실시간 얼굴 검출을 위한 임베디드 시스템에서의 구현방법

유혜빈*, 박성현*, 정혜원*, 박명숙*, 김상훈*
*한경대학교 전기전자제어공학과
e-mail: gpqls7669@naver.com, psh_mail@naver.com, jhw5631@naer.com,
nicems@nate.com, kimsh@hknu.ac.kr

Real-time face detection in embedded system

Hye-Bin Yoo*, Sung-Hyun Park*, Hye-Won Jeong*, Myung-Suk Park*,
Sang-Hoon Kim*
*Dept of Electrical, Electronic and Control, Hankyong National University

요 약

본 논문에서는 임베디드 GPU 보드를 탑재한 로봇에서의 검출 결과를 원격지에서 확인할 수 있는 방법에 대해 기술하였다. 딥러닝 모델의 연산량을 줄이는 방법 대신 Nvidia에서 제공하는 라이브러리를 이용하여 성능을 개선하였고, 로봇의 배터리 소모를 최소화하기 위해 실시간 영상 통신이 아닌 검출이 되었을 시에만 통신이 되게 하여 보다 긴 구동 시간을 얻도록 하였다.

1. 서론

인공지능은 1950년 Alan Turing이 제안한 Tuning 테스트, Learning Machine을 시작으로 현재까지 계속 연구가 진행되고 있는 분야이다. 이후 2012년 AlexNet의 등장으로 딥러닝의 전성기를 맞이하게 되었다. 딥러닝 전문 국제 학술대회(ICLR:International Conference for Learning Representations) 2019 논문 주제 Top 1은 압도적인 차이로 강화학습이었고, 2020 논문 주제도 Top 1과 2는 각각 딥러닝과 강화학습이었다[1]. 딥러닝의 발달로 영상처리와 관련된 부분도 더욱 활발한 연구가 진행되는 상황이다.

본 논문에서는 이러한 딥러닝 기술을 제한된 플랫폼에 적용하고, 원격으로 확인했을 때 충분한 성능이 보장되는지를 분석한다.

2.본론

2.1. 객체 검출

영상 분석 및 컴퓨터 비전 분야의 기본 요소 기술로 객체 검출은 그동안 많은 연구가 진행되어왔다. 객체 검출은 정해진 클래스의 객체들과 위치 정보를 찾는 문제이다. 이 기술은 기술 개발 및 성능의 객관적 비교를 위해 데이터 셋(Dataset)이 필수적이며, 특히 공개 데이터 셋인 이미지넷(ImageNet)을 이용하여 평가하는 ILSVRC(ImageNet Large Scale Visual Recognition Challenge)대회가 진행되며 기술의 발전에 크게 기여한 것을 그림 1을 통해 알 수 있다.

그림 1. ILSVRC 객체 검출 성능 변화[2]

대회를 대략적으로 살펴보면 2012년 AlexNet의 등장으로 기술적 흐름을 딥러닝 기반으로 변경하게 되는 중요한 계기가 된다. 2014년에서는 현재 딥러닝 기술 개발에 활용되고 있는 주요한 구조인 VGGNet이 등장하였고, Inception 모듈에 기반한 구조도 등장하였다. 2017년은 ILSVRC의 마지막 대회였고, 이후 대회는 Kaggle에서 관리 및 운영될 예정이다.

이처럼 객체 검술 기술은 ILSVRC 대회를 중심으로 발전해왔으며, 이러한 기술들을 경량화 및 고속화를 하려는 연구가 계속 진행되고 있다. 대표적으로는 Mobilenet으로, 기존 모델 내의 기본 모듈을 변경하여 파라미터와 연산량을 획기적으로 줄이는 방법이 고안되었다. 모델뿐만 아니라 개발 도구에서도 모바일 환경을 지원하기 위한 Tensorflow Lite, caffe2, CoreML등이 공개되고 있는 추세이다[2].

2.2. Jetson TX2

최근 데이터의 고속 처리를 위한 여러가지 임베디드 GPU 보드가 출시되었는데, 그중 가장 대표적인 임베디드 GPU 보드로는 Nvidia에서 만든 Jetson TX2가 있다.[4] Jetson TX2 중에서도 TX2 4GB, TX2, 그리고 TX2i 3개의 종류로 나뉜다. 그중에서도 실험에 사용할 TX2 모듈의 기본 사양에 대해 간단하게 말하자면 256개의 NVIDIA CUDA 코어 장착한 GPU, 듀얼코어와 쿼드코어를 장학한 CPU, 8GB의 메모리, 보드에 장착된 와이파이가 있다[3]. 아래의 그림 2는 Jetson TX2 모듈이다.

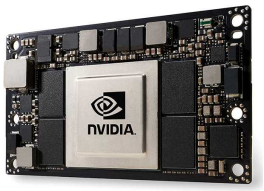

그림 2. Jetson TX2[3]

Jetson TX2는 Jetson TX1 모듈의 모든 기능을 지원하면서도 더 크고 복잡한 딥 뉴럴 네트워크를 지원한다. Jetson 시리즈 자체가 주로 AI 및 딥러닝 응용 프로그램을 위해 개발되었는데, 그중에서도 Jetson TX2는 저전력 엔벨로프에서 높은 계산 성능이 필요한 애플리케이션에 이상적이다. 본 논문에서는 원격에 있는 로봇에 탑재된 임베디드 GPU 보드에서의 성능을 알아보기 위함이기 때문에 TX2는 가장 적합한 모듈이라고 볼 수 있다[4].

추가적으로 NVIDIA에서 제공하는 Jetpack SDK를 Jetson TX2에 설치하여 딥러닝, 컴퓨터 비전 등의 라이브러리를 지원받을 수 있다. Jetpack을 설치함으로써 TensorRT, cuDNN, CUDA 툴킷 등을 제공받는다. 임베디드 시스템에서 이러한 라이브러리들을 사용하여 딥러닝 모델의 연산량을 줄이지 않아도 모델의 성능을 개선할 수 있다.

2.3. TCP/IP 무선 통신

Jetson TX2에는 보드에 와이파이와 블루투스가 기본적으로 장착되어 있다. 기본적으로 두 개의 통신방식은 서로 다른 기술표준을 사용하기 때문에 충돌이 나지 않는다.

블루투스 통신의 경우 데이터 전송속도가 크게 빠르지 않아도 되는 경우 사용한다. 사용 주파수는 2.4 GHz이고, 대역폭은 800kbps 정도로 낮고, 통신 범위는 5~30m 정도로 넓지 않다. 만약 블루투스로 로봇과 원격지가 통신을 하게 된다면 그 범위가 매우 좁아질 것이다. 블루투스 통신을 상징하는 마크는 그림 3과 같다.

그림 3. Bluetooth

와이파이 통신의 경우 많은 데이터 송수신과 빠른 통신 속도가 필요한 경우에 사용한다. 사용 주파수는 2.4 GHz, 3.6 GHz, 5 GHz이다. 대역폭은 11Mbps로 블루투스보다 높은 편이며 통신 범위는 실외 최대 95m까지 가능하다. 와이파이는 안테나 범위 확장이 가능하여 더 넓은 거리로 통신이 가능하다. 와이파이 통신을 상징하는 마크는 그림 4와 같다.

그림 4. Wifi

본 논문에서는 보다 넓은 통신 거리가 필요하기 때문에 와이파이 통신 방법을 사용한다. 무선 통신에서 사용할 수 있는 프로토콜 중 통신속도와 패킷 오류가 없는 전송 또한 필요하기 때문에 TCP/IP 소켓 통신을 이용한다.

네트워크 프로그래밍에서의 소켓이란 데이터를 송수신할 수 있도록 네트워크 환경에 연결할 수 있게 만들어진 연결부이다. 네트워크에 연결하기 위해서는 프로토콜(Protocol)에 맞게 만들어져야 한다. 보통 OSI 7 Layer(Open System Interconection 7 Layer)의 네 번째 계층인 TCP상에서 동작하는 소켓이 TCP/IP 소켓이다. 두 개의 프로세스가 소켓을 통해 연결되기 위해서는 클라이언트 소켓과 서버 소켓이 필요하다. 이 둘은 태생적으로 구조가 다른 소켓이 아닌, 호출되는 API 함수의 종류와 순서들이 다를 뿐 동일하다. 기본적인 소켓 API는 처음 소켓을 생성한 다음, 서버 측에 연결을 요청한다. 이후 서버 소켓에서 연결이 받아들여지면 데이터를 송수신하고 모든 처리가 완료되면 소켓을 닫는 이러한 흐름으로 이루어져 있다[5].

3. 실험 및 고찰

Jetson TX2에서 검출한 화면을 원격지에서 오류 없이 계속해서 확인할 수 있도록 하는 것이 본 논문의 목표이다. 관리자가 사용할 리모트 컨트롤러는 자바를 사용한 안드로이드 스튜디오를 이용한다. 우선 그림 5와 같이 기본적인 로봇의 동작을 제어하는 버튼, Jetson TX2 서버와 연결하는 버튼, Jetson TX2를 탑재한 로봇이 얼굴을 감지했을 경우 사용할 이미지 요청 버튼과 좌표 요청 버튼, 이미지를 보여주는 버튼으로 기본 디자인을 완성하였다. 이후 각각의 버튼과 View들에 맞는 코드를 작성한다. 네트워크 연결 방식은 TCP/IP 소켓 통신방식을 채택하여 얼굴 검출이 된 화면을 깨지지 않고 가져오도록 하였다.

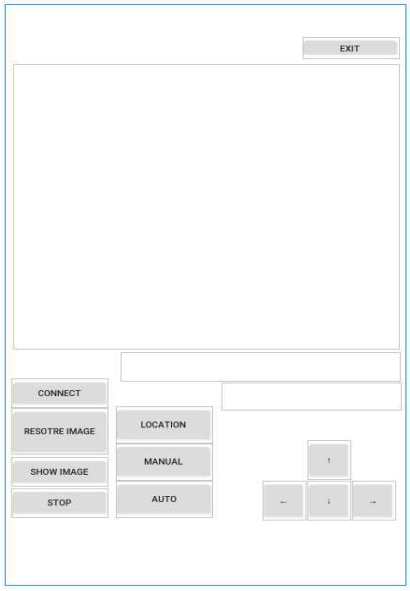

그림 5. 기본 디자인

안드로이드 버전 3(허니컴) 이후부터는 메인 쓰레드에서 네트워킹 처리를 할 수 없어 별개의 스레드를 만드는 방식이 필요하다. 물론 안드로이드 버전이 낮춰 사용할 수 있지만, 허니컴 이전의 점유율은 이제 채 5%도 되지 않는다. 이때, 새로운 스레드가 아닌 AsyncTask를 생성하여 네트워킹 처리를 하게 되면, 보다 간단하게 사용할 수 있다. 그림 6은 본 논문에서 사용한 안드로이드 스튜디오 코드 중 AsyncTask의 일부이다.

```
public class NetworkTask extends AsyncTask<Void, Void, Void>{
    String dstAddress;
    NetworkTask(String addr){
        dstAddress = addr;
    }
    @Override
    protected Void doInBackground(Void... arg0){
        try{
            Socket socket = new Socket( host: "dstAddress",  port: 6666);
```

그림 6. android studio AsyncTask code

그림 7은 검출과 통신에서 사용할 전체적인 알고리즘이다. 로봇에 탑재된 Jetson TX2가 얼굴을 검출하여 경계박스가 생성되면 설정해놓은 특정 디렉토리에 검출된 이미지가 생성된다. 서버 측에서 해당 디렉토리에 이미지가 새롭게 생성이 된 것을 확인하면 리모트 컨트롤러에 특정 신호를 전송한다. 디렉토리의 파일 변화를 감지하는 방법으로는 변수로 파악하는 방법도 있지만, 자바에서 제공하는 WatchService라는 기능이 있다. 이를 이용하여 디렉토리의 여러 변화들을 감지할 수 있다. 그림 8은 본 논문에서 사용한 WatchService 코드의 일부분이다. 리모트 컨트롤러가 디렉토리에 변화가 있다는 신호를 수신하면 로봇에게 이미지와 좌표를 요구한 후 띄우게 하는 알고리즘이다.

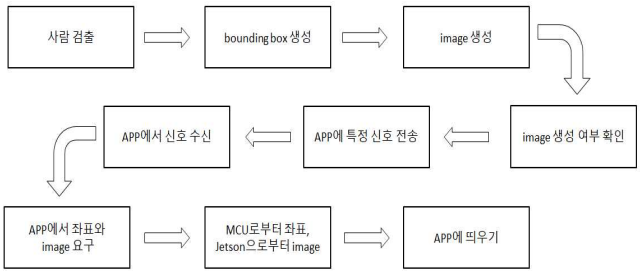

그림 7. 통신 알고리즘

```
for(WatchEvent<?> event : events) {
    Kind<?> kind = event.kind();
    Path pth = (Path)event.context();
    if(kind.equals(StandardWatchEventKinds.ENTRY_MODIFY)) {
        try {
            sout = comPort.getOutputStream();
            String signal = "signal";
```

그림 8. WatchService

이때, 실시간으로 영상을 보여주는 형식이 아닌 이미지로 전송을 하는 이유는 Jetson TX2가 로봇에 탑재되어 리튬 배터리로 전원을 공급받기 때문이다. 실시간으로 영상을 보여주는 형식으로 실험해본 결과, 영상에 노이즈가 끼거나 깨지는 형식으로 전송이 되는 경우도 빈번하였고, 로봇의 구동 시간이 이미지로 전송하는 방식에 비해 짧았기 때문에 실시간 영상 송수신 대신 경계 박스가 생성이 될 때마다 이미지를 만드는 형식으로 알고리즘을 구성하였다. 그림 9는 해당 코드의 일부분이다.

```
def draw_bboxes(self, img, boxes, confs, clss):
    """Draw detected bounding boxes on the original image."""
    for bb, cf, cl in zip(boxes, confs, clss):
        cl = int(cl)
        x_min, y_min, x_max, y_max = bb[0], bb[1], bb[2], bb[3]
        color = self.colors[cl]
        cv2.rectangle(img, (x_min, y_min), (x_max, y_max), color, 2)
        txt_loc = (max(x_min+2, 0), max(y_min+2, 0))
        cls_name = self.cls_dict.get(cl, 'CLS{}'.format(cl))
        txt = '{} {:.2f}'.format(cls_name, cf)
        img = draw_boxed_text(img, txt, txt_loc, color)
        cv2.imwrite("/home/hyebin/image/frame.jpg", img)
```

그림 9. 검출 시 이미지 생성

그림 10은 Jetson TX2를 탑재한 로봇과 리모트 컨트롤러로 통신을 하여 경계박스가 생성되었을 시의 프레임과 좌표를 저장하여 해당 이미지와 로봇의 좌표를 띄운 결과이다. 이미지 패킷이 깨지지 않고, 좌표 또한 정상적으로 출력이 된 것을 확인할 수 있다.

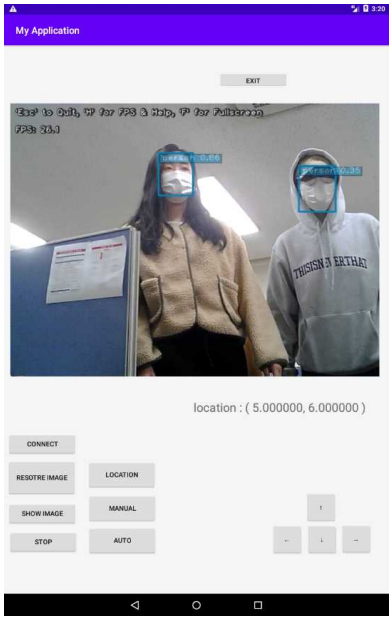

그림 10. 실제 화면

4. 결론 및 향후 연구계획

본 논문에서는 임베디드 보드를 탑재한 로봇과 멀리 떨어진 위치에서의 얼굴 검출을 확인하는 방법에 대해 기술하였다. Jetson TX2를 모니터를 이용하여 바로 확인하게 되면 검출 결과를 바로 알 수 있지만 이를 로봇에 탑재하게 되면 바로 알 수 있게 통신하는 방식이 필요하기 때문에 본 논문의 아이디어를 고안했다. 또한, 지속적으로 전원을 인가받을 수 없는 상황이기 때문에 전력 소모가 적으며 보다 정확하고 뚜렷한 검출 결과를 얻는 것이 중요하기 때문에 오류제어, 혼잡제어 등의 기능이 있는 TCP/IP 통신방식을 선택하여 경계박스가 생성되었을 때만 이미지를 전송할 수 있게 하였다. 결과적으로 비교적 긴 시간동안 경계박스가 생성될 때마다 뚜렷한 이미지를 얻을 수 있게 되었다.

향후 실시간 영상 송수신을 하며 최대한 적은 배터리 전력 소모로 로봇을 구동할 수 있는 방안을 모색해서 실험을 해볼 예정이다.

감사의 글

이 논문은 2020년도 정부(교육부)의 재원으로 한국연구재단 기초연구사업의 지원을 받아 수행된 연구임 (No.2020R1F1A1067496)

참고문헌

[1] AI와 최신 딥러닝 기술 동향 - AI빅데이터연구소 이주열 연구소장

[2] 딥러닝 기반 객체 분류 및 검출 기술 분석 및 동향 - 이승재, 이근동, 이수웅, 고종국, 유원영

[3] https://www.nvidia.com/ko-kr/autonomous-machines/embedded-systems/jetson-tx2/

[4] 고성능 IoT 데이터 스트림 처리를 위한 Nvidia Jetson TX2의 성능 평가 - 남윤성, 신현일, 최석원, 유병훈, 엄현상

[5]https://ko.wikipedia.org/wiki/%EB%84%A4%ED%8A%B8%EC%9B%8C%ED%81%AC_%EC%86%8C%EC%BC%93