im.data: 파이프라인 병렬처리를 통한 딥 러닝 훈련성능 개선 기법

(The method for improving deep learning training performance by using pipeline parallelism)

요 약 딥

**키워드** :

***Abstract***

Key words :

**1. 서론**

딥 러닝(Deep learning)은 기계학습(Machine learning) 한 분야로 최근 컴퓨팅 성능향상과 막대한 양의 데이터가 지속적으로 생성되면서 대규모 데이터를 학습한 딥러닝 관련 기술이 주목받고 있다. NAVER의 HyperCLOVA[]는 820억개의 파라미터를 가진 대규모 자연어처리(Natural language processing) 모델로 약 2089GB의 텍스트기반 데이터를 학습하여 자유대화, QnA에서 높은 추론능력으로 상대방의 대답을 분석하여 대화 의도를 예측해 인간과 자연스러운 대화가 가능할 정도의 높은 수준의 성능을 보였다. 이는, 번역, 이메일 작성기 등 실제 운영중인 서비스에 시범적용되어 사용되고 있으며 차후 virtual assistant[], 스마트홈[]과 같은 다양한 서비스에서도 적용 영역을 넓혀가고 있다[].

이러한 대규모 모델을 비롯한 딥러닝 모델은 병렬처리 성능이 높은 GPU에서 동작한다. 이에 GPU메모리 용량과 같은 하드웨어의 한계로 인해 훈련을 위해서는 CPU가 원본데이터를 스토리지로부터 읽어 들이고 전처리하여 훈련과정동안 학습모델에 지속적인 훈련데이터 공급이 요구되는데, 이러한 일련의 학습모델 훈련과정을 딥러닝에서 파이프라인이라고 한다. 최근 훈련데이터의 규모가 증가하면서 파이프라인 효율증가와 병렬처리에 대한 여러 연구들이 진행되고 있다. tf.data[]는 google에서 제공하는 딥러닝 라이브러리인 텐서플로우(TensorFlow)[]에 포함된 데이터 파이프라인 API로 사용자에게 데이터 전처리, 셔플링과 같은 다양한 기능과 높은 수준의 접근성을 제공한다. 그림 1은 tf.d

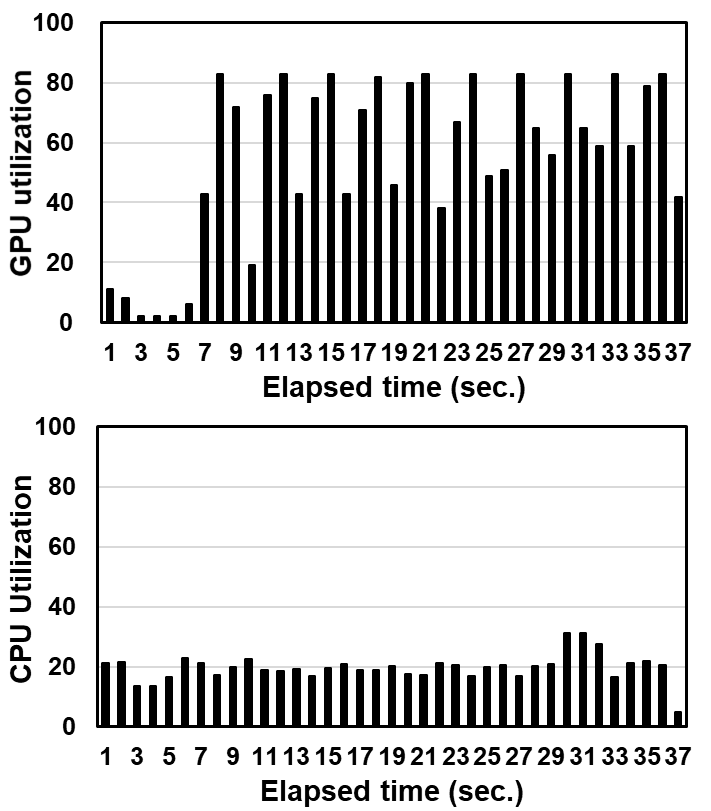


그림 mnist TFDATA파이프라인

그림 n im.data 파이프라인

ata를 사용하여 파이프라인을 구성하여 MNIST[] 데이터셋을 사용하여 CNN[]모델을 훈련시켰을 때 GPU와 CPU 사용률을 측정한 결과이다.

**2. 배경지식**

2.1

**3. 제안기법**

앞서 언급했던 바와 같이 기존 딥러닝 모델의 데이터 파이프라인 기법은 시스템 리소스 활용율이 낮아 하드웨어의 성능을 온전히 이끌어내지 못하는 문제가 존재한다는 것을 보였다. 본 장에서는 이러한 문제점을 해결하기 위해 새로운 파이프라인 병렬처리 기법(im.data, interleaved multithreading data)을 제안한다. 제안하는 기법은 파이프라인을 개별 쓰레드로 구성하여 학습모델의 훈련-검증 과정이 중첩되도록 동작한다. 또한 학습모델 검증 과정에서는 학습모델과 데이터를 멀티 쓰레드로 분할하여 병렬처리를 통해 전체적인 수행시간을 단축시킬 수 있다. 다음 절에서는 im.data의 구현과 동작과정에 대해 세부적으로 설명한다.

3.1 train-validation pipeline interleaving

im.data는 모델 훈련과정과 검증과정을 개별 쓰레드로 구분한다. 본 장에서는 학습모델의 훈련과정을 수행하는 쓰레드를 훈련 쓰레드, 검증과정을 수행하는 쓰레드를 검증 쓰레드로 부르기로 한다. 훈련 쓰레드는 데이터 전처리와 훈련데이터를 batch 단위로 분할하는 과정을 포함하고 학습모델을 훈련하여 역전파를 통해 가중치를 갱신하는 역할을 수행한다. 검증 쓰레드는 검증 데이터를 분할하고 학습모델의 추론과정으로 훈련 오차인 Loss를 계산하는 과정을 수행한다.

im.data 파이프라인은 훈련 쓰레드와 검증 쓰레드가 병렬적으로 동작하는데, 훈련 쓰레드가 종료되고 검증 쓰레드가 호출될 때 다음 Epoch에 대한 훈련 쓰레드가 호출되면서 이전 Epoch 검증 쓰레드와 다음 Epoch 훈련 쓰레드가 중첩되어 동작하도록 구성되어 있다.

3.2 model interference

각 쓰레드에서 동작하는 학습모델은 전역변수 객체로 정의되어 있다. 검증 쓰레드는 호출 시 학습모델의 객체를 복제하여 할당 받는다. 이는 훈련 쓰레드와 검증 쓰레드가 동시에 병렬적으로 동작하는 과정에서 동일한 학습모델 객체를 사용하게 될 경우 훈련 쓰레드가 가중치를 갱신하는 도중에 검증 쓰레드가 동일한 학습모델의 객체를 사용하여 추론을 수행하게 되면서 **가중치 값이 변경되면서** 이전 Epoch의 훈련 결과에 대한 검증을 수행할 수 없는 문제가 발생할 수 있기 때문이다. 이러한 가중치 간섭 문제를 해결하기 위해 학습모델 객체를 깊은 복사를 통해 복제하여 해당 Epoch에 대한 가중치 값을 보존하여 검증 쓰레드로 할당한다.

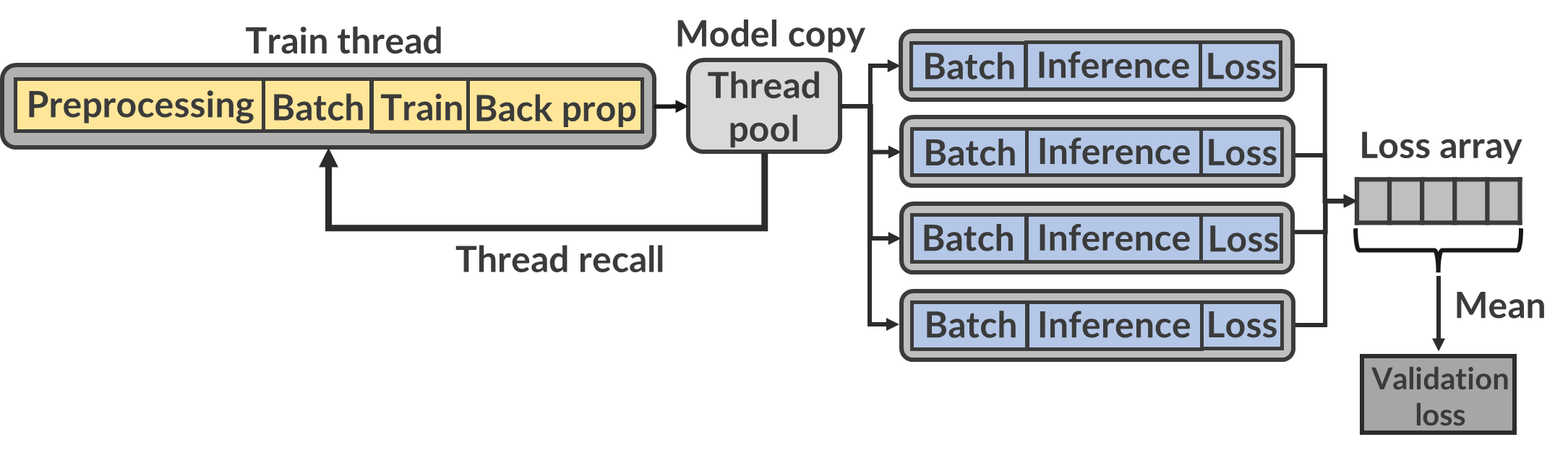


그림 n im.data 파이프라인

그림 n im.data 파이프라인

3.3 validation multithreading

검증 쓰레드는 Thread pool을 통해 멀티쓰레드로 동작한다. 훈련 쓰레드는 멀티쓰레딩을 사용하지 않는다. 이는, 만약 훈련과정을 멀티쓰레드로 구성하고 훈련데이터를 분할, 할당하여 병렬 처리할 경우 학습모델이 훈련데이터를 통해 모델 가중치를 갱신하는 시점이 각 쓰레드마다 다르기 때문에 가중치 stall[]로 인해 학습성능이 저하될 수 있기 때문이다. 또한, 여러 쓰레드가 하나의 학습모델에 동시에 접근하면서 발생하는 경쟁상태에 따른 성능저하 문제도 존재한다. [][아무개 .et al.]은 이러한 문제를 해결하기 위해 GPU 클러스터 환경에서 학습모델을 분할하여 개별 GPU에 할당해서 훈련시간을 단축시키는 결과를 보였다. 다만, 본 연구에서는 단일 GPU환경에서 진행하였기 때문에 훈련 쓰레드를 단일 쓰레드로 구성하였다.

그림 n은 i.data 파이프라인의 구조를 명시하는데, 훈련 쓰레드 동작이 종료되면 Thread pool에서 사용자가 지정한 개수만큼 검증 쓰레드를 생성한다. 이 과정에서 검증 데이 터를 쓰레드 수만큼 분할한다. 또한 공유자원

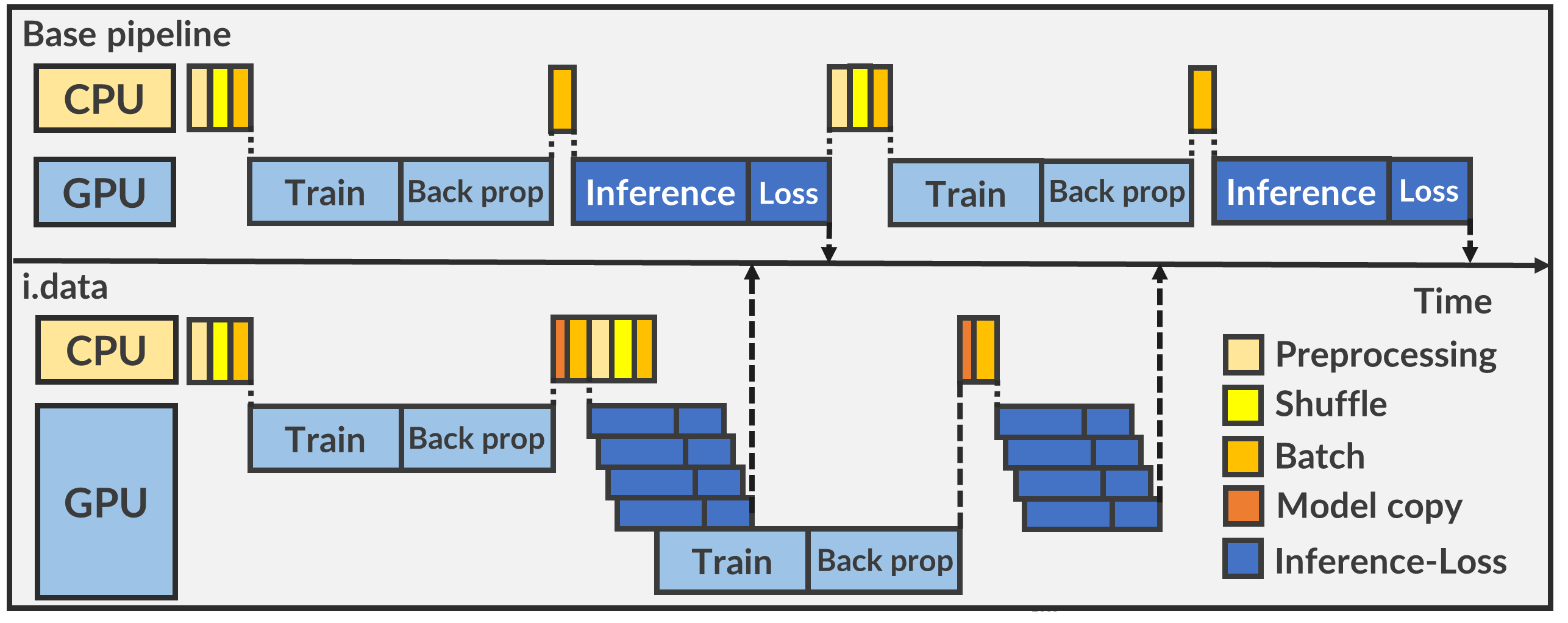


그림 n+1 i.data 파이프라인

그림 n+1 i.data 파이프라인

경쟁상태로 인해 발생할 수 있는 성능저하를 방지하기 위해 가중치가 갱신된 전역변수 학습모델 객체를 쓰레드 수만큼 복제하여 검증데이터와 각 쓰레드에 할당한다. 이후 검증쓰레드가 호출되는 동시에 다음 Epoch에 대한 훈련쓰레드가 호출되어 병렬적으로 동작한다.

검증 쓰레드는 검증과정이 종료되면 각 쓰레드에서 출력된 Loss의 평균값을 Epoch의 평균값으로 정의한다. 그림 n+1은 기존 파이프라인과 im.data 파이프라인이 하드웨어에서 동작하는 과정의 예시를 보인다. 초기상태에서는 **스토리지로부터 훈련데이터를 읽어들여 학습모델이 인식 가능한 데이터형식으로 변환하고 편향을 줄이기 위해 훈련데이터를 셔플[]하는 전처리 과정과 변환된 데이터를 분할하여 GPU로 전송하는 batch 과정이 동일하게 수행된다.**

기존 파이프라인과 im.data는 훈련 쓰레드가 CPU에서 데이터 전처리 후 GPU에서 학습모델의 훈련과 모델 가중치 갱신을 수행한다. 훈련이 종료되면 기존 파이프라인 구조는 1Epoch에 대해 훈련된 모델의 성능을 검증하기 위해 검증데이터를 batch 크기만큼 분할하여 추론과정을 통해 Loss를 계산한다. 이 과정에서 CPU는 유휴상태가 된다. 반면, im.data 파이프라인은 모델의 훈련이 종료되는 시점에 CPU에서 모델 객체를 사용자가 지정한 검증 쓰레드 개수만큼 복제하고 검증데이터를 분할하여 Thread pool을 통해 각 쓰레드로 할당한다. 그리고 GPU에서 검증 쓰레드가 동작하는 동시에 훈련 쓰레드를 호출하여 CPU는 2Epoch에 대한 훈련데이터 전처리와 분할을 수행하고 종료 후 학습모델 훈련을 수행한다. 이후 2Epoch에 대한 훈련이 종료되면 검증 쓰레드와 다음 훈련 쓰레드가 호출되면서 병렬적으로 중첩되어 동작한다. 이러한 수행 과정을 통해 기존 파이프라인 구조보다 하드웨어 리소스 활용률을 높여 훈련시간을 최대 16.6%까지 단축시킬 수 있었다.

**4. 실험 및 분석**

본 장에서는 논문에서 제안한 im.data 파이프라인의 성능평가를 위해 Numpy라이브러리를 사용한 파이프라인과 텐서플로우[]에서 지원하는 파이프라인 API인 tf.data를 비교군으로 사용한다. 실험방법은 각 파이프라인에서 학습모델의 훈련시간과 CPU와 GPU의 사용량을 측정하였다.

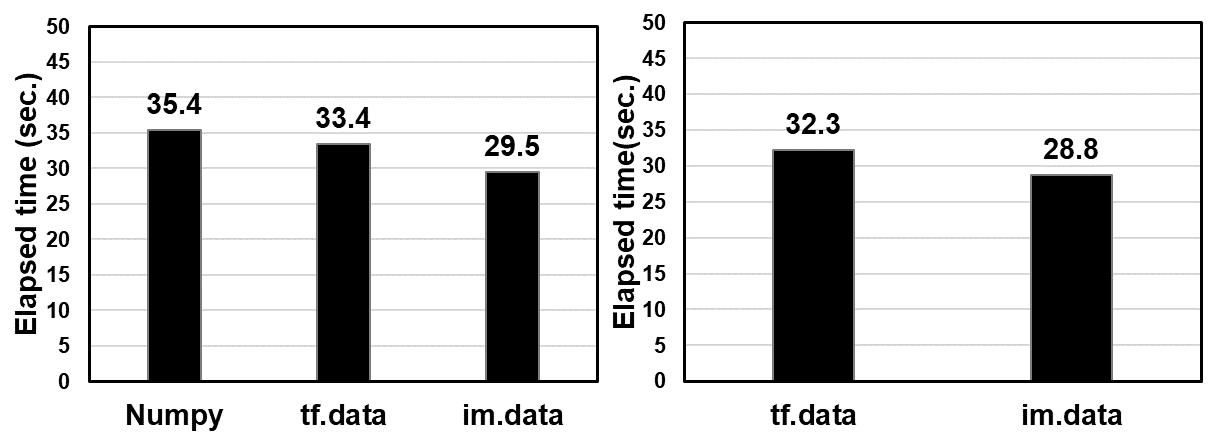
4.1 실험환경

실험환경은 Intel Core i5-9500 (3.00GHz, 6 Core) CPU, 32GB DRAM, Nvidia GeforceGTX 1650 4GB GPU가 장착된 호스트 PC를 리눅스 커널 4.15 버전의 Ubuntu 16.04 LTS 환경에서 사용하였다.

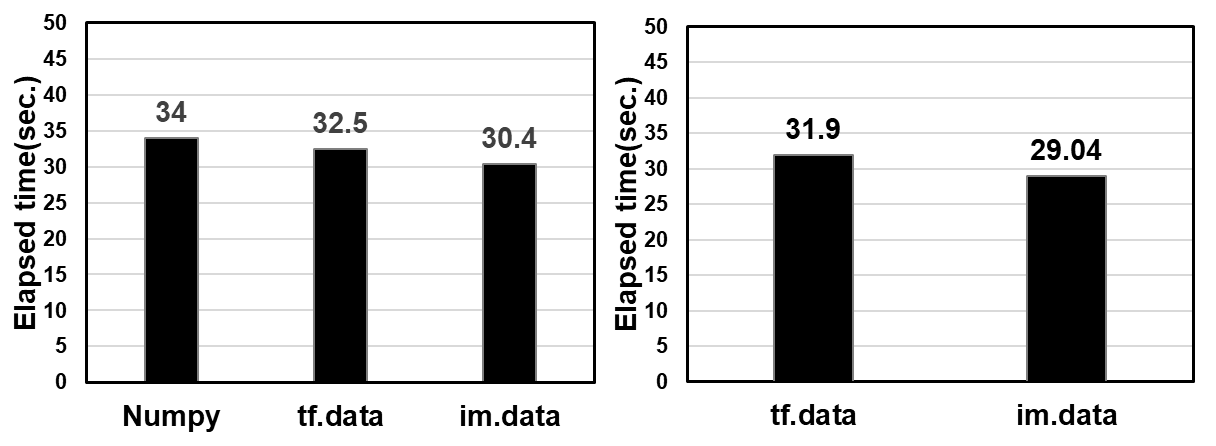
학습모델은 Keras 2.5.0, TensorFlow 2.5.0버전을 사용하여 이미지분류 모델로 가장 많이 사용되는 CNN(합성곱 신경망)모델로 구성했다. 데이터셋은 이미지 분류모델의 성능평가에 사용되는 테스트 데이터인 MNIST[]데이터셋과 Fashion MNIST[] 데이터셋을 사용하였으며 CPU, GPU 리소스 사용량 측정도구는 cpustat[], gpustat[]을 사용하여 실험을 진행했다.

4.2 처리시간 비교

본 절에서는 그림 n+2는 MNIST와 Fashion MNIST 데이터셋을 사용했을 때 각 파이프라인별 처리시간을 비교한다. 또한 im.data 파이프라인 멀티 쓰레딩으로 인한 성능향상 효과를 확인하기 위해 검증쓰레드의 개수가 1개일 경우, 6개일 경우로 나누어서 측정했다.



1. MNIST

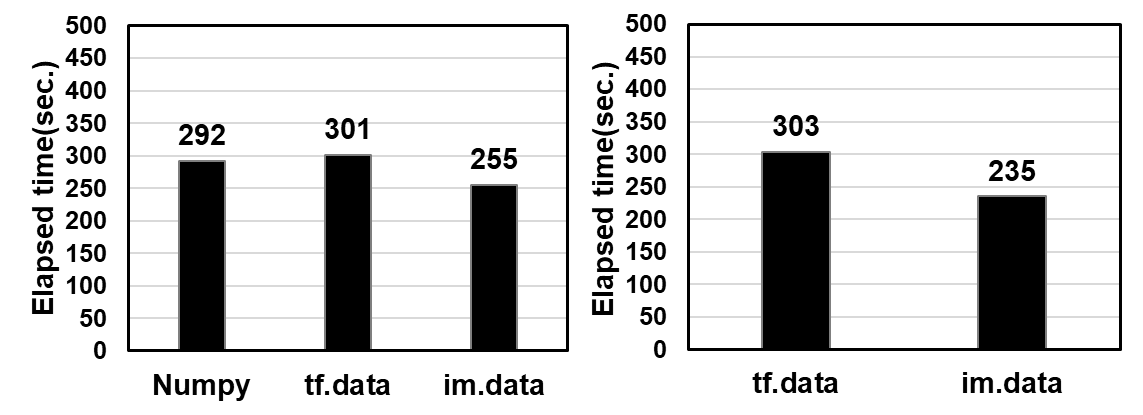


1. Fashion MNIST

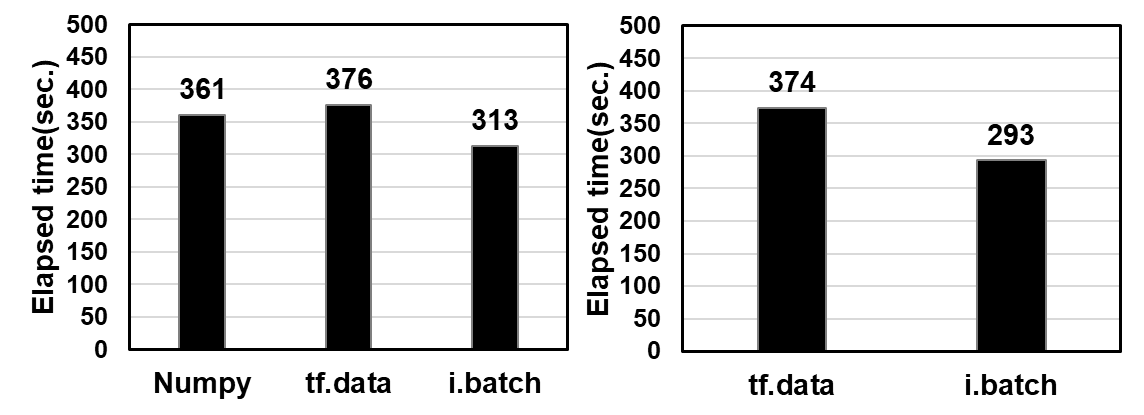
그림 n+3 파이프라인별 처리시간 비교

그림 n+3 파이프라인별 처리시간 비교

그림n\_+2는 데이터셋 별 파이프라인별 모델 훈련시간을 비교를 나타낸다. 훈련 데이터셋을 파이프라인 훈련 쓰레드 1개와 6개로 구성했을 때 측정결과를 왼쪽과 오른쪽 그래프로 나누어 표현하였는데 MNIST와 Fashion MNIST 데이터셋 모두 파이프라인별 훈련시간이 비슷한 분포를 보였다. 검증 쓰레드가 1개일 경우 im.data 파이프라인이 29.4초로, Numpy 라이브러리로 구현한 파이프라인보다 훈련시간이 약 16.6%, tf.data 11.6% 감소하는 결과를 보였다. 이는 im.data의 훈련쓰레드와 검증 쓰레드 인터리빙으로 인한 성능 향상으로 보인다. tf.data의 파이프라인 병렬처리 옵션으로 num\_parallel\_calls 파라미터를 6으로 지정하고 im.data의 검증 쓰레드를 6개로 설정했을 경우, 처리시간이 tf.data 보다 약 10.8% 감소하는 결과를 보였다. 이는 추론-검증 과정의 멀티쓰레딩으로 학습모델의 추론시간이 단축됨으로 인해 인터리빙 간격이 줄어들어 처리성능이 향상된 것으로 보인다. Numpy 파이프라인의 경우 별도의 멀티쓰레딩 옵션을 제공하지 않아 멀티쓰레딩 비교군에서는 제외되었다.



1. 1GB MNIST



1. 1GB Fashion MNIST

그림 n+4 1GB 데이터셋 파이프라인별 처리시간 비교

그림 n+4 1GB 데이터셋 파이프라인별 처리시간 비교

용량이 큰 데이터셋에서 검증 쓰레드의 개수에 따른 처리성능 변화를 확인하기 위해 기존 MNIST와 Fashion MNIST 데이터셋을 복제하여 각 1GB 데이터셋을 생성했다. 그림 n+4에서 MNIST의 경우 im.data는 검증쓰레드가 6개로 구성했을 때 1개일 때 보다 처리시간이 7.8% Fashion MNIST는 6.3% 감소하는 결과를 보였다.

그림 n+3에서 데이터셋 용량이 작을 경우보다 처리시간 감소량이 더 높은데, 이는 데이터셋 용량이 증가하면서 학습모델을 복제하고 데이터를 분할하여 검증쓰레드에 할당하면서 발생하는 오버헤드보다 멀티쓰레딩으로 인한 처리성능 향상이 수행시간에 더 큰 영향을 미치는 것으로 보인다. 반면 tf.data의 기존 데이터셋 크기에서의 처리시간 감소량과 동일하거나 오히려 증가하는 결과를 보였다.

4.3 파이프라인 리소스 사용률 비교

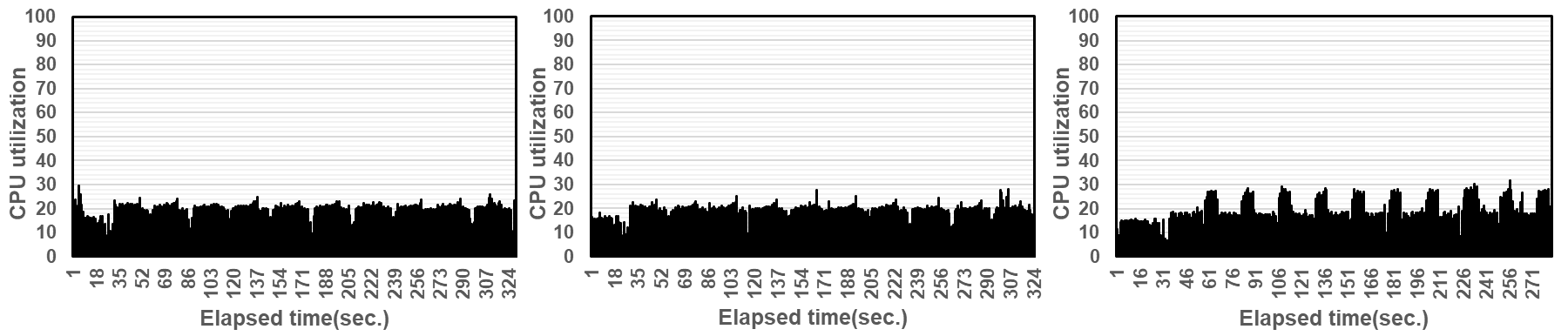
이러한 처리시간 차이에 따른 하드웨어 리소스 사용률과의 관계성을 확인하기 위해 각 파이프라인별 학



|  |  |  |
| --- | --- | --- |
| (a) Numpy | (b) tf.data | (c) im.data |

그림 n+5 파이프라인별 CPU 사용률

그림 n+5 파이프라인별 CPU 사용률



|  |  |  |
| --- | --- | --- |
| (a) Numpy | (b) tf.data | (c) im.data |

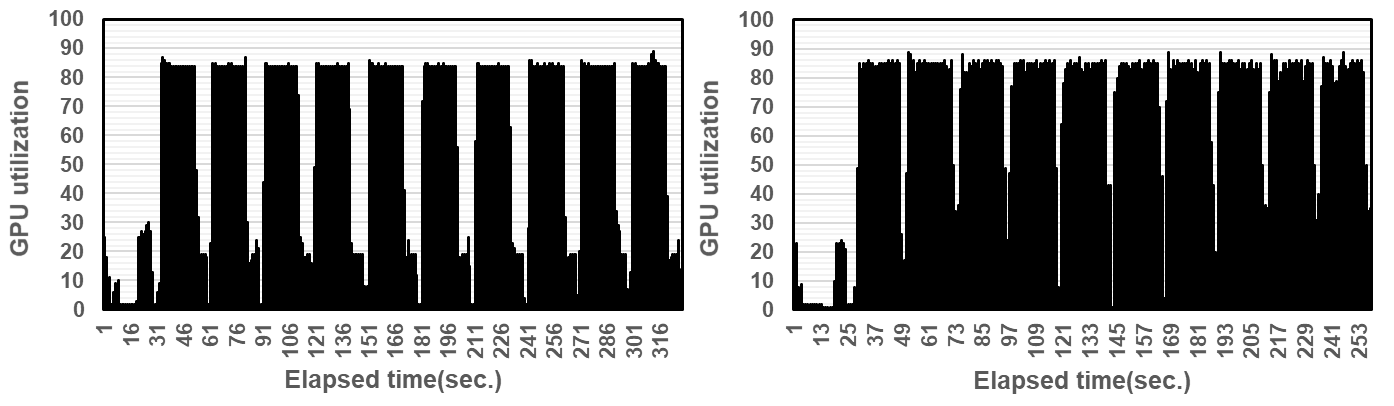
그림 n+5 파이프라인별 CPU 사용률

그림 n+5 파이프라인별 CPU 사용률

습모델 훈련시간동안 초당 CPU와 GPU 사용률을 측 정하는 실험을 진행했다. 훈련데이터는 1GB MNIST 데이터셋을 사용하였으며 Epoch은 10으로 설정했다. 사용률 측정은 리소스 모니터링 도구인 mpstat[]과 gpustat[]을 사용하였다. 그림 n+4와 n+5는 각 파이프라인을 단일 쓰레드로 구성했을 때 훈련시간동안 CPU와 GPU 사용률 패턴을 보인다.

그림 n+4의 (a)와 (b)는 각각 Numpy와 tf.data로 구성한 파이프라인의 GPU 사용률 패턴으로, 사용률이 높은 구간과 낮은 구간이 반복되는 패턴이 나타난다. GPU 사용률이 높은 구간은 사용률이 평균 약 84%로 학습모델이 파이프라인으로부터 GPU로 전달받은 데이터를 학습하여 옵티마이저에 의한 역전파로 가중치를 갱신하는 과정이고 사용률이 낮은 구간은 평균 약 18%로 훈련데이터를 통해 가중치가 갱신된 학습모델이 검증데이터를 통해 추론하는 과정이다. 추론과정이 끝나면 사용률이 10% 이하로 떨어지는 구간이 존재하는데 이는 CPU가 Loss를 계산하고 다음 학습데이터를 전처리하는동안 GPU가 대기하는 시간인 것으로 보인다. 그래프에서 이러한 사용률 고저 주기가 10번 나타나는데, 이는 Epoch 설정값인 10과 동일한 횟수인 것을 확인할 수 있다. (C)의 im.data 파이프라인에서는 사용률이 낮은 구간의 간격이 Numpy나 tf.data 파이프라인에 비해 좁은 것을 확인할 수 있다. 사용률이 높은 구간의 경우 다른 파이프라인보다 2% 높은 평균 약 86%의 사용률을 보이는데, 훈련 쓰레드와 검증 쓰레드 인터리빙에 따른 병렬처리인한 사용률 상승으로 보인다. 다만 GPU 활용률이 10% 이하로 떨어지는 구간이 동일하게 존재하는데, 이는 CPU에서 검증 쓰레드에 할당할 학습모델 복제와 검증데이터 분할하여 쓰레드풀에서 검증 쓰레드를 동작시키는 추가적인 처리과정에 의해 GPU가 대기하는 시간으로 보인다. 이는 그림 n+5의 (c)에서 im.data의 GPU 사용률이 떨어지는 구간에 CPU 사용률은 약 27%로 높아지는 것을 확인할 수 있다. 검증 쓰레드에 데이터가 할당되고 동작하는 동시에 훈련 쓰레드가 동시에 동작하면서 사용률이 높아지는 것으로 보인다. 그림 n+5의 (a), (b)는 쓰레드 병렬처리를 사용하지 않기 때문에 CPU 사용률의 뚜렷한 고저 패턴이 확인되지 않는 것을 알 수 있다.

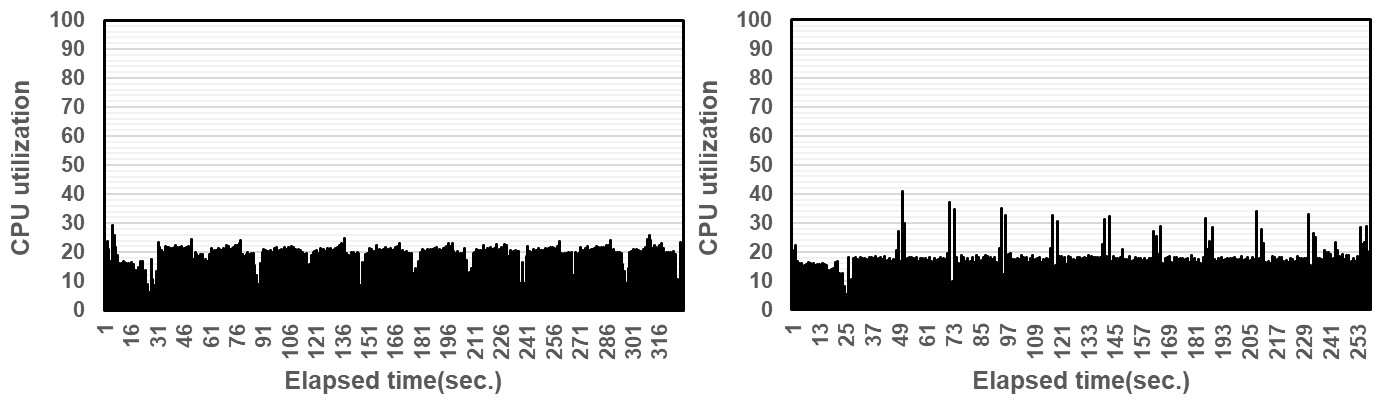
그림 n+6, n+7은 검증쓰레드를 6개로 구성했을 경우 GPU, CPU 사용률에 대한 패턴으로 (b)의 im.data의 GPU 패턴은 검증쓰레드가 1개일 때와 비슷한 패턴을 보인다. 반면 CPU 사용률 패턴의 경우 사용률이 상승하는 구간이 기존의 경우보다 약 8% 더 높은 사용률과 66.6% 짧은 수행시간을 보인다. 이는 검증 쓰레드 6개와 훈련



|  |  |
| --- | --- |
| (a) tf.data | (c) im.data |

그림 n+5 파이프라인별 CPU 사용률

그림 n+5 파이프라인별 CPU 사용률



|  |  |
| --- | --- |
| (a) tf.data | (c) im.data |

그림 n+5 파이프라인별 CPU 사용률

그림 n+5 파이프라인별 CPU 사용률

쓰레드의 전처리과정이 동시에 수행되면서 CPU 수행 시간이 줄어들고, 전체적인 CPU 사용률이 증가한 것으로 보인다. 반면, tf.data의 경우 사용률 패턴의 변화를 쉽게 확인할 수 없었다. 이는 텐서플로우 커널 내부에서 병렬 처리로 최적화되어 기존 사용률에서 더 높아지지 않는 것으로 보인다.

**5. 결론**

본 논문에서는 딥러닝에서 데이터 파이프라인의 훈련-검증 과정을 멀티쓰레딩을 사용한 병렬처리를 통해 훈련시간을 단축시킬 수 있는 새로운 기법을 제안하였다. 제안하는 기법은 데이터 파이프라인을 훈련 쓰레드와 검증 쓰레드로 구분하여 각 쓰레드가 중첩되어 동작한다. 이를통해 기존 파이프라인보다 하드웨어 리소스 사용률이 GPU를 20%, CPU를 5.3% 더 높였으며, 이를 통해 훈련시간을 16.6% 단축시키는 결과를 보였다. 향후 연구에서는 파이프라인의 과정 중 대규모 데이터에서 스토리지로부터 훈련데이터를 메모리에 로드하는 과정을 멀티쓰레딩을 통해 수행시간을 단축할 수 있을 것으로 기대된다.

**Reference**

[1] J. Kim, H. Pyo, J. Ha, C. Lee, J. Kim “Deep learning algorithms and applications,” Journal of KIISE, 33(8), 8, 25-31, 2015. (in Korean)

[2] S. Ko, C. Park, J. Ahn, “Predicting Chemical Structure of Drugs Using Deep Learning,” Journal of KIISE, 48(2), 234-242, 2021. (in Korean)

[3] B. Kim, Y. Choi, “Sports Broadcasting with Deep Learning,” Journal of KIISE, 46(10), 2019.10, 1020-1024, 2019. (in Korean)

[4] J. Choi, J. Seo, S. Choi, “Analysis of Action Recognition Performance According to Depth of Deep Neural Network,” Proc. of the KIISE Korea Computer Congress 2018, 1827-1829, 2018. (in Korean)

[5] C. Shorten, T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," Journal of Big Data 6.1, 1-48, 2019.

[6] S. Kumawat., G. Kanojia, S. Raman, “ShuffleBlock: Shuffle to Regularize Deep Convolutional Neural Networks,” arXiv preprint arXiv:2106.09358, 2021.