화재 대응을 위한 불꽃 인식 시스템 연구

김효준 이동찬 장준영

경희대학교 컴퓨터공학과

rmfotjgkgk@khu.ac.kr fulleast@khu.ac.kr vnsltdl0152@khu.ac.kr

Study of Fire Detection System for Fire Response

Hyojun Kim Dongchan Lee Junyoung Jang

School of Computer Science and Engineering, Kyung Hee University

요 약

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(Edge Computing) 수준의 불꽃 인식 시스템을 제안한다. 이 시스템에서는 객체 인식(Object detection) 모델을 사용하여 불꽃의 존재와 위치를 파악한다. 본 연구를 통해 기존 화재 감지기가 가지는 한계를 벗어나고 빠른 화재 대응을 도울 수 있는 시스템을 구현한다.

1. 서 론

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 제약이 존재한다. 만약 클라우드 컴퓨팅 환경에서 네트워크의 상태에 따른 지연이 발생한다면, 이는 치명적인 손실을 가져올 수 있다. 특히 화재와 같은 재난을 경고하는데 지연이 발생하면 돌이킬 수 없는 피해를 초래한다.

클라우드에 접근하기 위한 시간을 배제하기 위해서 본 논문에서는 엣지 컴퓨팅(Edge Computing) 수준의 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) 기법과 MQTT를 이용하여 효율을 올린다. 이를 통해 빠르게 불꽃을 예측하고 화재의 대응을 도울 수 있는 시스템을 구현한다.

2. 관련 기술 및 연구

2.1 객체 인식(Object Detection) 모델

컴퓨팅 자원의 발전에 따라 딥러닝(Deep Learning) 기술은 빠르게 발전했다. 그 중 이미지 내에서 객체를 분류하고 그 위치를 특정 짓는 객체 인식 연구 역시 꾸준히 발전되고 있다. 초기의 연구들은 모델의 성능을 높이는 것에 집중했으나, 최근의 연구들은 모델의 크기를 줄이면서도 성능을 잃지 않는 연구들이 이어졌다. 예를 들어, 모델의 적절한 scaling 방법을 찾거나 [1], 학습 후의 추론 과정의 모델을 가볍게 만드는 방법 [2] 등이 연구되었다.

모델의 크기를 줄이는 연구가 많이 진행된 것은 사실이나, 딥러닝 모델은 여전히 수많은 매개변수를 가진다. 모델의 학습은 추론을 잘할 수 있도록 이 매개변수들을 찾아가는 과정이다. 이 과정은 많은 연산을 요구하며 이는 많은 컴퓨팅 자원을 요구함을 의미한다. 즉, 컴퓨팅 기술이 발전했더라도 높은 성능을 가진 컴퓨터를 소유하고 있지 않다면 모델의 학습에 많은 시간이 소비되거나 학습이 불가할 수 있다.

학습을 위해서는 컴퓨팅 자원 외에도 학습할 데이터가 필요하다. 객체 인식의 경우 찾으려는 객체가 포함된 많은 이미지가 요구된다. 해당 시스템이 목표로 하는 화재 상황의 경우, 이러한 데이터 셋을 구성하는데 어려움을 갖는다. 이미지를 모으기 위해

여러 환경에 방화를 행하는 것은 현실적으로 어려운 일이며, 인터넷 상에 존재하는 이미지들은 모델이 학습하기에 부적절한 경우도 많다.

본 연구에서는 이러한 문제를 해결하기 위해 클라우드 위에서의 학습과 전이학습을 제안한다. 학습을 위한 컴퓨팅 자원이 충분하지 않은 상황을 해결하기 위해 GCP(Google Cloud Platform)를 사용한다. 또 부족한 데이터 셋을 위해 사전 학습된(pre-trained) 모델을 사용하는 전이학습(Transfer Learning)을 수행한다. [3] 이를 통해 불꽃을 인식할 수 있는 객체 검출 모델을 구현한다. 이후 TensorFlow의 양자화를 통해 추론 과정의 모델 크기를 줄이고, 비교적 낮은 컴퓨팅 자원을 가진 라즈베리 파이에서 모델이 구동되는 것을 목표로 한다.

2.2 통신 기술

화재를 대응하기 위해서는 빠른 알림이 필요하다. 최소한의 정보를 제공하면서, 가볍고 빠른 통신이 필요하다. 일반적인 경우의 통신 프로토콜은 HTTP(Hypertext Transfer Protocol)가 사용한다. HTTP는 클라이언트와 서버 사이에 이루어지는 응답 및 요청 프로토콜로 확장성이 높다는 장점이 있다. 그러나 배터리 소모가 심하고 비교적 무거워 라즈베리 파이에서 사용하기도 어렵다는 문제가 있다.

이러한 문제점을 해결하기 위해 HTTP가 가진 확장성을 줄이지만, 라즈베리 파이에서 구동하기 쉽게 가볍고, 배터리 소모가 적은 MQTT(Message Queuing Telemetry Transport) [4] 프로토콜을 사용한다. MQTT 프로토콜을 이용해 본 시스템의 효율을 높이면서, 동시에 화재 대응을 위한 신속한 통신의 기반을 구현한다.

2.3 실시간 스트리밍(Realtime Streaming)

불꽃이 실제로 인식되면 알람이 가는 것뿐만 아니라, 실시간으로 사용자가 상황을 판단하여 조치할 수 있도록 화재 상황을 보여줄 수 있어야 한다. 본 연구에서는 라즈베리 파이의 카메라를 통해 촬영하고 있는 영상을 Octoprint [5] 의 웹캠(Webcam) 기능을 활용하여 사용자에게 제공한다.

3. 프로젝트

본 논문이 목표로 하는 시스템은 그림 1과 같다. 라즈베리 파이의 카메라를 통해 입력을 받고 객체 인식 모델을 통해 불꽃을 인식한다. 이후 OctoPrint를 통해 사용자에게 실시간 스트리밍을 제공하며 화재가 인식된 경우에 어플리케이션으로 경고한다.

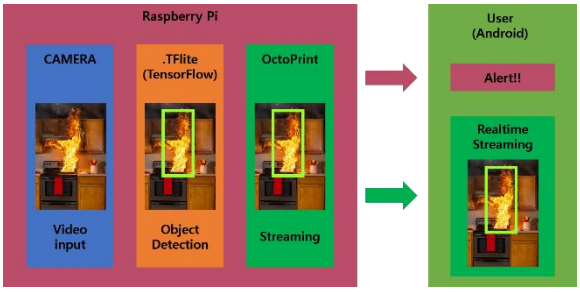

그림 1 해당 시스템의 구성

3.1 객체 인식(Object Detection) 모델의 구현

딥러닝 모델의 구현은 그림 2의 과정을 따른다.

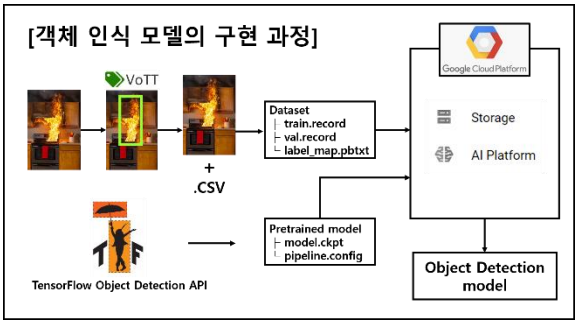

그림 2 모델의 구현 과정

먼저 데이터 셋을 구성하기 위한 화재 이미지를 웹 또는 Kaggle을 통해 수집한다. 수집된 이미지를 VoTT(Visual Object Tagging Tool)를 이용하여 객체의 위치에 bounding box를 그리는 annotation 작업을 수행한다. 이를 통해 각 좌표가 명시된 csv(comma separated value) 파일을 만들어낸다. 여기서 이미지와 label을 묶은 TFRecord를 만든다. 또 label의 정보가 담겨 있는 label map을 pbtxt로 표현하는 것으로 데이터 셋을 완성한다.

다음으로 전이학습을 위해 TensorFlow Object Detection API에서 사전 학습된 객체 인식 모델을 선택한다. 선택한 모델에 대해 학습을 위한 하이퍼 파라미터(Hyper Parameter)를 담고 있는 config 파일을 수정한다. 이후 Google CLI를 이용하여 데이터 셋과 모델의 checkpoint 파일, config 파일을 Google Storage에 업로드한다. 마찬가지로 Google AI Platform를 통해 TPU를 통해 빠르게 학습하고 GPU로 평가한다.

그림 2의 과정에 따라 300x300 이미지를 입력으로 하는 MobileNet [5] 기반의 SSD [6] 모델을 학습했다. 데이터 셋은 300장의 훈련집합과 65장의 검증집합으로 이루어졌다. 전체 학습 과정인 50,000 steps에서 32,700 step일 때, mAP@50IoU가 64%로 가장 높은 성능을 보였다.

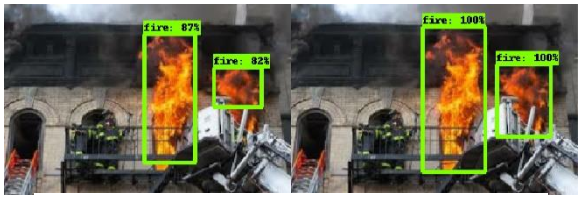

그림 3 학습된 모델의 추론(좌), 실제(우)

3.2 MQTT(Message Queuing Telemetry Transport)

모델을 통해 화재가 인지되면 MQTT를 통해 메시지를 보낸다. MQTT의 broker는 Mosquitto를 이용한다. 이는 QoS(Quality of Service)의 모든 레벨을 지원하기에 품질을 조금 낮추더라도 더 빠른 통신을 가능하게 해준다. MQTT를 통해 보낸 메시지를 받으면, 어플리케이션을 통해 화재 발생 알림을 보낸다.

3.3 OctoPrint

Octoprint는 3D 프린터를 위한 웹 인터페이스(web interface)이다. 3D 프린터는 수행 시간이 비교적 길고 높은 수준의 출력을 요구하기 때문에, 3D 프린터를 언제 어디서든 제어하고 출력물을 지속적으로 관찰할 필요가 있다. 바로 Octoprint가 이를 수행하기 위해 만들어진 오픈 소스(Open source)이다.

본 연구에서는 Octoprint의 웹캠 기능을 활용한다. Octopi를 라즈베리 파이에 삽입될 SD카드에 설치한 후, SSH을 통한 원격 제어를 위해 Octopi의 와이파이(Wi-fi)를 설정한다. 이후 Octoprint 서버에서 카메라로 입력되는 영상을 확인할 수 있다.

3.4 사용자 인터페이스(User Interface) 구현

불꽃이 검출되면 사용자는 화재가 인식되었다는 알림을 받아야 하고, 실제 상황을 실시간 영상으로 제공받을 수 있어야 한다. 본 연구에서는 이를 어플리케이션의 제작을 통해 제공한다.

그림 4 User Interface 설계

본 연구에서 목표로 하는 UI의 설계는 그림 4와 같다. 첫 번째 화면은 기본 메인 화면으로 '실시간 영상 확인' 버튼을 제공한다. 버튼을 누르면 실제 카메라가 있는 라즈베리 파이의 Octoprint 서버로 연결되어, 사용자가 원하면 실시간 스트리밍 영상을 확인할 수 있게 한다. 두 번째 화면은 화재가 실제로 인식되면 팝업(Pop-up)될 알림 메시지 창을 보여준다. 메시지 창에서도 바로 실시간으로 영상을 확인할 수 있도록 버튼을 제공한다. 마지막 화면은 Octoprint 서버에 접속되어 실시간 영상이 제공되고 있음을 보여준다.

4. 결론 및 향후 연구

본 연구는 화재 대응을 위해 불꽃을 인식하고 대응을 돕는 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.

현재는 데이터 셋의 양이 많지 않아 모델의 성능이 떨어진다. 따라서 향후 화재 이미지를 더 수집할 계획이다. 또 Octoprint의 실시간 영상을 사용자의 모바일 디바이스에서 어떻게 제공받을지 추가적인 연구가 필요하다.

참 고 문 헌

[1] Mingxing Tan, Quoc V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", *ICML 2019*, May 2019.

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

[3] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, Chunfang Liu, "A Survey on Deep Transfer Learning", *ICANN 2018*, Aug 2018.

[4] 김동휘, "MQTT 프로토콜 기반 IoT 환경 및 솔루션의 효율성 향상을 위한 연구", 한밭대학교 정보통신전문대학원 석사 학위 논문, Aug 2017.

[5] Octoprint, https://octoprint.org/#full-remote-control-and-monitoring

[6] 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.

[7] 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.