JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 26, No. 6: 842~849, Jun. 2022

터커 분해 및 은닉층 병렬처리를 통한 임베디드 시스템의 다중 DNN 가속화 기법

김지민[1] · 김인모[2] · 김명선[3*]

Multi-DNN Acceleration Techniques for Embedded Systems with Tucker Decomposition and Hidden-layer-based Parallel Processing

Ji-Min Kim[1] · In-Mo Kim[2] · Myung-Sun Kim[3*]

[1]Undergraduate Student, Department of IT Convergence Engineering, Hansung University, Seoul, 02876 Korea
[2]Graduate Student, Department of IT Convergence Engineering, Hansung University, Seoul, 02876 Korea
[3*]Assistant Professor, Department of Applied Artificial Intelligence, Hansung University, Seoul, 02876 Korea

요 약

딥러닝 기술의 발달로 무인 자동차, 드론, 로봇 등의 임베디드 시스템 분야에서 DNN을 활용하는 사례가 많아지고 있다. 대표적으로 자율주행 시스템의 경우 정확도가 높고 연산량이 큰 몇 개의 DNN들을 동시에 수행하는 것이 필수적이다. 하지만 상대적으로 낮은 성능을 갖는 임베디드 환경에서 다수의 DNN을 동시에 수행하면 추론에 걸리는 시간이 길어진다. 이러한 현상은 추론 결과에 따른 동작이 제때 이루어지지 않아 비정상적인 기능을 수행하는 문제를 발생시킬 수 있다. 이를 해결하기 위하여 본 논문에서 제안한 솔루션은 먼저 연산량이 큰 DNN에 터커 분해 기법을 적용하여 연산량을 감소시킨다. 그다음으로 DNN 모델들을 GPU 내부에서 은닉층 단위로 최대한 병렬적으로 수행될 수 있게 한다. 실험 결과 DNN의 추론 시간이 제안된 기법을 적용하기 전 대비 최대 75.6% 감소하였다.

ABSTRACT

With the development of deep learning technology, there are many cases of using DNNs in embedded systems such as unmanned vehicles, drones, and robotics. Typically, in the case of an autonomous driving system, it is crucial to run several DNNs which have high accuracy results and large computation amount at the same time. However, running multiple DNNs simultaneously in an embedded system with relatively low performance increases the time required for the inference. This phenomenon may cause a problem of performing an abnormal function because the operation according to the inference result is not performed in time. To solve this problem, the solution proposed in this paper first reduces the computation by applying the Tucker decomposition to DNN models with big computation amount, and then, make DNN models run in parallel as much as possible in the unit of hidden layer inside the GPU. The experimental result shows that the DNN inference time decreases by up to 75.6% compared to the case before applying the proposed technique.

키워드 : 터커 분해, 다중 DNN, 멀티스트림, 임베디드 GPU

Keywords : Tucker Decomposition, Multi-DNN, Multi-Stream, Embedded GPU

Received 6 May 2022, **Revised** 13 May 2022, **Accepted** 27 May 2022

*** Corresponding Author** Myung-sun Kim (E-mail : kmsjames@hansung.ac.kr, Tel : +82-02-760-4045)
Assistant Professor, Department of Applied Artificial Intelligence, Hansung University, Seoul, 02876 Korea

Open Access **http://doi.org/10.6109/jkiice.2022.26.6.842** print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

최근 딥러닝 기술의 발전으로 인해 높은 정확도를 가지는 DNN 모델이 많아지고 있고, 딥러닝이 활용되는 분야 또한 계속해서 확장되고 있다. 많은 투자와 함께 활발한 연구가 진행되고 있는 자율주행, 드론, 로봇 분야에서도 DNN 모델을 활용하여 기능을 구성하는 경우가 많아지고 있다. 이들은 주로 임베디드 시스템 기반으로 구성되는데, 이는 임베디드 시스템이 크기가 작아 휴대성이 좋으며 낮은 소비전력을 가진다는 특징을 가지기 때문이다. 하지만 높은 성능을 가지는 GPU 다수를 클러스터 형식으로 사용하는 서버 환경에 비하여 낮은 성능을 가지기 때문에 높은 성능을 요구하는 DNN 모델 연산 시 제약사항이 있다.

자율주행, 드론, 로봇의 전체 시스템은 여러 기능으로 나누어져 있으며 하나의 기능을 수행하기 위해 여러 응용이 동시에 사용된다[1]. 예를 들어 자율주행 시스템의 경우 주행 기능에 필요한 차선 제어, 물체 식별, 보행자 감지 등은 DNN 모델을 사용하는 응용으로 구성되어 각 응용들을 동시에 수행하여 주행 기능이 동작한다. 이때 DNN 모델을 이용한 동작 시 잘못된 결과로 인한 오작동을 최소화하여 안전성을 보장하는 것이 중요하다[2]. 따라서 높은 정확도를 가지는 연산량이 큰 DNN모델로 응용을 구성하여 기능의 정확성을 높이는 것이 중요하다. 연산량이 큰 DNN 모델은 낮은 성능을 가지는 임베디드 환경에서 느린 추론 시간을 가지며, 추론 시간은 DNN 모델이 다중으로 실행되는 환경에서 더욱 느려진다. 자율주행 동작 시 연속적으로 입력되는 이미지에 대한 추론 시간이 지나치게 느려질 경우 추론 결과에 따른 동작이 제때 이뤄지지 않아 정상적인 기능을 못하는 문제가 발생한다.

본 논문에서는 정확도는 높지만 연산량이 큰 DNN 모델들이 다중으로 실행되는 환경에서 추론 시간이 현저하게 증가하는 문제를 해결하고자 한다.

Ⅱ. 문제 정의

대표적인 임베디드 시스템인 자율주행 자동차는 점점 더 높은 정확도를 가지는 연산량이 큰 DNN 모델을 여러 개 사용하는 형태로 발전되고 있다[2]. 자율주행 시스템에 사용되는 대표적인 DNN 응용인 객체 인식은 빠른 응답 특성을 달성하기 위해서 실시간으로 수집되는 입력 영상에 대한 DNN 모델 연산을 즉시 처리할 수 있어야 한다. 즉 DNN 응용 실행 시 DNN 모델은 빠른 추론 시간을 가져야 한다.

Table. 1 Comparison of the accuracy and the inference time between small and large DNN models

Model	ACC. (%)	Solo-run (ms)	Multi-run (ms)
Alex	79.07	10.8	58.3
Squeeze	80.42	16.6	94.3
VGG	90.38	52.5	334.8
RegY	95.34	232.8	1487.3

우리는 간단한 실험을 통하여 대표적인 DNN 모델들의 각 연산량에 따른 정확도, 추론 시간 그리고 여러 개가 함께 수행될 때의 특징 등을 비교하였다. 각 모델의 추론 시간을 확인하기 위해 자율주행 시스템에 널리 활용되는 NVIDIA의 Jetson AGX Xavier(이하 Xavier)[3]를 사용하여 실험을 진행하였다. 하나의 이미지에 대한 추론을 진행하여 추론이 완료되는 시점까지 걸리는 시간을 측정하였다. 실험에 사용된 DNN 모델들은 상대적으로 낮은 정확도를 가지면서 연산량이 작은 AlexNet[4], SqueezeNet[5]과 상대적으로 높은 정확도를 가지고 연산량이 큰 VGGNet16[6], RegNetY-32GF[7]를 사용하였다. 이하 각각의 모델 이름은 Alex, Squeeze, VGG, RegY로 표기하였다. 표 1에 표기된 각 모델의 정확도는 PyTorch[8]에서 제공하는 ImageNet 데이터 셋으로 훈련 시 Top-5 정확도를 의미한다[9]. Solo-run은 한 개의 DNN 모델이 Xavier의 시스템 리소스를 독점하여 실행한 경우이고, Multi-run은 6개의 DNN 모델을 동시에 실행한 경우를 의미한다. PyTorch, TensorFlow[10] 등의 딥러닝 프레임워크는 하나의 프로세스 당 하나의 DNN 모델의 추론을 진행한다. 따라서 Solo-run은 싱글 프로세스이고 Multi-run은 6개의 프로세스가 DNN 추론 과정을 수행하는 경우를 나타낸다.

앞서 말한대로 자율주행 시스템의 경우 점점 더 높은 정확도를 가지는 연산량이 큰 DNN 모델을 사용하는 것이 추세이다. 하지만 연산량이 큰 VGG와 RegY는 연산량이 작은 Alex와 Squeeze에 비하여 비교적 느린 추론 시간을 가진다. 이러한 연산량이 큰 DNN 모델의 추론 시간은 6개의 DNN 모델이 동시에 실행될 때 더 심해짐을 표를 통해서 알 수 있다. 연산량이 큰 DNN 모델은

DNN 모델 연산에 필수적인 GPU와 같은 시스템 자원을 경쟁하면서 사용할 때 긴 추론 시간을 가지기 때문에 DNN 모델 연산을 즉시 처리하기 힘들다.

이에 추가하여, 임베디드 GPU에서는 MPS (multi process service)[11]를 지원하지 않기 때문에 GPU는 한 시점에 하나의 프로세스에 대한 연산만 수행할 수 있다[12]. 즉, GPU 내부를 구성하는 병렬 수행이 가능한 복수 개의 SM (streaming multiprocessor)들을 여러 개의 프로세스가 공유하는 형태로 동시에 사용하는 것이 어렵다. 따라서 다수의 DNN 모델이 동시 실행되는 경우 GPU 내부의 하드웨어 스케줄러에 의하여 각 프로세스가 번갈아 가며 SM을 사용하게 된다. 이때 실행되는 프로세스가 바뀔 때마다 GPU 컨텍스트 스위칭 오버헤드에 의한 추가적인 지연시간이 발생한다[12]. 이러한 특성 때문에 동시 실행되는 DNN 모델의 개수가 증가할수록 각 DNN 모델의 추론 완료시간이 증가한다.

위 문제점을 해결하기 위하여 연산량이 큰 DNN 모델들의 추론 정확도는 유지하면서 연산량을 감소시키는 방법과 다수의 DNN 모델들이 동시에 수행될 때 최적으로 공유자원인 임베디드 GPU를 사용할 수 있는 프레임워크를 제안한다.

Ⅲ. 문제해결 방법 및 설계

본 장에서는 앞서 제시한 문제점을 해결하기 위한 두 가지 방법을 제시한다. 첫 번째는 DNN 모델의 추론 시간을 감소시키기 위해 DNN 모델 자체의 연산량을 감소시키는 방법이고, 두 번째는 다수의 DNN 모델이 동시에 수행되는 환경에서 임베디드 GPU를 최대한 병렬적으로 수행할 수 있는 DNN 레이어 단위 병렬처리 기법을 설명한다.

3.1. 터커 분해를 통한 연산량 감소

본 절에서는 텐서 분해 기법의 한 종류인 터커 분해 (Tucker decomposition) 기법에 대해 설명하고, DNN 모델 연산에 대부분을 차지하는 컨볼루션 은닉층 연산에 터커 분해를 적용하여 DNN 모델 전체의 연산량이 감소되는 과정을 설명한다. DNN 모델 연산량은 일반적으로 많이 사용하는 MAC (multiply-accumulate) 횟수로 정의한다.

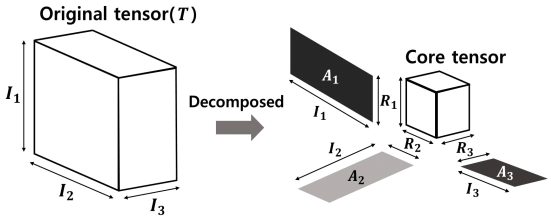

Fig. 1 Illustration of Tucker decomposition[13]

$I_1 \times I_2 \times \cdots \times I_m$의 크기를 가지는 고차원 텐서에 대해 터커 분해를 적용하면 한 개의 m차원 텐서 크기를 가지는 코어 텐서와 m개의 2차원 요인 행렬들로 분해된다[14]. 그림 1은 $I_1 \times I_2 \times I_3$크기를 가지는 3차원 원본 텐서(이하 T)를 기준으로 터커 분해의 동작 방식을 보여준다. 3차원 텐서 T가 터커 분해 과정을 거치면 한 개의 코어 텐서와 3개의 요인 행렬($A_1 \sim A_3$)로 분해된다. T의 각 축인 I_1, I_2, I_3을 행으로 고정하고 나머지 축을 펼쳐 만들어진 3개의 2차원 행렬에 대하여 각 특이값 분해를 적용해 각 축에 대한 랭크인 R_1, R_2, R_3를 구한다. 행렬에서 랭크란 행렬의 특이값 중 0인 값을 제외한 값의 개수를 의미한다. 예를 들어 I_1축을 행으로 고정하고 나머지 축을 펼쳐 만들어진 2차원 행렬은 $I_1 \times (I_2 * I_3)$의 크기를 가지고 있고, 이 행렬에 대해 특이값 분해를 적용하여 R_1을 구한다. R_1, R_2, R_3는 각 축을 기준으로 펼쳐 만들어진 2차원 행렬의 열의 값이 되어 $I_1 \times R_1$, $I_2 \times R_2$, $I_3 \times R_3$의 크기를 가지는 3개의 요인 행렬이 만들어지고, $R_1 \times R_2 \times R_3$의 크기를 가지는 T의 압축된 텐서로 표현되는 코어텐서가 만들어진다[13].

일반적으로 DNN 모델의 컨볼루션 은닉층의 필터는 4차원 텐서이고, 이를 본 논문에서는 가중치 텐서라고 한다. 가중치 텐서의 크기($K_w \times K_h \times C_i \times C_o$)는 순서대로 커널의 폭, 커널의 높이, 입력채널의 크기, 출력채널의 크기를 의미한다. 해당 가중치 텐서에 터커 분해 기법을 적용한다. 이 때 컨볼루션 은닉층의 커널 크기가 1×1일 경우 해당 은닉층의 가중치 텐서를 더 이상 분해할 필요가 없기 때문에, 이를 제외한 컨볼루션 은닉층에 터커 분해 기법을 적용한다.

가중치 텐서에 터커 분해를 적용하는 방식은 다음과 같다. 대부분의 컨볼루션 은닉층의 커널 축, 커널 높이를 나타내는 K_w와 K_h의 값은 3, 5, 7과 같이 비교적 작

은 값을 가지기 때문에 비교적 큰 값을 가지는 입력채널과 출력채널의 크기인 C_i와 C_o축에 대해서만 터커 분해를 적용한다. 즉, 가중치 텐서의 4개의 축에 대해 전부 분해를 하는 것이 아니라 C_i와 C_o축에 대해서만 분해를 하기 때문에 두 축에 대한 랭크인 R_3, R_4에 따라 $K_w \times K_h \times R_3 \times R_4$크기를 가지는 1개의 코어 텐서와 각각 $C_i \times R_3$, $C_o \times R_4$크기를 가지는 2개의 요인 행렬로 분해된다. 다시 말해 하나의 컨볼루션 은닉층의 가중치 텐서는 3개의 텐서로 분해되고, 분해된 3개의 텐서는 각 하나의 컨볼루션 은닉층의 가중치 텐서로 적용한다. 3개로 분해된 컨볼루션 은닉층의 가중치 텐서는 각각 $1 \times 1 \times C_i \times R_3$, $K_w \times K_h \times R_3 \times R_4, 1 \times 1 \times R_4 \times C_o$의 크기를 가진다. 터커 분해 적용 전 컨볼루션 은닉층의 파라미터 수는 $K_w * K_h * C_i * C_o$이고, 터커 분해를 적용하여 3개로 분해된 컨볼루션 은닉층의 파라미터 수는 $K_w * K_h * R_3 * R_4 + C_i * R_3 + R_4 * C_o (R_3 \leq C_i, R_4 \leq C_o)$이다. $R_3 = C_i, R_4 = C_o$을 만족하는 경우도 존재할 수 있지만 일반적인 경우 $R_3 \ll C_i, R_4 \ll C_o$ 관계를 가진다. 이를 통해서 터커 분해가 적용된 컨볼루션 은닉층의 파라미터 개수는 기존의 컨볼루션 은닉층의 파라미터 개수보다 작은 값을 가짐을 알 수 있다. 따라서 터커 분해를 적용하면 해당 DNN 모델의 MAC 횟수가 감소한다.

3.2. 은닉층 단위 병렬처리 프레임워크

그림 2는 3.1절에서 설명한 터커 분해 기법을 적용한 DNN 모델들이 DNN 레이어 단위로 임베디드 GPU를 최대한 병렬적으로 사용하여 동작하는 과정을 보여준다. 그림의 가로 점선을 기준으로 시스템이 시작하기 전인 오프라인 과정과 시스템 시작 후인 온라인 과정으로 구분한다.

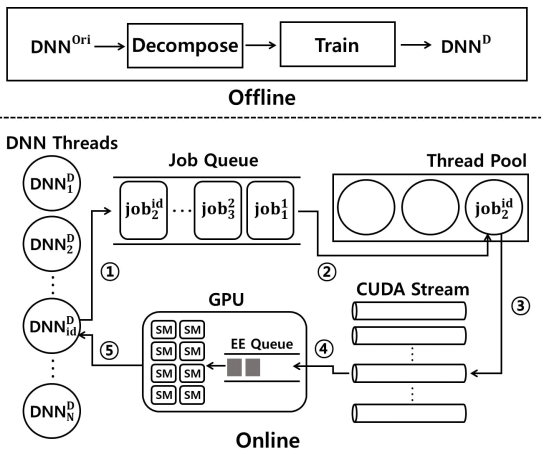

Fig. 2 Overview of the proposed framework

먼저 오프라인에서 터커 분해 기법을 적용한 n개 종류의 DNN 모델을 생성한다. DNN^{Ori}는 터커 분해 기법이 적용되지 않은 원본 DNN 모델을 의미한다. 오프라인에서 DNN^{Ori}에 터커 분해 기법을 적용하여 컨볼루션 은닉층을 분해한다. 분해 시 가중치 텐서 정보 손실로 인해 정확도가 떨어지게 되고, 떨어진 정확도를 최대한 회복시키기 위하여 재학습을 실시한다. 재학습이 끝난 DNN 모델을 그림에서는 DNN^D로 표기하였다. 오프라인 과정에서 터커 분해와 재학습을 통한 DNN^D의 생성과정은 모델 종류의 개수인 n번 수행한다.

시스템이 시작하면 추론하고자 하는 DNN^D의 개수인 N개의 DNN 모델 쓰레드를 생성하고 각각의 쓰레드에서 DNN^D를 실행시킨다. 이때 각 쓰레드에서 실행되는 DNN 모델을 $DNN_{id}^D (1 \leq id \leq N)$으로 표기하며 id는 DNN 모델 쓰레드의 인덱스 번호를 나타낸다. 그림 2의 온라인 과정은 메인 프로그램이 실행됐을 때의 임의의 id번째 DNN 모델 DNN_{id}^D의 동작 과정을 나타낸다. job은 DNN^D를 구성하는 연속된 은닉층의 집합을 의미하고, GPU를 점유하여 연산을 수행하는 단위이다. job을 구성하는 연속된 은닉층 집합에는 컨볼루션 은닉층, 풀링(pooling), 완전 연결(fully-connected) 은닉층 등이 포함될 수 있다. 이때 컨볼루션 은닉층의 경우, 터커 분해가 적용되지 않은 컨볼루션 은닉층 1개 또는 터커 분해가 적용된 3개의 컨볼루션 은닉층의 두가지 경우가 존재한다. 터커 분해가 적용된 실행 가능한 임의의 DNN 모델을 $DNN_{id}^D = \{job_1^{id}, job_2^{id}, \cdots, job_j^{id}\}$로 나타낼 경우, j는 해당 DNN_{id}^D의 총 job의 개수를 의미하고, 해당 모델을 구성하는 전체 레이어 개수가 k개라면 $j < k$을 만족한다.

그림 2에 나타낸 ①~⑤는 DNN_{id}^D의 추론이 수행되는 과정을 나타낸다. ①은 DNN_{id}^D를 실행중인 DNN 모델 쓰레드가 job_2^{id}를 FIFO 방식인 Job Queue에 추가하는 것을 나타낸다. 이어서 ②는 Job Queue에 있는 job_2^{id}가 빠져나갈 때 쓰레드 풀의 비어있는 쓰레드에 할당됨을

나타낸다. ③에서는 job을 할당받은 쓰레드는 job_2^{id}가 가지고 있는 연속적 레이어에 대한 연산을 수행하기 위해 해당 job_2^{id}에 대한 커널 함수를 발행하여 CUDA 스트림에 대기시킨다. ④에서 CUDA 스트림은 자신의 스트림에 존재하는 커널 함수를 GPU의 EE Queue[15]에 추가한 후 해당 커널 함수에 대한 GPU 연산을 수행한다. ⑤의 과정은 GPU에서 job_2^{id} 연산이 완료되면 대기중인 DNN_{id}^D에게 신호를 보내 DNN_{id}^D가 다음 job인 job_3^{id}를 Job Queue에 추가하도록 하는 과정을 보여준다.

다수의 DNN 모델을 동시 실행할 때 각각의 DNN^D를 하나의 프로세스 내에서 쓰레드로 생성하여 다중 프로세스로 DNN 모델의 추론을 수행할 때 발생할 수 있는 컨텍스트 스위칭 오버헤드를 제거하였다. 이에 더하여 각 쓰레드들이 하나의 스트림을 독립적으로 할당받아 사용할 수 있게 함으로써 GPU 내부의 SM들을 스트림 별로 동시에 사용할 수 있게 하였다. 결과적으로 GPU의 병렬 연산 능력을 최대한 사용하여 추론에 걸리는 시간을 효과적으로 줄일 수 있다.

Ⅳ. 실 험

본 장에서는 본 논문에서 제안한 기법의 효용성을 실험을 통해서 검증한다. 먼저 실험에 사용된 대상 시스템을 설명하고 이어서 실험의 결과들과 이에 필요한 분석 내용을 설명한다.

4.1. 실험 대상 시스템

2장에서 언급했던 Xavier를 대상 임베디드 시스템으로 사용하여 실험을 진행하였고, 표 2는 대상 시스템의 구체적인 사양을 나타낸다. 실험에 사용되는 DNN 모델들은 VGGNet16, ResNet152[16], Inception V3[17], RegNetY-32GF로 총 4종류를 사용하고, 이는 표기의 단순화를 위하여 각각 VGG, Res, Inception, RegY로 각 실험 결과 그래프에 표기한다. 학습에 사용된 데이터셋은 CIFAR10[18]을 사용하였다. 그림 3~4에서 $Ori.$는 DNN^{Ori}을 CIFAR10 데이터셋으로 학습한 모델을 사용한 결과를 의미하고, $Dcomp.$는 DNN^{Ori}에 터커 분해 적용 후 CIFAR10 데이터셋으로 재학습 시킨 DNN^D을 사용한 결과를 의미한다.

Table. 2 Target system specification

GPU	512-core Volta GPU with Tensor cores
CPU	8-Core ARM v8.2 64-bit CPU, 8MB L2, 4MB L3 Cache
Memory	32GB 256-bit LPDDR4x 2133MHZ - 137GB/s
OS	Linux kernel version 4.9.140
JetPack	ver. 4.2
CUDA ver.	CUDA 10.0.166

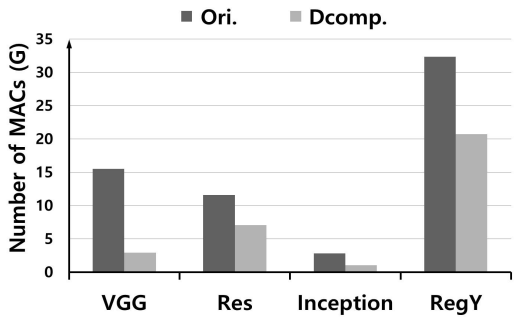

Fig. 3 Comparing the amount of computations per each DNN model

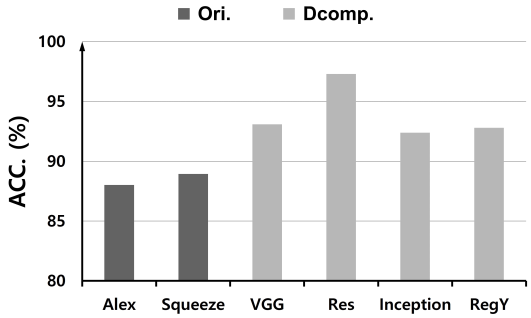

Fig. 4 Accuracy comparison between DNN models with small MAC counts and DNN models with large MAC counts approximated by Tucker decomposition

4.2. 터커 분해 적용에 따른 연산량 비교

그림 3은 터커 분해 적용 전, 후에 따른 DNN 모델의 연산량을 비교한 결과이다. 그림을 통해서 알 수 있듯이 터커 분해 적용 전 대비 VGG는 81%, Res는 38.6%, Inception은 62.8%, RegY는 35.8% 각각 연산량이 감소하였다. 모델 구조 특성상 각 컨볼루션 은닉층의 연산량이 상대적으로 크고, 1x1 컨볼루션 은닉층을 갖고 있지 않아서 모든 컨볼루션 은닉층에 터커 분해가 적용될 수 있는 VGG가 연산량 감소 비율이 가장 높았다.

4.3. 터커 분해 적용 시 정확도에 미치는 영향 분석

3.2절에서 설명하였듯이 본 논문에서는 큰 연산량을 수반하고 정확도가 상대적으로 높은 DNN 모델들의 MAC 횟수를 줄여 연산량을 줄이되 정확도의 희생을 최소화하는 것이 첫 번째 목표이다. 제안된 기법이 이를 만족하는지 확인하기 위하여 우리는 연산량이 큰 DNN 모델들에 터커 분해를 적용한 후 이들이 나타내는 정확도와 작은 연산량을 나타내는 DNN 모델들의 터커 분해가 적용되지 않았을 때의 정확도를 비교하였다.

그림 4는 이의 결과를 나타낸다. 연산량이 작은 Alex와 Squeeze는 터커 분해를 적용하지 않아도 최대 88.9%의 정확도를 가진다. 하지만 터커 분해를 적용한 모델들 중 정확도가 최소인 Inception의 경우 92.4%의 정확도를 가져 모든 실험에 사용된 연산량이 큰 DNN 모델들이 Alex나 Squeeze보다 높은 결과를 나타내었다.

4.4. 추론 시간 비교

본 절에서는 다수의 DNN 모델을 세 가지 환경 $Ori.(MP)$, $Dcomp.(MP)$, $Dcomp.(MT)$로 실행하여 추론 시간을 비교한다. 그림 5~6에서 $Ori.(MP)$, $Dcomp.(MP)$는 각각 CIFAR10으로 학습한 DNN^{Ori}, DNN^{D}를 멀티 프로세스 환경에서 실행한 결과이다. $Dcomp.(MT)$는 CIFAR10으로 학습한 DNN^{D}를 3.2절에서 제안한 멀티 쓰레드 환경에서 실행한 결과이다.

4.4.1. 다수의 동일 종류 DNN 모델 동시 실행

그림 5는 다수의 동일 종류 DNN 모델을 동시 실행했을 때의 추론 시간을 비교한 결과이다. VGG, Res, Inception은 8개, RegY는 7개의 DNN 모델을 동시 실행하였다. VGG와 RegY의 경우 하나의 DNN 모델당 추론 시간이 $Ori.(MP)$ 대비 $Dcomp.(MP)$에서 각 34.4%, 42.9% 감소하였다. 또한 $Ori.(MP)$ 대비 $Dcomp.(MT)$에서 각 62.9%, 75.6% 감소하였다.

Res와 Inception은 위의 경우와 다르게 $Ori.(MP)$ 대비 $Dcomp.(MP)$에서의 추론 시간이 각 12.4%, 16.3% 증가하였다. VGG와 RegY는 DNN 모델의 특성상 각 컨볼루션 은닉층의 연산량이 크기 때문에 GPU 리소스가 많이 요구된다. 이에 비해 Res와 Inception의 경우 연산량이 작은 여러 개의 컨볼루션 은닉층으로 이루어져 있기 때문에 각 컨볼루션 은닉층 연산의 GPU 리소스 요구량이 작다. 위와 같은 특성을 가진 DNN 모델에 터커 분해를 적용하게 되면, GPU 리소스 요구량이 더 작은 여러 개의 컨볼루션 은닉층으로 분해되어 전체 실행시간이 증가하게 된다.

$Ori.(MP)$ 대비 $Dcomp.(MT)$에서의 하나의 DNN 모델당 추론 시간은 각 56.1%, 47.9% 감소하였다. 멀티 쓰레드 환경에서는 멀티 스트림을 이용하여 여러 DNN 모델 쓰레드들의 커널 함수를 동시에 연산한다. 작은 연산량을 가진 컨볼루션 은닉층의 개수가 늘어나도 동시에 여러 DNN들이 GPU 리소스를 사용하기 때문에 추론 시간 감소 효과를 얻을 수 있다.

4.4.2. 다수의 다른 종류 DNN 모델 동시 실행

그림 6은 다수의 다른 종류의 DNN 모델을 동시 실행했을 때의 각 DNN 모델당 추론 시간을 비교한다. 실험에 사용된 서로 다른 DNN 모델은 VGG 3개, Res 1개, Inception 2개, RegY 2개로 총 8개의 DNN 모델을 사용했다. 4종류의 DNN 모델의 추론 시간은 $Ori.(MP)$ 대비 $Dcomp.(MP)$에서 최대 42% 감소하였다. 또한 $Ori.(MP)$ 대비 $Dcomp.(MT)$에서의 추론 시간은 최대 74.6% 감소하였다.

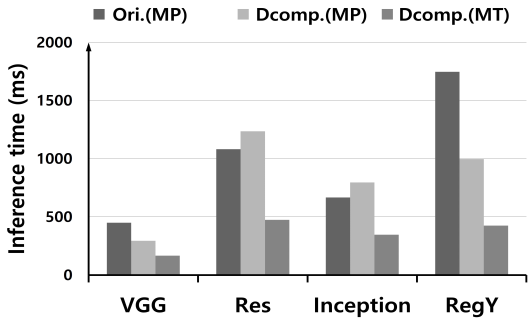

Fig. 5 Comparing the inference time of multiple homogeneous DNN models

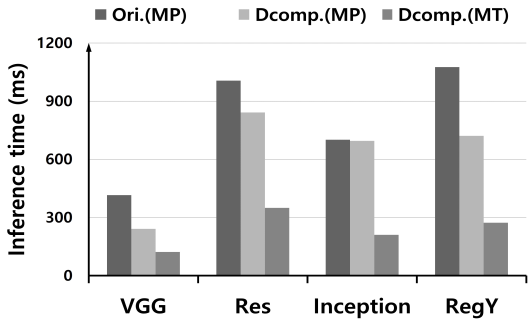

Fig. 6 Comparing the inference time of multiple heterogeneous DNN models

Ⅴ. 결 론

본 논문에서는 임베디드 시스템 환경에서 정확도가 높고 연산량이 큰 DNN 모델 다수를 동시 추론 시 각 DNN 모델의 추론 시간이 현저하게 증가하는 문제를 해결하고자 두 가지의 기법을 제안하였다. 첫 번째로 연산량이 큰 DNN 모델의 연산량을 감소시키기 위해 DNN 모델 연산에서 많은 부분을 차지하는 컨볼루션 은닉층에 터커 분해 기법을 적용하였다. 두 번째로 다수의 DNN 모델들이 GPU를 최대한 병렬적으로 수행할 수 있도록 하는 은닉층 병렬처리 프레임워크를 제안하였다. 이는 각 DNN 모델을 하나의 프로세스 내에서 쓰레드로 생성하여 쓰레드들이 하나의 스트림을 독립적으로 할당받아 GPU 내부 SM들을 스트림 별로 동시에 사용할 수 있다. 터커 분해 기법을 적용한 결과 실험에 사용된 DNN 모델의 연산량은 최대 81% 감소하였다. 두 가지 기법을 모두 적용한 결과 DNN 모델 추론 시간은 다수의 동일 종류 DNN 모델 동시 실행 시에 최대 75.6% 감소하였고, 다수의 다른 종류 DNN 모델 동시 실행 시에 최대 74.6% 감소하였다.

ACKNOWLEDGEMENT

This research was financially supported by Hansung University.

REFERENCES

[1] J. Peng, L. Tian, X. Jia, H. Guo, Y. Xu, D. Xie, H. Luo, Y. Shan, Y. Shan, and Y. Wang, “Multi-task ADAS system on FPGA,” in *Proceedings of the IEEE Interntaional Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Hsinchu, Taiwan, pp. 171-174, 2019. DOI: 10.1109/AICAS.2019.877615.

[2] J. Chandrasekaran, Y. Lei, R. Kacker, and D. R. Kuhn, “A Combinatorial Approach to Testing Deep Neural Network-based Autonomous Driving Systems,” in *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Proto de Galinhad, Brazil, pp. 57-66, 2021. DOI: 10.1109/ICSTW 52544.2021.00022.

[3] Jetson AGX Xavier Developer Kit [Internet]. Available: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolution neural networks,” in *Proceedings of the 25th Conference of Neural Information Processing Systems*, Lake Tahoe, Neveda, pp. 1106-1114, 2012.

[5] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size,” *arXiv preprint arXiv:1602.07360*, 2016. DOI: 10.48550/arXiv.1602.07360.

[6] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Proceedings of the International Conference on Learning Representations*, San Diego: CA, USA, pp. 1-14, 2015.

[7] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollar, “Designing Network Design Spaces,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle: WA, USA, pp. 10428-10436, 2020.

[8] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Brabury, G. Chanan, T. Kileen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Proceedings of the Neural Information Processing Systems (NeurIPS)*, Vancouver: BC, Canada, pp. 8024-8035, 2019.

[9] ImageNet 1-crop error rates [Internet]. Available: https://pytorch.org/vision/stable/modes.html.

[10] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steniner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A System for Large-Scale Machine Learning," in *Proceedings of the 12th USENIX Symposium on Operating System Design and Implementation (OSDI`16)*, Savannah: GA, USA, pp. 265-283, 2016.

[11] MPS(multi-process service) [Internet]. Available: https://docs.nvidia.com/deploy/mps/index.html.

[12] C. Lim and M. Kim, “ODMDEF: On-Device Multi-DNN Execution Framework Utilizing Adaptive Layer-Allocation on General Purpose Cores and Accelerators,” *IEEE Access*, vol. 9, pp. 85403-85417, Jun. 2021. DOI: 10.1109/ACCESS. 2021.308861.

[13] T. G. Kolda and B. W. Bader “Tensor Decompositions and Applications,” *SIAM Review*, vol. 51, no. 3, pp. 455-500, Sep. 2009.

[14] L. R. Tucker, “Some Mathematical notes on three-mode factor analysis,” *Psychometrika*, vol. 31, no. 3, pp. 279-311, Sep. 1966.

[15] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith, “GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed,” in *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Paris, France, pp. 104-115, 2017. DOI: 10.1109/RTSS.2017.00017.

[16] L. D. Nguyen, D. Lin, Z. Lin, and J. Cao, “Deep CNNs for microscopic image classification by exploiting transfer learning and feature concatenation,” in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, Italy, pp. 1-5, 2018.

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas: NV, USA, pp. 2818-2826, 2016.

[18] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” M. S. theses, University of Tront, Toronto: ON, Canada, 2009.

김지민(Ji-Min Kim)

2019~현재 한성대학교 IT융합공학부

※관심분야 : GPU Programing, DNN framework, Deep Learning

김인모(In-Mo Kim)

2015~2021 한성대학교 IT응용시스템공학과

2021~현재 한성대학교 대학원 IT융합공학과

※관심분야 : GPU Programing, DNN framework, Deep Learning

김명선(Myung-Sun Kim)

2000년~2002년 LG전자 전송연구소 주임연구원

2002년~2011년 삼성전자 DMC 연구소 책임연구원

2016년 서울대학교 전기컴퓨터공학부 박사 졸업

2016년~2019년 삼성전자 SR연구소 수석연구원

2019~현재 한성대학교 AI응용학과 조교수

※관심분야 : NPU(인공지능 프로세서), Linux kernel, HW/SW Co-design, 영상/음성 인식