FEDGKT

GROUP KNOWLEDGE TRANSFER: FEDERATED LEARNING OF LARGE CNNS AT THE EDGE

REFERENCE

- He, Chaoyang, Murali Annavaram, and Salman Avestimehr.
  "Group Knowledge Transfer: Federated Learning of Large CNNs at the Edge."
  Advances in Neural Information Processing Systems 33 (2020).

ABSTRACT

ABSTRACT

- CNN의 크기를 키우는 것이
  - 모델 정확도를 유효하게 향상시키는 것으로 알려져 있음
  - 폭, 깊이 등
- 그러나 큰 모델 크기는 학습이 힘듦
  - 자원이 제한된 엣지 디바이스(edge device)에서는 더욱 그러함
  - 연합 학습(FL)에서는 더욱 그러할 것

ABSTRACT

- 이러한 문제를 다루기 위해 FL을 재설계함
  - FedGKT (Federated Group Knowledge Transfer)
  - 그룹 지식 전이 학습 알고리즘

ABSTRACT

- FedGKT
  - 엣지 노드에서는 작은 CNN을 학습
  - 주기적으로 지식을 지식 증류(knowledge distillation)를 통해
    - 큰 서버 CNN으로 전이
- 엣지에서의 연산을 줄이고
- 커뮤니케이션 비용이 낮고
- 비동기 학습이 가능
- 정확도는 FedAvg와 유사하면서도

ABSTRACT

- CNN을 ResNet-56과 ResNet-110을 이용해
  - CIFAR-10, CIFAR-100, CINIC-10 데이터셋으로 학습
  - Non-IID하게
- 그 결과 FedGKT가 FedAvg 대비
  - 비슷하거나 심지어 조금 더 높은 정확도를 보임
- 무엇보다도 엣지 학습이 가능하다는 점이 중요
  - FedAvg 대비 9-17배 정도 연산력을 덜 사용하고
  - 엣지 CNN에서 54-105배 정도 적은 수의 파라미터만을 요구

INTRODUCTION

INTRODUCTION

- CNN의 크기를 키우면 정확도가 오름
  - 그러나 큰 CNN의 학습은
  - 자원이 제한된 엣지 디바이스에게는 도전적
    - 스마트폰
    - IoT 디바이스
    - 엣지 서버 등

INTRODUCTION

- FL에 대한 관심이 올라가며
  - 엣지 기반 학습의 요구가 증가
- FL은 분산 학습 패러다임
  - 글로벌 모델을 여러 엣지 디바이스에서
  - 중앙화된 데이터셋 필요 없이 학습

INTRODUCTION

- FedAvg와 같은 FL 방법론은
  - 로컬 SGD와 모델 평균을 통해
  - 커뮤니케이션 빈도를 줄였지만
  - 작은 크기의 CNN에서만 수렴성 평가가 진행되거나
  - 클라이언트가 큰 CNN을 학습할 수 있는 충분한 연산력이 있다 가정함
  - 이는 현실성이 떨어짐

INTRODUCTION

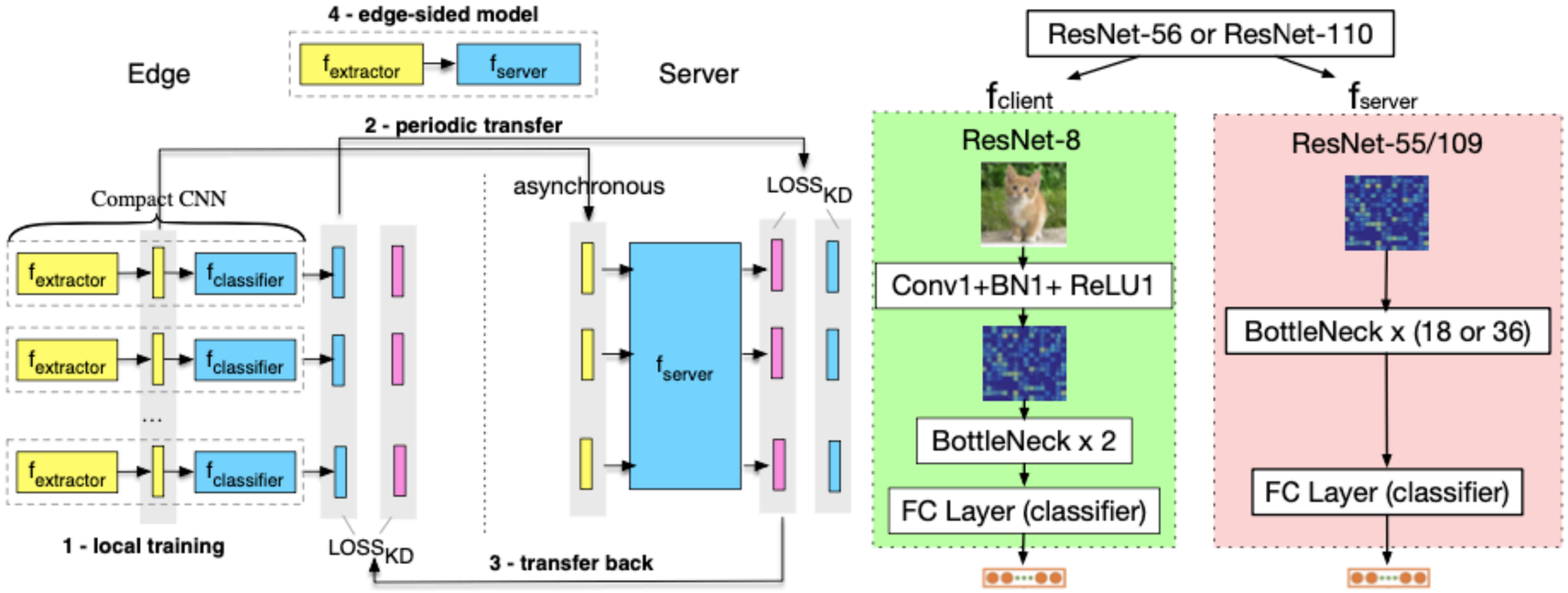
- 엣지 노드의 연산 한계를 다루기 위해
- 모델 병렬화 기반 **분할 학습(Split Learning, SL)**이 제안됨
  - 큰 모델을 파티셔닝하고
  - 일부 신경망 구조를 클라우드에 오프로딩
- 그러나 SL은
  - 싱글 미니배치 반복이
  - 서버와 엣지 간 여러 통신을 요구한다는 단점이 있음

INTRODUCTION

- FedGKT
  - 자원이 제한된 엣지 디바이스를 위한
  - 효율적인 FL 프레임워크
- FedAvg와 SL의 강점을 더하고자 함
  - FedAvg의 로컬 SGD를 사용한 학습
  - SL의 엣지에서의 낮은 연산력 요소

INTRODUCTION

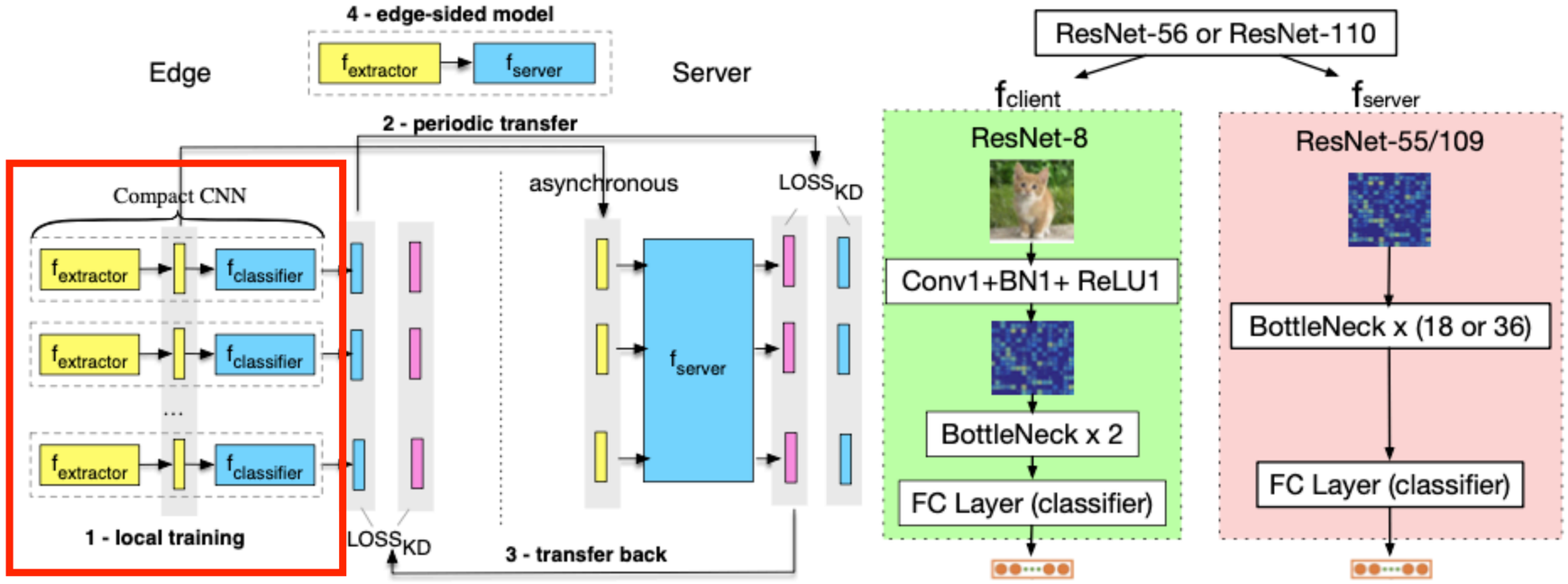
- 많은 수의 엣지에서 학습한 작은 CNN으로부터
- 클라우드 서버에서 학습하는 큰 CNN으로 지식을 전이



(a) Alternating and periodical knowledge transfer (b) CNN architectures on the edge and server

INTRODUCTION

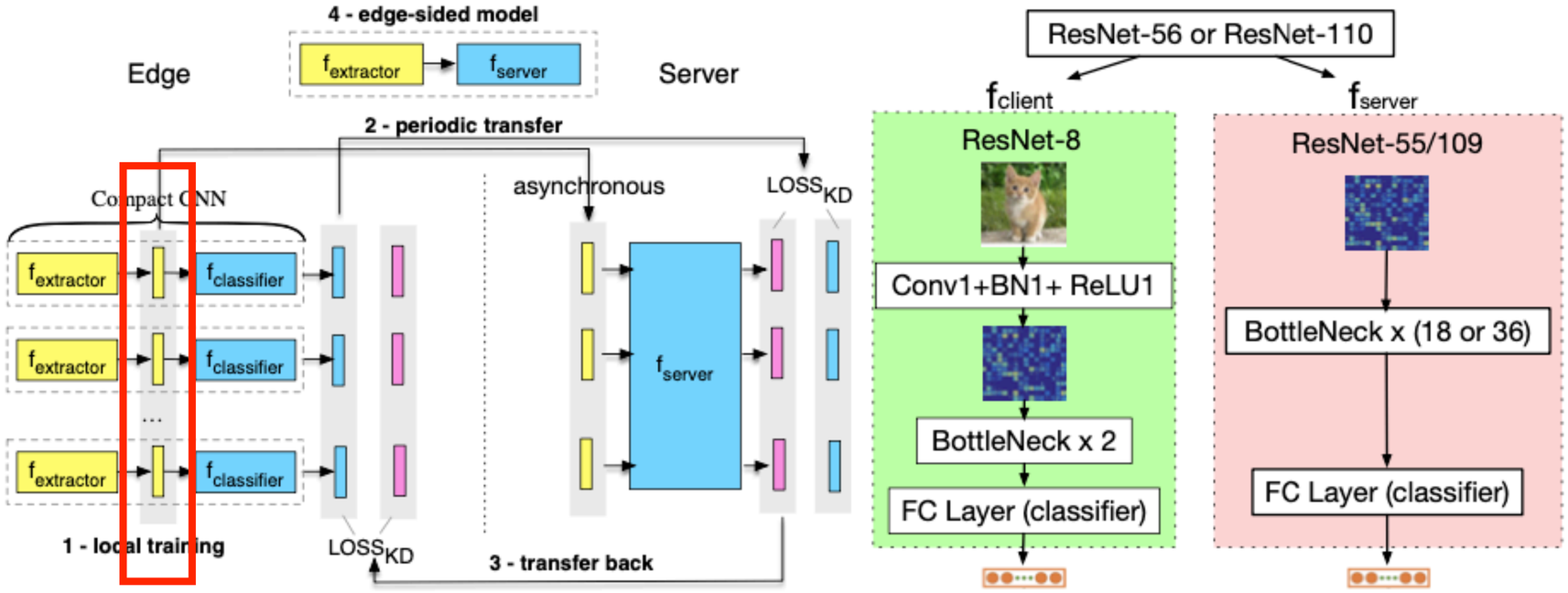
- 1. 로컬 학습
  - 프라이빗 데이터를 통해 엣지 디바이스가 작은 CNN을 학습 (특징 추출기 및 분류기)



(a) Alternating and periodical knowledge transfer (b) CNN architectures on the edge and server

INTRODUCTION

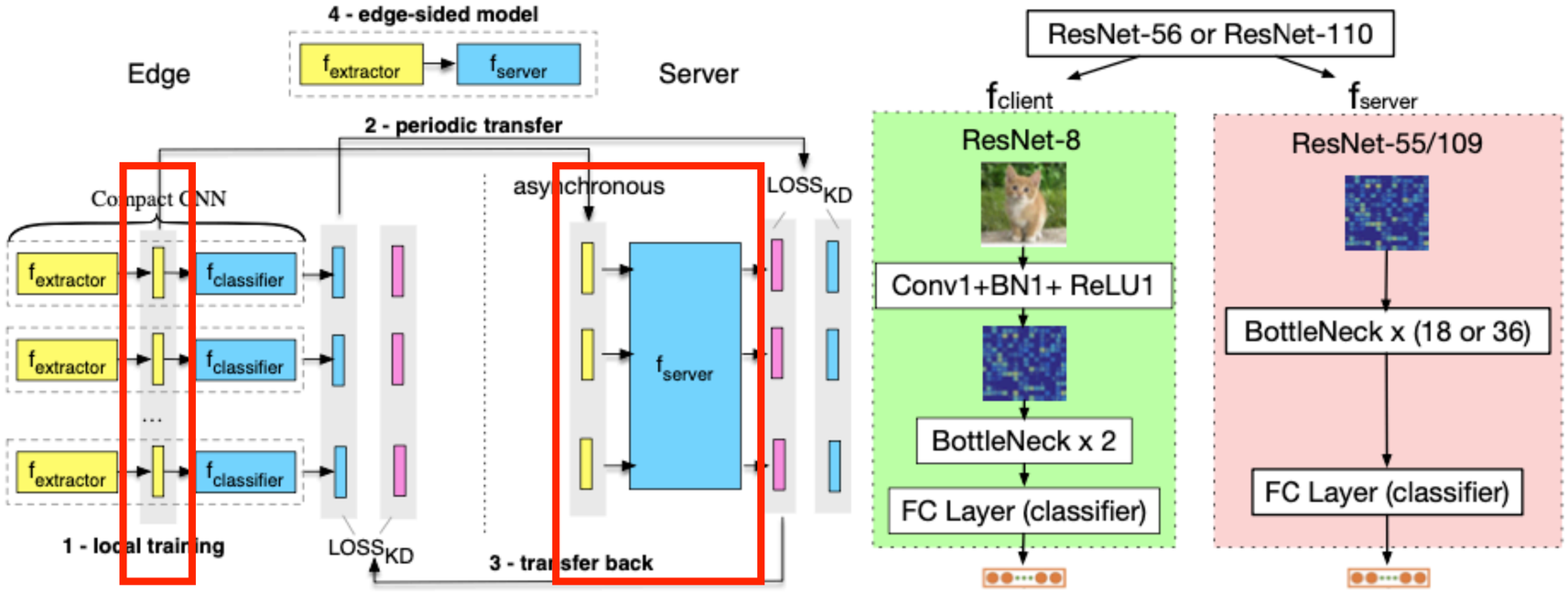
- 모든 엣지 노드가 특징 추출기로부터 정확히 같은 텐서 차원의 출력을 생성



(a) Alternating and periodical knowledge transfer (b) CNN architectures on the edge and server

INTRODUCTION

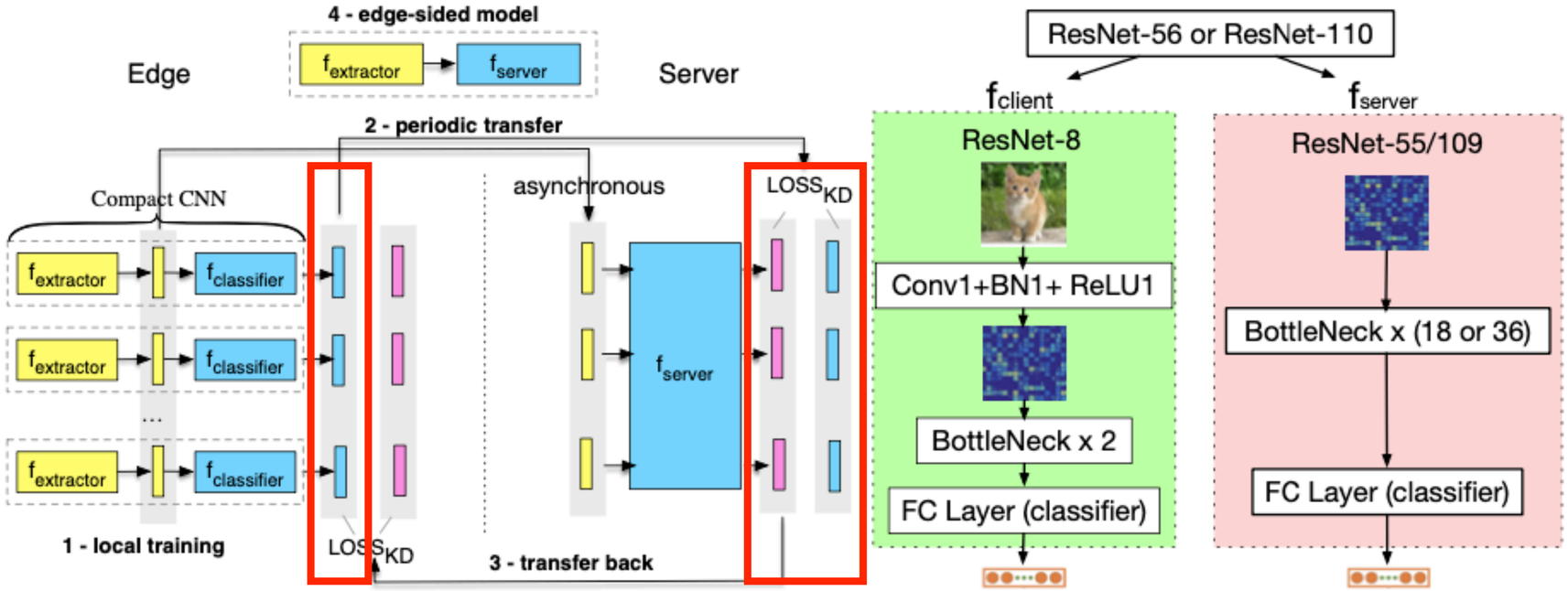
- 2. 주기적 전이
  - 큰 서버 모델은 특징들을 취해 학습



(a) Alternating and periodical knowledge transfer (b) CNN architectures on the edge and server

INTRODUCTION

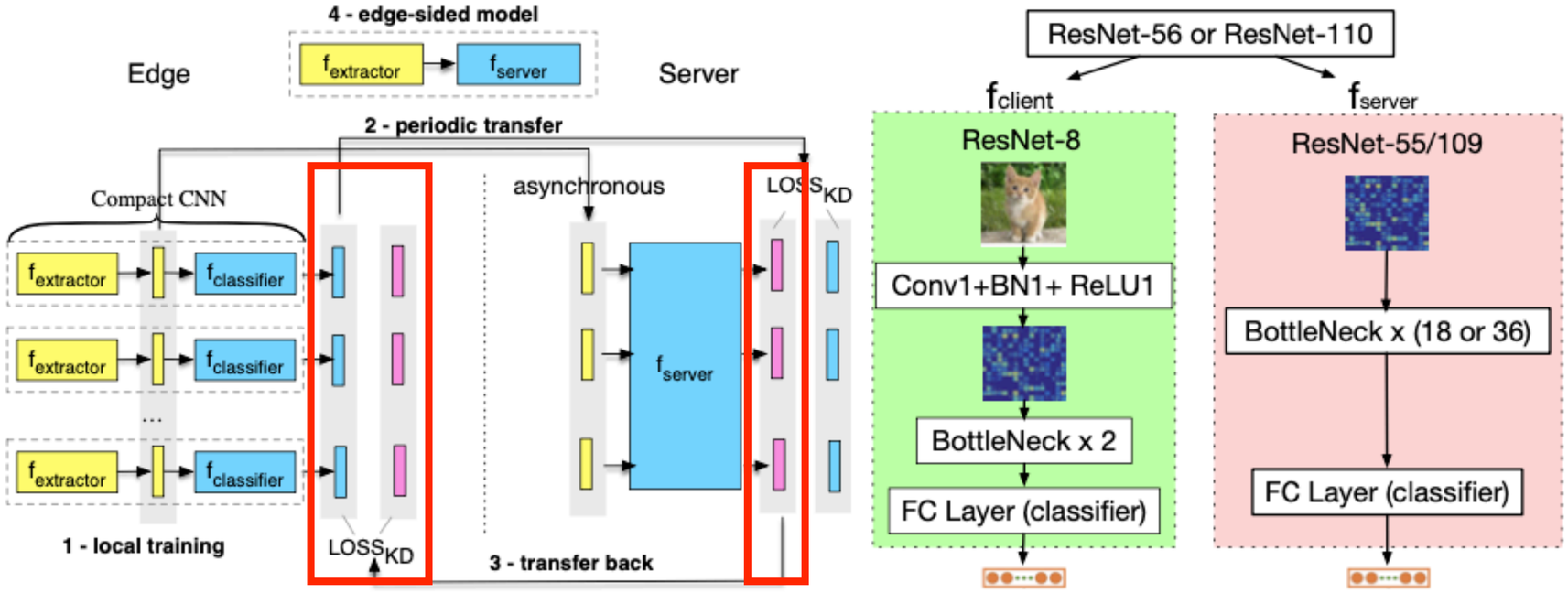
- KD 기반 손실 함수를 사용, 진짜 레이블과 엣지가 예측한 소프트 레이블 간의 간극을 최소화함
  - 서버 모델의 학습에 엣지 모델의 출력을 사용한다는 의미



(a) Alternating and periodical knowledge transfer (b) CNN architectures on the edge and server

INTRODUCTION

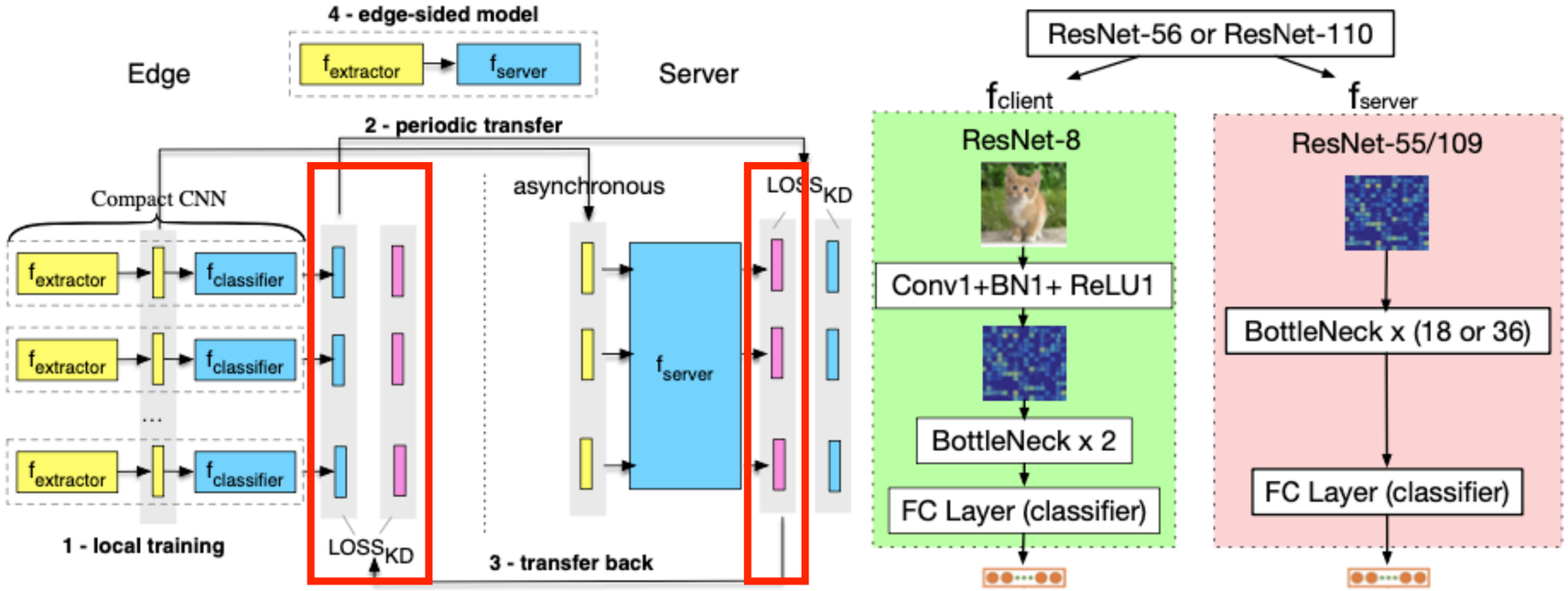
- 3. 역(Back) 전이
  - 엣지 모델의 성능을 향상시키기 위해, 서버가 소프트 레이블을 엣지에게 전송



(a) Alternating and periodical knowledge transfer (b) CNN architectures on the edge and server

INTRODUCTION

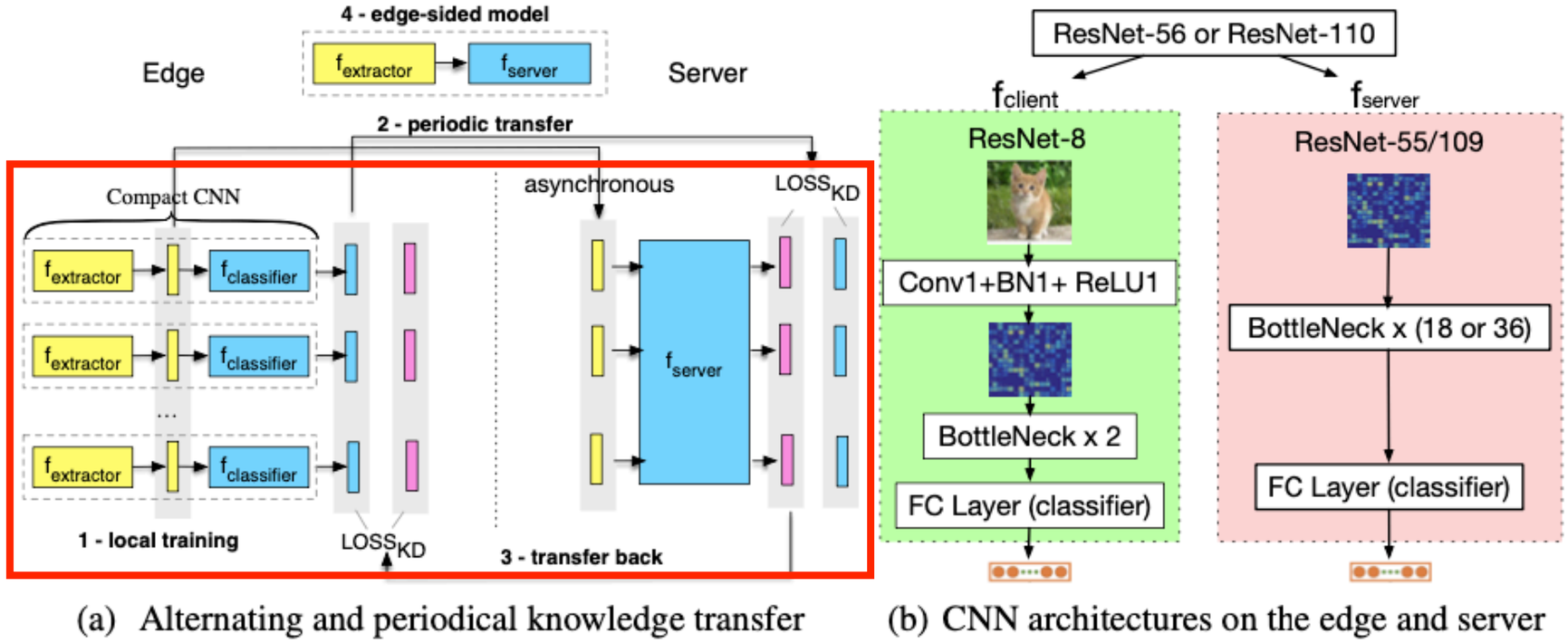
- 서버의 소프트 레이블을 이용해, 엣지가 로컬 데이터셋으로 KD-기반 손실 함수를 이용해 학습
  - 엣지 모델의 학습에 서버 모델의 출력을 사용한다는 의미



(a) Alternating and periodical knowledge transfer (b) CNN architectures on the edge and server

INTRODUCTION

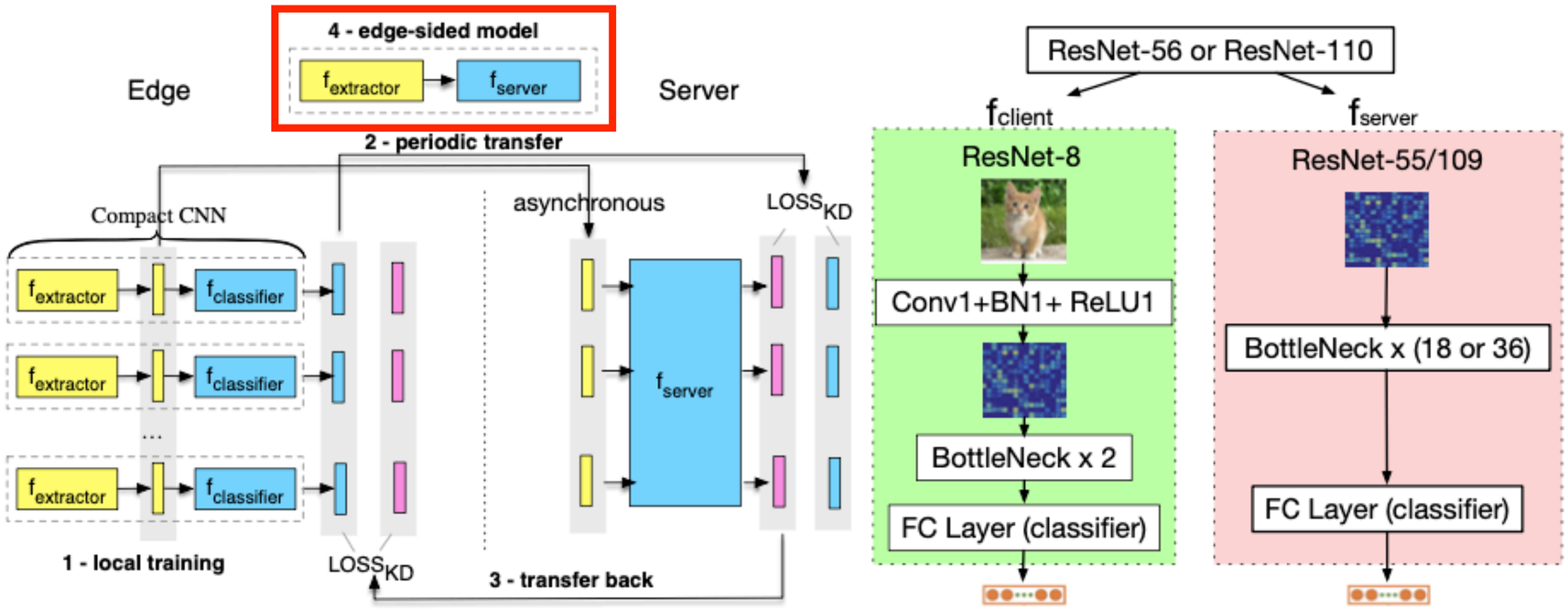
- 서버와 엣지는 서로 지식 전이를 이용해 성능을 향상시키는 것임



(a) Alternating and periodical knowledge transfer (b) CNN architectures on the edge and server

INTRODUCTION

- 4. 엣지 모델
  - **학습이 완료되면** 최종 모델을 형성: 로컬 특징 추출기와 공유된 서버 모델의 결합



(a) Alternating and periodical knowledge transfer (b) CNN architectures on the edge and server

INTRODUCTION

- 주요 트레이드-오프는
  - FedGKT가 연산을
  - 엣지 디바이스에서
  - 강력한 서버로 옮겼다는 것

INTRODUCTION

- FedGKT는 여러 강점을 하나의 프레임워크로 병합함
  - 1. SL과 유사하게 메모리 및 연산 효율적
  - 2. FedAvg와 유사하게 로컬 SGD의 학습이 가능
    - 통신 횟수의 절감
  - 3. FedAvg에서는 전체 모델을 전송해야 했으나
    - SL과 유사하게 특징(hidden feature)만 전송하면 됨
    - 대역폭의 절감

INTRODUCTION

- FedGKT는 여러 강점을 하나의 프레임워크로 병합함
  - 4. FedGKT는 본질적으로 비동기 학습을 지원
    - SL의 여러 동기화 문제를 해결
    - 서버 모델은 임의 클라이언트로부터 입력을 수신하면
    - 언제든 즉시 학습을 시작할 수 있음

GROUP KNOWLEDGE TRANSFER

GROUP KNOWLEDGE TRANSFER

- 큰 CNN을
  - 여러 자원이 제한된 디바이스에서 학습하고자 함
    - GPU 가속에 적합하지 않은
  - 각 디바이스의 데이터셋을 서버에 전송하지 않고서

GROUP KNOWLEDGE TRANSFER

- 지도 학습을 고려할 것
  - C 카테고리가 존재
  - 전체 데이터셋 D
- K 명의 클라이언트(엣지 디바이스)를 가정
  - k-th 노드는 고유의 데이터셋 $D^k := \{(X_i^k, y_i)_{i=1}^{N^{(k)}}\}$을 가짐
    - $y_i \in \{1,2,...,C\}$
    - $N^{(k)}$: 데이터셋 D^k의 개수

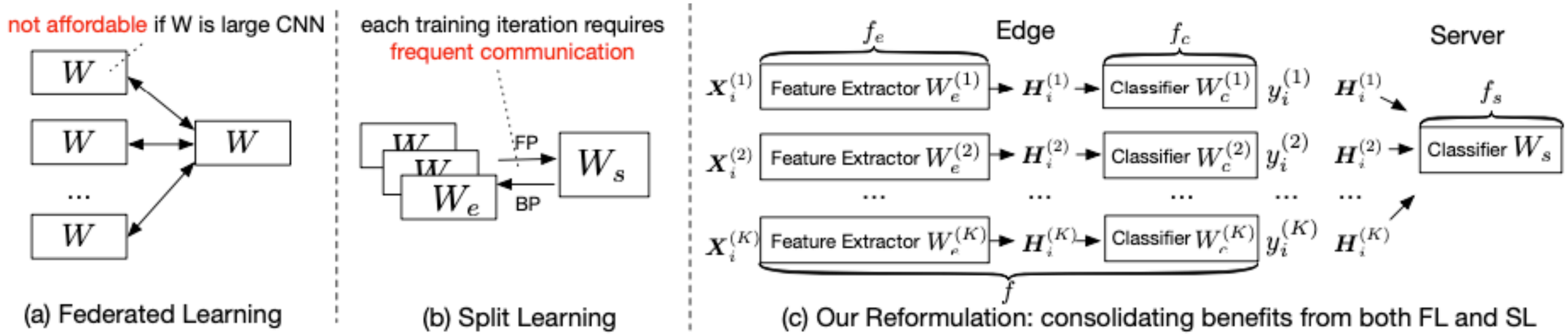
GROUP KNOWLEDGE TRANSFER

- CNN-기반 연합 학습은
  - 분산 최적화 문제로 공식화할 수 있음

$$\min_{\boldsymbol{W}} F(\boldsymbol{W}) \stackrel{\text{def}}{=} \min_{\boldsymbol{W}} \sum_{k=1}^{K} \frac{N^{(k)}}{N} \cdot f^{(k)}(\boldsymbol{W}), \text{where } f^{(k)}(\boldsymbol{W}) = \frac{1}{N^{(k)}} \sum_{i=1}^{N^{(k)}} \ell(\boldsymbol{W}; \boldsymbol{X}_i, y_i)$$

  - W : 글로벌 CNN의 가중치
  - $f^{(k)}(W)$: k 클라이언트의 로컬 목적 함수
  - l : 글로벌 CNN의 손실 함수

GROUP KNOWLEDGE TRANSFER



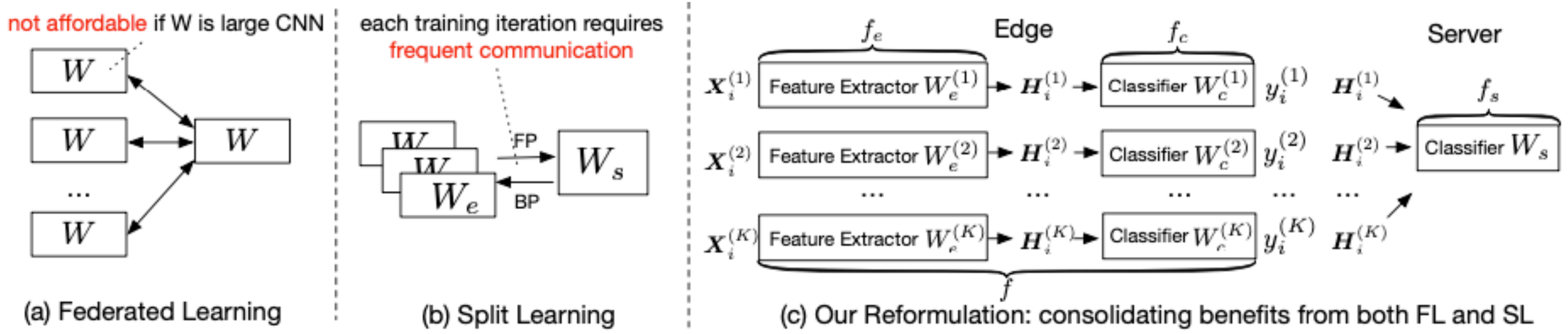
- 그러나 이 방법(a)은 자원이 제한적인 엣지 디바이스에서는 어려운 일
- 모델 병렬화 기반 분할 학습(b)
  - W를 분할함으로써 연산 문제를 다루고자 함
  - 신경망의 큰 부분을 서버에 오프로딩

GROUP KNOWLEDGE TRANSFER

- 그러나 SL에서는
  - 한 번의 미니배치 반복을 위해
  - 원격 forward 전파와 back 전파가 필요함
- 엣지 컴퓨팅에서 이러한 높은 빈도의 동기화 메커니즘은 문제가 됨
  - 현저히 프로세스의 속도를 늦춤

GROUP KNOWLEDGE TRANSFER

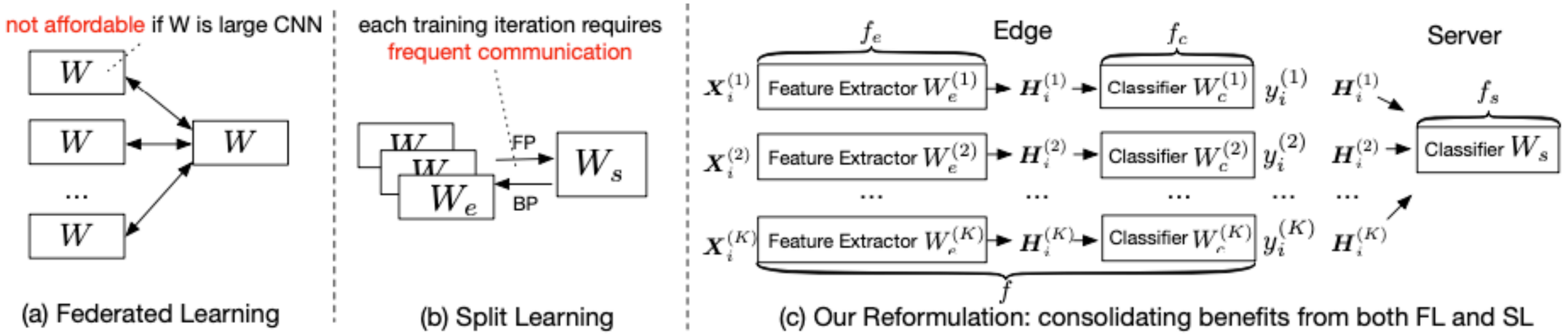
- FL과 SL을 재공식화하자!



- 그림(c)에서처럼 글로벌 CNN W를 두 부분으로 나누자
  - 작은 특징 추출기 W_e
  - 큰 서버측 분류기 W_s

GROUP KNOWLEDGE TRANSFER

- FL과 SL을 재공식화하자!



- W_e를 위한 분류기 W_c도 존재
  - 엣지 측에서 전체 분류(신경망 전체 프로세스)가 가능하도록 하는 분류기
  - 작은 크기

GROUP KNOWLEDGE TRANSFER

- 그룹 지식 전이
- Z
  - 마지막 전연결층 출력
  - 지식

Algorithm 1 Group Knowledge Transfer. The subscript s and k stands for the server and the kth edge, respectively. E is the number of *local* epochs, T is the number of communication rounds; η is the learning rate; $\boldsymbol{X}^{(k)}$ represents input images at edge k; $\boldsymbol{H}^{(k)}$ is the extracted feature map from $\boldsymbol{X}^{(k)}$; $\boldsymbol{Z}_s$ and $\boldsymbol{Z}_c^{(k)}$ are the logit tensor from the client and the server, respectively.

ServerExecute():
for each round $t = 1, 2, ..., T$ **do**
  for each client k **in parallel do**
    // the server broadcasts $\boldsymbol{Z}_c^{(k)}$ *to the client*
    $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)} \leftarrow$ **ClientTrain**$(k, \boldsymbol{Z}_s^{(k)})$
  $\boldsymbol{Z}_s \leftarrow$ empty dictionary
  for each local epoch i from 1 to E_s **do**
    for each client k **do**
      for $idx, \boldsymbol{b} \in \{\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
        $\boldsymbol{W}_s \leftarrow \boldsymbol{W}_s - \eta_s \nabla \ell_s(\boldsymbol{W}_s; \boldsymbol{b})$
        if $i == E_s$ **then**
          $\boldsymbol{Z}_s^{(k)}[idx] \leftarrow f_s(\boldsymbol{W}_s; \boldsymbol{h}^{(k)})$
  // illustrated as "transfer back" in Fig. 1(a)
  for each client k **in parallel do**
    send the server logits $\boldsymbol{Z}_s^{(k)}$ to client k

ClientTrain$(k, \boldsymbol{Z}_s^{(k)})$:
// illustrated as "local training "in Fig. 1(a)
for each local epoch i from 1 to E_c **do**
  for batch $\boldsymbol{b} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Z}_s^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
    // $\ell_c^{(k)}$ is computed using Eq. (7)
    $\boldsymbol{W}^{(k)} \leftarrow \boldsymbol{W}^{(k)} - \eta_k \nabla \ell_c^{(k)}(\boldsymbol{W}^{(k)}; \boldsymbol{b})$
// extract features and logits
$\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)} \leftarrow$ empty dictionary
for *idx*, batch $\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
  $\boldsymbol{h}^{(k)} \leftarrow f_e^{(k)}(\boldsymbol{W}_e^{(k)}; \boldsymbol{x}^{(k)})$
  $\boldsymbol{z}_c^{(k)} \leftarrow f_c(\boldsymbol{W}_c^{(k)}; \boldsymbol{h}^{(k)})$
  $\boldsymbol{H}^{(k)}[idx] \leftarrow \boldsymbol{h}^{(k)}$
  $\boldsymbol{Z}_c^{(k)}[idx] \leftarrow \boldsymbol{z}_c^{(k)}$
return $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}$ to server

GROUP KNOWLEDGE TRANSFER: SERVER-SIDE

- 각 라운드에 대해
- 각 클라이언트에 대해
  - 병렬적으로

Algorithm 1 Group Knowledge Transfer. The subscript s and k stands for the server and the kth edge, respectively. E is the number of *local* epochs, T is the number of communication rounds; η is the learning rate; $\boldsymbol{X}^{(k)}$ represents input images at edge k; $\boldsymbol{H}^{(k)}$ is the extracted feature map from $\boldsymbol{X}^{(k)}$; $\boldsymbol{Z}_s$ and $\boldsymbol{Z}_c^{(k)}$ are the logit tensor from the client and the server, respectively.

ServerExecute():
for each round $t = 1, 2, ..., T$ **do**
  for each client k **in parallel do**
    // the server broadcasts $\boldsymbol{Z}_c^{(k)}$ *to the client*
    $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)} \leftarrow$ **ClientTrain**$(k, \boldsymbol{Z}_s^{(k)})$
  $\boldsymbol{Z}_s \leftarrow$ empty dictionary
  for each local epoch i from 1 to E_s **do**
    for each client k **do**
      for *idx*, $\boldsymbol{b} \in \{\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
        $\boldsymbol{W}_s \leftarrow \boldsymbol{W}_s - \eta_s \nabla \ell_s(\boldsymbol{W}_s; \boldsymbol{b})$
        if $i == E_s$ **then**
          $\boldsymbol{Z}_s^{(k)}[idx] \leftarrow f_s(\boldsymbol{W}_s; \boldsymbol{h}^{(k)})$
  // illustrated as "transfer back" in Fig. 1(a)
  for each client k **in parallel do**
    send the server logits $\boldsymbol{Z}_s^{(k)}$ to client k

ClientTrain$(k, \boldsymbol{Z}_s^{(k)})$:
// illustrated as "local training "in Fig. 1(a)
for each local epoch i from 1 to E_c **do**
  for batch $\boldsymbol{b} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Z}_s^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
    // $\ell_c^{(k)}$ is computed using Eq. (7)
    $\boldsymbol{W}^{(k)} \leftarrow \boldsymbol{W}^{(k)} - \eta_k \nabla \ell_c^{(k)}(\boldsymbol{W}^{(k)}; \boldsymbol{b})$
// extract features and logits
$\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)} \leftarrow$ empty dictionary
for *idx*, batch $\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
  $\boldsymbol{h}^{(k)} \leftarrow f_e^{(k)}(\boldsymbol{W}_e^{(k)}; \boldsymbol{x}^{(k)})$
  $\boldsymbol{z}_c^{(k)} \leftarrow f_c(\boldsymbol{W}_c^{(k)}; \boldsymbol{h}^{(k)})$
  $\boldsymbol{H}^{(k)}[idx] \leftarrow \boldsymbol{h}^{(k)}$
  $\boldsymbol{Z}_c^{(k)}[idx] \leftarrow \boldsymbol{z}_c^{(k)}$
return $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}$ to server

GROUP KNOWLEDGE TRANSFER: SERVER-SIDE

- 각 클라이언트의 학습
  - 서버 지식을 이용
- 출력
  - 특징, 지식, 정답

Algorithm 1 Group Knowledge Transfer. The subscript s and k stands for the server and the kth edge, respectively. E is the number of *local* epochs, T is the number of communication rounds; η is the learning rate; $\boldsymbol{X}^{(k)}$ represents input images at edge k; $\boldsymbol{H}^{(k)}$ is the extracted feature map from $\boldsymbol{X}^{(k)}$; $\boldsymbol{Z}_s$ and $\boldsymbol{Z}_c^{(k)}$ are the logit tensor from the client and the server, respectively.

ServerExecute():
for each round $t = 1, 2, ..., T$ **do**
  for each client k **in parallel do**
    // the server broadcasts $\boldsymbol{Z}_s^{(k)}$ to the client
    $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)} \leftarrow$ **ClientTrain**$(k, \boldsymbol{Z}_s^{(k)})$
  $\boldsymbol{Z}_s \leftarrow$ empty dictionary
  for each local epoch i from 1 to E_s **do**
    for each client k **do**
      for *idx*, $\boldsymbol{b} \in \{\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
        $\boldsymbol{W}_s \leftarrow \boldsymbol{W}_s - \eta_s \nabla \ell_s(\boldsymbol{W}_s; \boldsymbol{b})$
        if $i == E_s$ **then**
          $\boldsymbol{Z}_s^{(k)}[idx] \leftarrow f_s(\boldsymbol{W}_s; \boldsymbol{h}^{(k)})$
  // illustrated as "transfer back" in Fig. 1(a)
  for each client k **in parallel do**
    send the server logits $\boldsymbol{Z}_s^{(k)}$ to client k

ClientTrain$(k, \boldsymbol{Z}_s^{(k)})$:
// illustrated as "local training" in Fig. 1(a)
for each local epoch i from 1 to E_c **do**
  for batch $\boldsymbol{b} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Z}_s^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
    // $\ell_c^{(k)}$ is computed using Eq. (7)
    $\boldsymbol{W}^{(k)} \leftarrow \boldsymbol{W}^{(k)} - \eta_k \nabla \ell_c^{(k)}(\boldsymbol{W}^{(k)}; \boldsymbol{b})$
// extract features and logits
$\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)} \leftarrow$ empty dictionary
for *idx*, batch $\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
  $\boldsymbol{h}^{(k)} \leftarrow f_e^{(k)}(\boldsymbol{W}_e^{(k)}; \boldsymbol{x}^{(k)})$
  $\boldsymbol{z}_c^{(k)} \leftarrow f_c(\boldsymbol{W}_c^{(k)}; \boldsymbol{h}^{(k)})$
  $\boldsymbol{H}^{(k)}[idx] \leftarrow \boldsymbol{h}^{(k)}$
  $\boldsymbol{Z}_c^{(k)}[idx] \leftarrow \boldsymbol{z}_c^{(k)}$
return $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}$ to server

GROUP KNOWLEDGE TRANSFER: CLIENT-SIDE

- 로컬 학습
  - 로컬 SGD

Algorithm 1 Group Knowledge Transfer. The subscript s and k stands for the server and the kth edge, respectively. E is the number of *local* epochs, T is the number of communication rounds; η is the learning rate; $\boldsymbol{X}^{(k)}$ represents input images at edge k; $\boldsymbol{H}^{(k)}$ is the extracted feature map from $\boldsymbol{X}^{(k)}$; $\boldsymbol{Z}_s$ and $\boldsymbol{Z}_c^{(k)}$ are the logit tensor from the client and the server, respectively.

ServerExecute():
for each round $t = 1, 2, ..., T$ **do**
  for each client k **in parallel do**
    // the server broadcasts $\boldsymbol{Z}_c^{(k)}$ *to the client*
    $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)} \leftarrow$ **ClientTrain**$(k, \boldsymbol{Z}_s^{(k)})$
  $\boldsymbol{Z}_s \leftarrow$ empty dictionary
  for each local epoch i from 1 to E_s **do**
    for each client k **do**
      for $idx, \boldsymbol{b} \in \{\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
        $\boldsymbol{W}_s \leftarrow \boldsymbol{W}_s - \eta_s \nabla \ell_s(\boldsymbol{W}_s; \boldsymbol{b})$
        if $i == E_s$ **then**
          $\boldsymbol{Z}_s^{(k)}[idx] \leftarrow f_s(\boldsymbol{W}_s; \boldsymbol{h}^{(k)})$
  // illustrated as "transfer back" in Fig. 1(a)
  for each client k **in parallel do**
    send the server logits $\boldsymbol{Z}_s^{(k)}$ to client k

ClientTrain$(k, \boldsymbol{Z}_s^{(k)})$:
// illustrated as "local training "in Fig. 1(a)
for each local epoch i from 1 to E_c **do**
  for batch $\boldsymbol{b} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Z}_s^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
    // $\ell_c^{(k)}$ is computed using Eq. (7)
    $\boldsymbol{W}^{(k)} \leftarrow \boldsymbol{W}^{(k)} - \eta_k \nabla \ell_c^{(k)}(\boldsymbol{W}^{(k)}; \boldsymbol{b})$
// extract features and logits
$\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)} \leftarrow$ empty dictionary
for *idx*, batch $\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
  $\boldsymbol{h}^{(k)} \leftarrow f_e^{(k)}(\boldsymbol{W}_e^{(k)}; \boldsymbol{x}^{(k)})$
  $\boldsymbol{z}_c^{(k)} \leftarrow f_c(\boldsymbol{W}_c^{(k)}; \boldsymbol{h}^{(k)})$
  $\boldsymbol{H}^{(k)}[idx] \leftarrow \boldsymbol{h}^{(k)}$
  $\boldsymbol{Z}_c^{(k)}[idx] \leftarrow \boldsymbol{z}_c^{(k)}$
return $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}$ to server

GROUP KNOWLEDGE TRANSFER: CLIENT-SIDE

- 특징 및 지식 추출
- 특징 추출기의 출력
  - 특징
- 분류기의 출력
  - 지식

Algorithm 1 Group Knowledge Transfer. The subscript s and k stands for the server and the kth edge, respectively. E is the number of *local* epochs, T is the number of communication rounds; η is the learning rate; $\boldsymbol{X}^{(k)}$ represents input images at edge k; $\boldsymbol{H}^{(k)}$ is the extracted feature map from $\boldsymbol{X}^{(k)}$; $\boldsymbol{Z}_s$ and $\boldsymbol{Z}_c^{(k)}$ are the logit tensor from the client and the server, respectively.

ServerExecute():
for each round $t = 1, 2, ..., T$ **do**
  for each client k **in parallel do**
    // the server broadcasts $\boldsymbol{Z}_c^{(k)}$ *to the client*
    $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)} \leftarrow$ **ClientTrain**$(k, \boldsymbol{Z}_s^{(k)})$
  $\boldsymbol{Z}_s \leftarrow$ empty dictionary
  for each local epoch i from 1 to E_s **do**
    for each client k **do**
      for $idx, \boldsymbol{b} \in \{\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
        $\boldsymbol{W}_s \leftarrow \boldsymbol{W}_s - \eta_s \nabla \ell_s(\boldsymbol{W}_s; \boldsymbol{b})$
        if $i == E_s$ **then**
          $\boldsymbol{Z}_s^{(k)}[idx] \leftarrow f_s(\boldsymbol{W}_s; \boldsymbol{h}^{(k)})$
  // illustrated as "transfer back" in Fig. 1(a)
  for each client k **in parallel do**
    send the server logits $\boldsymbol{Z}_s^{(k)}$ to client k

ClientTrain$(k, \boldsymbol{Z}_s^{(k)})$:
// illustrated as "local training "in Fig. 1(a)
for each local epoch i from 1 to E_c **do**
  for batch $\boldsymbol{b} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Z}_s^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
    // $\ell_c^{(k)}$ is computed using Eq. (7)
    $\boldsymbol{W}^{(k)} \leftarrow \boldsymbol{W}^{(k)} - \eta_k \nabla \ell_c^{(k)}(\boldsymbol{W}^{(k)}; \boldsymbol{b})$
// extract features and logits
$\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)} \leftarrow$ empty dictionary
for *idx*, batch $\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
  $\boldsymbol{h}^{(k)} \leftarrow f_e^{(k)}(\boldsymbol{W}_e^{(k)}; \boldsymbol{x}^{(k)})$
  $\boldsymbol{z}_c^{(k)} \leftarrow f_c(\boldsymbol{W}_c^{(k)}; \boldsymbol{h}^{(k)})$
  $\boldsymbol{H}^{(k)}[idx] \leftarrow \boldsymbol{h}^{(k)}$
  $\boldsymbol{Z}_c^{(k)}[idx] \leftarrow \boldsymbol{z}_c^{(k)}$
return $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}$ to server

GROUP KNOWLEDGE TRANSFER: CLIENT-SIDE

- 특징, 지식, 정답 전송

Algorithm 1 Group Knowledge Transfer. The subscript s and k stands for the server and the kth edge, respectively. E is the number of *local* epochs, T is the number of communication rounds; η is the learning rate; $\boldsymbol{X}^{(k)}$ represents input images at edge k; $\boldsymbol{H}^{(k)}$ is the extracted feature map from $\boldsymbol{X}^{(k)}$; $\boldsymbol{Z}_s$ and $\boldsymbol{Z}_c^{(k)}$ are the logit tensor from the client and the server, respectively.

ServerExecute():
for each round $t = 1, 2, ..., T$ **do**
  for each client k **in parallel do**
    // the server broadcasts $\boldsymbol{Z}_c^{(k)}$ *to the client*
    $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)} \leftarrow$ **ClientTrain**$(k, \boldsymbol{Z}_s^{(k)})$
  $\boldsymbol{Z}_s \leftarrow$ empty dictionary
  for each local epoch i from 1 to E_s **do**
    for each client k **do**
      for $idx, \boldsymbol{b} \in \{\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
        $\boldsymbol{W}_s \leftarrow \boldsymbol{W}_s - \eta_s \nabla \ell_s(\boldsymbol{W}_s; \boldsymbol{b})$
        if $i == E_s$ **then**
          $\boldsymbol{Z}_s^{(k)}[idx] \leftarrow f_s(\boldsymbol{W}_s; \boldsymbol{h}^{(k)})$
  // illustrated as "transfer back" in Fig. 1(a)
  for each client k **in parallel do**
    send the server logits $\boldsymbol{Z}_s^{(k)}$ to client k

ClientTrain$(k, \boldsymbol{Z}_s^{(k)})$:
// illustrated as "local training "in Fig. 1(a)
for each local epoch i from 1 to E_c **do**
  for batch $\boldsymbol{b} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Z}_s^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
    // $\ell_c^{(k)}$ is computed using Eq. (7)
    $\boldsymbol{W}^{(k)} \leftarrow \boldsymbol{W}^{(k)} - \eta_k \nabla \ell_c^{(k)}(\boldsymbol{W}^{(k)}; \boldsymbol{b})$
// extract features and logits
$\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)} \leftarrow$ empty dictionary
for *idx*, batch $\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
  $\boldsymbol{h}^{(k)} \leftarrow f_e^{(k)}(\boldsymbol{W}_e^{(k)}; \boldsymbol{x}^{(k)})$
  $\boldsymbol{z}_c^{(k)} \leftarrow f_c(\boldsymbol{W}_c^{(k)}; \boldsymbol{h}^{(k)})$
  $\boldsymbol{H}^{(k)}[idx] \leftarrow \boldsymbol{h}^{(k)}$
  $\boldsymbol{Z}_c^{(k)}[idx] \leftarrow \boldsymbol{z}_c^{(k)}$
return $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}$ to server

GROUP KNOWLEDGE TRANSFER: SERVER-SIDE

- 서버측 지식 계산
- 각 로컬 에폭
- 각 클라이언트에 대해
- 서버 분류기를 학습
  - 출력이 지식이 됨

Algorithm 1 Group Knowledge Transfer. The subscript s and k stands for the server and the kth edge, respectively. E is the number of *local* epochs, T is the number of communication rounds; η is the learning rate; $\boldsymbol{X}^{(k)}$ represents input images at edge k; $\boldsymbol{H}^{(k)}$ is the extracted feature map from $\boldsymbol{X}^{(k)}$; $\boldsymbol{Z}_s$ and $\boldsymbol{Z}_c^{(k)}$ are the logit tensor from the client and the server, respectively.

ServerExecute():
for each round $t = 1, 2, ..., T$ **do**
  for each client k **in parallel do**
    // the server broadcasts $\boldsymbol{Z}_c^{(k)}$ *to the client*
    $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)} \leftarrow$ **ClientTrain**$(k, \boldsymbol{Z}_s^{(k)})$
  $\boldsymbol{Z}_s \leftarrow$ empty dictionary
  for each local epoch i from 1 to E_s **do**
    for each client k **do**
      for $idx, \boldsymbol{b} \in \{\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
        $\boldsymbol{W}_s \leftarrow \boldsymbol{W}_s - \eta_s \nabla \ell_s(\boldsymbol{W}_s; \boldsymbol{b})$
        if $i == E_s$ **then**
          $\boldsymbol{Z}_s^{(k)}[idx] \leftarrow f_s(\boldsymbol{W}_s; \boldsymbol{h}^{(k)})$
  // illustrated as "transfer back" in Fig. 1(a)
  for each client k **in parallel do**
    send the server logits $\boldsymbol{Z}_s^{(k)}$ to client k

ClientTrain$(k, \boldsymbol{Z}_s^{(k)})$:
// illustrated as "local training "in Fig. 1(a)
for each local epoch i from 1 to E_c **do**
  for batch $\boldsymbol{b} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Z}_s^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
    // $\ell_c^{(k)}$ is computed using Eq. (7)
    $\boldsymbol{W}^{(k)} \leftarrow \boldsymbol{W}^{(k)} - \eta_k \nabla \ell_c^{(k)}(\boldsymbol{W}^{(k)}; \boldsymbol{b})$
// extract features and logits
$\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)} \leftarrow$ empty dictionary
for idx, batch $\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
  $\boldsymbol{h}^{(k)} \leftarrow f_e^{(k)}(\boldsymbol{W}_e^{(k)}; \boldsymbol{x}^{(k)})$
  $\boldsymbol{z}_c^{(k)} \leftarrow f_c(\boldsymbol{W}_c^{(k)}; \boldsymbol{h}^{(k)})$
  $\boldsymbol{H}^{(k)}[idx] \leftarrow \boldsymbol{h}^{(k)}$
  $\boldsymbol{Z}_c^{(k)}[idx] \leftarrow \boldsymbol{z}_c^{(k)}$
return $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}$ to server

GROUP KNOWLEDGE TRANSFER: SERVER-SIDE

- 이를
  - 클라이언트에게 전송

Algorithm 1 Group Knowledge Transfer. The subscript s and k stands for the server and the kth edge, respectively. E is the number of *local* epochs, T is the number of communication rounds; η is the learning rate; $\boldsymbol{X}^{(k)}$ represents input images at edge k; $\boldsymbol{H}^{(k)}$ is the extracted feature map from $\boldsymbol{X}^{(k)}$; $\boldsymbol{Z}_s$ and $\boldsymbol{Z}_c^{(k)}$ are the logit tensor from the client and the server, respectively.

ServerExecute():
for each round $t = 1, 2, ..., T$ **do**
  for each client k **in parallel do**
    // the server broadcasts $\boldsymbol{Z}_c^{(k)}$ *to the client*
    $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)} \leftarrow$ **ClientTrain**$(k, \boldsymbol{Z}_s^{(k)})$
  $\boldsymbol{Z}_s \leftarrow$ empty dictionary
  for each local epoch i from 1 to E_s **do**
    for each client k **do**
      for $idx, \boldsymbol{b} \in \{\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
        $\boldsymbol{W}_s \leftarrow \boldsymbol{W}_s - \eta_s \nabla \ell_s(\boldsymbol{W}_s; \boldsymbol{b})$
        if $i == E_s$ **then**
          $\boldsymbol{Z}_s^{(k)}[idx] \leftarrow f_s(\boldsymbol{W}_s; \boldsymbol{h}^{(k)})$
  // illustrated as "transfer back" in Fig. 1(a)
  for each client k **in parallel do**
    send the server logits $\boldsymbol{Z}_s^{(k)}$ to client k

ClientTrain$(k, \boldsymbol{Z}_s^{(k)})$:
// illustrated as "local training "in Fig. 1(a)
for each local epoch i from 1 to E_c **do**
  for batch $\boldsymbol{b} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Z}_s^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
    // $\ell_c^{(k)}$ is computed using Eq. (7)
    $\boldsymbol{W}^{(k)} \leftarrow \boldsymbol{W}^{(k)} - \eta_k \nabla \ell_c^{(k)}(\boldsymbol{W}^{(k)}; \boldsymbol{b})$
// extract features and logits
$\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)} \leftarrow$ empty dictionary
for *idx*, batch $\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)} \in \{\boldsymbol{X}^{(k)}, \boldsymbol{Y}^{(k)}\}$ **do**
  $\boldsymbol{h}^{(k)} \leftarrow f_e^{(k)}(\boldsymbol{W}_e^{(k)}; \boldsymbol{x}^{(k)})$
  $\boldsymbol{z}_c^{(k)} \leftarrow f_c(\boldsymbol{W}_c^{(k)}; \boldsymbol{h}^{(k)})$
  $\boldsymbol{H}^{(k)}[idx] \leftarrow \boldsymbol{h}^{(k)}$
  $\boldsymbol{Z}_c^{(k)}[idx] \leftarrow \boldsymbol{z}_c^{(k)}$
return $\boldsymbol{H}^{(k)}, \boldsymbol{Z}_c^{(k)}, \boldsymbol{Y}^{(k)}$ to server

GROUP KNOWLEDGE TRANSFER

- 이 과정의 반복
  - 여러 라운드에 걸쳐
- 각 라운드에서는
  - 클라이언트의 지식이 서버에게 전이
  - 서버의 지식이 클라이언트에게 전이

EXPERIMENTS

EXPERIMENTS

- 서버 노드
  - 4개의 NVIDIA RTX 2080Ti GPUs
  - 큰 모델을 학습하기에 충분한 GPU 메모리
- 클라이언트 노드
  - 다양한 CPU 기반
  - 작은 CNN을 학습
- 16 클라이언트와 하나의 GPU 서버를 사용

EXPERIMENTS

- CIFAR-10, CIFAR-100, CINIC-10의 이미지 분류
  - 학습 샘플을 K개 균등하지 않은 크기의 Non-IID로 분할
- 각 라운드에 대해 테스트를 위한 이미지가 사용됨
- Top 1 정확도를 여러 다른 방법들에 대해 기록

EXPERIMENTS

- FedGKT를
  - FedAvg 및 SL과 비교
- FedProx는 비교하지 않음
  - 큰 CNN에 대해 FedAvg보다 못한 성능을 보이므로
- FedMA도 비교하지 않음
  - 배치 정규화 레이어를 포함한 최신 DNN 구조에서 동작하지 않으므로
  - ResNet은 배치 정규화 레이어가 포함됨

EXPERIMENTS

- 모델 구조
  - 두 최신 CNN 구조가 평가됨
  - ResNet-56 및 ResNet-110
- FedAvg
  - 모든 엣지 노드가 이 두 CNN을 학습하는데 사용됨

EXPERIMENTS

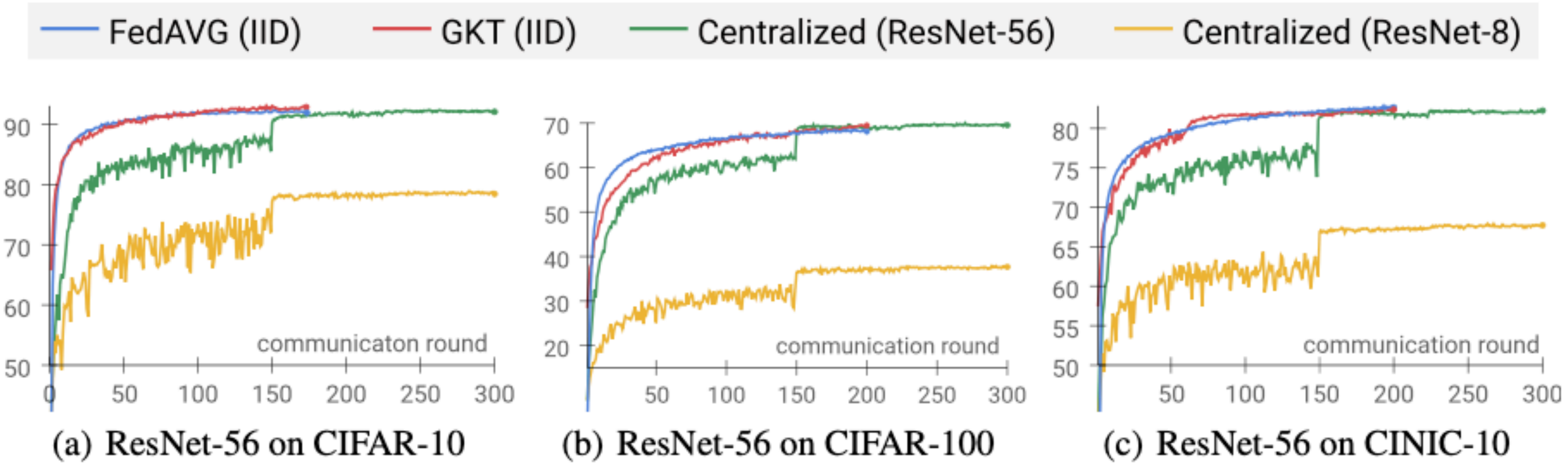
- FedGKT
  - 엣지에서는 RedNet-8 사용
  - 8 컨볼루션 레이어를 포함한 작은 CNN
- 서버에서는
  - ResNet-55 및 ResNet-109가 사용됨
  - 엣지측 특징 추출기의 출력을 입력으로 하는

EXPERIMENTS

- SL
  - 엣지측은 ResNet-8
  - 서버측은 ResNet-55 및 ResNet-109 사용

EXPERIMENTS: MODEL ACCURACY

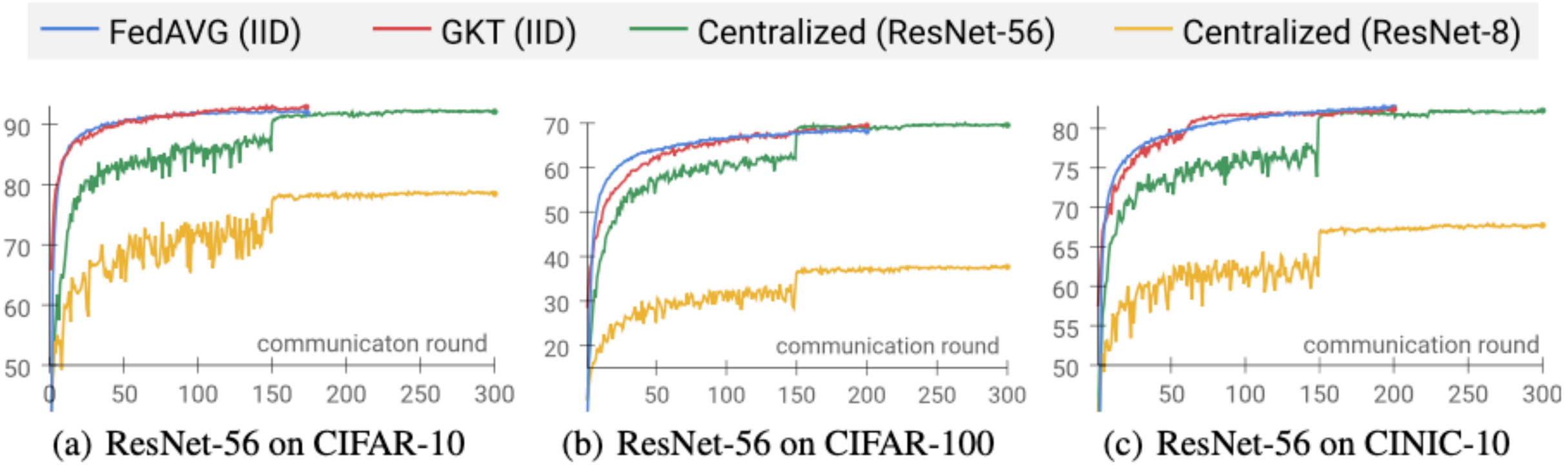
- 중앙화된 학습도 평가에 포함



(a) ResNet-56 on CIFAR-10 (b) ResNet-56 on CIFAR-100 (c) ResNet-56 on CINIC-10

EXPERIMENTS: MODEL ACCURACY

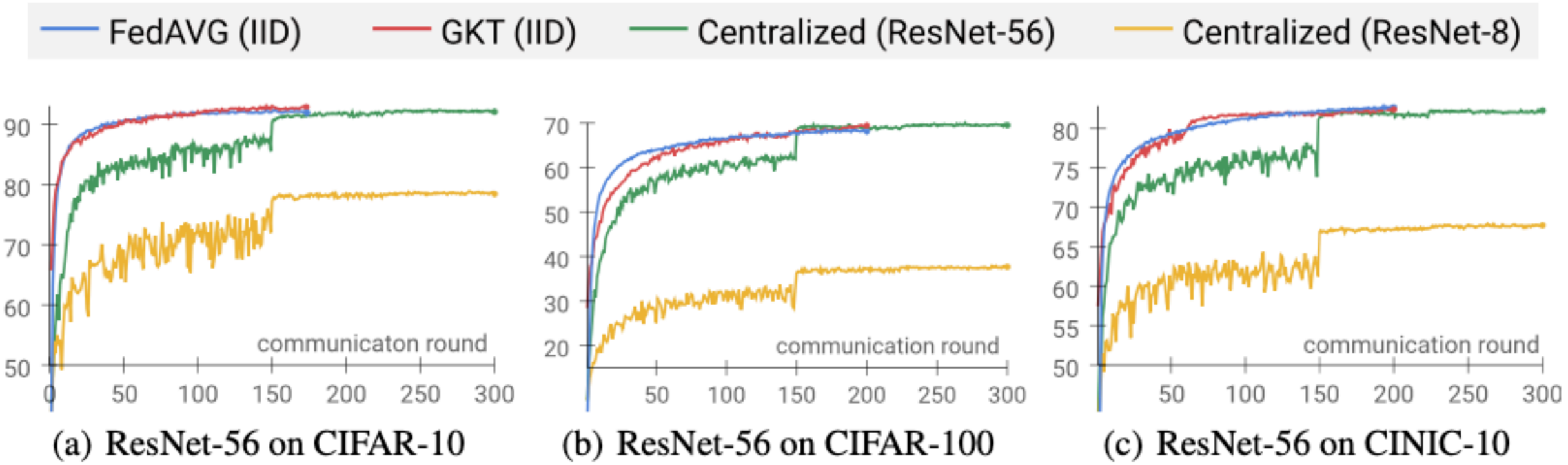
- FedGKT가 FedAVG에 필적하는 성능을 보임



(a) ResNet-56 on CIFAR-10 (b) ResNet-56 on CIFAR-100 (c) ResNet-56 on CINIC-10

EXPERIMENTS: MODEL ACCURACY

- FedGKT와 FedAVG가 중앙화 모델에 필적하는 성능을 보임



(a) ResNet-56 on CIFAR-10 (b) ResNet-56 on CIFAR-100 (c) ResNet-56 on CINIC-10

EXPERIMENTS: MODEL ACCURACY

- IID와 non-IID를 포함한 보다 상세한 결과

Model	Methods	CIFAR-10		CIFAR-100		CINIC-10	
		I.I.D.	non-I.I.D.	I.I.D.	non-I.I.D.	I.I.D.	non-I.I.D.
ResNet-56	**FedGKT (ResNet-8, ours)**	**92.97**	**86.59**	**69.57**	**63.76**	**81.51**	**77.80**
	FedAvg (ResNet-56)	92.88	86.60	68.09	63.78	81.62	77.85
	Centralized (ResNet-56)	93.05		69.73		81.66	
	Centralized (ResNet-8)	78.94		37.67		67.72	
ResNet-110	**FedGKT (ResNet-8, ours)**	**93.47**	**87.18**	**69.87**	**64.31**	**81.98**	**78.39**
	FedAvg (ResNet-56)	93.49	87.20	68.58	64.35	82.10	78.43
	Centralized (ResNet-56)	93.58		70.18		82.16	
	Centralized (ResNet-8)	78.94		37.67		67.72	

EXPERIMENTS: EFFICIENCY EVALUATION

- 연산 요구량을 비교하기 위해
  - 부동소수점 연산의 수를 측정 (petaFLOPs)
- 통신 효율성을 비교하기 위해 (GBytes)
  - SL을 기준으로 삼음
  - 데이터 압축 기술 없음

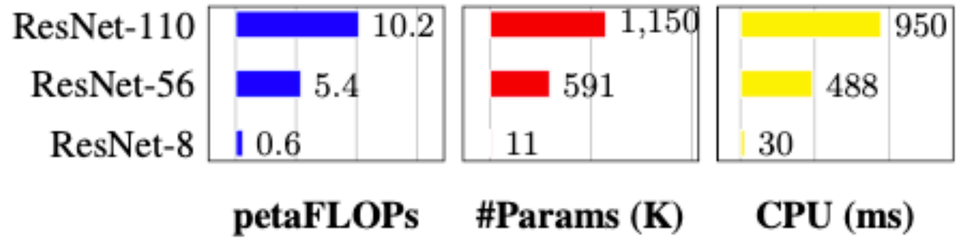


Figure 4: Edge Computational Efficiency (CIFAR-100)

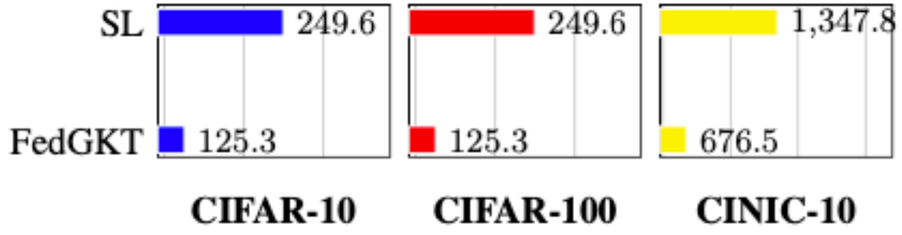


Figure 5: Communication Efficiency (ResNet-56)

CONCLUSION

CONCLUSION

- FedGKT
  - 연산 제한적인 실정을 위한
  - 그룹 지식 전이 학습 알고리즘
- 효율적으로 엣지의 작은 CNN을 학습하고
  - 주기적으로 지식을 지식 증류를 통해 서버측 CNN에 전달
  - Vice-versa
- 비동기 특성 역시 가지고 있음

CONCLUSION

- 두 신경망 구조와 세 가지 데이터셋으로 평가
  - FedGKT가 기존 방법들인 FedAvg 및 SL 대비
  - 효율적이고 정확도도 유사하거나 살짝 더 높음
- 무엇보다 엣지 학습을 가능하게 만들었다는 점이 중요
  - FedAvg 대비 9-17배 연산력이 적게 필요(FLOPs)
  - 54-105배 적은 파라미터만 요구

FEDGKT

GROUP KNOWLEDGE TRANSFER: FEDERATED LEARNING OF LARGE CNNS AT THE EDGE