Collaborative Execution of Deep Neural Networks on Internet of Things Devices

Hadidi, Ramyad, Jiashen Cao, Micheal S. Ryoo, and Hyesoon Kim

"Collaborative Execution of Deep Neural Networks on Internet of Things Devices."
arXiv preprint arXiv:1901.02537 (2019).

Motivation

- 영상처리, 자연어 처리, 기계 번역과 같은 문제를 해결하기 위해 DNN이 사용되고 있음
- IoT 장치는 DNN 응용을 실행하기엔 성능이 부족함
- DNN 응용에 대한 연구는 HPC에 집중되어 있음
- 소비자가 DNN 응용을 사용하기 위해서는 클라우드를 이용해야 함
- 이 방법은 클라우드 서비스와 네트워크에 대한 의존성을 만들며 사적 데이터에 대한 프라이버시 문제를 야기

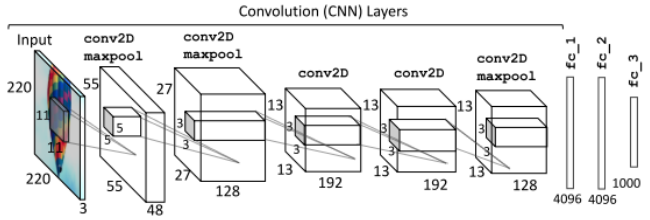
Motivation

- 로컬에서 생성된 데이터를 로컬에서 처리하는 방법이 요구

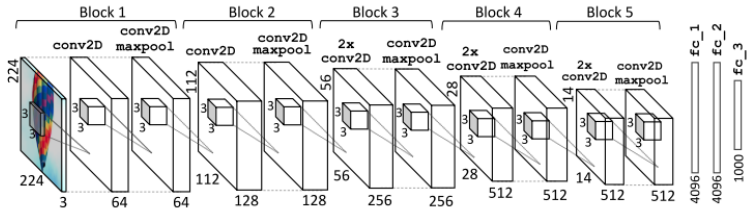
- IoT 장치를 이용해 DNN 응용을 분산처리 하는 것이 목표
  - 클라우드 리소스와 고품질 네트워크에 대한 의존 감소
  - 사용자의 개인 정보 보호
  - 특정 모델이나 데이터, 하드웨어의 종속되지 않는 솔루션

Models

- AlexNet
  - 5개의 conv layer, 3개의 fc layer
  - 매개변수 40M

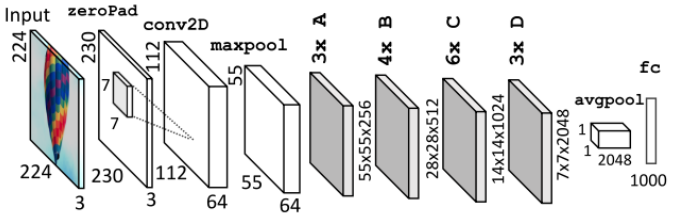


- VGG16
  - 13개의 conv layer, 3개의 fc layer
  - 매개변수 140M

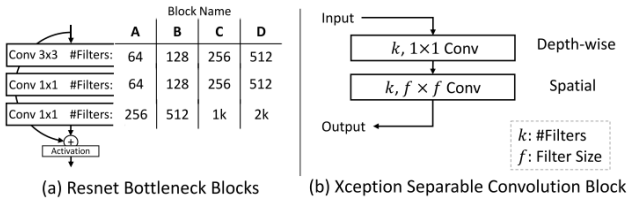


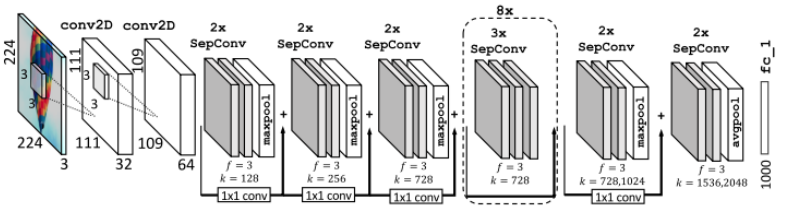
Models

- ResNet50
  - 50개의 레이어로 구성
  - 블록 사이에 shortcut이 존재
  - 매개변수 25M



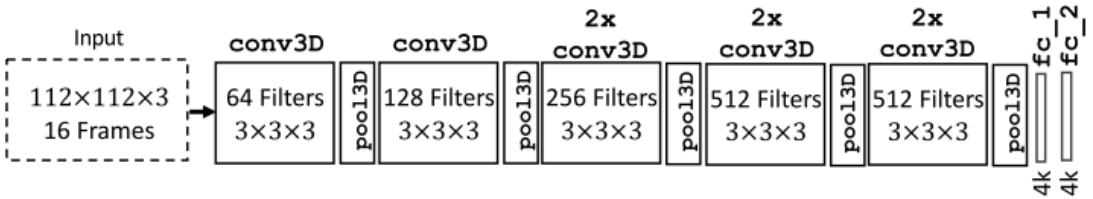
- Xception
  - 34개의 separable conv layer
  - 입력 채널에 대한 깊이 방향의 conv 연산 후,
    각 결과에 대해 독립적인 공간 conv 연산을 진행
  - 매개변수 23M





Models

- C3D
  - 비디오를 처리하도록 설계
  - 동작 인식 및 장면 분류에 사용
  - 8개의 3D conv layer로 구성
  - 매개변수 80M



분산 처리 방법

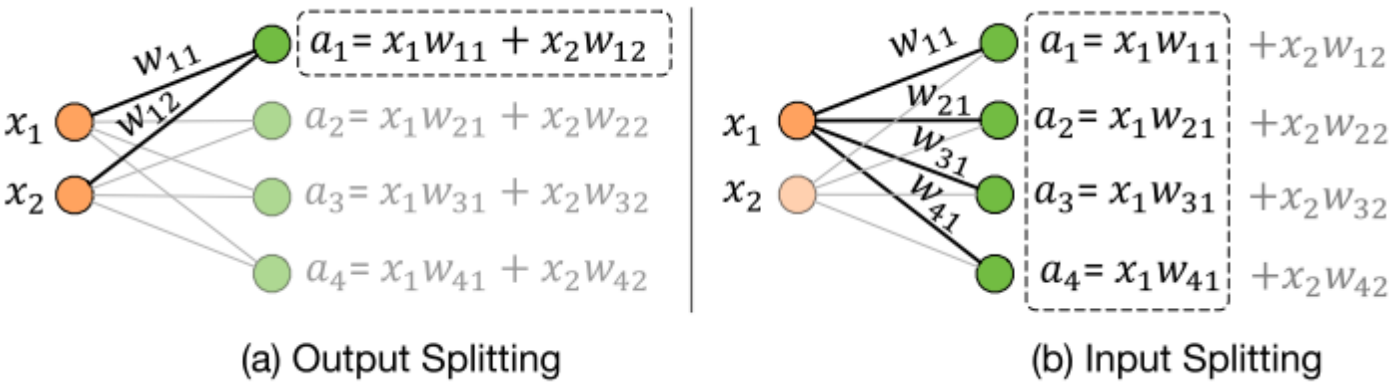
- 데이터 병렬 처리
  - 동일한 연산을 수행하는 노드를 복제
  - 입력 데이터를 분산, 병렬 처리하기 때문에
    각 입력을 처리하는 시간은 그대로이며, 초당 추론 수는 증가

  - 무거운 레이어의 경우 전체 데이터가 메모리에 올라가지 않아
    연산에 지연이 발생
  - IoT 장치에서 실시간 처리를 진행할 경우
    데이터 병렬 처리를 위한 충분한 양의 데이터가 제공되지 않을 수 있음

- 모델 병렬 처리
  - 모델 병렬 처리는 연산을 여러 노드에 분산
  - 따라서 입력을 처리하는 시간이 단축됨
  - 높은 수준의 병렬 처리를 사용하려면 각 계층의 연산 방법,
    병렬 처리가 데이터 통신, 연산 및 집계에 미치는 영향에 대한 지식이 필요

모델 병렬 처리

- FC Layer
  - Output Splitting
    - 각 노드에 모든 입력을 전송. 각 노드는 모든 가중치를 보유
    - 각 노드가 연산을 완료하면 결과를 병합
    - 활성화 함수를 각 노드에서 적용할 수 있음
  - Input Splitting
    - 입력의 일부를 각 노드에 전송. 각 노드는 처리하는 입력에 대한 가중치만 보유
    - 각 노드가 연산을 완료하면 결과를 병합
    - 활성화 함수는 병합 이후에만 적용이 가능

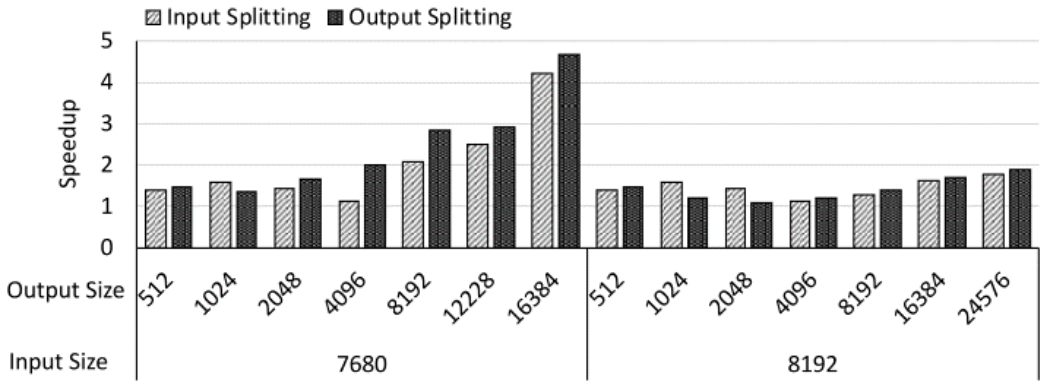


(a) Output Splitting (b) Input Splitting

모델 병렬 처리

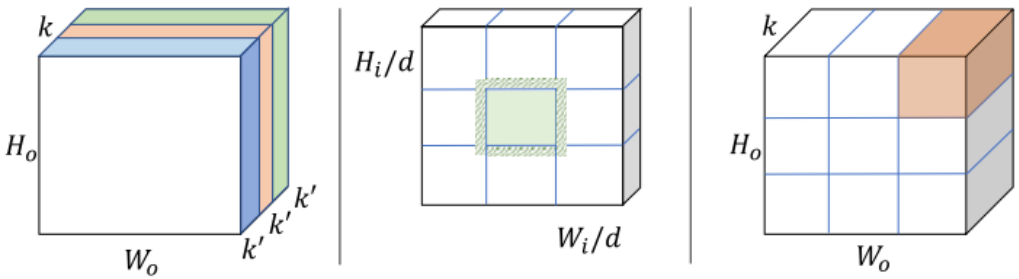
- FC Layer
  - 두 개의 장치에서 분산 처리 테스트
  - 입력 크기가 7680이고 출력 크기가 크면 속도 증가 폭이 커짐
  - 이 경우 오프칩 저장, 스왑이 사용되기 때문
  - 입력 크기가 8192인 경우 DNN 프레임워크는 액세스를 최적화해 스왑을 피함

  - Output Splitting은 각 장치에서 활성화 함수를 적용할 수 있기 때문에 Input Splitting보다 대체적으로 성능이 높음



모델 병렬 처리

- Convolutional Layer
  - Channel Splitting
    - 각 노드에 모든 입력을 전송. 각 노드는 분할된 필터만 보유
    - 각 노드마다 하나의 채널에 해당하는 결과를 출력
  - Spatial Splitting
    - 각 노드에 지역적으로 분할된 입력을 전송. 각 노드는 모든 필터를 보유
    - 각 노드는 분할된 영역에 대한 완성된 결과를 출력



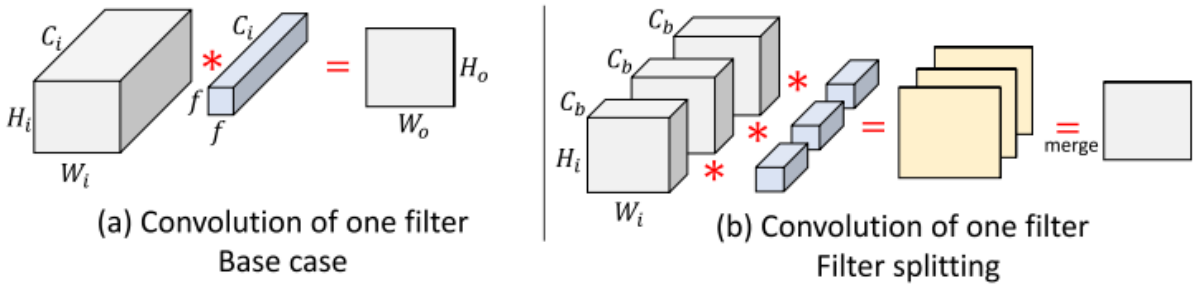
(a) Channel Splitting (output)

(b) Spatial Splitting (input)

(c) Spatial Splitting (output)

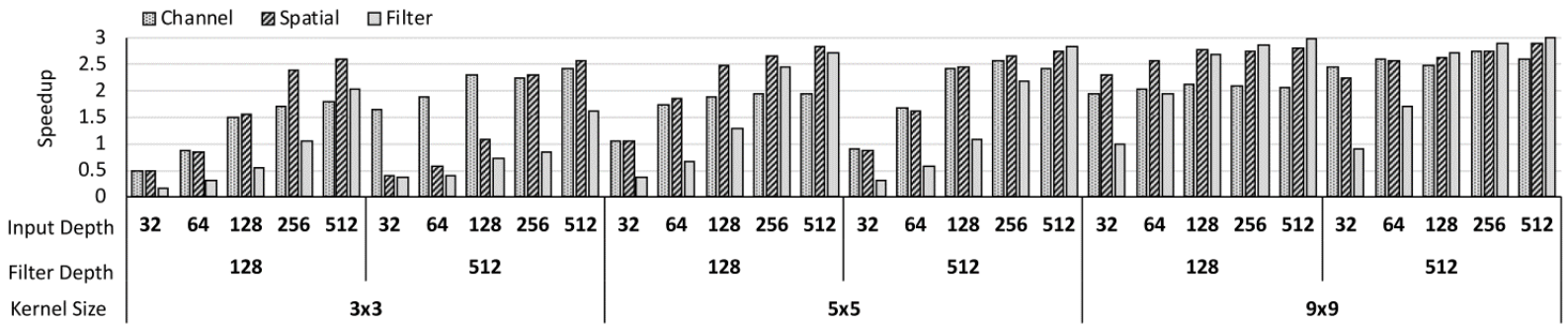
모델 병렬 처리

- Convolutional Layer
  - Filter Splitting
    - 각 노드에 채널 범위만큼 분할된 입력을 전송
    - 각 노드는 채널 범위만큼 분할된 필터를 보유
    - 각 노드는 부분적인 결과를 출력
    - 결과값을 병합하기 전에 활성화 함수를 적용할 수 없음



모델 병렬 처리

- Convolutional Layer
  - 세 개의 장치를 사용해 테스트
  - 입력, 필터, 커널 크기가 작을 때는 속도 향상이 없음
    - 작업 분배 후의 연산이 매우 적기 때문
  - 입력이 커지면, 연산이 적절히 분배되기 때문에 연산 속도가 향상
  - 커널 크기가 작을 때 Spatial Splitting의 속도 향상이 비교적 큼
    - 입력에서 오버래핑의 크기가 적기 때문에 오버헤드가 적음
    - 커널 크기가 커지면 spatial splitting의 이점이 줄어듬



분산 처리 방법

- 통신 오버헤드는 고정적으로 발생하기 때문에 불필요한 분할은 피해야 함
- 각 노드마다 충분한 양의 작업을 할당해
  병렬 처리의 이점이 통신 오버헤드로 상쇄되지 않도록 유의
- 각 병렬 처리의 특성과 각 레이어의 프로파일링 결과를 고려해
  어떤 레이어에 어떤 방식을 적용할지 선택

분산 처리 방법

1. 모든 레이어의 메모리 사용량을 분석

2. 레이어의 메모리 사용량이 노드의 메모리보다 크다면 모델 병렬 처리

3. 분할된 레이어들의 순차적 종속성과 대기시간을 고려해
   같은 그룹의 레이어들은 같은 대기시간을 갖도록 조정

4. 메모리 사용량과 대기 시간을 모니터링하면서 필요에 따라 이 과정을 반복

성능 평가

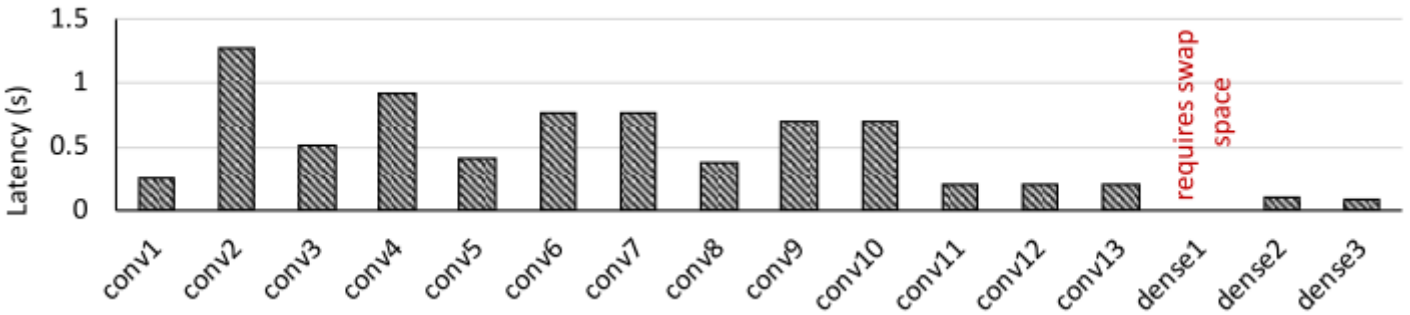
- 실험 환경
  - Raspberry Pi 3
  - TensorFlow, Keras를 이용해 만든 모델
  - 94.1Mbps로 측정된 네트워크 환경

- 모델 병렬 처리만을 분석

성능 평가

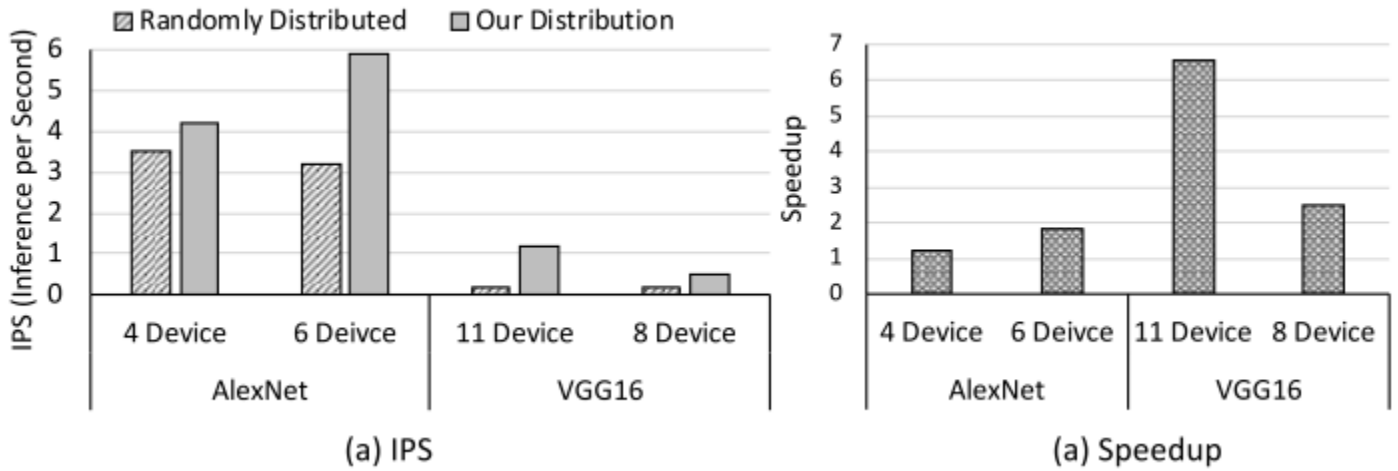
- AlexNet
  - 첫 번째 fc layer가 메모리를 초과하기 때문에 Output Splitting을 적용
  - 나머지 레이어들은 유휴 노드에 분배

- VGG16
  - 보다 연산 집약적인 레이어로 구성
  - 첫 번째 fc layer가 메모리를 초과하기 때문에 Output Splitting을 적용
  - Conv2와 같이 대기시간이 긴 레이어를 중심으로 묶어서 분산처리



성능 평가

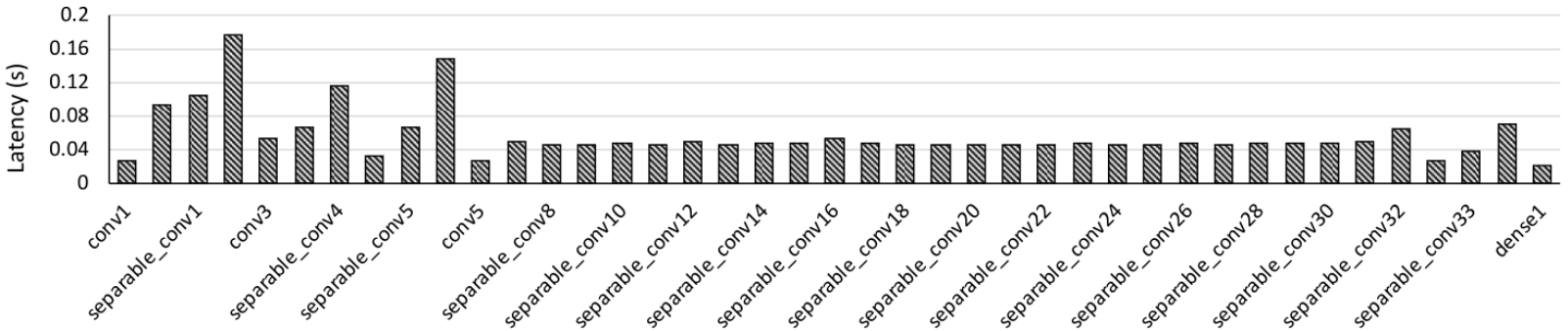
- AlexNet
- VGG16



성능 평가

- ResNet50 & Xception
  - 모든 레이어가 낮은 대기시간을 보임
  - 가장 큰 대기 시간도 0.2초보다 낮음
  - 이런 경우 모델 병렬 처리는 불필요

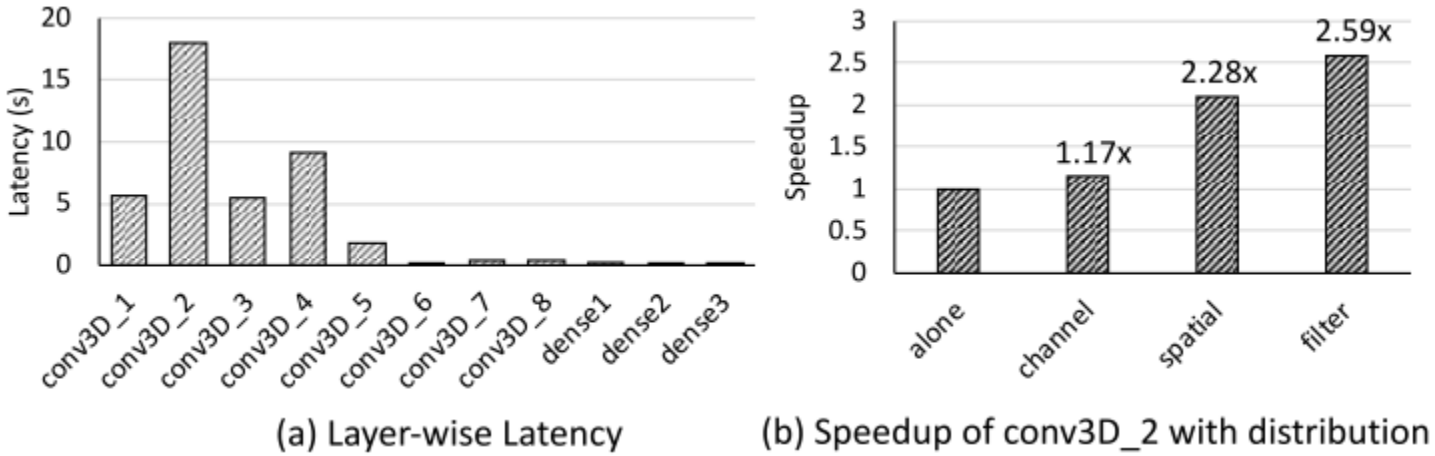
  - 데이터 병렬 처리를 통해 성능 향상 가능



성능 평가

- C3D
  - 3D convolution 연산을 하기 때문에 대기 시간이 높음
  - 메모리 사용량은 높지 않음

  - conv3D_2를 세 개의 장치를 통해 분산 처리한 결과
    최대 2.59배의 성능 향상을 얻을 수 있음



(a) Layer-wise Latency (b) Speedup of conv3D_2 with distribution

결론

- 장치간 공동 작업을 활용해 DNN 추론을 위한 파이프라인을 구성
- 이를 통해 리소스가 제한되는 IoT 장치를 위한 DNN 솔루션 제공
- 고정 비용인 통신 오버헤드를 줄일 수 있는 방법 연구
- 번역, 음성 인식과 같은 다양한 DNN 모델로 확장