논문 22-47-06-03 The Journal of Korean Institute of Communications and Information Sciences '22-06 Vol.47 No.06
https://doi.org/10.7840/kics.2022.47.6.818

AI 기반 모바일 서비스를 위한 딥러닝 모델 파티셔닝 기법

임 정 아*, 김 영 진°

DNN Model Partitioning in AI-Based Mobile Services

Jeong-A Lim*, Yeongjin Kim°

요 약

최근 무선 네트워크 기술의 발전을 통해 모바일 단말에서 물체 인식 및 비디오 분석과 같은 실시간성 모바일 비전 어플리케이션이 여러 분야에서 적용되고 있다. 이러한 어플리케이션은 높은 정확도의 딥러닝 모델을 사용하기 위하여 모바일 엣지 컴퓨팅 (MEC)을 활용하지만, 네트워크 비용 및 지연으로 인해 낮은 사용자 경험 품질 (QoE)을 보인다. 이를 해결하기 위해서, 모바일 단말과 MEC 서버 사이 딥러닝 추론 연산을 레이어 단위로 나누는 딥러닝 모델 파티셔닝 기술이 주목받고 있다. 기존 연구는 모바일 비전 어플리케이션의 사용자 경험 품질 향상을 위하여 단대단 지연 시간, 에너지 소모, 추론 속도 (fps) 중 한 가지, 혹은 두 가지 지표만을 개선할 수 있는 딥러닝 모델 파티셔닝 방법을 제시하였다. 본 연구에서는 위 세 가지 성능 개선지표의 향상을 위하여 1) 모델 파티션 포인트, 2) 입력 프레임 중 처리할 프레임 수, 3) 모바일 단말의 GPU 클럭 주파수를 제어하는 알고리즘을 제안한다. 시뮬레이션 결과를 통해, 제안하는 방법이 동일 단대단 지연 시간에서 모바일 단말 단독 추론 알고리즘과 비교했을 때, 90.2%의 평균 에너지 소모 성능 개선을 보이고, MEC 서버 단독 추론 알고리즘과 비교했을 때, 91.8%의 처리 fps 성능 개선을 보임을 확인하였다.

Key Words : DNN model partitioning, mobile vision application, quality of user experience, mobile edge computing, optimization

ABSTRACT

Through the advancement of wireless network technology, real-time mobile vision applications such as object detection and image analysis in mobile devices are being used in various fields. Such applications leverage mobile edge computing (MEC) to utilize high-accuracy deep learning models, but show low QoE due to network overhead. To tackle this, deep model partitioning has emerged that splits processing for inference between a mobile device and MEC server. Existing works proposed deep learning model partitioning algorithms to improve one or two metrics among end-to-end latency, energy consumption, and frame per second (fps) to enhance the QoE of mobile vision applications. In this paper, we propose an algorithm to jointly control (i) the model partitioing point, (ii) the number of frames to be processed among the input frames, and (iii) the GPU clock frequency of the mobile device to improve the performance of the above three metrics. With trace-driven simulation, we verify that our RT-DMP can save 90.2% of energy consumption than mobile processing algorithm, and improve processed fps by 91.8% compared to MEC algorithm.

※ 본 연구는 2022 년도 정보통신기획평가원 (2020-0-01389 인공지능융합연구센터 지원, 2021-0-02201 사용자 프라이버시를 보존하는 비디오 캐싱을 위한 연합 학습 시스템, 2022-0-00448 인간처럼 회상이 가능한 인공 신경망 지속학습 플랫폼 개발), 및 한국연구재단 (2020R1F1A1065638)의 지원을 받아 수행되었습니다.

◆ First Author : Inha University Department of Electrical and Computing Engineering, jeong.lim@inha.edu, 학생회원
° Corresponding Author : Inha University Department of Electronic Engineering, yj.kim@inha.ac.kr, 정회원
논문번호 : 202204-049-B-RU, Received April 1, 2022; Revised April 19, 2022; Accepted April 19, 2022

Ⅰ. 서 론

최근 B5G/6G 네트워크의 출현으로 인하여 자율주행 자동차, 증강 현실 (AR) 등과 같이 딥러닝을 활용한 모바일 비전 어플리케이션이 주목받고 있다. 이러한 어플리케이션 사용자의 경험 품질 (QoE) 기준은 크게 1) 한정된 배터리 용량에 의존하는 모바일 단말의 에너지 효율성, 2) 추론 결과에 대한 정확도, 3) 단일 프레임을 처리하는 데 발생하는 총 처리 시간인 단대단 지연 시간, 4) 초당 처리되는 프레임의 수 (fps)로 총 네 가지이다. 이때, 모바일 사용자의 요구나 어플리케이션의 종류에 따라서 특정 기준이 다른 기준보다 더 중요할 수 있다.

모바일 단말은 한정적인 연산 능력과 메모리 크기, 배터리 용량을 갖기 때문에, 모바일 비전 어플리케이션은 주로 모바일 엣지 컴퓨팅 (MEC)의 도움을 받는다. 이때 MEC 서버는 충분한 연산 능력 (GPU)를 갖고 있다. 따라서 모바일 단말은 무선 네트워크 (5G, WiFi) 상태와 배터리 상태에 따라서 모바일 단말 자체에서 딥러닝 추론(모바일 컴퓨팅)을 진행하거나, 입력 데이터를 MEC 서버로 전송하여 MEC 서버가 대신 딥러닝 추론 (MEC)을 진행할 수 있다. 일반적으로 모바일 컴퓨팅은 네트워크 오버헤드를 발생시키지 않지만, 부족한 연산 능력으로 인하여 프로세싱 오버헤드를 발생시킨다[1]. 반면 MEC 서버는 충분한 연산 능력으로 인하여 모바일 단말의 처리 오버헤드는 줄여줄 수 있지만, 모바일 단말과 MEC 서버 사이 데이터 송·수신 과정에서 네트워크 오버헤드가 발생할 수 있다. 과거에는 모바일 단말의 DVFS (Dynamic Voltage and Frequency Scaling)과 실시간으로 변화하는 네트워크 환경을 고려하여 모바일 컴퓨팅을 할 것인지, MEC를 할 것인지를 결정하는 이분법적 연구들이 진행되어왔다[1-8].

한편, 모바일 비전 어플리케이션의 경험 품질 향상을 위하여 딥러닝 모델 파티셔닝에 대한 연구가 진행되고 있다[9-13]. 이러한 연구는 기존의 이분법적 연구와는 달리, 딥러닝 모델이 연속된 레이어로 구성되어 있는 점을 활용하여 레이어 별 추론을 모바일 단말과 MEC 서버가 나누어 처리하게 된다. 딥러닝 모델이 나누어지는 지점을 파티션 포인트라 하며, 기존 연구와 같이 DVFS, 네트워크 제어와 함께 파티션 포인트를 추가로 제어함으로써, 모바일 단말과 MEC 서버 사이의 네트워크 및 프로세싱 오버헤드를 조절할 수 있다. 이를 통해, 추가적인 사용자 경험 품질 향상을 기대할 수 있다. 하지만 기존 딥러닝 모델 파티셔닝을 다룬 연구는 앞서 소개한 네 가지의 사용자 경험 품질 기준 중, 하나 혹은 두 가지의 기준만을 다루었다.

본 연구에서는 딥러닝 모델 파티셔닝이 사용자 경험 품질 기준 중 1) 모바일 단말의 에너지 소모, 2) 단대단 지연 시간, 3) 처리 fps 세 가지 기준에 대해 높은 변동성을 보인다는 점을 활용하여, 동적 딥러닝 모델 파티셔닝 및 프로세싱/네트워크 자원 관리 모델을 제안한다. 해당 모델은 위 세 가지 경험 품질에 대한 성능을 보장한다.

이를 위하여 본 연구에서는 모바일 단말의 프로세싱과 네트워크 사용에 따른 에너지 소모와 처리 fps로 이루어진 사용자 만족도에 대한 비용을 최소화하는 문제를 생성한다. 이때 시간 평균 단대단 지연 시간을 보장한다. 이를 가상 큐 기반 Lyapunov 최적화를 활용하여 1) 파티션 포인트, 2) 모바일 단말의 DVFS, 3) 처리 fps를 동적으로 제어하는 ***RT-DMP*** 알고리즘으로 도출한다. 제안한 ***RT-DMP*** 알고리즘은 목표한 단대단 지연 시간을 위반하지 않으면서 비용을 최소화함을 증명한다. 또한, 시뮬레이션을 통해 제안한 ***RT-DMP*** 알고리즘을 검증한다.

Ⅱ. 시스템 모델

2.1 서비스 모델

그림 1은 본 연구의 시스템 구조로, 모바일 단말이 비전 어플리케이션을 사용하고, 모바일 단말과 MEC 서버 모두 사전 학습된 딥러닝 모델 m을 메모리에 올려두고 있다. 예를 들어, 모바일 단말에서 물체 인식을 위하여 모바일 단말과 MEC 서버 모두 YOLOv4-tiny 모델을 메모리에 올려둔다. 본 연구에서는 $t \in T = \{0,1,\cdots,T-1\}$로 나누어진 환경으로, 단위 시간 간격은 Δt (second)로 둔 시간에 따라 동작하는 시스템을 고려한다. 모바일 비전 어플리케이션은 매 타임슬롯 시작 추론을 위하여 w (bits) 크기의 $a(t)$ (frames) 개를 생성하고, 이는 모바일 단말의 추론을 위한 입력 데이터로 사용한다. 모바일 단말은 입력되는 데이터에 대해서 처리하고자 하는 데이터의 양 $b(t) \in \{0,\cdots,a(t)\}$을 결정한다.

사전 학습된 딥러닝 모델 m은 convolution, maxpool과 같이 특정 작업을 수행하는 여러 레이어로 구성된다. 이러한 레이어의 출력 결과는 연속된 다음 레이어의 입력이 된다. 본 연구에서는 모델 m을 N_m개의 레이어 그룹으로 나누어, $\mathrm{Seq}_m = \left\{l_m^1, l_m^2, \cdots l_m^{N_m}\right\}$로 표현하였다. 레이어 그룹 $l \in \mathrm{Seq}_m$은 추론 연산을

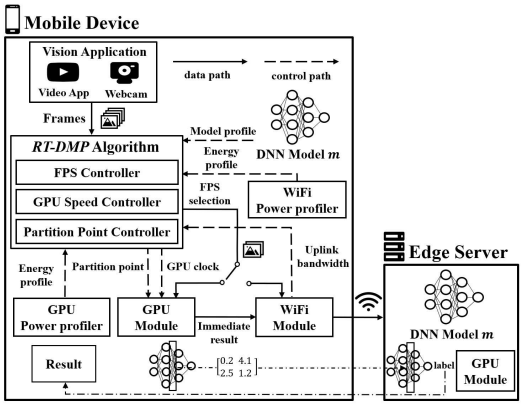

그림 1. 모바일 단말-엣지 서버 기반 딥러닝 모델 파티셔닝 시스템 구조
Fig. 1. Architecture of mobile-edge based DNN model partitioning system

분할하기 위한 가장 작은 단위이다. 이때, 레이어 그룹 l은 단일 레이어뿐 아니라, 병렬적으로 구성된 여러 개의 레이어와 shortcut connection으로 구성된 residual block 등을 모두 포함할 수 있다. 모바일 단말과 MEC 서버 모두에서 $b(t)$개의 프레임을 처리하기 위해 각 프레임은 Seq_m의 모든 레이어 그룹 연산을 거치게 된다. 따라서 일부 상위 레이어 그룹은 모바일 단말에서 프로세싱된 후, 그에 따른 결과를 MEC 서버로 전송한 후, 남아있는 나머지 레이어 그룹은 MEC 서버에서 프로세싱하여 추론을 진행할 수 있다. 즉, 딥러닝 모델 파티셔닝을 통하여 모바일 단말과 MEC 서버가 추론 연산을 분담한다. 이에 따라 모바일 단말은 모바일 단말과 MEC 서버 간 프로세싱 연산을 나누기 위하여 매 타임슬롯 t마다 파티션 포인트 $i(t) \in \{1,2,\cdots,N_m+1\}$를 결정한다. 결정된 파티션 포인트 $i(t)$에 따른 동작은 다음과 같다.

$$\begin{cases} i(t)=1: \text{MEC 서버가 } \mathrm{Seq}_m \text{를 처리}, \\ i(t)=N_m+1: \text{모바일 단말이 } \mathrm{Seq}_m \text{를 처리}, \\ otherwise: \text{모바일 단말이 } \left\{l_m^1,\cdots,l_m^{i(t)-1}\right\}\text{를 처리 후}, \\ \qquad\qquad \text{MEC 서버가 } \left\{l_m^{i(t)},\cdots,l_m^{N_m}\right\}\text{를 처리} \end{cases}$$

각 레이어 그룹 l_m^n은 구성된 레이어, 수행 작업, 필터의 영향으로 서로 다른 특성을 보인다. 예를 들어, 서로 다른 역할을 하는 레이어를 처리하는데 필요한 연산량은 다를 것이다. 또한, 같은 종류의 레이어라고 해도, 특성 맵을 생성하는 데 필요한 필터의 구성에 따라서 필요한 연산량은 다르다. 이러한 각 레이어 그룹의 특성을 반영하기 위하여 단일 비트를 처리하는 데 필요한 GPU 사이클 수를 나타내는 processing density ρ_m^n(cycles/bit)를 정의하였다. 또한, 각 레이어 그룹별로 입력 데이터 대비 출력 데이터 크기가 다르다. 이러한 비율을 bit conversion ratio σ_m^n(bits/bit)로 정의하였다.

파티션 포인트 $i(t)$에 따른 모바일 단말 (md)와 MEC 서버 (es)의 processing density와 bit conversion ratio는 다음과 같이 표현된다.

$$\begin{cases} \rho_{m,\mathrm{md}}^{i(t)}=\sum_{j=0}^{i(t)-1}\rho_m^j\prod_{i=0}^{j-1}\sigma_m^i, & \rho_{m,\mathrm{es}}^{i(t)}=\sum_{j=i(t)}^{N_m}\rho_m^j\prod_{i=0}^{j-1}\sigma_m^i \\ \sigma_{m,\mathrm{md}}^{i(t)}=\prod_{i=0}^{i(t)-1}\sigma_m^i, & \sigma_{m,\mathrm{es}}^{i(t)}=\prod_{i=0}^{N_m}\sigma_m^i \end{cases} \quad (1)$$

2.2 자원 및 에너지 모델

모바일 단말은 처리를 위하여 GPU를 탑재하고 있고, DVFS가 가능하여 매 타임슬롯 모바일 단말의 GPU 클럭 주파수 $s(t)=[0,s_{\max}]$ (cycles/second)를 조절할 수 있다. 모바일 단말은 결정된 파티션 포인트 $i(t)$에 따라서 $s(t)$를 조절한다. $s(t)$에 따른 GPU 에너지 소모량 $E_s(s(t))$는 다음과 같다.

$$E_s(s(t))=(\alpha s(t)^{\gamma}+\beta)\Delta t \ \text{ (Joule)} \quad (2)$$

α,β,γ는 모바일 단말의 GPU 모델에 따라 결정되는 상수이다. MEC 서버는 모바일 단말보다 연산 능력이 좋으므로 $s_{\max}$보다 높은 s_{es} (cycles/second)의 고정된 GPU 클럭 주파수를 갖는다.

모바일 단말은 매 타임슬롯 t마다 파티션 포인트 $i(t)$에 따라서 MEC 서버로 5G, WiFi와 같은 무선 네트워크를 통해 데이터를 전송한다. 이때, 네트워크 무선 채널은 채널 상태, 모바일 단말의 path loss 등에 의해 매 타임슬롯 변화한다. 따라서 매 타임슬롯마다 업링크 네트워크를 통해 전송 가능한 데이터의 양을 $d(t)\in[0,d_{\max}]$(bits)라 정의하였다. 이로 인해 모바일 단말은 매 타임슬롯마다 MEC 서버로 $b(t)w\sigma_{m,\mathrm{md}}^{i(t)}$를 전송한다. 네트워크 에너지 소모량 $E_n(t)$는 다음과 같다.

$$E_n(t)=E_n^{\max}\frac{b(t)w\sigma_{m,\mathrm{md}}^{i(t)}}{d(t)} \ \text{ (Joule)} \quad (3)$$

$E_n^{\max}$는 업링크 자원 $d(t)$를 최대로 사용했을 때

발생하는 최대 네트워크 에너지 소모량이다. 따라서 매 타임슬롯 t에 모바일 단말에서 발생하는 총에너지 소모 $E(t)$는 $E_s(s(t))$와 $E_n(t)$의 합으로 표현된다.

2.3 단대단 지연 모델

단일 프레임을 처리하는 데 발생하는 단대단 지연 시간은 다음과 같이 네 가지로 이루어진다.

1) 모바일 단말에서의 프로세싱 시간: $i(t)$에 따라 모바일 단말에서 처리해야 하는 GPU 사이클 수는 $w\rho_{m,\mathrm{md}}^{i(t)}$ (cycles)이고, GPU 클럭 주파수는 $s(t)$ (cycles/second)이므로, 모바일 단말 프로세싱에 걸리는 시간은 $w\rho_{m,\mathrm{md}}^{i(t)}/s(t)$ (seconds)이다.

2) 결과 업로드 시간: 모바일 단말에서 MEC 서버로 전송하는 데이터의 양은 $w\sigma_{m,\mathrm{md}}^{i(t)}$ (bits)이고, 업링크 대역폭은 $d(t)/\Delta t$ (bits/second)이므로 데이터 전송 시간은 $w\sigma_{m,\mathrm{md}}^{i(t)}\Delta t/d(t)$이다.

3) MEC 서버에서의 프로세싱 시간: MEC 서버에서 처리해야하는 GPU 사이클 수는 $w\rho_{m,\mathrm{es}}^{i(t)}$ (cycles)이고, GPU 클럭 주파수는 s_{es} (cycles/second)이므로 MEC 서버 프로세싱에 걸리는 시간은 $w\rho_{m,\mathrm{es}}^{i(t)}/s_{\mathrm{es}}$ (seconds)이다.

4) 최종 결과 다운로드 시간: 비전 어플리케이션에서 최종 추론 결과 (bounding box 좌표, 신뢰도, 클래스)는 전체 추론시간 대비 매우 적은 비중을 차지하므로 최종 데이터 전송에 대한 시간은 생략한다.

따라서 단대단 지연 시간 $R(t)$은 다음과 같다.

$$R(t)=\frac{w\rho_{m,\mathrm{md}}^{i(t)}}{s(t)}+\frac{w\sigma_{m,\mathrm{md}}^{i(t)}}{d(t)}\Delta t+\frac{w\rho_{m,\mathrm{es}}^{i(t)}}{s_{\mathrm{es}}}\ (\text{seconds}) \tag{4}$$

Ⅲ. 문제 정의 및 제안 알고리즘

3.1 문제 생성

본 절에서는 모바일 비전 어플리케이션 경험 품질 개선 문제를 세우고, 이를 해결하기 위한 알고리즘에 대해 살펴본다. 우선 파티션 포인트 $i(t)$와 모바일 단말 내에서 입력되는 프레임 중 처리할 프레임 수 $b(t)$, 모바일 GPU 클럭 주파수 $s(t)$를 제어하여 추론을 진행하며 평균 단대단 지연 시간을 보장하는 동시에 long-term 모바일 단말 에너지 소모 (E)- fps 유틸리티 (U)의 가중 합을 최소화하는 문제는 다음과 같다.

$$\text{(P1):}\quad \min_{\boldsymbol{i},\boldsymbol{b},\boldsymbol{s}}\left[\overline{E}-WU\left(\frac{\overline{b}}{a}\right)\right]$$

$$s.t.\begin{cases}(C1):\overline{R}\le r_{\max},\\ (C2):b(t)w\sigma_{m,\mathrm{md}}^{i(t)}\le d(t),\ \forall t\in T,\\ (C3):b(t)w\rho_{m,\mathrm{md}}^{i(t)}\le s(t)\Delta t,\ \forall t\in T,\\ (C4):i(t)\in\{1,2,\cdots,N_m+1\},\ \forall t\in T,\\ (C5):b(t)\in\{0,\cdots,a(t)\},\ \forall t\in T,\\ (C6):s(t)=[0,s_{\max}],\ \forall t\in T.\end{cases}$$

$\boldsymbol{i}=\{i(t)|t\in T\}$, $\boldsymbol{b}=\{b(t)|t\in T\}$, $\boldsymbol{s}=\{s(t)|t\in T\}$이며, W는 에너지 감소와 fps 유틸리티 사이 가중치이다. 또한, 유틸리티 함수 $U(\cdot)$은 연속 미분 가능하고, 증가하는 성질을 지닌 concave 함수이다. (C1)은 시간 평균 단대단 지연 시간에 대한 제약사항, (C2), (C3)은 업링크 네트워크와 GPU 프로세싱 용량의 제약사항, (C4), (C5), (C6)은 파티션 포인트 $i(t)$, 처리하고자 하는 프레임 수 $b(t)$, 모바일 GPU 클럭 주파수 $s(t)$이 선택 가능한 영역이다. 본 연구에서는 시스템 상태 $\boldsymbol{\Omega}(t)=(a(t),d(t))$가 매 타임슬롯 i.i.d 하다고 가정한다.

유틸리티 함수 $U(\cdot)$는 시간 평균에 대하여 비선형 함수이므로, 매 타임슬롯마다 문제를 풀어나가기 어렵다. 따라서 매 타임슬롯 푸는 문제로 변환하기 위해서 straight line set과 보조 변수를 추가한다. 먼저 임의의 상수 $(\kappa_{\min},\kappa_{\max})$로 이루어진 straight line set $K=\{\kappa|\kappa_{\min}\le\kappa\le\kappa_{\max}\}$을 정의한다. (P1)에 제약사항을 추가한 문제는 다음과 같다.

$$\text{(P2):}\quad \min_{\boldsymbol{i},\boldsymbol{b},\boldsymbol{s}}\left[\overline{E}-WU\left(\frac{\overline{b}}{a}\right)\right]$$

$$s.t.\quad\begin{cases}(C1)-(C6),\\ \dfrac{\overline{b}}{a}\in K.\end{cases}$$

이때, (P2) 목적함수의 최솟값은 추가된 제약사항으로 인해 (P1)의 최솟값보다 작을 수 없다. 하지만 straight line set K가 매우 크다면, (P2)는 (P1)과 동일한 최적값을 달성한다. 다음으로, (P2)를 시간 평균의 함수로 나타내기 위하여 보조 변수 $\kappa(t)\in K$를 소개한다. $\kappa(t)$는 Jensen's inequality로 인하여 다음과 같은 관계를 갖는다.

$$\overline{\kappa}\in K\quad\text{and}\quad\overline{U(\kappa)}\le U(\overline{\kappa}) \tag{5}$$

마지막으로, 보조 변수 $\kappa(t)$를 추가한 문제는 다음

과 같다.

$$\text{(P3):} \quad \min_{\boldsymbol{i},\boldsymbol{b},\boldsymbol{s},\boldsymbol{\kappa}} [\overline{E} - WU(\overline{\kappa})]$$
$$s.t. \quad \begin{cases} (C1)-(C6), \\ (C7): \overline{\kappa} \le \dfrac{\overline{b}}{a}, \\ (C8): \kappa(t) \in K, \forall t \in T. \end{cases}$$

이때, $\boldsymbol{\kappa} = \{\kappa(t) | t \in T\}$ 이며 보조 변수 $\kappa(t)$는 가상의 처리 fps로, $\frac{b(t)}{a(t)}$는 $\kappa(t)$를 시간 평균 관점에서 쫓아가는 방식으로 동작하게 된다. $(i(t), b(t), s(t))$를 제어하여 시간 평균 에너지 소모와 시간 평균 $\frac{b(t)}{a(t)}$로 이루어진 fps 유틸리티의 가중합을 최소화하는 (P1)의 문제는 보조 변수 $\kappa(t)$의 추가로, $(i(t), b(t), s(t), \kappa(t))$를 제어하여 시간 평균 에너지 소모와 시간 평균 fps 유틸리티 가중치 합을 최소화하는 문제로 변환되었다. 즉, 보조 변수 $\kappa(t)$의 추가로 매 타임슬롯 풀어나갈 수 있는 문제로 변환되었다. (P3)의 목적함수의 최적값은 (P1)의 목적함수 최적값과 같다.

3.2 알고리즘 유도

먼저, 초기 큐 값 $F(0), G(0) < \infty$을 갖는 가상 큐 $F(t), G(t)$를 정의한다. $(F(t), G(t))$은 다음 관계를 갖는다.

$$F(t+1) = [F(t) - r_{\max}]^+ + R(t), \tag{6}$$

$$G(t+1) = \left[G(t) - \frac{b(t)}{a(t)} \right]^+ + \kappa(t). \tag{7}$$

이때 만약 큐 $F(t)$와 $G(t)$가 발산하지 않는다면, 제약사항 (C1)과 (C7)이 만족하게 된다.

가상의 큐 $F(t)$와 $G(t)$를 사용하여, Lyapunov drift와 시간 평균 에너지와 fps 유틸리티 가중치 합에 V를 곱한 합을 나타내는 Lyapunov drift-plus-penalty를 최소화하는 알고리즘으로 도출할 수 있다.

3.3 알고리즘 설명

본 절에서는, 실시간 딥러닝 모델 파티셔닝 방법인 ***RT-DMP***를 제안한다. 제안하는 방법은 오직 입력되는 가상의 큐 $(F(t), G(t))$, 프레임의 수 $a(t)$, 업링크 네트워크 자원 $d(t)$ 정보만을 이용하여 $(\kappa(t), i(t), b(t), s(t))$을 결정한다.

1) 보조 변수 $\kappa(t)$: 매 타임슬롯 t마다 가상의 큐 $G(t)$에 따라 ***RT-DMP***는 다음 문제를 해결한다.

$$\min_{\kappa(t)} -VWU(\kappa(t)) + G(t)\kappa(t), \quad s.t.\ \kappa(t) \in K. \tag{8}$$

식 (8)은 convex 최적화 문제이기 때문에. 다음과 같은 해가 도출된다.

$$\kappa(t) = \left[(U')^{-1}\left(\frac{G(t)}{VW}\right) \right]_{\kappa_{\min}}^{\kappa_{\max}} \tag{9}$$

유틸리티 함수 $U(\cdot)$이 연속 미분 가능하면서 증가하는 concave 함수이기 때문에, $(U')^{-1}(\cdot)$은 감소함수이다. 식 (9)에서 $G(t)$가 감소하게 되면, 즉, 제약사항 (C4)를 만족하게 되면, 간접적으로 $\frac{b(t)}{a(t)}$를 증가시키기 위하여 $\kappa(t)$를 증가하여 $G(t)$가 증가한다.

2) 그 외 변수 $(i(t), b(t), s(t))$: 매 타임슬롯 t마다 가상의 큐 $F(t)$와 $G(t)$에 따라 ***RT-DMP***는 다음 문제를 해결한다.

$$\begin{aligned} &\min_{i(t),b(t),s(t)} [\mathrm{IBS}(i(t), b(t), s(t)) \\ &= V\left\{ (\alpha s(t)^\gamma + \beta)\Delta t + E_n^{\max} \frac{b(t) w \sigma_{m,\mathrm{md}}^{i(t)}}{d(t)} \right\} \\ &+ F(t)\left\{ \frac{w\rho_{m,\mathrm{md}}^{i(t)}}{s(t)} + \frac{w\sigma_{m,\mathrm{md}}^{i(t)}}{d(t)}\Delta t + \frac{w\sigma_{m,\mathrm{es}}^{i(t)}}{s_{\mathrm{es}}} \right\} - G(t)\frac{b(t)}{a(t)} \Bigg] \\ &s.t. \begin{cases} b(t) w \sigma_{m,\mathrm{md}}^{i(t)} \le d(t), \\ b(t) w \rho_{m,\mathrm{md}}^{i(t)} \le s(t)\Delta t, \\ i(t) \in \{1, 2, \cdots, N_m + 1\}, \\ b(t) \in \{0, \cdots, a(t)\}, \\ s(t) = [0, s_{\max}]. \end{cases} \end{aligned} \tag{10}$$

식 (10)에서 $(i(t), b(t), s(t))$는 업링크 네트워크 자원 제약과 GPU 프로세싱 자원 제약으로 인해 서로 얽혀있는 문제로, 독립적인 문제로 분해될 수 없다. 따라서 삼차원의 도메인에서 가능한 모든 경우의 수를 확인 후, 변수 결정이 가능하다. 문제의 복잡도를 낮추기 위해 고정된 $i(t) \in \{1, 2, \cdots, N_m + 1\}$와 $b(t) \in \{0, \cdots, a(t)\}$에 대해서 식은 다음 $s(t)$에 대한 일차원의 convex 최적화 문제로 변환된다.

$$s_{i,b}(t) = \arg\min_{s(t)} \left[V\alpha s(t)^\gamma \Delta t + F(t)\frac{w\rho_{m,\mathrm{md}}^{i}}{s(t)} \right]$$

$$s.t. \begin{cases} b(t) w \rho_{m,\mathrm{md}}^{i(t)} \le s(t) \Delta t, \\ s(t) = [0, s_{\max}]. \end{cases} \quad (8)$$

식 (11)의 $s_{i,b}(t)$는 convex 함수로, 이에 대한 최솟값으로 최적값이 구해진다.

$$s_{i,b}(t) = \left[\sqrt[\gamma+1]{\frac{F(t) w \rho_{m,\mathrm{md}}^{i}}{V \alpha \gamma \Delta t}} \right]_{\frac{b w \rho_{m,\mathrm{md}}^{i}}{\Delta t}}^{s_{\max}} \quad (12)$$

이를 가능한 파티션 포인트 $i(t)$와 처리 프레임 개수 $b(t)$에 대해 비교하는 방식의 알고리즘은 다음과 같다.

$(i(t), b(t), s(t))$ 결정 알고리즘

Input: $F(t), G(t), a(t), d(t)$,
Output: $(i(t), b(t), s(t))$,
(Initialization)
1: $x \leftarrow \infty$.
(Iteration)
2: for $i(t) \in \{1,2,\cdots,N_m+1\}$ do
3: for $b(t) \in \{0,\cdots,a(t)\}$ do
4: Find $s_{i,b}(t)$ by eq (12).
5: Update $x = \min[x, \mathrm{IBS}(i,b,s_{i,b}(t))]$.
6: if $x = \mathrm{IBS}(i,b,s_{i,b}(t))$ then
7: Update $(i(t), b(t), s(t)) = (i,b,s_{i,b}(t))$.
8: end if
9: end for
10: end for

$(i(t), b(t), s(t))$ 결정 알고리즘은 $a(t)$의 최댓값이 $A_{\max}$일 때, $(N_m+1)A_{\max}$ 회의 반복을 요구한다.

3.4 알고리즘 성능

본 연구에서 제안한 알고리즘 ***RT-DMP***는 시간 평균 에너지 소모와 fps 유틸리티의 가중 합에 대해 모든 제약사항을 만족하면서 다음과 같은 성능을 보인다.

$$\overline{E} - W\overline{U(\kappa)} \le \overline{E_{P3}^{*}} - W\overline{U(\kappa_{P3}^{*})} + \frac{B}{V} \quad (13)$$

이때, $\overline{E_{P3}^{*}} - W\overline{U(\kappa_{P3}^{*})}$는 (P3)의 에너지 소모와 fps 유틸리티의 가중치 합의 최솟값이다.

Ⅳ. 시뮬레이션

이 절에서는 시뮬레이션을 통하여 ***RT-DMP*** 알고리즘이 모바일 비전 어플리케이션 환경에서의 동작을 검증한다.

4.1 시뮬레이션 환경

본 시뮬레이션에서는 모바일 단말 사용자가 사물 추론 비전 어플리케이션을 사용하는 것을 고려하였다. 사용하는 딥러닝 모델은 YOLOv4-tiny[14]로, 모델을 12개의 레이어 그룹으로 나눈 뒤 각 특성을 그대로 반영하였다. 입력되는 프레임의 수는 평균 60개로 설정하였고, 모바일 단말은 (114-1300)MHz의 GPU 클럭 주파수 설정이 가능하며, MEC 서버의 GPU 클럭 주파수는 RTX3090 스펙으로 설정하였다. 모바일 단말의 에너지 및 하드웨어 특성은 Nvidia Jetson TX2의 스펙으로 설정하였다. 또한, 업링크 네트워크로 보낼 수 있는 데이터 양은 (10.864-63.196)Mbps로 설정하였다. 모바일 단말의 GPU 에너지 모델에 따른 값은 $\alpha = 3.8351 \times 10^{-8}$, $\beta = 0.7312$, $\gamma = 2.6343$로 설정하였다. 시뮬레이션은 총 1000 타임슬롯 동안 실행되었다.

4.2 시뮬레이션 결과

그림 2는 본 연구에서 제안한 알고리즘의 성능 검증을 위한 시뮬레이션 결과이다. 그림 2-(a)는 요구 단대단 지연 시간에 따른 처리 fps와 프레임당 발생하는 에너지에 관한 결과를 보여준다. 결과에 따르면, 요구 단대단 지연이 길어질수록 더 높은 처리 fps를 갖는 것을 확인할 수 있다. 이러한 결과는 요구되는 단대단 지연 시간이 길어질수록 프로세싱 및 네트워크 자원을 더 유연하게 사용할 수 있으므로 시스템은 더 많은 수의 프레임을 처리할 수 있기 때문이다. 하지만 모바일 단말의 프로세싱 자원 부족으로 인하여 모바일 단말의 프로세싱 한계에 다다르게 되면 처리 fps가 더 이상 증가하지 못하게 된다. 또한, 그림 2-(a)에서 단대단 지연 시간이 길어질수록 GPU 에너지는 감소한다. 이는 단대단 지연 시간이 길어질수록 모바일 단말이 더 작은 크기의 GPU 클럭 주파수 설정을 통하여 처리하기 때문이다. 이와 함께 긴 단대단 지연 시간에서는 MEC 서버로 더 많은 양의 프로세싱을 분담하여 네트워크 에너지는 증가한다. 하지만 모바일 단말의 에너지 소모는 GPU 사용에 따른 에너지가 주를 이루기 때문에, 단대단 지연 시간이 길어질수록 총에너지 소모는 감소한다.

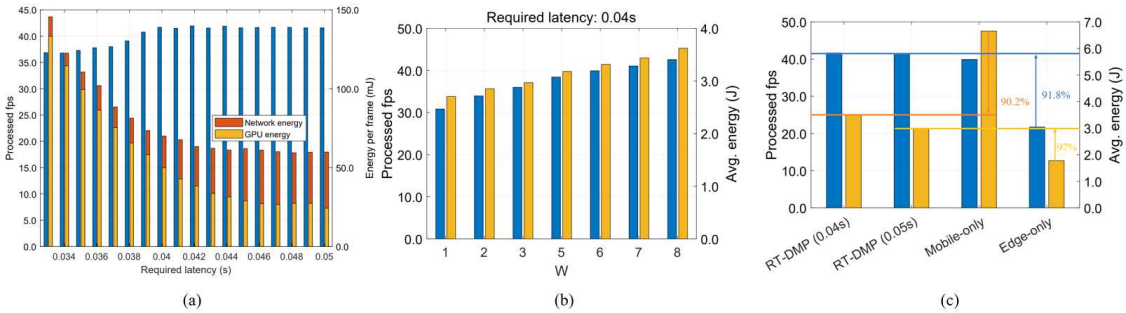

그림 2. 시뮬레이션 결과 (a) 단대단 지연 시간에 따른 처리 fps, 에너지 소모 (b) 가중치에 따른 처리 fps, 에너지 소모 (c) 타 알고리즘의 처리 fps, 에너지 소모
Fig. 2. Simulation result (a) processed fps, energy consumption with different target E2E latency (b) processed fps, energy consumption with different weight parameter (c) processed fps, energy consumption of other algorithm,

그림 2-(b)는 요구 단대단 지연 시간이 0.04초일 때, 에너지 감소와 처리 fps 사이 가중치 W 변화에 따른 처리 fps와 모바일 단말의 평균 에너지에 관한 결과를 보여준다. W의 증가는 시스템이 에너지 감소보다 처리 fps 증가를 중점으로 동작함을 의미한다. 결과에 따르면 W의 증가와 함께 처리 fps와 에너지 모두 증가한다. 이는 W의 증가함에 따라서 시스템이 처리 fps에 더 집중한 결과로, 처리 fps 증가에 따라 모바일 단말의 GPU 클럭 주파수 증가와 네트워크 사용 증가로 인한 결과라 볼 수 있다.

그림 2-(c)는 제안한 방법 *RT-DMP*와 타 알고리즘 간 처리fps와 평균 에너지의 비교 그래프이다. 타 알고리즘은 1) *Mobile-only*: 처리할 프레임의 개수와 GPU 클럭 주파수를 제어하는 모바일 단말 단독 추론 알고리즘과, 2) *Edge-only*: 처리할 프레임의 개수를 제어하는 MEC 서버 단독 추론 알고리즘이다. *RT-DMP* (0.04s)와 Mobile-only 알고리즘은 요구 단대단 지연 시간을 0.04초로 설정, *RT-DMP* (0.05s)와 *Edge-only* 알고리즘은 요구 단대단 지연 시간을 0.04초로 설정하여 비교하였다. 먼저 *RT-DMP* (0.04s)는 *Mobile-only*와 비교했을 때 4.5%의 처리 fps 성능 개선 및 90.2% 평균 에너지 소모 성능 개선을 보인다. 이는 *Mobile-only*가 요구 단대단 지연 시간의 만족과 높은 처리 fps 달성을 위하여 매우 높은 GPU 클럭 주파수 설정 때문이다. 다음으로 *RT-DMP* (0.05s)를 *Edge-only*와 비교했을 때 91.8%의 처리 fps 성능 개선을 보이지만,평균 에너지 소모 측면에서는 좋은 성능을 보이지 못한다. 이는 *Edge-only*가 GPU 에너지를 발생시키지 않기 때문이다. 하지만 *Edge-only*는 네트워크 상태에 큰 영향을 받기 때문에 0.05초보다 짧은 단대단 지연 시간을 보장하지 못하고, 22 이상의 처리 fps를 달성시키지 못한다. 즉, 모바일 단말의 에너지 소모를 제외한 사용자의 경험 품질 개선이 이루어지지 않았다. 이러한 결과는 딥러닝 모델 파티셔닝과 DVFS, 네트워크의 복합적 제어가 단대단 지연 시간, 처리 fps, 에너지 소모 성능 개선에 효과를 보임을 확인시켜 준다.

V. 결 론

모바일 비전 어플리케이션 사용자의 경험 품질은 모바일 단말의 에너지 소모, 하나의 프레임의 처리 시간을 나타내는 단대단 지연 시간, 처리 속도를 나타내는 처리 fps에 큰 영향을 받는다. 본 연구에서는 모바일 비전 어플리케이션 사용자 경험 품질 향상을 위한 딥러닝 모델 파티셔닝 기반 제어 알고리즘을 제안한다. 이는 모델 파티션 포인트, 입력 프레임 중 처리할 프레임의 개수, 모바일 단말의 GPU 클럭 주파수를 제어하여 단대단 지연 시간을 보장하면서 에너지 소모 감소 및 fps 유틸리티 증가시킬 수 있다. 시뮬레이션을 통해 제안하는 알고리즘이 요구된 단대단 지연 시간에 따라서 처리 fps와 에너지 소모 개선이 이루어짐을 검증하였다. 또한, 모바일 단독 추론 알고리즘과 MEC 서버 단독 추론 알고리즘과 비교하여 사용자 경험 품질 개선에 높은 효과가 있음을 검증하였다.

References

[1] J. Kwak, Y. Kim, J. Lee, and S. Chong, "DREAM: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE J. Sel. Areas in Commun.*, vol. 33, no. 12, pp. 2510-2523,

Dec. 2015.

[2] E. Lee and S. Lee, “Task offloading algorithm for mobile edge computing,” *J. KICS*, vol. 46, no. 2, pp. 310-313, 2021.

[3] Y. Kim, J. Kwak, and S. Chong, “Dual-side optimization for cost-delay tradeoff in mobile edge computing,” *IEEE Trans. Veh. Technol.*, vol. 67, no. 2, pp. 1765-1782, Feb. 2018.

[4] A. Galanopoulos, J. Romero, D. Leith, and G. Iosifidis, “AutoML for video analytics with edge computing,” in *Proc. IEEE INFOCOM*, pp. 1-10, May. 2021.

[5] M. Hanyao, Y. Jin, Z. Qian, S. Zhang, and S. Lu, “Edge-assisted online on-device object detection for real-time video analytics,” in *Proc. IEEE INFOCOM*, pp. 1-10, May. 2021.

[6] X. Ran, H. Chen, X. Zhu, and J. Chen, “DeepDecision: A mobile deep learning framework for edge video analytics,” in *Proc. IEEE INFOCOM*, pp. 1421- 1429, Honolulu, HI, Apr. 2018.

[7] L. Liu, H. Li, and M. Grueser, “Edge assisted real-time object detection for mobile augmented reality,” in *Proc. ACM MobiCom*, pp. 1-16, Los Cabos, Mexico, Oct. 2019.

[8] Y. Kim, H.-W. Lee, and S. Chong, “Mobile computation offloading for application throughput fairness and energy efficiency,” *IEEE Trans. Wireless Commun.*, vol. 18, no. 1, pp. 3-19, 2019.

[9] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, “Neurosurgeon: Collaborative intelligence between the cloud and mobile edge,” in *Proc. ACM ASPLOS*, pp. 615-629, Apr. 2017.

[10] E. Li, Z. Zhou, and X. Chen, “Edge intelligence: On-demand deep learning model co-inference with device-edge synergy,” in *Proc. 2018 Wkshp. Mob. Edge Commun.*, pp. 31-36, 2018.

[11] C. Shi, L. Chen, C. Shen, L. Song, and J. Xu, “Privacy-aware edge computing based on adaptive DNN partitioning,” in *Proc. IEEE GLOBECOM*, pp. 1-6, 2018.

[12] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, “Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services,” *IEEE Trans. Mob. Comput.*, vol. 20, no. 2, pp. 565-576, 2021.

[13] C. Hu, W. Bao, D. Wang, and F. Liu, “Dynamic adaptive DNN surgery for inference acceleration on the edge,” in *Proc. IEEE INFOCOM*, pp. 1423-1431, 2019.

[14] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: Optimal speed and accuracy of object detection,” Retrieved Mar. 15, 2022, from https://github.com/AlexeyAB/darknet

임 정 아 (Jeong-A Lim)

2020년 2월 : 인하대학교 전자공학과 졸업

2020년 8월~현재 : 인하대학교 전기컴퓨터공학과 석사 과정

<관심분야> 모바일 엣지 컴퓨팅

[ORCID:0000-0003-1604-3972]

김 영 진 (Yeongjin Kim)

2011년 2월 : 한국과학기술원 전자공학과

2013년 2월 : 한국과학기술원 전자공학과 석사

2018년 2월 : 한국과학기술원 전자공학과 박사

2018년 3월~2020년 2월 : 삼성전자 senior engineer

2020년 3월~현재 : 인하대학교 전자공학과 조교수

<관심분야> 모바일, 엣지, 클라우드 컴퓨팅

[ORCID:0000-0003-4482-2287]