화재 대응을 위한 불꽃 시스템 연구

경희대학교 컴퓨터공학과 캡스톤디자인1

2015104170 김효준 2015104195 이동찬 2015104216 장준영

요약

화재는 인명 및 재산의 손해를 가져오는 치명적 재난이다. 따라서 신속한 화재 인지와 경고는 화재 피해를 줄이기 위해 중요하다. 본 논문에서는 카메라 감시 시스템과 인공지능 기술을 통한 엣지 컴퓨팅 수준의 실시간 불꽃 인식 시스템을 제안한다. 이 시스템에서는 객체 인식 모델을 사용하여 불꽃의 존재와 위치를 파악한다. 본 연구를 통해 기존 화재 감지기와 더불어 빠른 화재 대응을 도울 수 있는 시스템을 구현한다.

1. 서론

1.1. 연구 배경

우리는 수많은 사물 인터넷(Internet of Things) 장치들을 기반으로 초연결(Hyperconnected) 사회에 진입해 있다. 얻어진 방대한 양의 데이터를 빅 데이터 기술 및 인공지능 기술을 통해 데이터를 가공하고 정보를 만들어 제공한다. 이 과정에서 많은 컴퓨팅 자원이 요구되며 이를 위해 클라우드 컴퓨팅(Cloud Computing)을 사용한다. 그러나 실시간 시스템(Real Time System)은 시간적인 제약을 갖는다. 만약 클라우드 컴퓨팅 환경에서 네트워크의 상태에 따른 지연이 발생한다면, 이는 치명적인 손실을 가져온다. 특히 화재와 같은 재난을 경고하는데 지연이 발생하면 돌이킬 수 없는 피해를 초래한다.

통계청의 통계 결과, 최근 10년간의 통계를 보면 국내 재난사고 발생 중 화재와 산불이 약 19%으로 약 78%인 도로교통 다음으로 높았다. 전국의 화재 건수는 17년 기준 44,178건, 18년 42,338건, 19년 40,102건으로 매년 줄어들고 있다. 그러나 화재는 약 4만 건으로 지속적으로 발생하고 있으며, 이는 많은 인명 및 재산 피해로 이어지고 있다. 모든 사고와 재난은 초기

대응을 적절히 했을 때 피해를 최소화할 수 있는 것이 사실이다. 따라서 화재에 따른 빠른 대응이 가능하도록 화재의 발생을 인지하고 알리는 것은 중요하다.

본 연구는 클라우드 의존성을 배제하고 실시간 시스템을 보장하는 엣지 컴퓨팅(Edge Computing)[1] 시스템을 제안한다. 여러 SBC(Single Board Computer) 중, 라즈베리 파이(Raspberry Pi)에서 객체 인식(Object Detection) 모델을 구동하여 불꽃을 인식하고 다른 모바일 장치와 통신한다. 비교적 낮은 컴퓨팅 자원을 가진 라즈베리 파이 위에서 시스템을 구동하기 위해 TensorFlow의 양자화(Quantization)[2] 기법과 MQTT(Message Queueing Telemetry Transport)[3]를 이용하여 효율을 올린다. 이를 통해 빠르게 불꽃을 예측하고 화재의 대응을 도울 수 있는 시스템을 구현한다.

1.2. 연구 목표

첫째, 딥러닝 기반 인공지능을 통해 불꽃을 인식한다. 영상 속에서 불꽃의 위치를 확인할 수 있도록 object detection 모델을 구현한다. 또한, 모델이 다른 환경에 비해 비교적 자원이 한정된 raspberry pi 위에서 구동될 수 있게 모델을 압축한다.

둘째, 현재 상황에 대한 정보를 사용자에게 전달한다. 현재 감시하고 있는 환경을 사용자가 확인할 수 있도록 실시간 스트리밍과 화재 발생 시 사용자에게 보여질 정보를 제공할 수 있는 API를 구현한다.

셋째, 사용자 인터페이스를 통해 사용자의 올바른 대응을 돕는다. 제공 받은 정보를 사용자가 직관적으로 이해할 수 있도록 표현한다. 사용자에게 상황을 인지시키고 사용자의 판단에 따른 대응이 가능한 인터페이스를 구현한다.

위 3개의 목표에 따라 시스템의 기능을 구현하여 화재 대응을 위한 불꽃 인식 시스템을 완성한다.

2. 관련 연구

2.1. 기존 연구

2.1.1. 딥러닝(Deep Learning)

딥러닝은 데이터의 관계를 데이터를 통해 학습시켜 인공지능을 구현하는 기계학습의 한 방법이다. 컴퓨팅 자원의 발전에 따른 빠른 연산과 높은 정확도로 딥러닝 모델은 일부 과제에서 사람보다 뛰어난 능력을 보이고 있다. 그러나 수많은 매개변수를 가지고 있기 때문에 추론과정에서도 많은 연산을 요구한다. 따라서 학습된 딥러닝 모델은 컴퓨팅 자원이 뒷받침되지 않을 때 속도의 지연을 야기할 수 있으며 이는 실시간 시스템을 보장하지 않는다.

2.1.2. Existing communication technology of Raspberry Pi

기존의 라즈베리파이 통신기술로는 크게 유 / 무선 통신이 있다. 이 중 우리가 중점적으로 보는 부분은 무선 통신이다. 무선 통신의 종류로는 bluetooth, wifi 등이 있다.

블루투스는 수 미터에서 수십 미터 정도의 거리를 둔 정보기기 사이에, 전파를 이용해서 간단한 정보를 교환하는데 사용된다. 블루투스 통신은 단거리의 크기가 작은 데이터를 대상으로 하는 것이 특징이다. 이로 인해 블루투스는 단거리 기기간의 통신에 주로 하여 사용된다.

와이파이는 전자기기들이 무선랜에 연결할 수 있도록 하는 기술로, 대게 기기와 인터넷을 연결하여 web상의 데이터에 접근하도록 하게 해준다.

	와이파이(Wi-Fi)	블루투스(Bluetooth)
주파수(Frequency)	2.4GHz, 5GHz	2.4GHz
사용 목적	무선 인터넷 접속(WLAN)	휴대용 전자기기 간의 무선 연결(WPAN)
전송 속도	802.11B : 22Mbps 802.11G : 54Mbps 802.11N : 600Mbps 802.11AC : 1750Mbps	와이파이보다 느리나 버전업하면서 점차 빨라짐 25Mbps (ver.4.0)
스트리밍 데이터의 압축	압축 없이 무손실 전송 가능(AoIP)	대역폭이 좁아서 손실 압축하여야 함
오디오 신호의 지연(Latency)	VoIP : 20~30msec AoIP : 20msec 이하	SBC(Standard Bluetooth Codec): 약 250msec atpX 코덱: 약 180msec atpX LL 코덱: 약 40msec
장점	높은 전송속도 거리가 길다.	저비용 낮은 소비전력
단점	고비용 높은 소비전력	낮은 전송속도 거리가 짧다.

[그림1] Wi-Fi 와 Bluetooth

실시간을 요구하는 프로젝트에서는 wifi가 더 적절하다.

2.1.3. 클라우드 컴퓨팅

클라우드 컴퓨팅은 인터넷으로 연결되어 장치들이 가상의 시스템 리소스를 사용하는 방식이다. 사물인터넷의 장치들은 저전력, 저용량 등의 한계를 극복하기 위해 클라우드 컴퓨팅 기술과 접목되어 사용되고 있다. 그러나 데이터센터가 많은 장치들과 연결되면서 처리할 양이 기하급수적으로 증가하고 있으며 데이터센터가 수용할 수 있는 데이터 양을 초과하면 전송과 처리 속도의 지연이 발생한다. 또 클라우드에 접근하기 위해서는 데이터센터와 장치들이 연결되어 있어야 하는데 연결이 온전하지 않은 경우 지연 문제를 가져온다. 이러한 지연 문제로 클라우드 컴퓨팅은 실시간 시스템에 적용되기에 적합하지 않다.

2.1.4. Webcam streaming & Application UI design(Android studio)

본 연구에서는 불꽃이 인식되면 사용자에게 알람이 가고, 불꽃이 인식된 장소의 상황을 application의 형태로 사용자에게 제공한다. raspberry pi의 webcam을 통해 촬영되는 영상에서 불꽃이 인식되면 실제로 불꽃의 존재를 직접 확인할 수 있는 방법이 필요하다. 본 연구에서는 이를 webcam streaming을 이용해 사용자에게 실시간 영상을 제공한다.

또 화재가 발생했다는 알람을 받을 수 있고, 위에서 언급한 streaming 영상을 볼 수 있는 환경을 제공하기 위해, android studio를 이용하여 간결하고 쓰기 용이한 UI로 구성된 application을 구현한다.

2.2. 기존 연구의 한계

2.2.1. 성능을 위한 높은 비용

여러 연구자들이 지적하고 있듯이 딥러닝 모델의 성능과 모델의 무게는 trade-off 관계에 놓여있다. 지금까지 Object detection의 성능에서 엄청난 발전이 있었지만 그에 따라 많은 연산량과 매개 변수를 갖는 것이 사실이다. 따라서 한정된 컴퓨팅 자원에서 높은 성능을 보이는 object detection 모델을 구동하는 것에는 한계가 존재할 수 있다.

본 문서에서 제안하는 연구의 경우, 학습된 object detection 모델을 비교적 한정된 자원인 raspberry pi 위에서 실시간으로 구동되는 것을 하나의 목표로 두고 있다. 따라서 단순히 성능이 좋은 모델을 선택하여 시스템을 구성할 수 없으며 이를 해결할 수 있는 방안이 필요하다.

2.2.2. 비교적 무거운 프로토콜

우리가 일반적으로 웹에서 자주 사용하는 HTTP 의 경우, www 상에서 정보를 주고받을 수 있는 프로토콜이다. 주로 html 문서를 주고받는데 사용되며, HTTP는 클라이언트 / 서버 사이에 이루어지는 응답 / 요청 프로토콜이다. 하지만 이는 모바일 기기를 위해 항상 최적화된 것은 아니다. HTTP가 가지는 장점은 확장성이 좋다는 것이지만 그 장점에 비해 raspberry pi 에서 사용하기에는 비교적 무거워 배터리 소모가 심하고, 3G network 측정 결과에서도 타

프로토콜에 비해 throughput이 좋지 않았다. 확장성을 조금 줄이고, 라즈베리파이에서 사용할 수 있는 가벼운 프로토콜을 사용해야 한다.

2.2.3. 안정적인 스트리밍 영상, UI 제공의 필요성

성공적으로 연구의 목적을 이루려면 object detection까지 완료된 영상이 성공적으로 사용자에게 전송되어야 한다. 따라서 webcam streaming 영상을 실시간으로 안전하고 끊김없이 사용자에게 제공할 수 있는 환경이 필요하다. 또한 raspberry pi에서 영상의 모델링과 전송이 모두 이루어지므로 간단하고 편리하게 webcam streaming을 할 수 있어야 한다. 따라서 이 환경을 제공할 수 있는 적절한 기능을 raspberry pi에 구현해야 한다.

2.3. 해결방안

2.3.1. 모델 압축

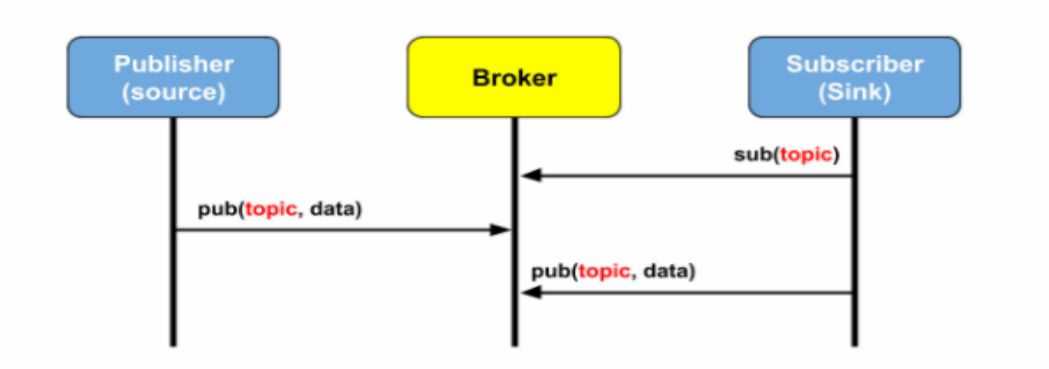
딥러닝 모델의 성능을 유지하며 모델의 크기를 줄이는 방법은 대표적으로 knowledge distillation과 quantization 방법이 있다. knowledge distillation은 잘 학습된 신경망의 softmax 분포를 이용하여 비교적 작은 크기의 신경망에서 학습시킬 때, 같은 크기의 신경망에서 일반적인 방법으로 학습시킨 것보다 성능이 좋음을 보여준다. 또 quantization은 모델의 매개변수를 나타낼 때 사용되는 숫자의 정밀도를 줄이는 것으로 더 작은 모델 크기와 더 빠른 연산이 가능하게 한다.

제안하는 시스템에서와 같이, raspberry pi 위에서 불꽃을 정확하고 지연 없이 인식하기 위해서는 모델 압축이 필요하다. 위와 같은 해결 방안을 토대로 실용성 있는 방법을 통해 모델을 압축한다.

2.3.2. MQTT

Message Queue Telemetry Transport의 약자인 MQTT는 통신장비, 모바일, 스마트폰 기기에 최적화된 가벼운 메시지 프로토콜이다. 현재 IoT에서 많이 사용되고 있다. MQTT가 IoT에 많이 사용되는 이유는 IoT와 같은 라이트 장비에 단순 메시지 교환을 위해 무거운 프로토콜을 사용하게 되면 성능 저하가 발생한다. 이를 해결할 수 있는 MQTT는 Broker라는 중계자가

존재해 중계자를 통해 Topic을 선정해서 subscribe하면 해당 Topic이 Publish 되었을 때 구독자들에게 메시지를 전달해주는 방식이다. 이를 간단하게 표현하면 아래 [그림2]과 같다.

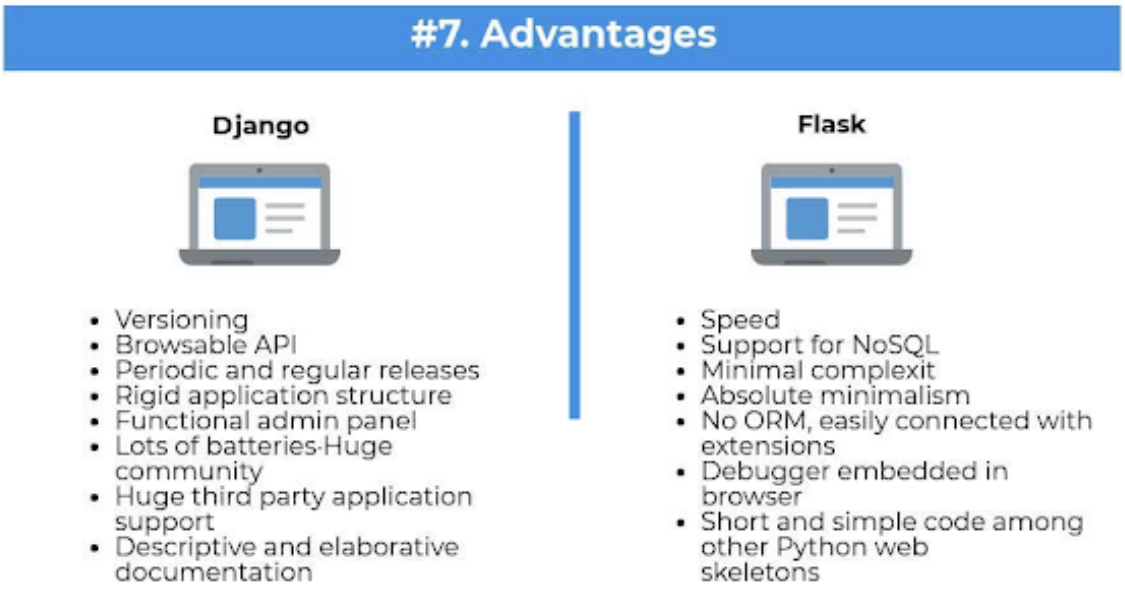


[그림2] MQTT의 동작

MQTT 브로커에는 다양한 종류가 있다. ActiveMQ, Apollo, JoramMQ, Mosquitto, RabbitMQ등 다양한 MQTT 브로커가 존재하는데, 이러한 MQTT 브로커를 통해 메시지를 주고받을 수 있고, 모바일 기기와의 통신도 가능하다. MQTT의 특징을 설명하면, 단순하고 가벼운 메시지 프로토콜이며 오버헤드를 최소화하기 위해 헤더 크기를 대폭 줄였다. 또한 클라이언트 서버간 연결이 끊어졌을 때 보정 기능을 제공한다는 장점이 있다. HTTP와 마찬가지로 TLS/SSL을 지원해 메시지 암호화도 할 수 있다.

2.3.3. Flask

딥러닝 모델을 통해 추론된 영상은 웹서버를 거쳐 사용자에게 제공된다. 화재가 검출되지 않는 상황에서도 사용자가 상황을 확인할 수 있게 라즈베리 파이의 파이카메라로 영상을 촬영하고 이를 Flask를 통해 볼 수 있는 웹서버를 구현했다. MJPG-streamer 또는 OctoPrint[4]를 활용하면 손쉽게 영상을 스트리밍할 수 있으나 학습한 영상을 스트리밍할 수는 없었고, Django와 비교했을 때 라즈베리 파이의 부하를 줄이기 위하여 비교적 가벼운 Flask를 사용한다.



[그림3] Django vs Flask

3. 프로젝트

3.1. 시스템 시나리오

3.1.1. 실시간 스트리밍

[그림4] 실시간 스트리밍 영상

실시간으로 화재가 발생했을 시, raspberry pi의 webcam 영상을 딥러닝 서버로 보내 처리한다. 그 다음 Flask를 통해 처리된 영상을 웹서버에 스트리밍 하고, 해당 url을 application내에서 연결한다. [그림4]는 application에서 보여지게 될 실시간 streaming 영상 화면이다. 이와 같이 object detection이 이루어진 영상을 사용자에게 제공함으로써, 실시간으로 화재 상황에 대한 영상정보를 제공할 수 있다.

3.1.2. 화재 발생 알림

[그림5] 알림 화면

raspberry pi에서 모델을 통해 object detection이 되면, MQTT 프로토콜을 통해 모바일로 알림을 보내게 된다. 알림 메시지는 Mosquitto broker 주소에 연결된 MQTT Client 객체가 subscribe 대기하는 것으로, 메시지가 도착했을 때 [그림5]와 같이 메인 스레드에서 실시간 영상을 확인 버튼을 포함한 대화상자의 형태로 띄워지도록 구현했다. 사용자는 알림에서 제공되는 버튼 클릭을 통해 화재가 인식되었을 때, 바로 불꽃이 detection된 영상에 대한 streaming service를 제공받을 수 있다.

3.1.3. 사용자 인터페이스(default GUI)

[그림6] 메인 화면

[그림7] 알림 화면

[그림8] 실시간 영상 확인

[그림6]은 본 연구에서 제공할 application의 메인 화면이다. 불을 인식하고 찾는다는 의미에서의 불 그림과 돋보기 모양, dr.fire이라는 이름을 통해 화재를 인식하는 application임을 나타냈다. 메인화면에서는 '실시간 영상 확인' 기능을 통해 사용자가 원할 때, 카메라가 설치된 장소의 streaming 영상을 볼 수 있다.

[그림7]는 실제 화재가 인식되었을 때 나타나는 알림 표시 화면으로써, 화재가 인식되어 mqtt를 통해 알림 메시지가 도착했을 때, 실시간 영상을 확인 버튼을 포함한 대화상자의 형태로 알림을 띄운다. 그 알림에 영상 확인 버튼을 제공함으로써, 사용자가 바로 상황을 확인할 수 있게 도와주도록 UI를 설계하였다.

[그림8]은 [그림7]에서 '예' 버튼이나 [그림6]에서 '실시간 영상 확인' 버튼을 클릭하였을 때 나타나는 실시간 영상 확인화면으로, 실제 화재가 발생했음을 나타내고 어디 화재가 나타났는지 object detection 처리된 영상을 통해 사용자에게 실시간으로 보여주게 된다.

3.2. 요구사항

3.2.1. 스트리밍 및 알림을 위한 API

앞서 말했던 MQTT에는 여러 broker가 있다. 이 broker를 분석한 그림은 다음과 같다.

SERVER	QoS0	QoS1	QoS2	bridge	SSL	Dynamic Topics
ActiveMQ	V	V	V	X	V	V
emitter	V	X	X	X	V	V
JoramMQ	V	V	V	V	V	V
Mosquitto	V	V	V	V	V	V
MQTT.js	V	V	V	X	V	V
RabbitMQ	V	V	X	X	V	V

[그림 9] MQTT broker 성능 분석 표

모든 기능이 필요한 것은 아니지만, 기본적으로 서비스의 질을 보장해주는 레벨인 QoS의 모든 레벨을 지원해주는 broker 중에서 자주 사용되는 Mosquitto를 이용한다. MQTT의 paho,mqtt.client api와 mosquitto에서 제공해주는 broker 사이트를 활용해 알림을 보낸다.

3.2.2. 스트리밍 및 알림을 위한 object detection 모델

Raspberry pi의 카메라 모듈을 통해 입력 받은 영상이 실시간으로 스트리밍 되기 이전에, 불꽃을 인식하는 object detection 모델을 거쳐야한다. 화재는 빠른 대응이 중요한 문제이므로 모델의 이미지 처리 속도를 50ms 내로 구현한다.

입력 이미지에 불꽃이 존재하여 알림 발생 시, 적절한 정보를 제공하기 위해 이미지의 불꽃 객체 위치에 bounding box가 그려진 이미지를 제공해야한다. 이를 위해 모델은 bounding box의 좌표들과 객체의 정확도를 최종 출력으로 한다.

3.2.3. 사용자 인터페이스

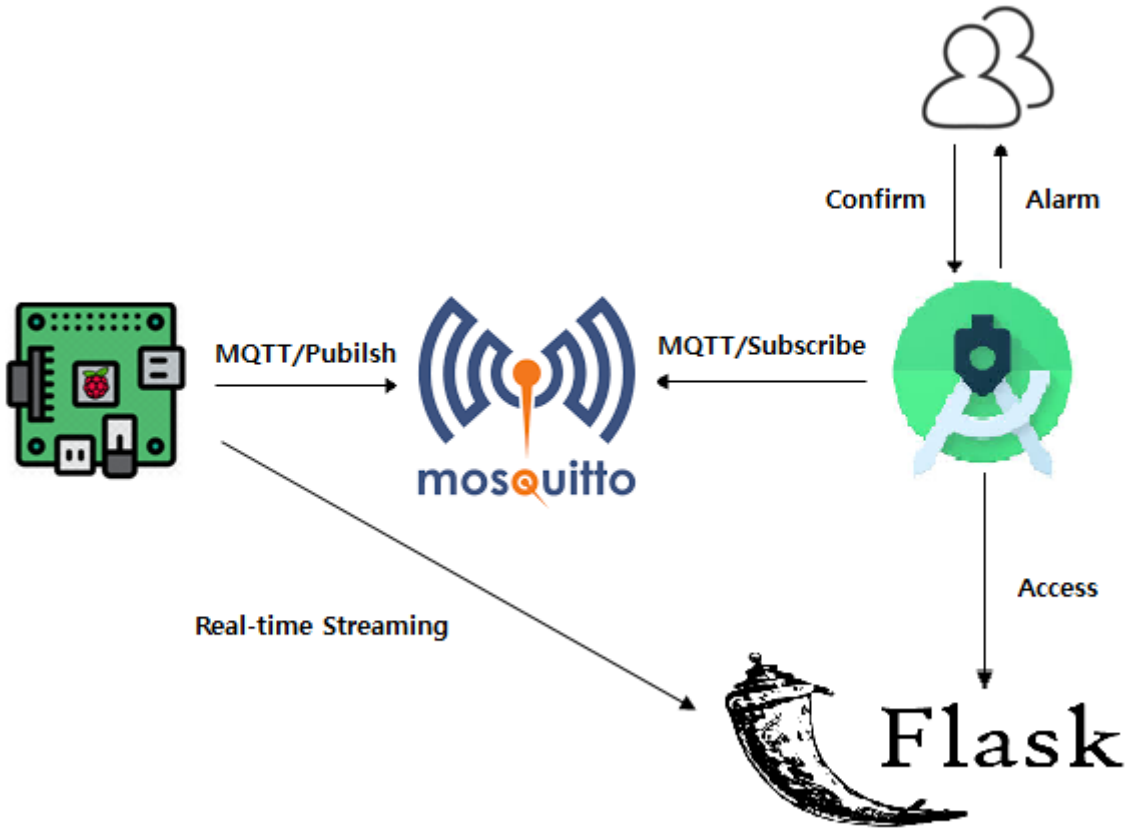
간결하고 시각적으로 보기 좋은 디자인을 선택해야 한다. 또한 화재와 관련된 application인 만큼, 붉은색 위젯 위주의 디자인으로 설계한다. 기본적인, 필수적인 디자인 외에는 불필요한 디자인을 추가하지 않는다.

스트리밍 영상을 보는데 지연이나 화질이 본래의 영상보다 최대한 나빠지지 않도록 재생 환경을 구축한다. 또한 알림이 왔을 때, 알림 화면이 뜨는데 지연이 생기면 application의 목적이 사라지므로 이에 유의하여 프로그램을 설계한다.

3.3. 시스템 설계

3.3.1. 시스템 구성도

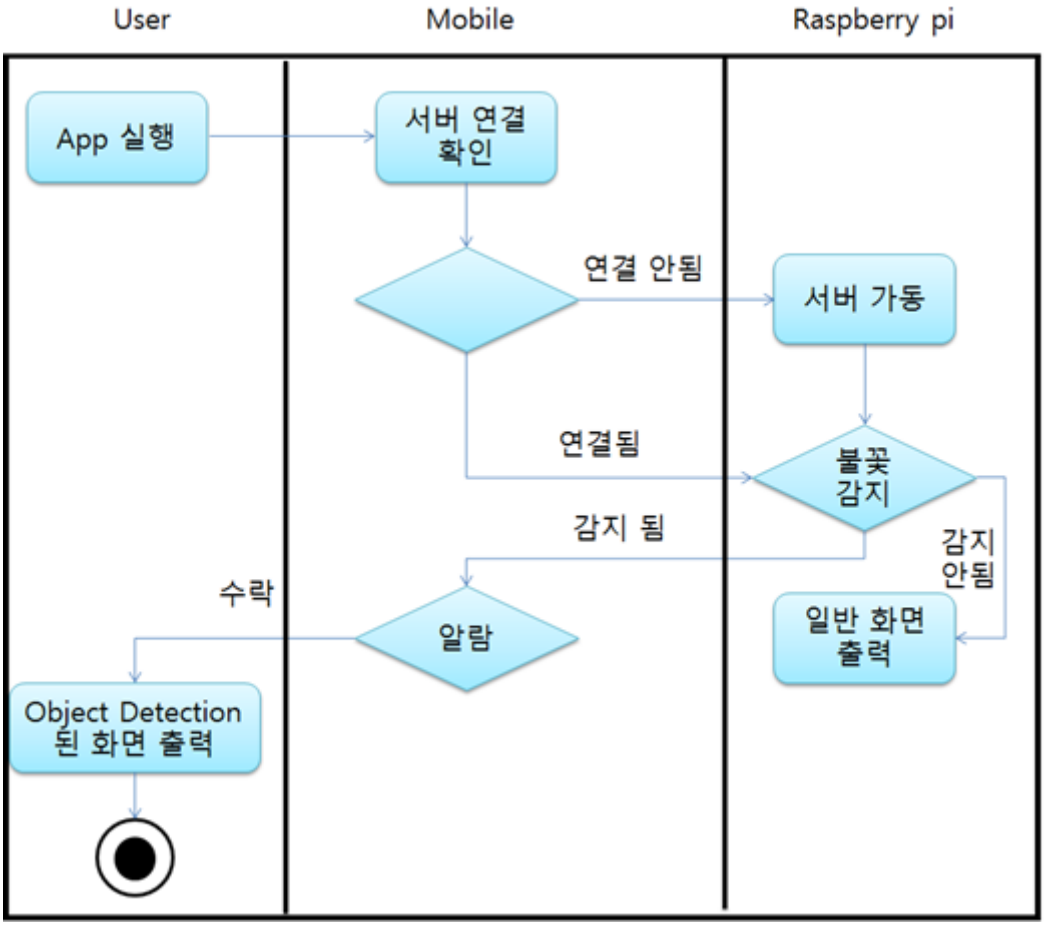
시스템 구성도는 [그림 10]과 같다.



[그림 10] 시스템 구성도

3.3.2. UML Diagram을 통한 시스템 모델링

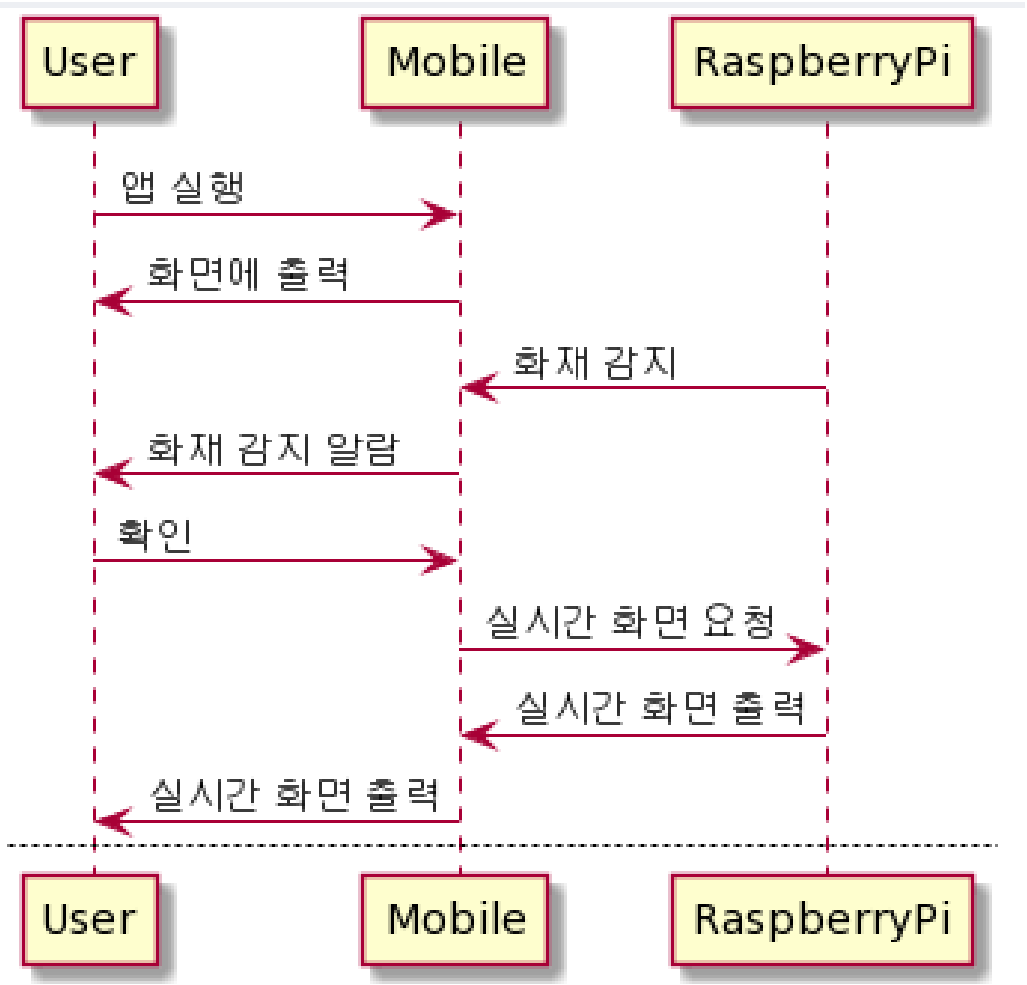
3.3.2.1. Activity Diagram



[그림 11] Activity Diagram

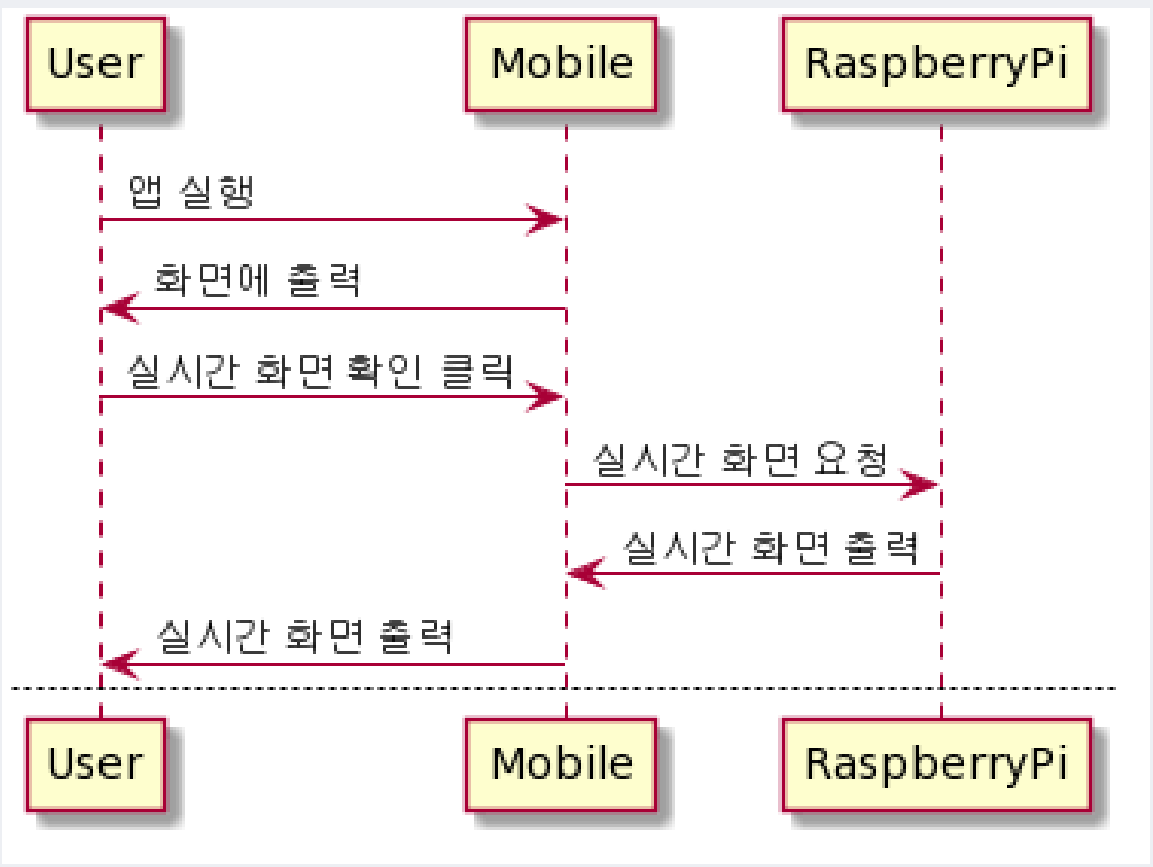
어플리케이션을 실행하면 서버 연결을 확인하고 서버가 연결이 되어있지 않은 경우 라즈베리 파이의 서버를 가동시킨다. 연결이 되어있다면, 계속해서 영상을 찍게 된다. 만약 불꽃을 감지하면 알람이 보내지고, 이를 수락시 detection된 화면이 출력되고, 감지되지 않은 경우 사용자가 요청하면 일반 스트리밍 화면을 출력해서 보여준다.

3.3.2.2. Sequence Diagram



[그림 12] 화재 감지시 Sequence Diagram

앱이 실행되면 화면이 출력된다. 만약 화재가 감지될 경우, 화재 알림을 사용자에게 보낸다. 사용자가 확인을 하게 되면 실시간 화면을 웹에 요청하게 되고, 화면을 출력해 사용자에게 제공한다.



[그림 13] 일반 스트리밍 확인 Sequence Diagram

앱이 실행되면 화면이 출력된다. 일반 스트리밍 경우, 사용자가 실시간 화면 확인을 클릭하면 실시간 화면을 웹에 요청하게 되고, 화면을 출력해 사용자에게 제공한다.

3.4. 구현

쿠 허브 주소: http://khuhub.khu.ac.kr/2020-1-capstone-design1/Spin_Project1

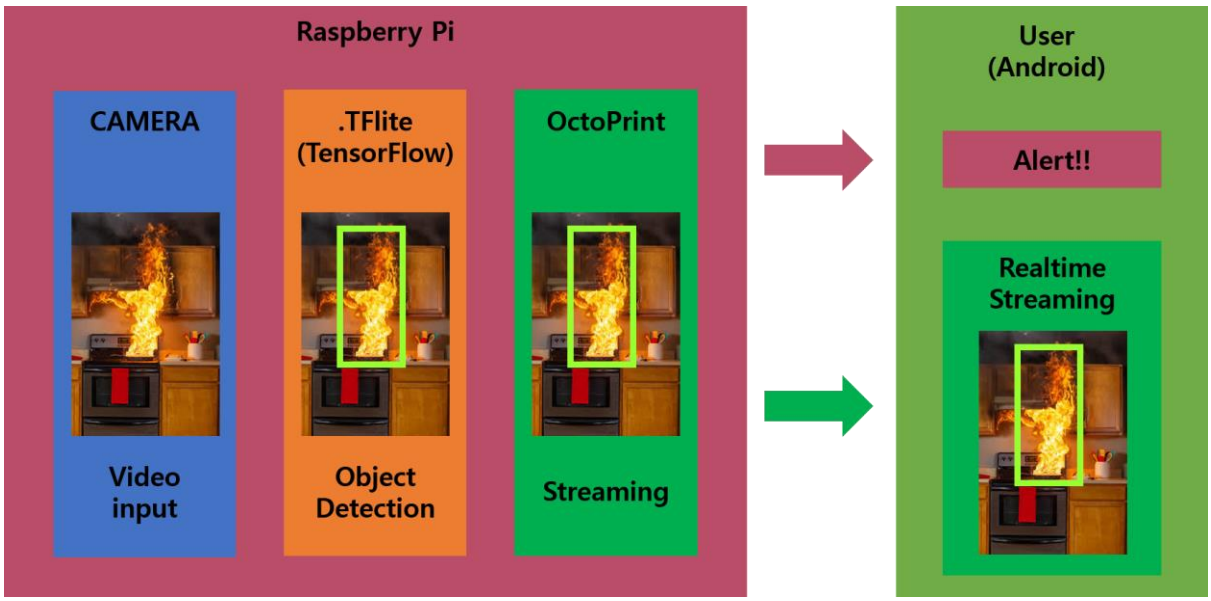
4. 프로젝트 결과

4.1. 연구 결과

이번 프로젝트에서 설계 구조상 가장 중요한 점은 raspberry pi에서 구동될 수 있는 압축된 작은 모델이다. 엣지 컴퓨팅과 딥러닝 모델의 양자화를 통한 실시간 시스템을 제안한다. 데이터 센터와 연결되지 않고 라즈베리 파이 위에서 양자화된 딥러닝 모델의 추론 과정을 수행하는 것으로 화재를 감지할 수 있는 실시간 시스템을 구현했다.

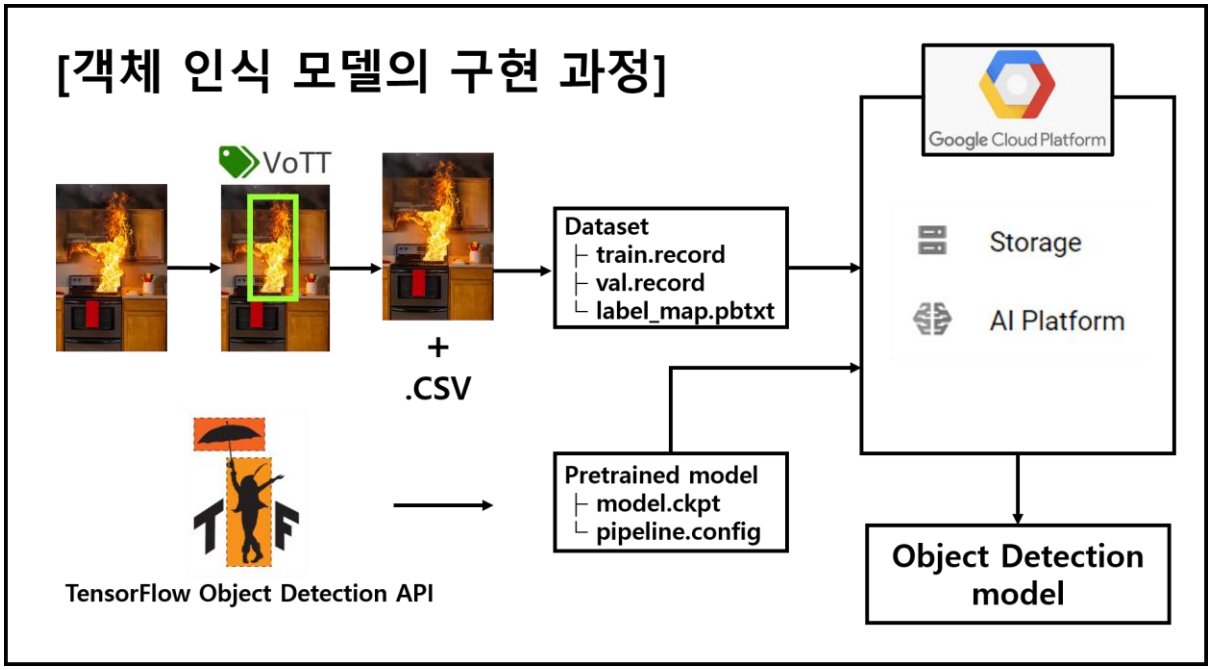
4.1.1. Dr.Fire

구현된 불꽃 인식 시스템(Dr.Fire)은 [그림 14]와 같다. 라즈베리 파이의 파이카메라(pi camera)를 통해 입력을 받고 객체 인식 모델을 통해 불꽃을 인식한다. 이후 촬영되고 추론한 이미지를 사용자에게 영상 스트리밍으로 제공하며 화재가 인식된 경우 MQTT를 통해 어플리케이션으로 알린다.



[그림 14] Dr.Fire

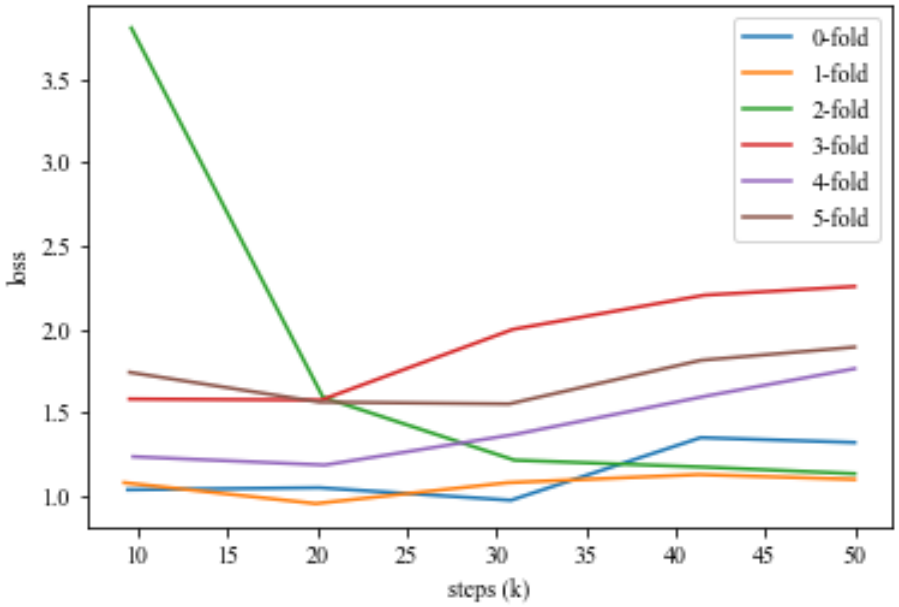
4.1.2. 객체 인식 모델의 구현 및 성능평가



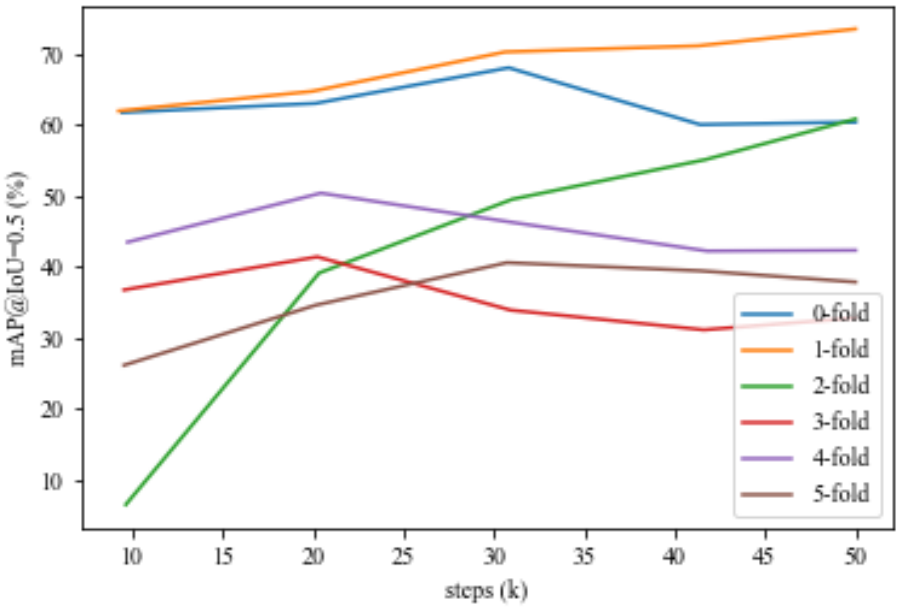
[그림 15] 객체 인식 모델의 구현 과정

객체 인식 모델의 구현은 [그림15]와 같다. 모델은 TensorFlow Object Detection API에서 사전 학습된 mobilenet[5] 기반 SSD[6] 모델을 선택했다. 모델은 전이학습을 통해 Google Cloud Platform의 TPU로 학습되고 GPU를 통해 평가되었다.

데이터 셋을 구성하기 위해 웹과 Kaggle에서 수집된 화재 이미지는 총 365장이었다. 데이터 셋이 작기 때문에 모델 평가의 정확성을 높이기 위하여 k-fold cross validation[7]을 통해 모델을 평가했다. 데이터 셋을 순차적으로 6개 fold로 나누었고 각 fold를 검증집합으로 50,000 steps을 학습시켰다. [그림 16]과 [그림 17]은 step에 따른 eval loss와 IoU(Intersection over Union)가 50% 이상일 때의 mAP(mean Average Precision)를 나타낸 것이다. 이는 데이터 셋의 검증집합에 따라 모델의 성능이 달라질 수 있음을 의미한다. 따라서 전체 fold의 모델 성능을 평균으로 계산하여 모델의 성능을 측정했다.



[그림 16] step과 loss 그래프



[그림 17] step과 mAP@IoU=0.5

n-fold	mAP@IoU=0.5	size of validation (images)
0-fold	68.06	61
1-fold	64.73	61
2-fold	60.84	61
3-fold	41.41	61
4-fold	50.39	61
5-fold	40.59	60
Average	54.34	

[그림 18] 각 fold에 대한 모델의 성능과 평균

[그림 18]과 같이 각 fold에서 검증집합에 대한 손실이 최소일 때 모델을 선택했고 평균을 계산한 결과 mAP@IoU=0.5에서 54.34%의 성능을 보였다.

5. 기대효과 및 결론

본 연구는 화재 대응을 위해 불꽃을 인식하고 대응을 돕는 실시간 시스템을 제안한다. 이는 딥러닝 모델이 낮은 컴퓨팅 성능을 갖는 환경에서 성공적으로 구동되는 것을 증명한다. 또 IoT에 적합한 통신 기법을 통해 시스템의 효율을 올리고 기존의 오픈 소스를 목표에 맞게 활용하는 방법을 보여준다.

기존의 화재감지기는 통풍이 잘되고 화재와 거리가 먼 경우 화재를 감지하지 못하는 문제점을 가지고 있다. 또 CCTV를 통해 사람이 화재의 발생 여부를 감시하는 것은 분명 한계를 가지고 있다. 본 연구의 결과물인 Dr.Fire는 카메라를 통해 딥러닝 모델이 실시간으로 화재를 감지한다. 이를 통해 기존의 화재 감시 방법이 가지는 한계를 극복하고 기존의 방법과 더불어 화재를 감지하고 신속한 대응을 도울 수 있으리라 기대한다.

6. 참고문헌

[1] 신성식, 민대홍, 안지영, 김성민, “엣지 컴퓨팅 시 장 동향 및 산업별 적용 사례”, ETRI, 2019 Electronics and Telecommunications Trends, Apr 2019.

[2] Google, TensorFlow Lite, https://www.tensorflow.org/lite/performance/model_optimization

[3] 김동휘, "MQTT 프로토콜 기반 IoT 환경 및 솔 루션의 효율성 향상을 위한 연구", 한밭대학교 정보 통신전문대학원 석사 학위 논문, Aug 2017.

[4] Octoprint, https://octoprint.org/#full-remotecontrol-and-monitoring

[5] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, arXiv preprint arXiv:1704.04861, Apr 2017.

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, “SSD: Single Shot MultiBox Detector”, ECCV 2016, Dec 2015.

[7] Sebastian Raschka, “Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning”, arXiv preprint arXiv:1811.12808, Nov 2018.