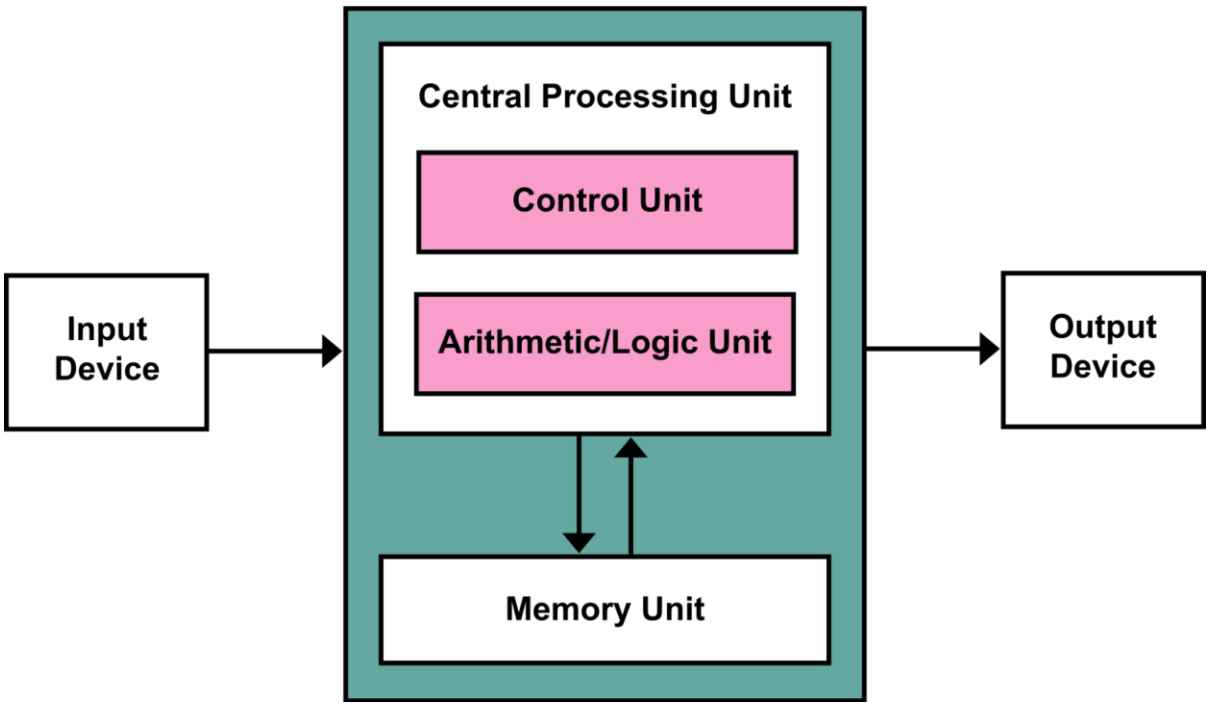
최근 AI 산업의 급격한 성장으로 인해 AI 전용 칩의 생산과 판매도 활발해지고 있습니다. 그런데 의외로 IT 나 실리콘 기술에 대해 어느 정도 지식이 있는 경우에도 CPU GPU TPU NPU 등의 용어에 대한 정확한 의미를 잘 모르는 경우가 있는 것 같습니다. 각각의 칩의 목적과 동작 방식에 대해 궁금하거나, 어렴풋이 알고는 있지만 정확하게 정리해 보고 싶은 분들에게 아래 글을 한번 읽어보시기를 권합니다.

CPU 계산 구조

AI 시대인 요즘의 키워드를 기준으로 하여 CPU GPU TPU NPU 네 가지 유형을 이야기하고 있지만, 이들 중에서 CPU 와 나머지 GPU, TPU, NPU 칩은 구조나 목적상으로 큰 차이점이 있습니다.

CPU 는 Centralized Processing Unit 의 축약어입니다. 현대의 Processing chip 들은 모두 폰노이만이 맨하탄 프로젝트 당시 논문에서 제안했던 전자계산기 기본 구조를 따라 발전해 왔습니다.

폰 노이만 아키텍쳐 (von Neumann architecture)



von Neumann architecture 출처: Wikipedia

현재의 CPU 기본 구조는 폰노이만 구조에 기반하여 발전하였습니다. 폰노이만 아키텍쳐는 유연한 계산 능력이라는 장점이 뚜렷한 구조입니다. 순차적인 계산을 통해 이전 계산 결과와 다음 계산 결과의 유기적인 연계가 가능한 구조이기도 합니다. CPU 는 폰노이만 아키텍쳐에서 Control, Cache 등의 메모리 영역이 고도화된 구조를 지향합니다. 여러 Level 의 캐시를 대량 집적하여 다양한 형태의 복잡한 프로그램을 효율적으로 실행하는 데에 그 목적이 있다고 할 수 있습니다. 하지만 직렬(순차) 실행으로 인한 문제점을 현대의 모든 CPU 가 동일하게 가지고 있기도 합니다.

바로 “von Neumann Bottleneck” 이라는 것인데, 모든 계산의 결과가 ALU (Arithmetic Logic Unit) 연산을 거쳐 반드시 메모리 어딘가에 저장이 되어야 한다는 것입니다. ALU

연산의 결과가 메모리 또는 칩 내부의 캐시에 저장되어야 하기 때문에 항상 한번에 하나의 트랜잭션만을 순차적으로 처리하게 됩니다.

클럭을 높여서 처리 속도가 빨라지더라도 높아진 클럭 만큼의 계산만 더 할 수 있습니다. 클럭의 개선이 발열 등의 문제로 인해 한계에 다다른 이후 CPU 의 발전은 multi-core, hyper-threading 과 같이 동시에 구동할 수 있는 코어의 개수를 늘리는 방향으로 이루어지고 있습니다.

CPU 에서 연산이 어떻게 이루어지는지 살펴봅시다

위에서 CPU 연산의 강점과 약점에 대해 이야기 했습니다. 간단히 CPU 에서 연산이 이루어지는 예제를 살펴보면 내용을 이해하는데 도움이 될 것입니다.

모든 CPU 는 기계어를 실행합니다. 그리고 CPU 종류마다 실행할 수 있는 기계어의 종류는 다를 수 있습니다. 그래서 CPU 를 비롯한 마이크로 프로세서는 자기가 실행할 수 있는 기계어의 모음을 가지고 있습니다. 이런 기계어의 모음을 ISA 라고 부르며, 이는 Instruction Set Architecture 의 축약어입니다.

예를 들어 CPU 에서 “1 + 2 = 3” 연산이 어떻게 수행되는지 살펴봅시다. 다음의 세 개의 명령어 세트만 가지고 있는 단순한 마이크로 프로세서가 있다고 가정해 보겠습니다. 그리고 이 마이크로 프로세서에는 3 개의 레지스터가 있으며, 각각 reg1, reg2, reg3 로 지정하여 프로그램할 수 있다고 가정해 보겠습니다. 실제로 이런 프로세서가 있다면 별로 쓸모는 없겠지만, 개념을 이해하기 위해 단순한 구조로 가정해 보는 것입니다.

명령어	설명
LDR	CPU 내부의 레지스터로 데이터 가져오기
STR	CPU 내부의 레지스터에서 메모리로 데이터 내보내기
ADD	두 개의 레지스터의 데이터 값을 더하여 결과를 다른 레지스터에 저장하기

Instrunction Set

위와 같은 ISA 를 가진 프로세서가 있다고 할 때, 아래와 같은 기계어 프로그램을 통해 계산을 수행할 수 있습니다.

```
LDR reg1, #1; // reg1 레지스터에 상수 1 로드
LDR reg2, #2; // reg2 레지스터에 상수 2 로드
ADD reg1, reg2, reg3; // reg1 값과 reg2 값을 더해 reg3 에 저장
STR reg3, [0x00040222h]; // reg3 의 값을 0x00040222 메모리로 내보냄
```

간단한 구조의 ISA 를 가정하여 작성한 어셈블리 프로그램으로, 첫 번째 레지스터에 1, 두 번째 레지스터에 2 를 저장한 뒤 ALU 를 이용해 두 레지스터에 저장된 값을 더하여 세 번째 레지스터에 저장합니다. 그리고 계산 결과는 메모리 주소를 지정하여 0x00040222h 로 내보내고 있습니다. (레지스터는 CPU 내부의 저장소이고, 메모리는 CPU 바깥의 저장소입니다.)

위의 예제를 von Neumann Architecture 에 대입하여 보면 CPU 의 구조와 동작이 어떤 방식으로 이루어지는지 짐작할 수 있습니다.

Controller 에 의해 어셈블리 프로그램은 순차적으로 Program Counter 로 올라가고 실행될 것입니다. 프로세서는 LDR, ADD, STR 과 같은 Instruction 에 대해 무엇을 해야 하는지 알고 있으므로, 해당 동작을 클럭이 뜰 때마다 기계적으로 수행합니다. 4 클럭이 지나면 계산이 완료되어 메모리에 결과값이 저장될 것입니다.

CPU 는 범용 계산기

초초 간단한 마이크로 프로세서를 가정하여 구동 방식을 살펴보았습니다. 하지만 실제 CPU 도 이런 방식에서 크게 다른 것이 없습니다. 다만, 효율적인 ISA, 엄청나게 많은 내부 데이터 저장소를 비롯해 복잡한 프로그램을 최대한 빠르게 실행할 수 있도록 설계된 정교한 다계층 캐시 구조 등 범용 계산을 효율적으로 실행하는 것에 포커스가 맞춰진 프로세서가 CPU 인 것입니다.

어떤 프로그램도 평균 이상 효율적으로 실행할 수 있는 칩이기 때문에 컴퓨터, 스마트 기기, 각종 임베디드 장비들의 main processor 대부분이 CPU 를 채용하고 있는 것입니다.

왜 GPU 를 사용해야 할까

GPU 는 Graphic Processing Unit 의 축약어입니다. 말 그대로 초기에는 그래픽 처리에 필요한 대량 연산을 수행하기 위한 Co-processor 형태로 출발하였습니다.

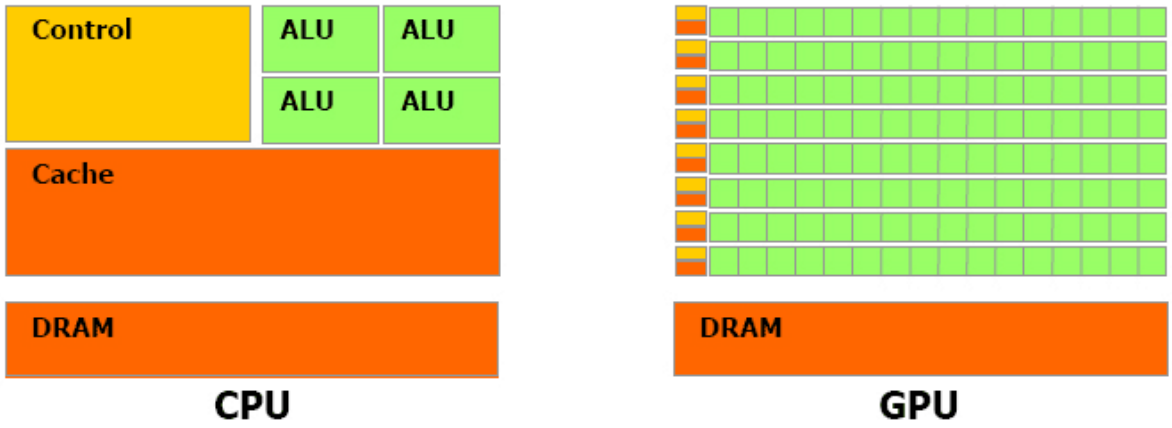
그래픽 처리할 때 일반적으로 빛과 관찰자의 위치를 어떻게 가정하느냐에 따라 수 많은 vector space conversion 이 발생하게 됩니다. 이에 따라 엄청나게 많은 쉐이더, 텍스쳐 연산과 모니터에 뿌려주기 위한 픽셀 렌더링 등이 발생하게 되지요. 이러한 연산들은 그래픽 해상도나 텍스쳐의 정교한 정도에 따라 다르지만 일반적인 CPU 연산에 비해 엄청나게 많은 수의 부동소수점 곱셈 연산을 발생시킵니다.

부동 소수점 곱셈 연산에 대해 잘 모른다면 정수(Integer number)와 정수간의 곱셈이 아닌 소수점(Real number)이 있는 두 개의 숫자간의 곱셈이라고 생각해도 됩니다.

엄청나게 많은 수의 부동소수점 곱셈 연산을 CPU 를 이용해 모두 처리해 주어도 됩니다만, 모든 폰노이만 아키텍처에 기반한 프로세서는 순차처리 방식이기 때문에 수 많은 부동수수점 곱셈 연산이 한번에 하나씩 순차적으로 처리될 것입니다. 이는 정작 중요한 프로그램의 구동이 단순 연산을 수행하느라 대기하게 되는 것을 의미하겠지요.

예를 들어 게임을 구동했는데 화면에 그림을 그리느라 키보드 입력이 안되는 문제가 발생한다는 것입니다.

CPU vs GPU 구조



CPU vs GPU 구조

여기 CPU 와 GPU 의 구조를 직관적으로 살펴볼 수 있는 이미지가 있습니다. 이미지를 보고 앞서 살펴보았던 내용을 다시 한번 짚어보겠습니다.

그래픽 처리에 필요한 계산에는 복잡한 명령어 세트나 많은 수의 레지스터가 필요하지 않습니다. 복잡한 캐시 구조도 필요 없습니다. 그냥 동일한 형태의 계산-부동 소수점 곱셈-을 대량으로 수행하면 됩니다.

CPU 는 복잡한 명령어세트를 실행하기 위한 정교한 아키텍쳐를 가지고 있습니다. 그렇기 때문에 어떠한 형태의 프로그램도 효과적으로 수행할 수 있는 똑똑한 녀석입니다. 하지만 똑똑한 만큼 개별 계산당 비용이 큽니다. CPU 도 당연히 부동소수점 연산을 500 억번 수행할 수는 있습니다. 하지만 그 시간에 다른 복잡한 일을 하는 것이 더 효율적이겠지요.

GPU 는 대규모 병렬 곱셈 계산기

GPU 는 단순한 형태의 대량 계산을 CPU 로부터 독립시키기 위하여 고안된 Co-processor 입니다. CPU 코어의 복잡한 구조를 단순화하여 개별 계산당 비용을 극단적으로 낮추고 대신 단순한 형태의 코어를 대량으로 집적하여 단순 연산을 병렬로 수행합니다. 이를 통해 그래픽 처리에 필요한 대량 부동소수점 연산을 저비용으로 수행할 수 있게 된 것입니다.

물론 GPU 가 화면에 그림을 그리기 위해 열심히 부동 소수점 계산을 수행할 동안 CPU 는 보다 중요한 복잡한 일들을 수행할 수 있습니다. 키보드, 마우스 입력을 받아 캐릭터를 이동 시키는 것과 같은 일 말이지요.

AI 연산에는 왜 GPU 를 사용할까

AI 추론이나 학습을 할 때 핵심적으로 필요한 연산은 매트릭스 합성곱(convolution) 연산입니다. 합성곱 연산을 통해 이미지에서 필터가 지정한 방향에 따른 경계선 모양을 대략적으로 알아낼 수 있습니다. 이런 작업을 딥러닝에서는 feature extraction 이라고 합니다. 여러 번 이런 식으로 추출한

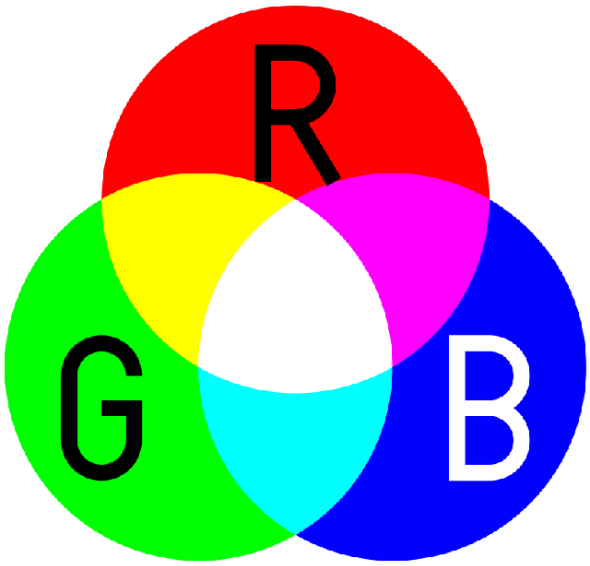
feature map 을 적절하게 변형한 다음 정답과 맞추어 보고 최대한 알고 있는 정답에 근접하도록 각 필터의 가중치를 수정해 나가는 것이 딥러닝 모델 학습 과정입니다.

추론 과정에서도 학습 과정과 동일한 계산이 필요합니다. 따라서 Convolution 연산이 보통 매우 많이 필요합니다.

Convolution 연산에 대해 조금만 알아봅시다

RGB 이미지

이미지를 표현하는 방법은 여러가지가 있습니다. 그 중 하나가 RGB 방식입니다. RGB 방식은 초등학교 미술시간에 배웠던 빛의 삼원색 빨강, 초록, 파랑의 조합으로 색깔을 표현하는 방식입니다.



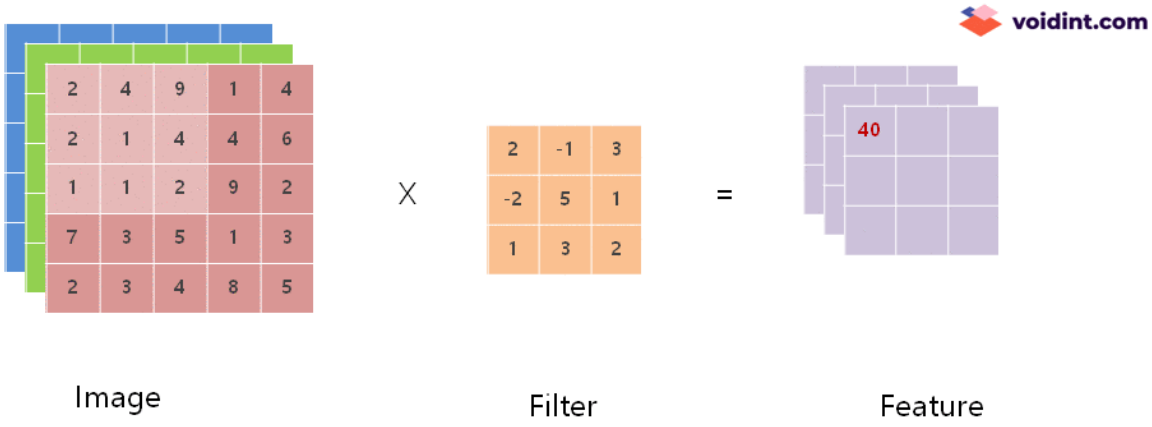
빛의 삼원색 – RGB

RGB 방식에서 하나의 픽셀은 빨강(Red), 초록(Green), 파랑(Blue) 세 가지 색깔의 요소를 가집니다. 각각의 색깔은 보통 256 단계의 강도를 가지도록 구분됩니다.

Convolution 연산

Convolution 연산은 최근의 메가 트렌드인 딥러닝을 통한 AI 의 핵심 연산이라고 할 수 있습니다.

Convolution 이 무엇이며 어떤 계산을 하게 되는지 약간만 살펴보겠습니다. 딥러닝에 대해 잘 모르셔도 됩니다. AI 연산에 왜 CPU 보다 GPU 가 더 적합한 것인지 이해하는 것이 목적입니다.



Convolution Example

위 그림은 아주 작은 이미지를 가지고 수행하는 Convolution example 입니다. 좌측에 Image 가 있고, 가운데에 Filter 가 있으며, 우측에 Feature 가 있습니다.

이미지의 하나의 칸은 하나의 pixel 입니다. 3 차원 매트릭스로 이미지의 RGB 픽셀이 표현되어 있습니다. 가로 5 pixel, 세로 5 pixel 짜리 아주 작은 이미지입니다.

Filter 는 커널이라고도 부르는데, Image 로부터 Feature 를 추출하기 위한 변수와 같은 역할을 합니다. 딥러닝 학습을 한다는 것은 많은 필터의 계수를 변경하여 정답에 가깝게 만드는 과정입니다.

위의 움직이는 이미지에서 보면 Image 에서 Filter 의 크기 만큼 범위가 점점 우측, 아래 방향으로 이동하면서 Feature 값이 채워지고 있는 것을 볼 수 있습니다.

한 칸의 계산을 자세히 살펴보겠습니다.

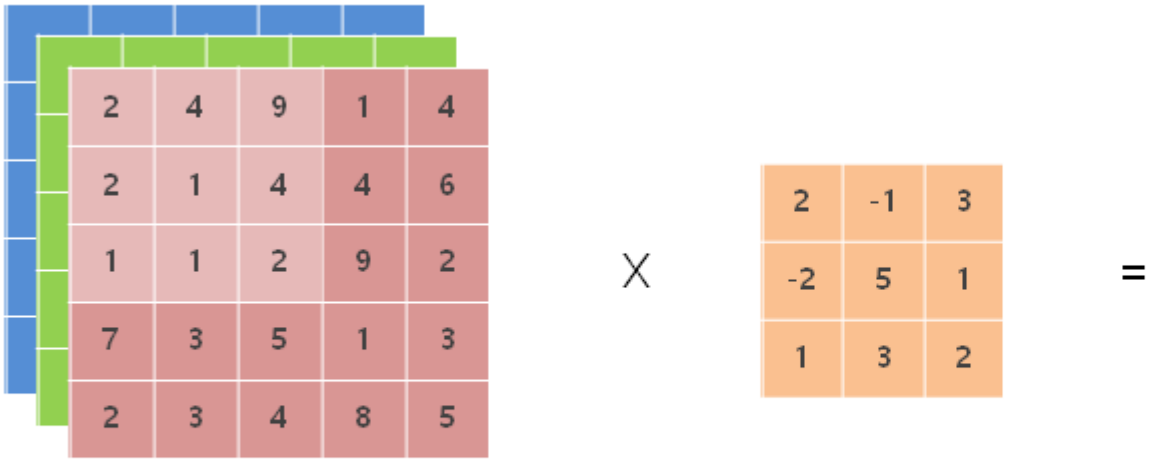


Image Filter

Convolution 연산 첫 번째 항목

Filter 의 크기와 같은 3×3 사이즈로 Image 의 빨강색 요소의 데이터를 뽑아서 Filter 와 항목 대 항목으로 곱셈한 뒤 모든 항목의 곱셈 결과를 더합니다. 아래 공식을 참조하세요.

$$Feature_{0,0} = \begin{pmatrix} 2 & 4 & 9 \\ 2 & 1 & 4 \\ 1 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} 2 & -1 & 3 \\ -2 & 5 & 1 \\ 1 & 3 & 2 \end{pmatrix}$$

$$= 2 \cdot 2 + 4 \cdot (-1) + 9 \cdot 3 +$$
$$2 \cdot (-2) + 1 \cdot 5 + 4 \cdot 1 +$$
$$1 \cdot 1 + 1 \cdot 3 + 2 \cdot 2$$
$$= 40$$

이렇게 계산된 결과 40 이 추출된 Feature 의 첫 번째 항목이 됩니다. 다음 항목은 위에 첨부한 움짤과 같이 차례로 계산하여 채워집니다.

왜 Convolution 연산에 CPU 보다 GPU 가 유리할까

예제로 작은 이미지에 대한 합성곱 연산을 약간 살펴보았습니다. 예제는 단지 5×5 픽셀의 실제로는 알아보기도 힘들 정도의 작은 이미지를 사용하고 단 1 개의 필터를 사용했으며, 모든 항목은 정수형 곱셈으로 이루어져 있습니다. 그렇더라도 프로세싱 해야 하는 곱셈 연산이 상당히 많은 것을 알 수 있을 것입니다.

경우에 따라 다르기는 하지만 딥러닝 네트워크를 학습하고 실행하는 데에 요즘에는 보통 608 픽셀 이상의 이미지를 사용합니다. 필터의 개수는 알고리즘에 따라 다르지만 몇 백 가지 내지 몇 천 개 단위의 네트워크도 많습니다. 개별 필터의 크기도 제각각이고 다양합니다. Edge inference 에는 Quantization 기술이 적용되어 8 bit 연산이 일반적인 사양이 되었지만, 최근까지 대부분 학습과 추론에는 32bit 실수가 사용되었습니다. 앞서 언급했지만 부동 소수점 간의 곱셈은 CPU 가 실행하기 까다로운 인스트럭션입니다.

이미지 한 장에서 사람, 동물 또는 자동차와 같은 객체를 뽑아내는 데에 저런 Convolution 연산이 엄청난 횟수가 발생합니다. Convolution 연산은 그 자체로 매우 많은 횟수의 곱셈으로 이루어집니다. 이것을 값비싸고 무슨 일이나 할 수 있는 CPU 를 사용해서 할 필요는 없겠지요? 이건 마치 건설 현장에서 벽돌 5 만 개를 건축 공학 박사에게 나르라고 하는 것과 마찬가지입니다. 벽돌을 나르는 일은 박사가

아니더라도 누구든 그냥 여러 사람을 불러 모아 시키는 것이 훨씬 경제적일 것입니다.

TPU, NPU

앞서 CPU 와 GPU 에 대한 이야기가 너무 길었습니다. 그런데 TPU, NPU 는 CPU 와 GPU 의 차이점에 대해 이해했다면 전혀 새로울 것이 없는 것들입니다. 딥러닝에서 딥뉴럴 네트워크를 학습하거나 추론하는 데에 실수의 곱셈 연산이 엄청나게 많이 필요하다고 이야기했습니다. 그래서 GPU 를 이용해 그런 계산을 CPU 보다 효율적으로 하고 있다고 했지요.

TPU, NPU 는 실상은 같은 것입니다. 그냥 Neural Processing Unit 입니다. GPU 를 이용해서 곱셈을 처리하던 시절, Neural Network 와 Neural Network 를 구성하는 뉴런에 해당하는

다양한 노드가 생겨났습니다. Convolution 노드도 신경망을 구성하는 하나의 뉴런에 해당한다고 볼 수 있습니다.

NPU 는 GPU 에 신경망에서 처리해야 하는 곱셈 연산을 나눠서 시키는 불편함과 자원의 낭비를 줄이거나, NPU 제작 업체가 필요한 사양을 추가하는 등의 Neural Network Processing 에 특화된 칩셋을 말합니다.

TPU 는 구글에서 제작한 NPU 의 이름입니다. Tensor Processing Unit 이라고 합니다. TPU 가 탑재된 Coral 시스템도 NVIDIA 또는 퀄컴에서 제작한 NPU 와 하는 일이 크게 다르지 않습니다. Tensorflow lite 가 조금 더 잘 호환된다고 하는 등의 제작사의 요구사항이 반영되어 있는 NPU 의 한 가지인 것입니다.

(글에 NPU 내용이 너무 부실하다고 지적하시는 분들이 많아 아래 글에 보강하였습니다. Ctrl+클릭하면 열립니다)