**고급소프트웨어실습**

**기말 리포트**

**(CSE 4152)**

**Due: 2020년, 12/17일, 오후 12시 (정오)**

**학번 : 20171672**

**반 번호 : 2**

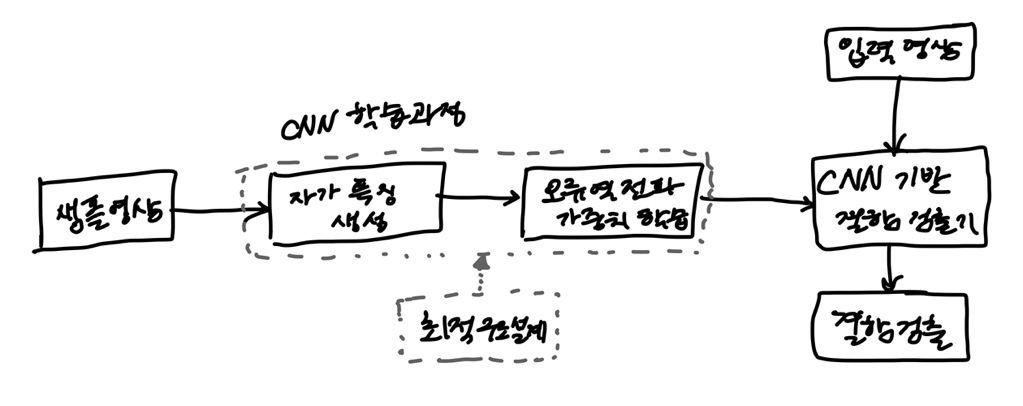
**이름 : 이정원**

**1. 영상 처리 분야에서 Deep Neural Network (DNN) 기술을 사용하여 상용화된 SW 또는 제품의 사례가 있는지 조사하고, 있다면 대표적인 한 가지에 사용된 DNN 기술에 대하여 구조도, 기능, 특징을 포함하여 한 페이지 이내로 설명하시오. 만약 상용화 단계의 사례가 없다면, 연구 개발 등에 많이 활용되는 대표적인 영상 처리 DNN 기술에 대하여 같은 방식으로 설명하시오.**

기존의 다양한 결함 검출 방법들은 영상에서 배경이나 결함의 특징을 이용하여 검출을 진행한다. 그러나 기존의 방법은 검사 대상물이 바뀔 때마다 새로운 특징에 대한 연구를 해야 하기 때문에 다양한 표면을 동시에 검사하는 방법은 아직 개발되지 않았다. 또한 하나의 표면에 여러 종류의 특징을 이용하여 결함을 검출할 경우 성능이 증가할 순 있으나 시간이 매우 늘어나고, 사람이 예측하지 못한 결함이 발생될 경우 다시 성능이 하락한다는 단점이 있다.

기존의 결함 방법들의 제한 조건을 극복할 수 있는 다양한 종류의 결함을 감지할 수 있는 결함 검사 방법 연구가 진행되고 있다. 그것은 바로 DNN 기반의 CNN(Convolution Neural Network)으로 다수의 학습 샘플을 이용하여 특징 추출을 자체적으로 수행하는 학습 기반 결함 검출 방법이다. 영상을 직접 분해하여 특징을 획득하고, 사람의 개입이 없기 때문에 매우 높은 검출 결과를 보여 준다.

CNN 기반 결함 검출 구조도는 다음과 같다.

****

위에서 CNN은 신경망을 기초한 결함 검출 방법으로 다양한 계층의 구성을 통해 부류를 구분한다. 영상을 학습 데이터로 이용하여 계층간의 연결에 사용되는 가중치를 학습한다. 컨볼루션 층에서는 하위 계층에서 상위 계층으로 신경망이 연결될 때마다 추상화 과정을 통해 특징을 생성하며, 폴링 층에서는 특징 맵을 감소시켜 오차를 줄이고, 다양한 크기의 고수준의 특징을 추출한다. CNN의 학습은 오류 역전파 기법을 사용하여 학습한다.

CNN은 다양한 부류를 동시에 검출함에도 다른 검출 방법과 달리 성능이 뛰어난데, 이에는 여러 장점이 있다. 첫째로 학습을 여러 번 진행할 필요 없이 표면의 종류가 많아도 학습에 부류를 추가함으로써 모든 표면의 검사를 진행한다는 것이다. 둘째로 각각의 표면에 대한 분류가 가능하기 때문에 표면의 종류가 순차적으로 입력되지 않고 다양한 영상을 무작위로 입력하더라도 검사가 가능하다는 것이다.

**2. 아래의 (CODE 1)은 다음과 같은 C/C++ 코드를 어떤 이전 버전의 Visual Studio의 DEBUG 모드에서 컴파일하여 생성한 어셈블러 코드의 일부이다.**

switch (val) {

   case 0: tmp = val\*misc; break;

   case 1: tmp = val + misc; break;

   case 2: tmp = val/misc; break;

**/\* here \*/**

}

**위의 C/C++ 코드의 /\* here \*/ 부분에 다음과 같은 문장을 추가한 후 동일 조건에서 컴파일하였을 때, (CODE 2)와 같은 결과를 얻었다.**

case 3: tmp = val - misc; break;

**수행 속도 차원에서 어떠한 이유로 (CODE 2)의 계산 방식이 (CODE 1)의 방식보다 우월할 수 있는지, 각 방법의 작동/코드 수행 원리를 반드시 설명하면서 상세히 기술하라.**

|  |  |
| --- | --- |
| 테이블이(가) 표시된 사진  자동 생성된 설명 | 텍스트이(가) 표시된 사진  자동 생성된 설명 |

위 C/C++ 코드는 입력받은 val에 따라 switch문을 이용하여 연산을 수행하는 코드이다.

(코드 2) 노란 부분을 보면 switch문에 있는 변수값 val이 3보다 큰지를 검사하고 있다. 3과 비교하는 이유는 case문에서 가장 큰 값이 3이기 때문이다. 3보다 크다면 jump할 case문이 없기 때문에 default로 이동한다. 초록 부분은 jump할 case문이 있을 경우 이 코드에 진입한다는 뜻이다. 이 코드에 진입한 후 $L568을 검사하고 val과 매핑되는 값으로 jump하도록 되어 있다. $L568에는(파란 부분) Jump Table이 저장되어 있고 이는 switch case문의 개수가 4 이상이기 때문에 jump table을 사용한 것을 뜻한다.

여기서 Jump Table이란 점프 명령어들을 이용해서 프로그램의 제어를 프로그램의 다른 부분으로 옮기는 방법을 뜻한다. 이 구조는 주로 Switch문을 구현하는 ,컴파일러에 의해 생성된 어셈블리어 프로그램에 사용된다. 파란 부분을 보면 switch 문의 각 연산을 수행하는 주소가 저장되어 있다.

이후 코드는 switch문에 명시되어 있는 연산을 수행하는 코드들이다. 또한 jump table을 사용할 경우 cmp 명령문 여러 번 사용할 필요 없기 때문에 비교 연산을 하기 위한 시간이 줄어든다는 것을 알 수 있다.

(코드 1) 그러나 코드 1은 코드 2와 달리 switch case문의 개수가 4 미만이기 때문에 Jump Table을 사용하지 않는다. 즉, if-else if문처럼 코드가 작동한다는 것이다. 이 때문에 val 값이 0인지, 1인지, 2인지 cmp로 일일이 조건 비교하여 jump하고 있으며, cmp 명령문을 여러 번 수행할 수 있기에 비교 연산 계산량이 늘어나 수행 속도가 느린 것이다. (최악의 경우 3번까지) 하지만 코드 2에서는 바로 default로 갈지 말지 cmp 명령문으로 한 번만 비교 연산을 수행하고 난 뒤 Jump Table을 이용해 계산을 수행하기 때문에 수행 속도가 코드 1보다 훨씬 빠르다. 이후 코드는 switch문에 명시되어 있는 연산을 수행하는 코드들이다.

**3. CUDA memory hierarchy에 대하여 아래한글 기준 1500글자(공백 제외) 이상의 분량으로 명료하게 기술하라. 강의자료의 내용을 copy-and-paste할 경우 답으로 인정하지 않으며, 반드시 자신의 언어로 CUDA에 대하여 잘 모르는 컴퓨터공학 전공 3학년에게 설명하는 수준에서 기술할 것. (제출 내용에 대하여 copy-check를 진행할 예정임)**

**============= 글자수(공백 제외): 1502 =============**

**역사(歷史, 문화어: 력사, 영어: history)는 인류 사회의 변천과 흥망의 과정, 지난 시대에 남긴 기록물, 이를 연구하는 학문 분야 등을 가리킨다. 또 인간이 거쳐온 모습이나 인간의 행위로 일어난 사실을 말하는 단어로도 쓰인다. 또한 역사는 시간의 흐름으로써 어떤 사람이 겪은 일에서 중요한 일들 중 후대에게 쉽게 잊히지 않는 과거의 사실을 의미하며 명예와는 미래에서 잊히지 않고 이어 전해진다는 차이가 존재한다.[1][2] 역사라는 말은 객관적 사실과 서정적 표현, 주관적 기술의 세 측면의 의미를 가지는 것으로 정의되고 있는데, 레오폴트 폰 랑케는 "있었던 그대로의 과거"를 밝혀내는 것이 역사가의 사명이라고 하여 객관적 사실을 강조하였다. 이에 비해 에드워드 핼릿 카는 과거의 사실을 보는 역사가의 관점과 사회 변화에 따라 역사가 달리 쓰일 수 있다고 하였다. 역사 연구란 이야기(narrative)를 통해 연속된 사건들을 검증하고 분석하며 종종 인과 관계를 객관적으로 규정하기도 한다. 역사학은 고고학, 사회학 등의 주변 학문과 밀접한 연관을 갖고 있다. 역사라는 개념은 또한 넓은 의미에서 사건이나 사물의 자취를 총칭하는 데 쓰이기도 한다. 역사학자들은 과거를 연구하기 위하여 많은 역사적 자료를 탐색한다. 이 자료들은 1차 사료와 2차 사료로 구분되는데, 1차 사료는 어떤 사건에 직접 참여한 사람들이 남긴 문서, 기록 등이다.**

**역사(歷史, 문화어: 력사, 영어: history)는 인류 사회의 변천과 흥망의 과정, 지난 시대에 남긴 기록물, 이를 연구하는 학문 분야 등을 가리킨다. 또 인간이 거쳐온 모습이나 인간의 행위로 일어난 사실을 말하는 단어로도 쓰인다. 또한 역사는 시간의 흐름으로써 어떤 사람이 겪은 일에서 중요한 일들 중 후대에게 쉽게 잊히지 않는 과거의 사실을 의미하며 명예와는 미래에서 잊히지 않고 이어 전해진다는 차이가 존재한다.[1][2] 역사라는 말은 객관적 사실과 서정적 표현, 주관적 기술의 세 측면의 의미를 가지는 것으로 정의되고 있는데, 레오폴트 폰 랑케는 "있었던 그대로의 과거"**

C/C++과 같은 컴퓨터 프로그래밍을 접한 사람이라면 어떠한 문제에 대해 정답 결과 값을 얻기 위하여 for 반복문을 이용해 계산을 수행해 본 적이 있을 것이다. 그러나 하나의 프로그램 내에 아무리 간단한 연산 수식이어도 for문이 많이 중첩될수록 계산량이 늘어나 수행 시간이 기하급수적으로 증가할 것이다. 앞서 든 예시처럼 계산에 이용되는 데이터들이 서로 독립적이며 같은 계산이 순차적으로 여러 번 반복되는 상황을 어떻게 하면 시간 및 메모리 측면에서 보다 더 효율적으로 만들 수 있을까? 코드의 효율을 중요시하는 프로그래머라면 이에 대한 해답으로 각 데이터에 대한 계산을 여러 구간으로 나누어 동시에 병렬적으로 수행하는 방법이 떠오를 것이다. 이를 가능하게 만드는 기술이 바로 CUDA(Compute Unified Unified Device Architecture)이다.

CUDA란, NVIDIA가 개발한 병렬 컴퓨팅 플랫폼 및 API 모델이며 GPU(그래픽 처리 장치)에서 수행하는 병렬 처리 등과 같은 알고리즘을 C/C++과 같은 언어를 사용하여 코드를 작성할 수 있는 기술이다. 일반적으로 CUDA는 세 가지 과정을 거친다. 첫 번째로, Host(CPU) 메모리에 할당된 데이터를 Device(GPU) 메모리에 복사한다. 두 번째로, GPU에서 병렬 연산을 수행한다. 세 번째로, Device 메모리에 저장되어 있는 연산 결과 데이터를 다시 Host 메모리로 복사한다.

CUDA는 실행 과정에서 다양한 종류의 메모리에 접근할 수 있는데, 개별적인 로컬 메모리(Private Local Memory), 공유 메모리(Shared Memory), 글로벌 메모리(Global Memory), 상수 메모리(Constant Memory), 텍스처 메모리(Texture Memory)로, 총 다섯 가지로 구성되어 있다.

첫 번째로, 개별적인 로컬 메모리에는 주로 Register 세트로 구현된 Local Memory가 있는데, 커널 입장에서 필요한 변수들을 저장하는 메모리 계층이다. 이는 각각의 스레드 내부에서 생성되고 해당 스레드 내에서만 참조가 가능하다. 커널 함수 내에서 지역 변수를 많이 사용할수록 레지스터 사용량도 늘어난다.

두 번째로, 공유 메모리는 각 스레드 블록 당 배정되고(실행 중인 스레드 블록 상에서만 유효하다.) 각 블록 안의 모든 스레드가 그들끼리 데이터를 공유할 수 있는 메모리 계층이다. 특히 이 공유 메모리는 CUDA의 L1 캐시라고 불리기도 하는데, CUDA 코드로 어떤 데이터를 입력할 것인지 명시해야 한다. 이는 데이터를 필요한 만큼만 읽어올 수 있고, 글로벌 메모리 접근보다 접근 시간이 적게 걸리는 장점이 있다. 공유 메모리를 사용할 때 주의사항이 있는데, 스레드 블록 내에 모든 스레드들이 접근하므로 동기화에 주의해야 한다. 이를 위해 \_\_syncthreads()를 호출하여 모든 스레드들이 해당 지점에 도달할 때까지 기다리며 동기화하는 기능을 수행한다.

세 번째로, 글로벌 메모리는 읽기 전용 메모리 계층으로 모든 응용 프로그램들이 생성한 그리드들이 공유하며 Host와 GPU간의 데이터 교환에도 사용된다. 메모리 용량과 접근 범위가 메모리 계층 중 가장 크고 넓지만, 로컬 메모리나 공유 메모리에 비해 수행 속도가 느리다는 단점이 있다.

네 번째로, 상수 메모리는 읽기 전용 메모리 계층으로 모든 스레드가 접근할 수 있다. 커널 함수에 사용되는 데이터 변수를 상수 메모리로 이용하면 커널 함수의 인자를 줄이고 글로벌 메모리보다 접근성이 좋으며 레지스터 사용도 줄일 수 있는 장점이 있다. 레지스터의 사용을 줄일수록 더 많은 스레드들이 활성화되기 때문에 수행 속도도 빨라져 프로그램이 효율적으로 돌아가게 된다.

다섯 번째로, 텍스처 메모리는 읽기 전용 메모리 계층으로 캐시 읽기를 지원한다. 속도가 빠르기 때문에 이미지의 블러, 샤프닝 등 처리에 사용하는 것에 매우 효율적으로 작용한다. 텍스처 메모리를 사용하는 것은 그래픽 API를 사용하는 것과 유사하여 CUDA 프로그래밍에서 사용하기에 적합하지 않을 때가 있다.