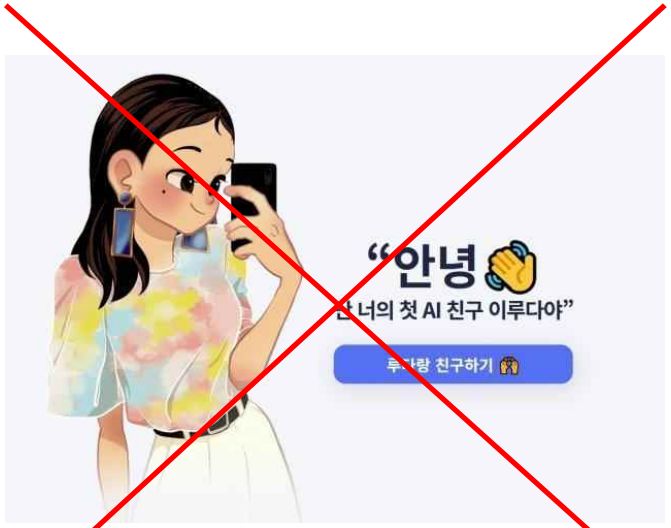
서버 관리 문제



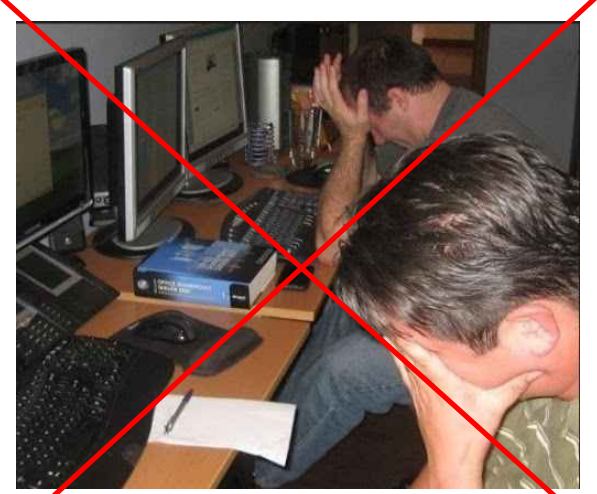
보안/기타등의 이슈로
인터넷 접속 불가

경량화

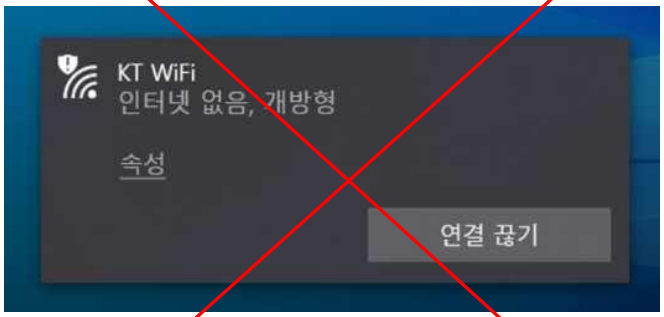
Edge/Mobile 디바이스에서 실시간 동작 가능할 정도로
모델을 경량화 한다면 모든 문제가 해결 된다.



개인정보 유출문제



서버 관리 문제



보안/기타등의 이슈로
인터넷 접속 불가

모델 경량화(Model Compression)

1. Low-precision training

2. Quantization

3. Pruning

4. Dedicated architectures

모델 경량화(Model Compression)

1. Low-precision training

   학습을 low precision으로 한다.

2. Quantization

   학습이 완료된 모델을 low precision으로 바꾼다.

3. Pruning

   기여가 적은 connection을 삭제한다.

4. Dedicated architectures

   경량화에 알맞은 네트워크 아키텍처를 설계한다.

모델 경량화(Model Compression)

1. Low-precision training

   학습을 low precision으로 한다.

2. Quantization

   학습이 완료된 모델을 low precision으로 바꾼다.

3. Pruning

   기여가 적은 connection을 삭제한다.

4. Dedicated architectures

   경량화에 알맞은 네트워크 아키텍처를 설계한다.

Quantization

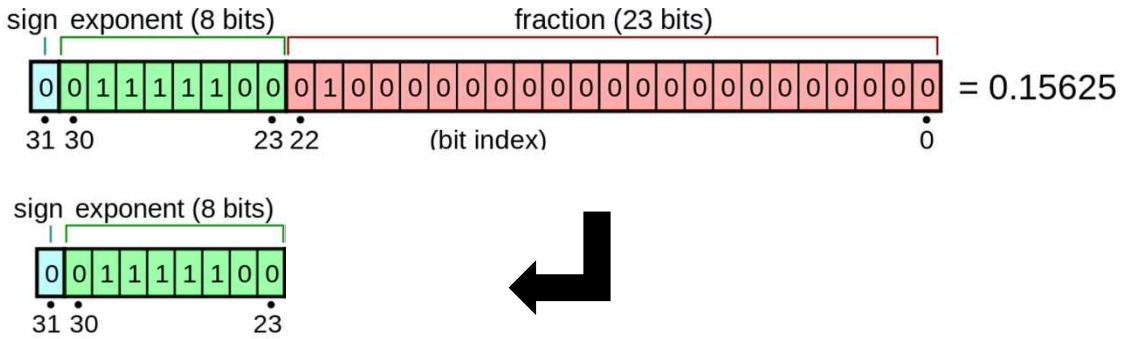
Quantization

데이터 타입을 더 덜 정확한 타입으로 바꾸다. (float → int)

[장점]

.

1. 연산 시 소모되는 메모리가 줄어든다.
2. 연산 속도가 빨라진다



[단점]

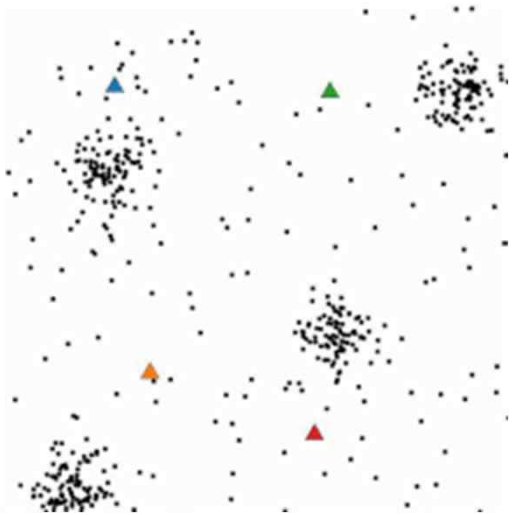
1. 모델의 추론 성능이 떨어질 수도 있다.

> 최대한 성능을 유지하면서, 모델 데이터 타입을 줄이는 것이 목표

Clustering

Quantization은 사실 Clustering 문제이다.

여러 데이터를 원본 데이터를 효율적으로 표현할 수 있는 더 작은 차원으로 사상하는 것.



K-means

- 기존 데이터들을 잘 표현할 수 있는 centroid를 찾는다.
- Floating Point를 int로…

Vector Quantization

Quantization은 사실 클러스터링 문제이다.

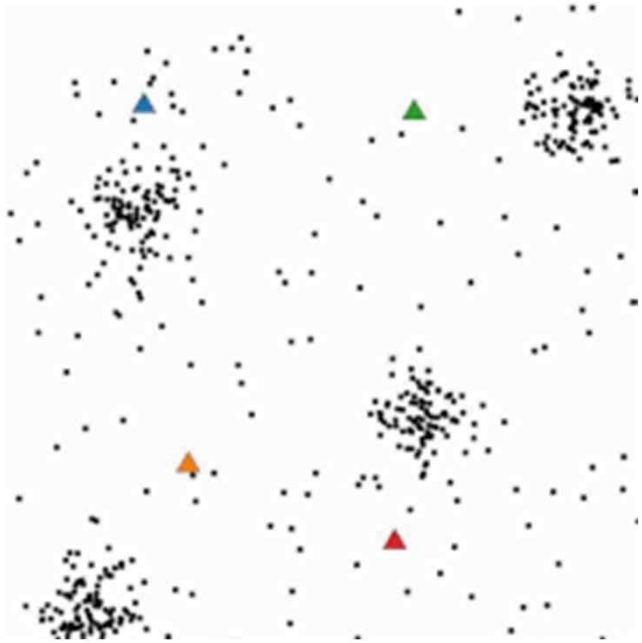
여러 데이터를 원본 데이터를 효율적으로 표현할 수 있는 더 작은 차원으로 사상하는 것.

D차원의 벡터를 k차원의 벡터로 매핑하는 함수 $q(x)$가 존재한다.

여기서 공간 C에는 총 k개의 centroid가 존재한다. $q(x) \in C$

$$C = c_i; i \in I,\ I = 0..k-1$$
$$V_i = x \in \mathbb{R}^D : q(x) = c_i$$

D차원의 데이터 V_i는 $Q(x)$에 입력되어 어떤 한 centroid C_i로 변경된다.



Objective of the Vector Quantization

K-means를 사용할 경우.

$\boldsymbol{\mu}_i$ 가 Clustering center(centroids)일 때

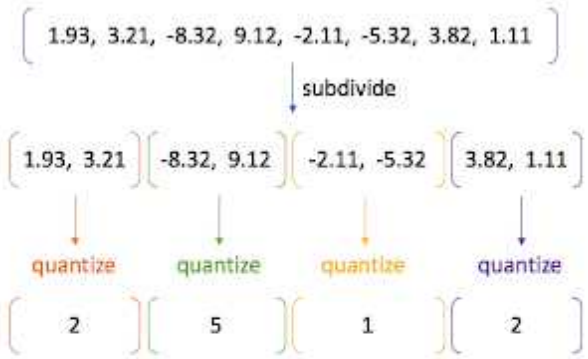
$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

각 클러스터 내 포인트들과 centroid간의 L2 distance 가 최소화 되는 방향으로 갱신.

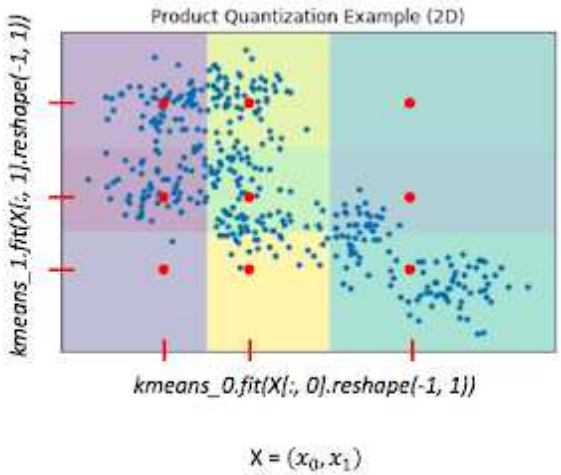
- 여기서 알 수 있는 것은, Vector Quantization은 기존 값들을 잘 압축할 수 있는 사상법을 찾는 과정임을 알 수 있다.

Product Quantization

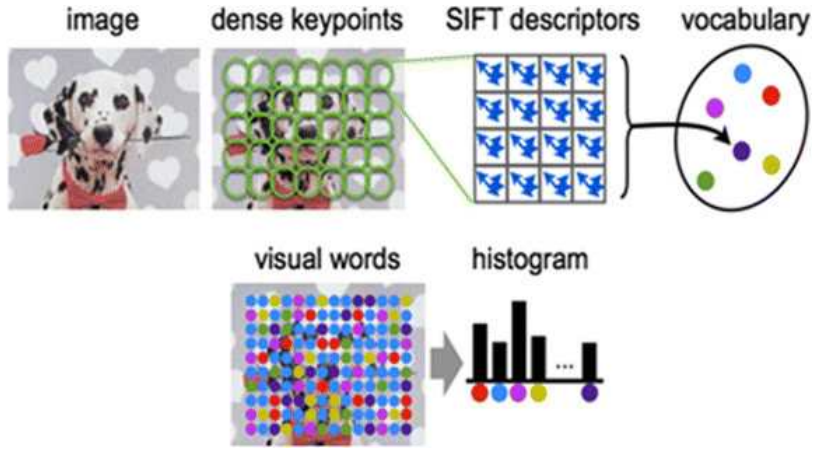
벡터 Euclidian distance 계산은 쿼리 벡터의 차원이 커질수록 오래 걸린다.
> 입력 벡터를 slicing하여 각각 quantization하고 이를 concat 한다.



8D example

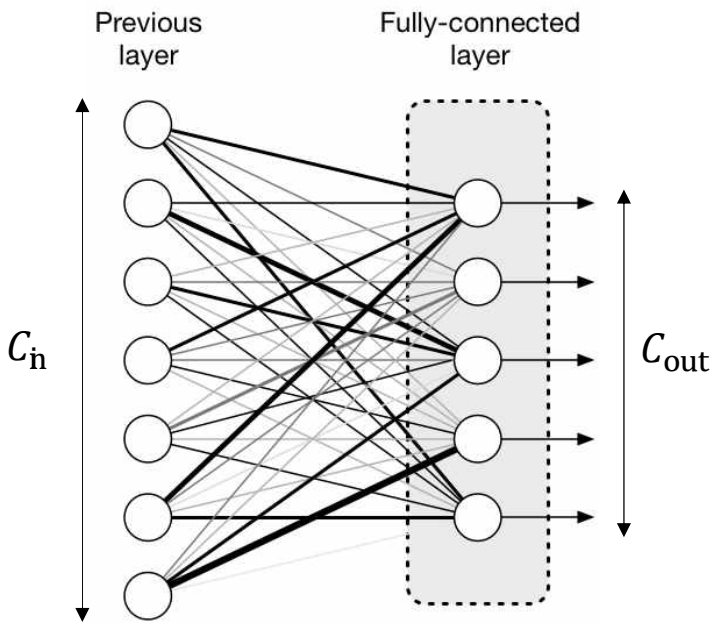


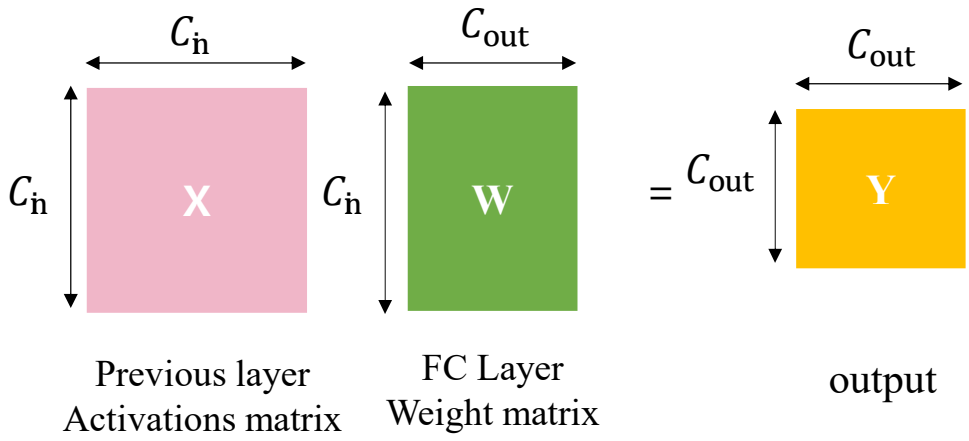
2D example (easier to visualize)



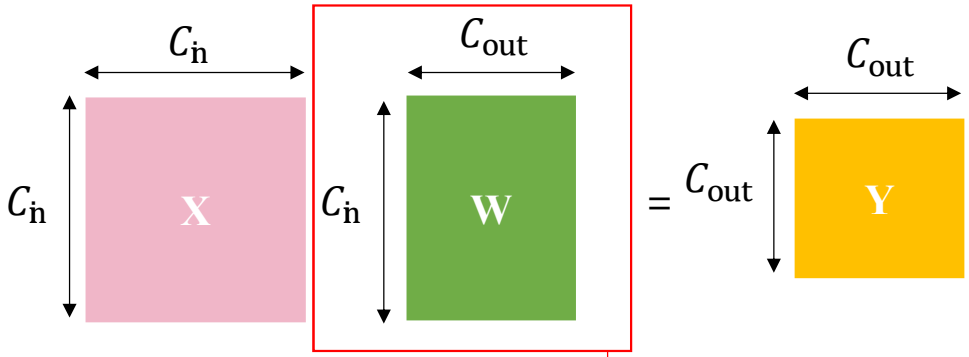
Dense SIFT 와 유사한 접근법

Fully Connected Layer PQ

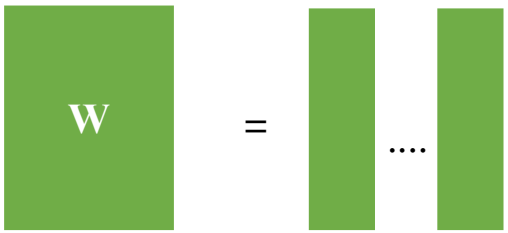




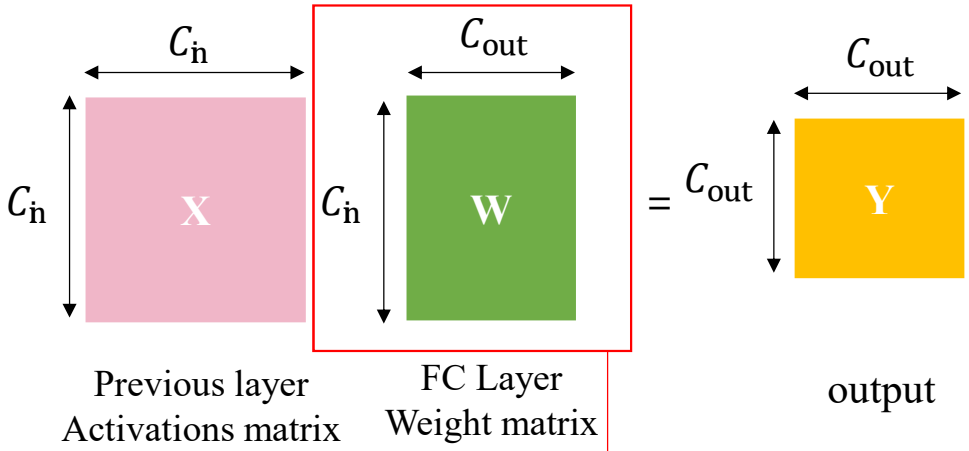
Fully Connected Layer PQ



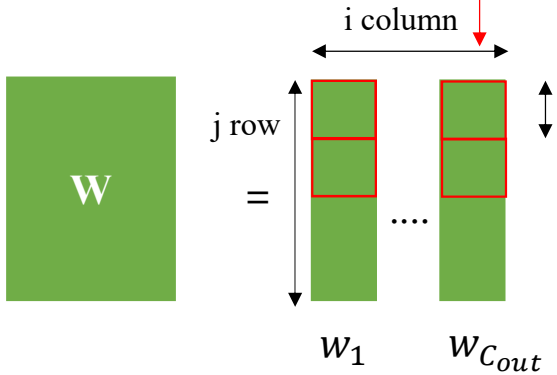
column vectors of W…



Fully Connected Layer PQ



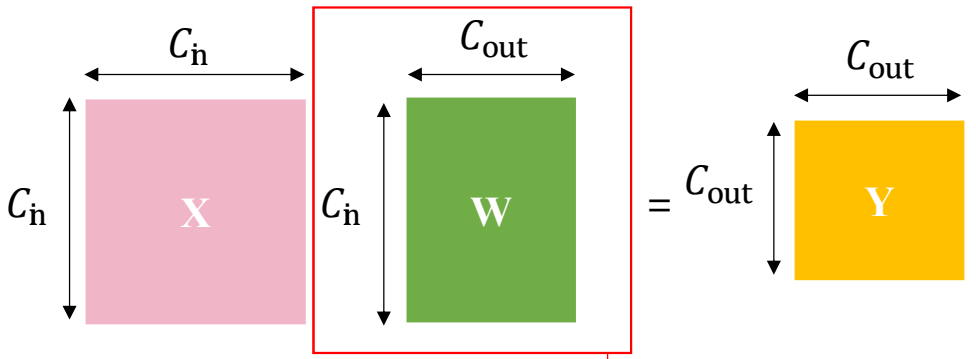
column vectors of W…



$$d = \frac{C_{in}}{m}$$

각 column vector들을 m개로 쪼갠다.
쪼개진 vector를 기준으로 clustering.

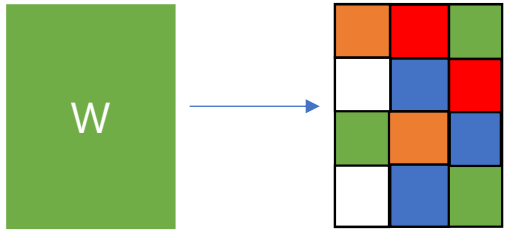
Fully Connected Layer PQ



$= C_{out}$ Y

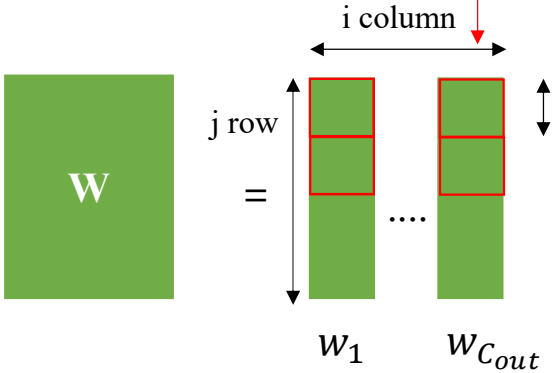
output

codebook $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$ contains k codewords of dimension d.



Subvector의 수는 mC_{out}

column vectors of W…

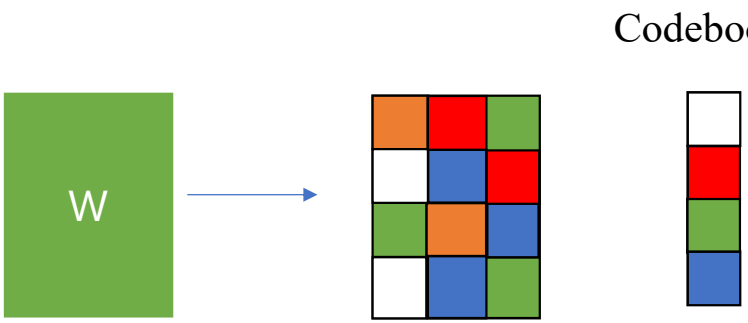


$d = \frac{C_{in}}{m}$

각 column vector들을 m개로 쪼갠다.
쪼개진 vector를 기준으로 clustering.

Codebook

Fully Connected Layer PQ : objective function



Subvector의 수는 mC_{out}

codebook $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$

Column $\mathbf{w}_j$ 에 대해서.

quantized version $\mathbf{q}(\mathbf{w}_j) = (\mathbf{c}_{i_1}, \ldots, \mathbf{c}_{i_m})$

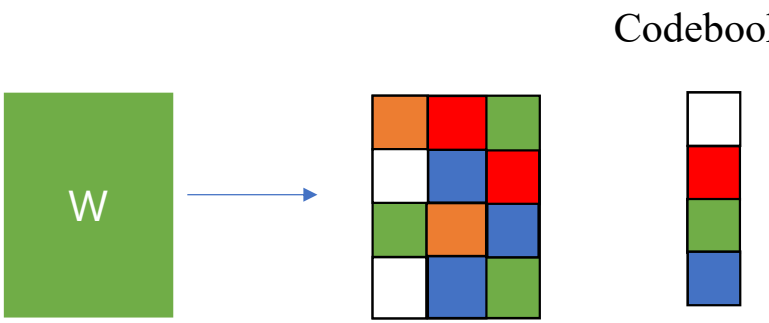
1~m번째 까지
Clustering된 centroid값

$$\|\mathbf{W} - \widehat{\mathbf{W}}\|_2^2 = \sum_{\cdot} \|\mathbf{w}_j - \mathbf{q}(\mathbf{w}_j)\|_2^2,$$

원래W와 Quantized Matrix $\widehat{W}$ 간의
Squared L2 norm 을 최소화 하는 방향으로 최적화.

즉 원래의 W를 최대로 반영하는 $\widehat{W}$ 가 되기 위한
최적화된 q(x) 매핑 함수를 찾는다.

Fully Connected Layer PQ



M=1 d = C_{in} 이면, PQ는 VQ와 같아진다.

M = C_{in} , d = 1 이면, k-means와 같아진다.

Subvector의 수는 mC_{out}

codebook $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$

Column $\mathbf{w}_j$ 에 대해서.

quantized version $\mathbf{q}(\mathbf{w}_j) = (\mathbf{c}_{i_1}, \ldots, \mathbf{c}_{i_m})$

1~m번째 까지
Clustering된 centroid값

Problem of PQ's Objective

$$\|\mathbf{W} - \widehat{\mathbf{W}}\|_2^2 = \sum_j \|\mathbf{w}_j - \mathbf{q}(\mathbf{w}_j)\|_2^2,$$

Quantized $\widehat{W}$와 Original W 와의 벡터간 거리를 줄이는 데만 집중.

실제 모델의 출력에 대해서는 고려하지 않는다.

오로지 W를 잘 재현하면 모델의 출력도 잘 재현될 것이라는 assumption 뿐.

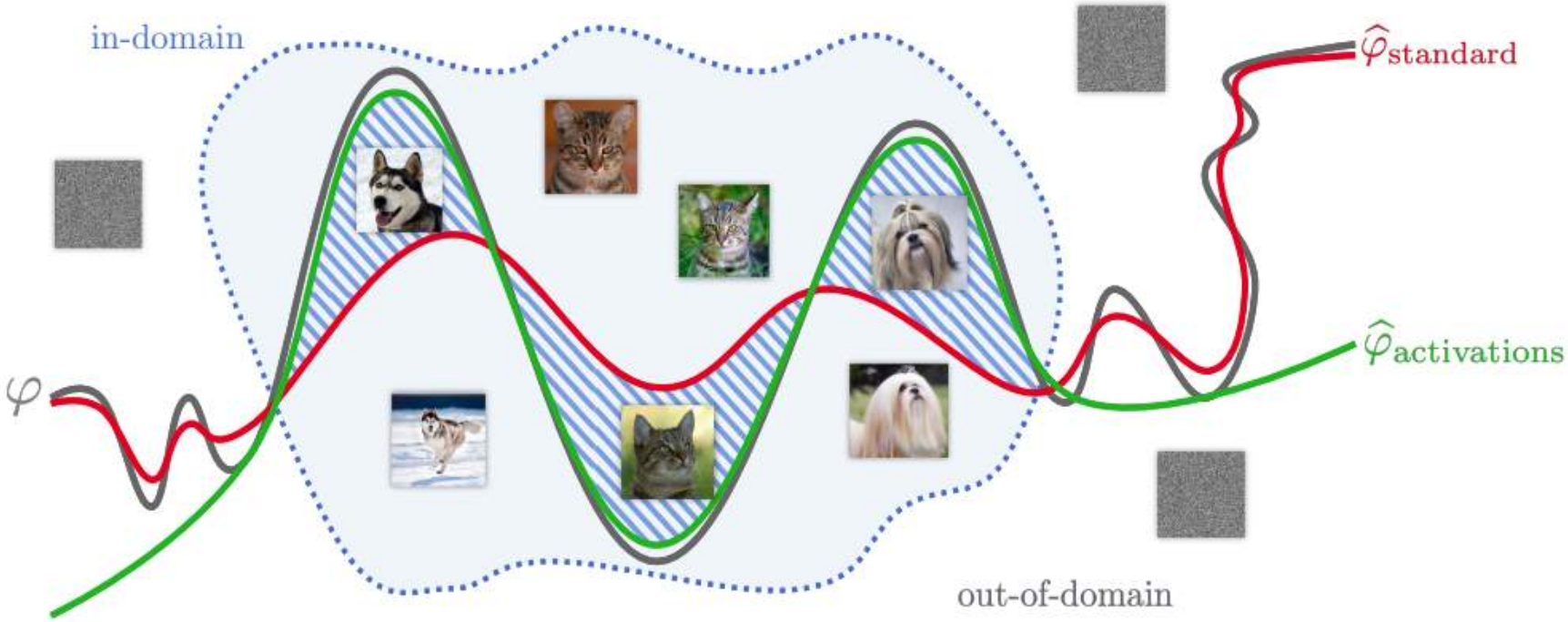
Problem of PQ's Objective

$$\|\mathbf{W} - \widehat{\mathbf{W}}\|_2^2 = \sum_j \|\mathbf{w}_j - \mathbf{q}(\mathbf{w}_j)\|_2^2,$$

Quantized $\widehat{W}$와 Original W 와의 벡터간 거리를 줄이는 데만 집중.

실제 모델의 출력에 대해서는 고려하지 않는다.

모델의 출력 값을 따라가게 Quantization을 하는 방법이 더 효율적이지 않을까?



회　색 : 원본 W
빨간색 : 기존 PQ의 approximation
초록색 : 제안된 방법의 approximation

φ: binary classifier

In-domain : 의미 있는 샘플들이 존재하는 공간
Out-of-domain : 의미 없는 샘플들이 존재하는 공간

- 거리 기반으로 최적화시 Out-of-domain까지 fitting
- 샘플에 대한 결과 기반으로 최적화시 in-domain에 대해서만 fitting

Proposed method

$$\|\mathbf{W} - \widehat{\mathbf{W}}\|_2^2 = \sum_j \|\mathbf{w}_j - \mathbf{q}(\mathbf{w}_j)\|_2^2, \qquad \mathbf{x} \in \mathbf{R}^{B \times C_{\text{in}}}$$

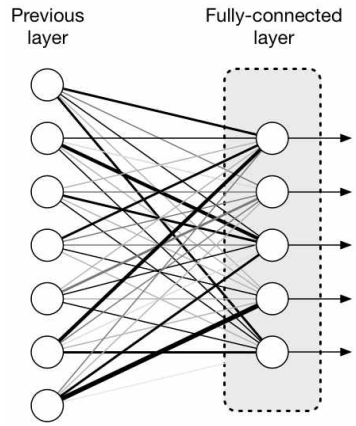
배치사이즈가B인 이전 활성화 값 (fc)의 입력

Weights를 reconstruction하는 방향이 아니라

출력 값을 reconstruction 하는 방향으로 최적화.

$$\|\mathbf{y} - \widehat{\mathbf{y}}\|_2^2 = \sum_j \|\mathbf{x}(\mathbf{w}_j - \mathbf{q}(\mathbf{w}_j))\|_2^2,$$

X*W



EM 알고리즘

$\widetilde{\mathbf{x}}$ 는 unroll된 X 즉 flatten 된 weight matrix

$\mathbf{x} \in \mathbf{R}^{B \times C_{\text{in}}}$ 배치사이즈가B인 이전 활성화 값
(fc)의 입력 : 벡터간 거리 연산을 위해..

Centroid를 최적화 하는 단계는 2가지.

1. 클러스터 중점들을 뿌린다. (cluster assignment) : E-step

$$\mathbf{c}_j = \underset{\mathbf{c} \in \mathcal{C}}{\operatorname{argmin}} \underbrace{\|\widetilde{\mathbf{x}}(\mathbf{c} - \mathbf{v})\|_2^2}_{y - \hat{y}\text{와 같음}}.$$

각 sub-vector $\mathbf{v}$ 는 codeword $\mathbf{c}_j$ 로 할당됨
> 모든 codeword 중 가장 가까운 것으로 할당.

> 뭔 말인가 하면, 각 클러스터별로 원본 W와 quantized W의 출력 값의 차이가 가장 적은 c로 sub vector 할당.

2. 클러스터 중점들을 움직인다. (codeword update) : M-step

$$\mathbf{c}^\star = \underset{\mathbf{c} \in \mathbf{R}^d}{\operatorname{argmin}} \sum_{p \in I_{\mathbf{c}}} \underbrace{\|\widetilde{\mathbf{x}}(\mathbf{c} - \mathbf{v}_p)\|_2^2}_{y - \hat{y}\text{와 같음}}.$$

할당된 codeword $\mathbf{c} \in \mathcal{C}$.
C에 속한 벡터들 $(\mathbf{v}_p)_{p \in I_{\mathbf{c}}}$

Then, we update $\mathbf{c} \leftarrow \mathbf{c}^\star$,

> 뭔 말인가 하면, 각 클러스터별로 원본 W와 quantized W의 출력 값의 차이를 최소화 하는 방향으로 c 이동

EM 알고리즘

(1) Target W에서 uniformly sampling k vectors > k 개 클러스터 center 초기화.

(2) E-step (sub vector 들에게 가장 가까운 cluster center 할당)

(3) 아무 벡터도 할당되지 못한 클러스터 존재 시

(3-1) 가장 많이 할당된 codeword $\mathbf{c}_0$ 를 찾고
(3-2) $\mathbf{c}_0$ 랜덤 노이즈를 더하고 빼서 클러스터를 2개로 쪼갠다..

$$\mathbf{c}_0' = \mathbf{c}_0 + \mathbf{e} \qquad \mathbf{c}_i' = \mathbf{c}_0 - \mathbf{e}$$

$$\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \varepsilon\mathbf{I})$$

$$\varepsilon = 1\mathrm{e}{-8}$$

정규분포를 기준으로
매우 적은 값으로 scaling된
Noise

(3-3) back to (2) [E-step] 반복

Proposed method in FC Quantization 정리

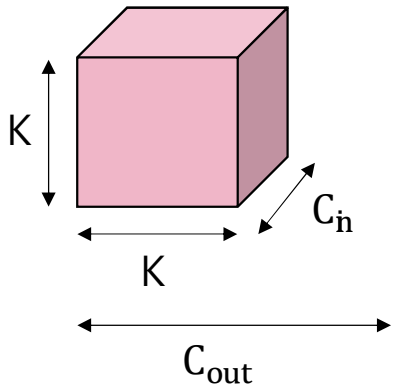
(1) 입력 벡터들을 잘 압축하는 클러스터 중점인 codeword C를 학습하는 것이 목표이다.

(2) 클러스터 중점은 각 iteration마다 업데이트 된다.

(3) 이때 입력 벡터와의 차이를 줄이는 방향이 아닌,
출력 값을 재현하는 방향으로 클러스터 중점 업데이트 된다.

Proposed method in Convolution layer

Visualize convolution weights



$R^{C_{out} \times C_{in} \times K \times K}$

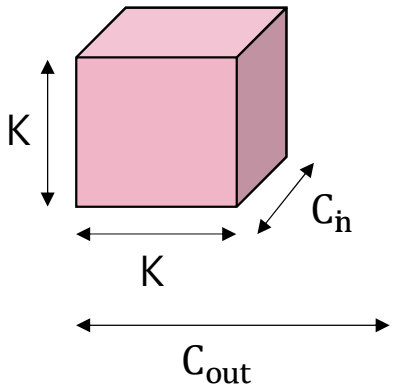
출력 피처맵의 채널 : C_{out}
입력 피처맵의 채널 : C_{in}
커널 사이즈 : K

```
input tensor torch.Size([1, 10, 224, 224])
after conv torch.Size([1, 20, 224, 224])
odict_keys(['conv.weight', 'conv.bias'])
torch.Size([20, 10, 3, 3])
```

```python
import torch


class Conv_layer(torch.nn.Module):

  def __init__(self):
    super().__init__()
    self.conv = torch.nn.Conv2d(
        in_channels = 10,
        out_channels = 20,
        kernel_size = 3,
        stride = 1,
        padding = 1
    )

  def forward(self, x):
    print("input tensor", x.shape)
    x = self.conv(x)
    print("after conv", x.shape)

net = Conv_layer()
dummy_input = torch.rand([1, 10, 224, 224])
output = net(dummy_input)

weight = net.state_dict()
print(weight.keys())

conv_weight = weight['conv.weight']
print(conv_weight.shape)
```

Proposed method in Convolution layer

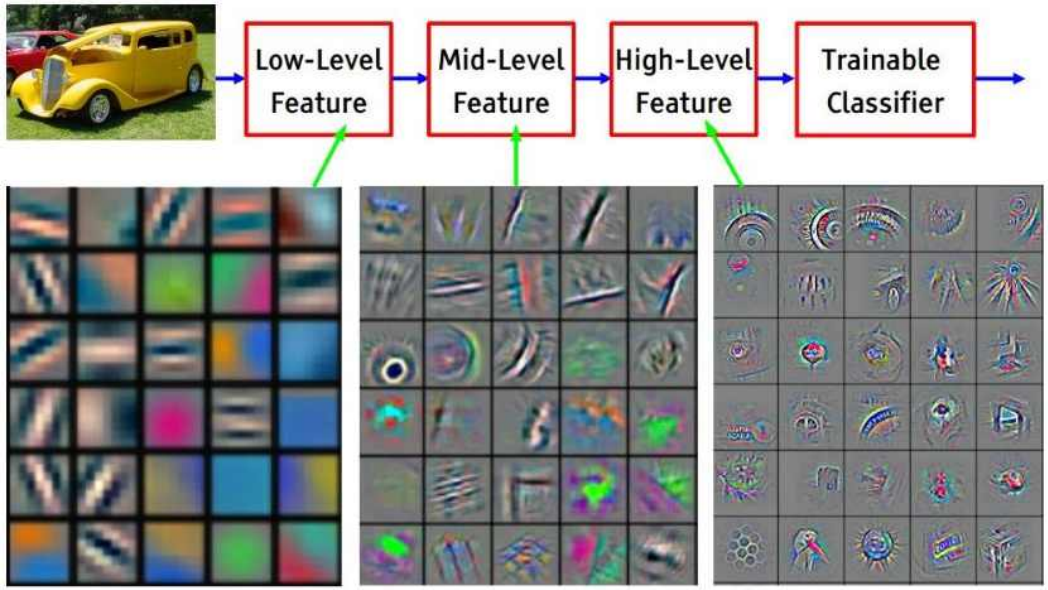
Visualize convolution weights



$R^{C_{out} \times C_{in} \times K \times K}$

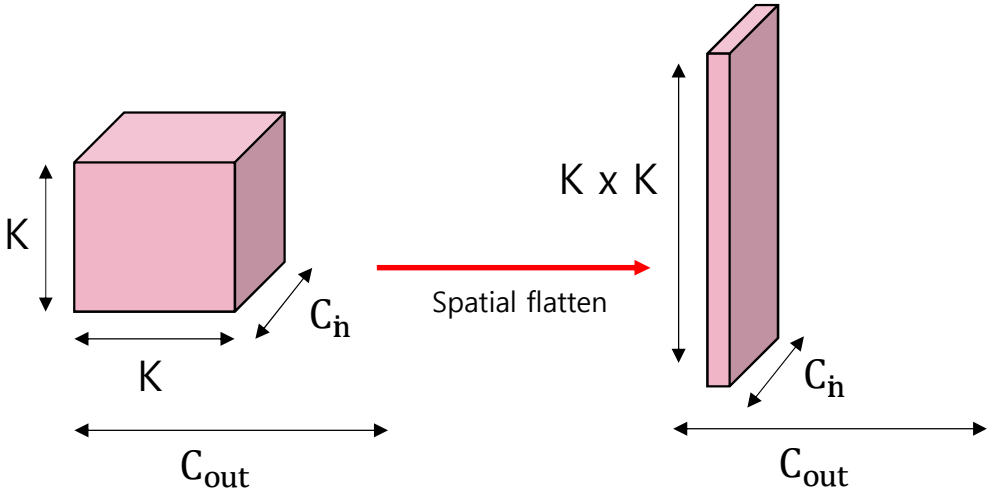
출력 피처맵의 채널 : C_{out}
입력 피처맵의 채널 : C_{in}
커널 사이즈 : K

Clustering을 그냥 하면 안되고
한 cluster안에는 무언가 correlation이 있어야한다.
하나의 커널 안에는 spatial-correlation 이 존재.
PQ의 Sub Vector를 spatial correlation을 고려해서 자르자!



Proposed method in Convolution layer

PQ를 하려면 Sub Vector 형태로 변형시켜야 한다.

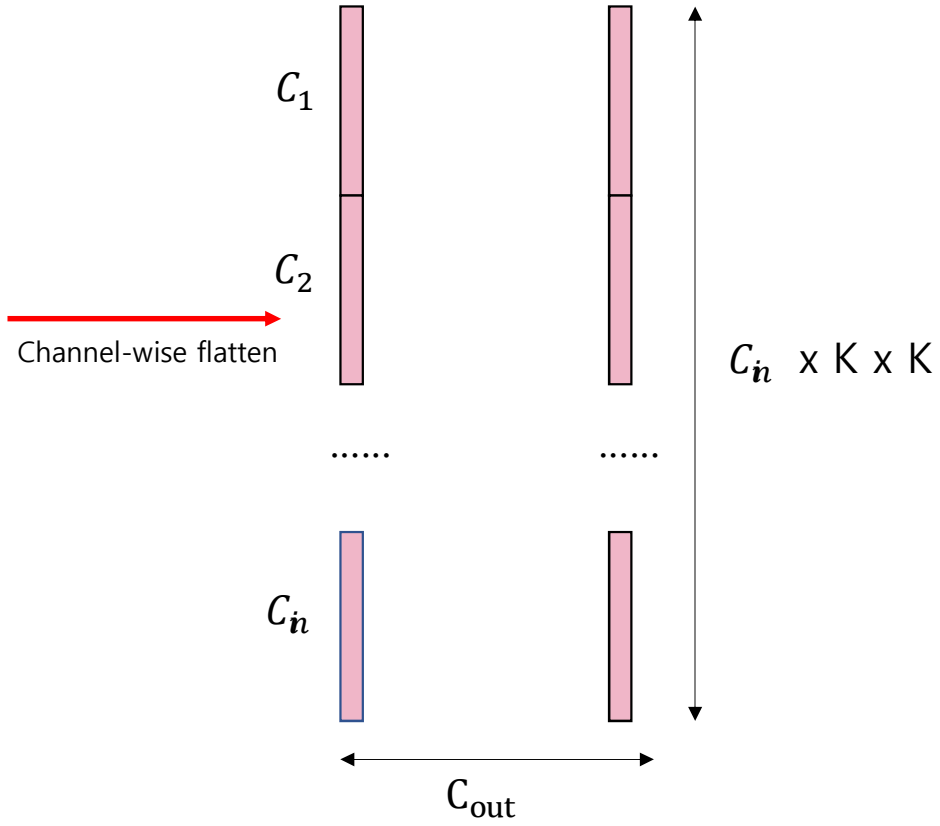


$R^{C_{out} \times C_{in} \times K \times K}$

$R^{C_{out} \times C_{in} \times (K \times K)}$

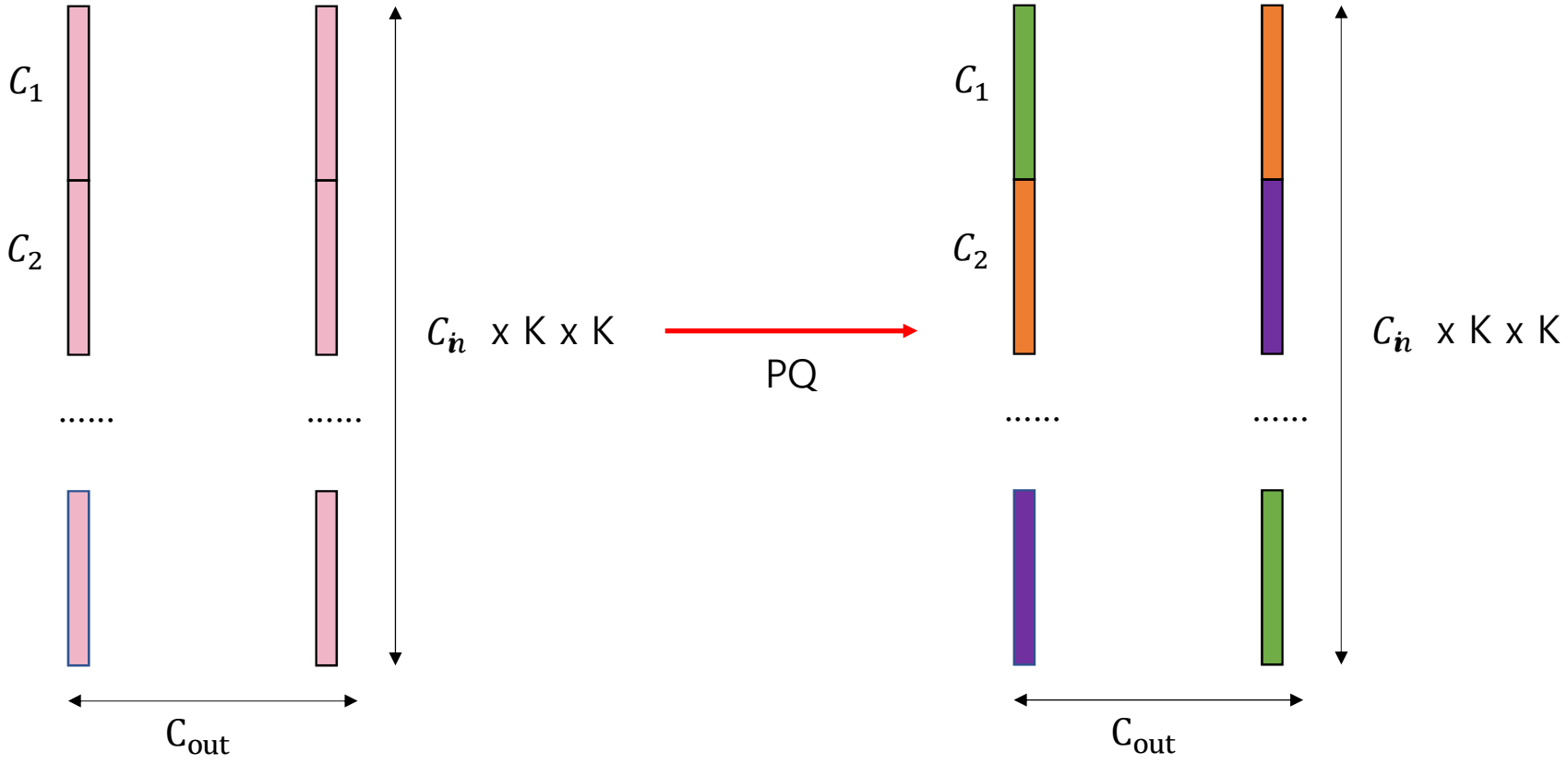
출력 피처맵의 채널 : C_{out}
입력 피처맵의 채널 : C_{in}
커널 사이즈 : K

각각의 sub-vector 안에는
하나의 필터에 대한 spatial 정보가 모두 들어있다.
Filter-wise quantization



Reshaped matrix W_r

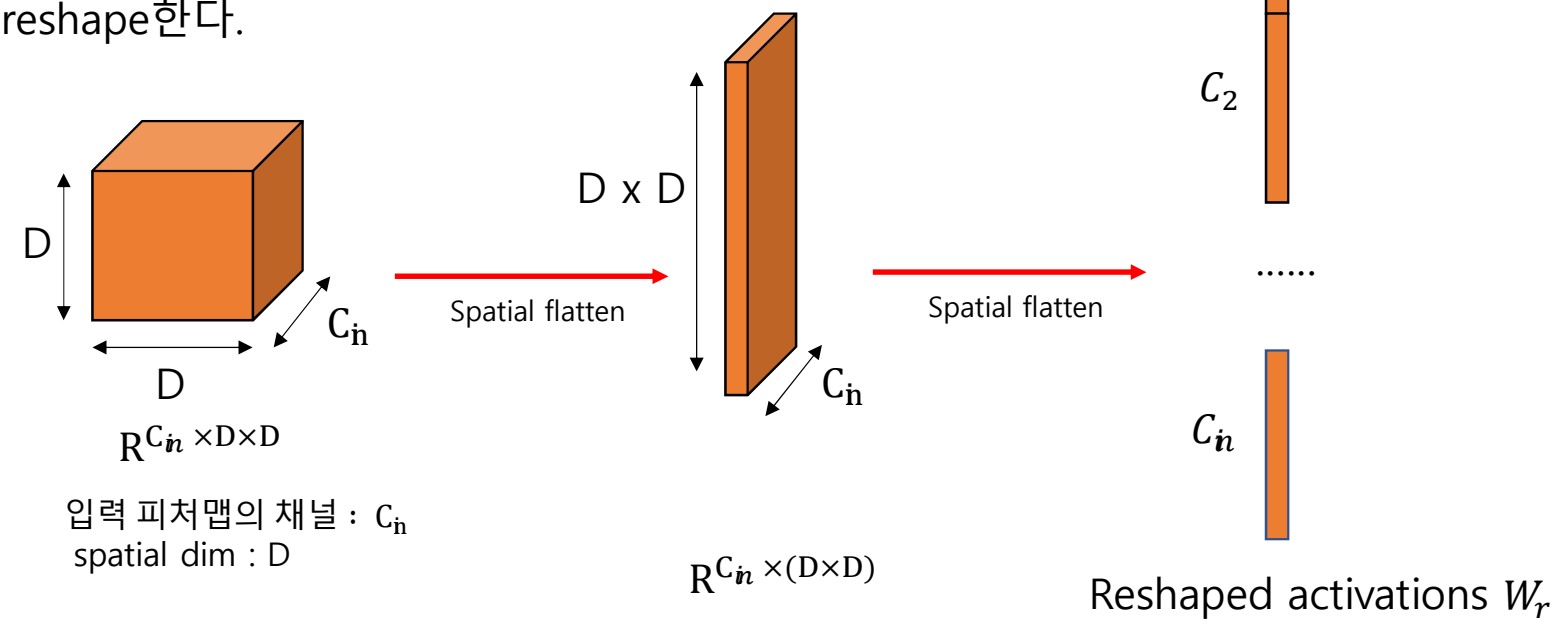
Proposed method in Convolution layer



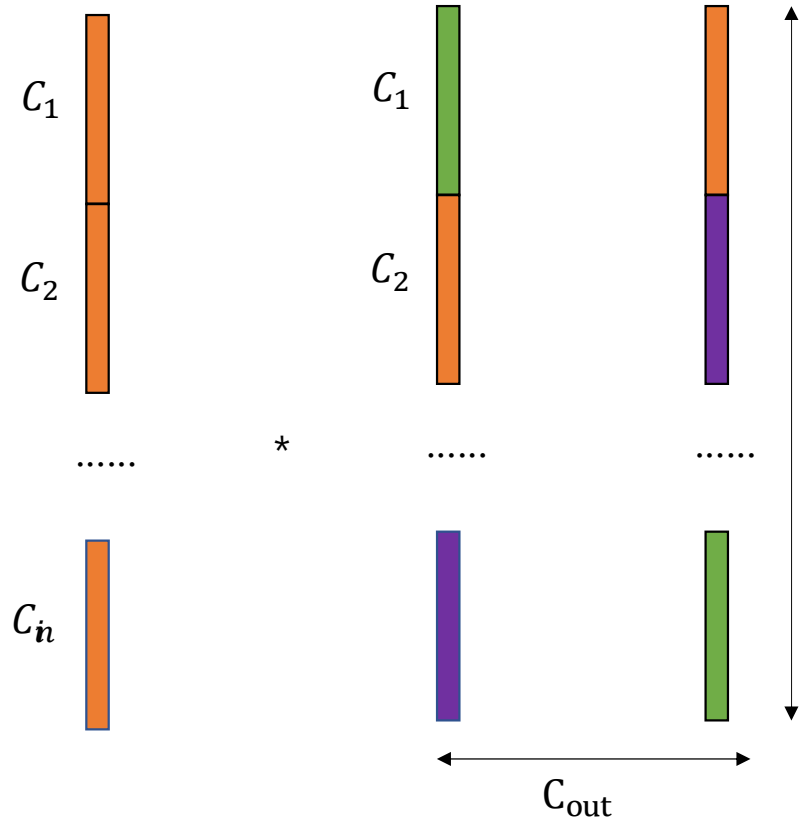
Proposed method in Convolution layer

출력을 비교해야 하기때문에 $x_r W_r$ 구할 수 있어야 한다.

W_r = Reshaped vector , x_r = Reshaped activations
입력도 reshape한다.



Proposed method in Convolution layer



C_{in} x K x K $= \hat{y} = X_r W_r$

최적화된 codebook 찾기는 이전과 동일.

Learning the codebook

1. 입력이랑 가장 가까운 레이어부터 연속적으로 Quantization.

2. Quantization 이후 codebook finetuning

   > 학습된 파라미터는 그대로 두고, BatchNorm의 mean/variance만 업데이트.

3. 원본 모델의 이전 layer의 output을 가지고 quantization하지 않고
   Quantization 된 이전 layer의 output을 이용해 quantization

   > 실험적으로, 성능이 더 잘 나왔던 방법.

Finetuning the codebook

Knowledge distillation 으로 codebook finetuning

Teacher : 원본 모델
Student : Quantized 모델

$$\mathcal{L} = \mathrm{KL}(\mathbf{y}_s, \mathbf{y}_t).$$

learning rate η

$$\mathbf{c} \leftarrow \mathbf{c} - \eta \frac{1}{|I_\mathbf{c}|} \sum_{p \in I_\mathbf{c}} \frac{\partial \mathcal{L}}{\partial \mathbf{b}_p}.$$

KL Divergence loss만 주기 때문에, label 필요 없음. 즉 아무 데이터나 넣어주어도 됨

레이블로 supervision 주었을 때보다도 더 좋은 결과를 얻음을 실험적으로 확인.
> 다량의 unannotated 데이터를 이용해 추가적인 성능향상이 가능할지도 모름.

Fine-tuning on the codewords is done
by averaging the gradients of each sub-vector assigned to a given codeword

IV. Experiments

Comparision with others

X축 : 용량
Y축 : top1 accuracy

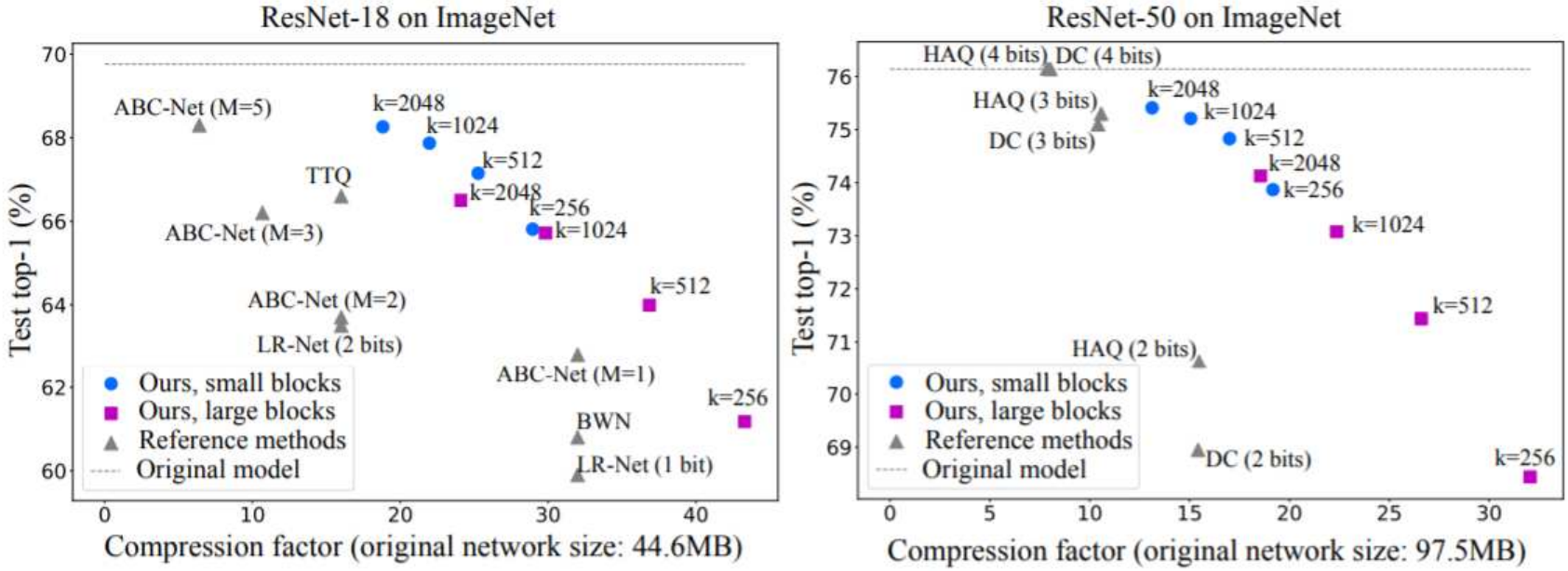
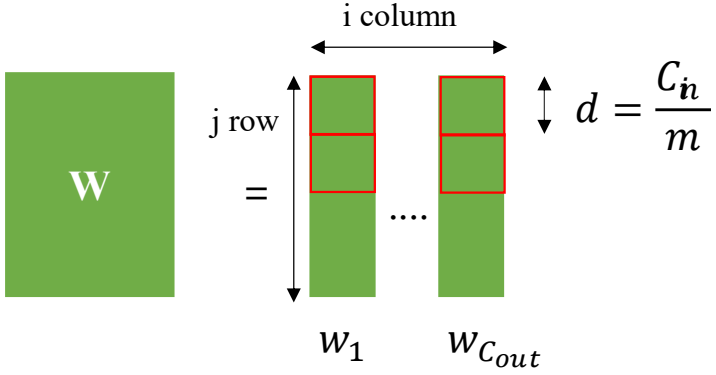


Figure 3: Compression results for ResNet-18 and ResNet-50 architectures. We explore two compression regimes as defined in Section 4.1: small block sizes (block sizes of $d=4$ and 9) and large block sizes (block sizes $d=8$ and 18). The results of our method for $k = 256$ centroids are of practical interest as they correspond to a byte-compatible compression scheme.

Results for ResNet-18 and ResNet-50

Table 1: Results for vanilla ResNet-18 and ResNet-50 architectures for $k = 256$ centroids.

Model (original top-1)	Compression	Size ratio	Model size	Top-1 (%)
ResNet-18 (69.76%)	Small blocks	29x	1.54 MB	**65.81** ±0.04
	Large blocks	43x	1.03 MB	**61.10** ±0.03
ResNet-50 (76.15%)	Small blocks	19x	5.09 MB	**73.79** ±0.05
	Large blocks	31x	3.19 MB	**68.21** ±0.04



쿼리벡터의 dim을 더 잘게 쪼갤 경우 : small blocks
> 더 많은 블록 사용 : 성능 좋음
쿼리벡터의 dim을 더 크게 쪼갤 경우 : large blocks
> 더 적은 블록 사용 : 성능 안 좋음

large block : d = 9
Small block : d = 18

Results on limited budget

SOTA와의 공정한 비교를 위해서 유사한 사이즈 시와 성능을 비교

Table 2: Best test top-1 accuracy on ImageNet for a given size budget (no architecture constraint).

Size budget	Best previous published method	Ours
~1 MB	**70.90%** (HAQ (Wang et al., 2018a), MobileNet v2)	64.01% (vanilla ResNet-18)
~5 MB	71.74% (HAQ (Wang et al., 2018a), MobileNet v1)	**76.12%** (semi-sup.ResNet-50)
~10 MB	75.30% (HAQ (Wang et al., 2018a), ResNet-50)	**77.85%** (semi-sup.ResNet-50)

Semi-supervised Resnet : YFCC-100M 데이터셋으로 finetuned 되었다.

Ablation study KD

Table 3: Ablation study on ResNet-18 (test top-1 accuracy on ImageNet).

Compression	Centroids k	No act + Distill	Act + Labels	**Act + Distill (ours)**
Small blocks	256	64.76	65.55	**65.81**
	512	66.31	66.82	**67.15**
	1024	67.28	67.53	**67.87**
	2048	67.88	67.99	**68.26**
Large blocks	256	60.46	61.01	**61.18**
	512	63.21	63.67	**63.99**
	1024	64.74	65.48	**65.72**
	2048	65.94	66.21	**66.50**

Centroid개수 일반 PQ + KD(label) 제안된 PQ + KD(label) **제안된 PQ + KL distill**

Mask R-CNN

Table 4: Compression results for Mask R-CNN (backbone ResNet-50 FPN) for $k = 256$ centroids (compression factor 26×).

Model	Size	Box AP	Mask AP
Non-compressed	170 MB	37.9	34.6
Compressed	6.65 MB	33.9	30.8

Conclusion

- 5Mb로 ImageNet에서 76.1%를 달성. (**레이블링된 데이터 없이**)

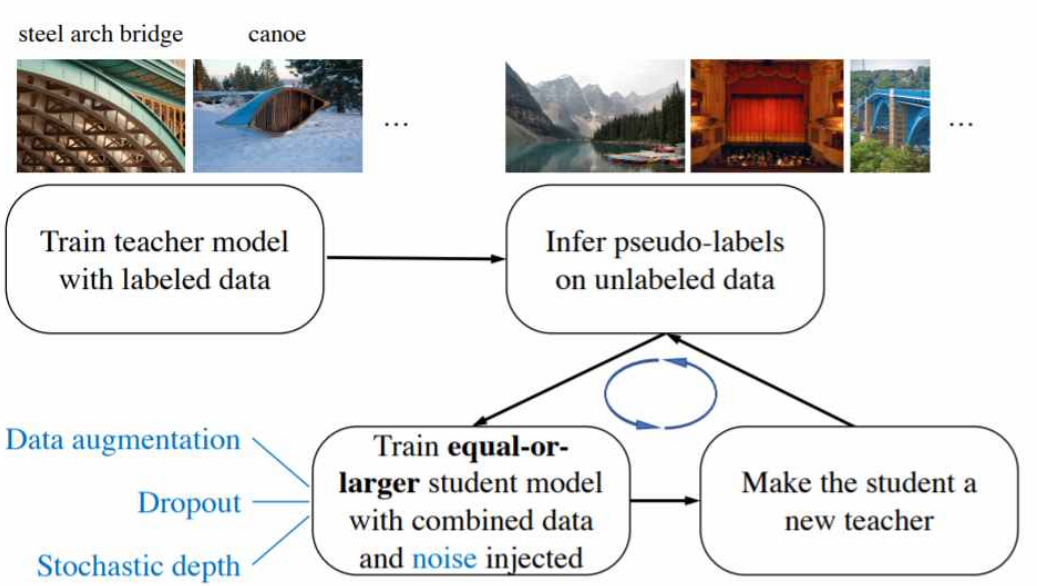
- 레이어의 weights를 재현하는 방향보다는 출력 값을 재현하는 방향으로 quantization하는 것이 좋다.

- 레이블 없이 pseudo softmax 와의 KL-divergence loss 만으로 finetuning 하는 것이 레이블보다 더 좋은 성능을 달성.

- 실제 활용 면에서는 Semi-Supervised 방식이 효과적임을 시사.

- Notation에 대한 생략이 있어 읽기 힘들었다....

supplementary



(1) 이미지넷으로 Teacher 학습

(2) Unlabeled 이미지의 pseudo label을 Teacher로 생성

(3) 이미지넷 + Inlabeled 이미지로 Student 학습
학습시 Noise 추가

(4) Student를 Teacher로 하여 반복

> 2020 이미지넷 SOTA 달성

아마 레이블을 사용하지 않았을 때 노이즈로
작용하여 성능이 올랐지 않았을 지.....

Q. Xie et al. "Self-Training With Noisy Student Improves ImageNet Classification", CVPR 2020

감사합니다

맛있는 점심 식사 되세요.