공 학 석 사 학 위 논 문

엣지 컴퓨팅과 비전 시스템을 이용한 포트홀 검출 및 포트홀 크기 측정에 대한 연구

A Study on The Detection of Pothole and Pothole Size Measurement Using Edge-Computing and Vision System

울 산 대 학 교 대 학 원

전 기 전 자 컴 퓨 터 공 학 과

김 민 혁

엣지 컴퓨팅과 비전 시스템을 이용한 포트홀 크기 검출 및 포트홀 크기 측정에 대한 연구

지 도 교 수　김 한 실

이 논문을 공학석사 학위논문으로 제출함.

2022年 7月

울 산 대 학 교 대 학 원

전 기 전 자 컴 퓨 터 공 학 과

김　민　혁

김민혁의 공학석사학위 논문을 인준함

심사위원장 김 현 철 (인)

심사위원 서 영 수 (인)

심사위원 김 한 실 (인)

울 산 대 학 교 대 학 원

2022年 7月

감사의 글

4학년 1학기 캡스톤2를 듣는 중에 지금은 연구실 선배이지만 그 당시 조교였던 선배의 권유로 연구실에 들어와 다양한 경험을 하며 생각에서 그치던 것들을 실현시키며 개발자의 길로 나아갈 수 있도록 이끌어주신 모든 분께 감사의 인사 드리고자 합니다.

먼저 대학원 석사과정을 한다고 했을 때 믿고 응원해주신 부모님과 형 동생에게 감사합니다. 회사 생활과 대학원 과정을 같이 진행하며 힘이 들 때에 잘 해낼 수 있다고 해주신 가족의 믿음에 꼭 보답하는 자식, 형, 동생이 되도록 하겠습니다.

석사 2년차에 회사 생활로 인해 중간에 연구실 과제에서 나가게 되었지만 끝까지 잘 챙겨주시고 많은 것을 알려주신 김한실 교수님께 감사함을 전하고 싶습니다. 전공 지식 이외에도 업체 미팅을 통해 아이디어가 시제품이 되기까지의 과정 및 다양한 경험을 쌓는 기회를 주셔 정말 감사합니다. 자랑이 될 수 있는 제자가 되도록 노력하겠습니다. 또한 본 논문 심사를 맡아주신 김현철 교수님, 서영수 교수님께 감사의 말씀 전합니다.

그리고 늘 편하게 대해주시며 제가 가고자 하는 방향을 잡아주신 시경이 형, 태우 형, 수현이 형, 수진이 누나 그리고 진희, 진기, 민식이, 채원이, 선주, 마지막으로 덤벙대던 저를 잘 챙겨주던 동기 관후형까지 연구실에서 많은 추억을 쌓게 해준 자동제어 연구실 모든 분들께 진심으로 감사합니다. 마지막으로 석사과정을 보내며 옆에서 응원해주었던 고등학교 친구들과 텔레파시 학생회 분들께도 감사의 인사 전합니다.

일련의 과정을 통해 제가 석사과정을 졸업할 수 있었던 것 같습니다. 연구실에서 배우고 느꼈던 것들로 사회에 나가 도움이 되는 개발자가 될 수 있도록 노력하겠습니다.

2022 년 7월

김 민 혁

[국 문 요 약]

엣지 컴퓨팅과 비전 시스템을 이용한 포트홀 검출 및 포트홀 크기 측정에 대한 연구

A Study on The Detection of Pothole and Pothole Size Measurement Using Edge-Computing and Vision System

울산대학교 대학원

전기공학부

김민혁

엣지 컴퓨팅은 IoT 기기의 발달로 많은 양의 데이터를 처리하면서 클라우드 컴퓨팅의 지연 발생과 네트워크 불안전시 차단 가능성의 문제를 해결하기 위해 나왔으며 실시간 데이터 처리를 위해 GPU가 부착된 엣지 컴퓨팅의 기술로 엣지에서 딥러닝이 가능하게 되었다. 본 논문에서는 엣지 컴퓨팅과 비전 시스템을 이용해 포트홀 검출 및 크기를 알아 내고자 한다. YOLO 알고리즘을 이용하여 실시간으로 포트홀을 검출할 때 연산량을 줄이기 위한 이미지 이진화와 ROI 영역 설정을 하기로 한다. 검출된 포트홀의 크기를 알아내기 위해 차와 포트홀 사이의 길이, 위치좌표, 픽셀 길이의 수식을 통해 포트홀의 크기를 알아내고자 한다. 여러 차례의 실험의 결과와 실제 결과값과의 비교를 통해 시스템의 신뢰성을 검증하였다.

[영 문 요 약]

A Study on The Detection of Pothole and Pothole Size Measurement Using Edge-Computing and Vision System

Min-Hyuk Kim
School of Electrical Engineering
The Graduate School,
University of Ulsan
Supervised by Prof. Han-Sil Kim

ABSTRACT

With the development of IoT devices, large amounts of data must be processed. Cloud computing is used to solve this problem, but there is a disadvantage in that data is not transmitted due to latency and network instability. To solve this problem, edge computing has been developed, and this edge computing has a GPU attached for real-time data processing, enabling deep learning on the edge.

In this paper, we use edge computing and vision systems to find out the porthole detection and size. When detecting a porthole in real time using the YOLO algorithm, image binarization and ROI region setting are decided to reduce the computation volume.

In order to find out the size of the detected porthole, we want to find out the size of the porthole through the equation of the length, location coordinates, and pixel length between the car and the porthole. The reliability of the system was verified by comparing the results of several experiments with the actual results

[국문요약]
[영문요약]
목 차
그림 목차
표 목차

그림 목차

표 목차

1. 서론

1.1 연구 배경 및 목적

그림 1. 도로위 발생된 포트홀

포트홀은 도로가 내려앉거나 부서져 생긴 원형으로 생긴 홈을 말한다. 도로의 노후, 시공불량, 기상현상, 화물차량의 지속적인 충격 등 다양한 이유가 있다. 이러한 이유로 벌어진 틈에 지속적으로 충격이 가해지거나 틈 사이로 빗물이 들어가 골재 분리가 발생하기 쉬운 상태가 되어 충격이 계속 누적되면 포트홀의 진행 속도가 빨라진다. 이렇게 생긴 포트홀은 차량 운전자가 인지하지 못한 상황에서 고속 주행 시 대형 교통사고로 이어질 수 있다. 이와 같은 사고를 방지하기 위해 포트홀을 신속히 파악하여 보수를 해야만 한다. 하지만 넓은 도로에서 발생한 다양한 포트홀을 모두 파악하는데 힘이 들며 한정된 인력으로 도로 곳곳을 돌아다니며 발견해 내기에는 시간적으로나 물리적으로 많이 부족한 상황이다.

이러한 이유로 연간 포트홀에 인한 사고가 지속적으로 발생되고 있다. 매년 3,000건 이상 발생하는 포트홀은 고속으로 달리는 고속도로에서는 대형사고로 이어질 수 있어 미리 대비하여 피해를 최소화하는 것이 중요하다. 한국도로공사에서 제공받은 자료를 보면 포트홀의 발생 건수는 줄었지만 피해보상액은 늘어나고 있는 추세다.

최근 5년간 연도별 포트홀 발생 현황

연도	2015년	2016년	2017년	2018년	2019년	2020. 6월
합 계	17,575	14,179	7,189	4,553	3,717	1,921

최근 5년간 포트홀에 의한 피해보상 현황

구 분	2015년	2016년	2017년	2018년	2019년
보상건수(건)	199	160	323	877	707
보상금액 (백만원)	153	141	215	595	646

그림 2. 한국도로공사 제공 포트홀 발생 및 피해보상 현황

예전에는 대부분이 시민들의 신고 혹은 현장 순찰 차에 의존하는 방식이라 신속하게 대응하기 어려웠으며 도로 위의 다양하게 생성된 포트홀의 크기에 따라 사고의 위험성이 올라가기 때문에 포토홀로 발생하는 피해를 줄이기 위해 안전한 도로 환경을 구축하려면 포트홀을 빨리 찾아 알리는 것이 중요하며 포트홀의 크기를 알아내어 보수공사를 진행할 때 우선순위를 두는 것 또한 중요하다. 이런 많은 이유들로 인해 최근 레이저를 이용하거나 적외선을 이용하거나 카메라를 이용하여 포트홀을 검출하고 있다. 하지만 제한된 센싱 영역의 문제, 높은 구축 비용 등 실용화에 활용하기 많이 어려움이 존재한다[5].

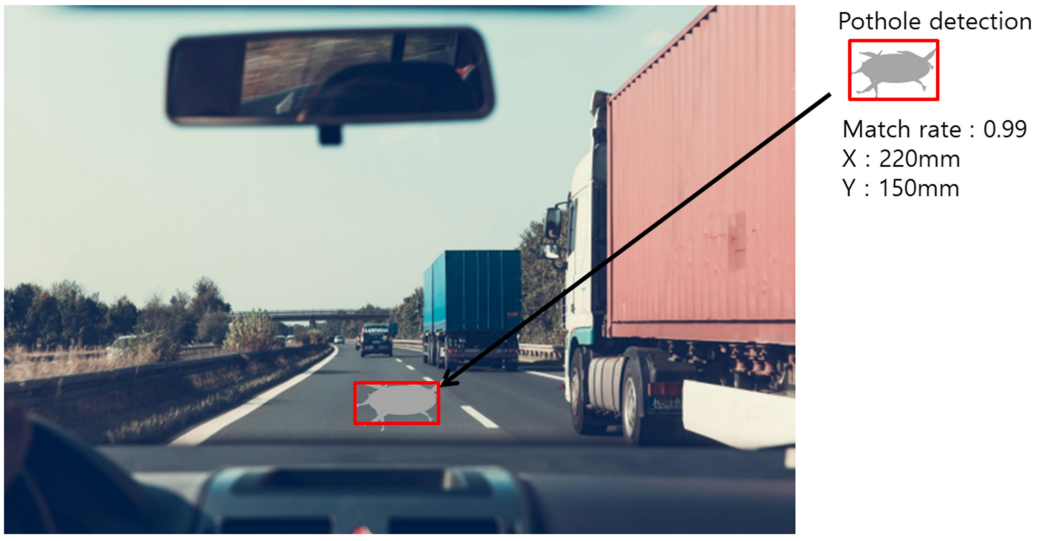

그림 3. 포트홀 실시간 검출 사진

본 연구는 이러한 문제를 해결하고자 실시간으로 포트홀을 검출하기 위해 비전 시스템과 엣지 컴퓨팅을 이용하여 기기와 서버 간의 연결이 아닌 기기와 엣지단과 연결하여 포트홀을 검출해내며 포트홀과 차의 거리를 측정 및 픽셀의 길이를 구하는 수식을 통해 포트홀의 크기를 알아내고자 한다.

1.2 선행 연구에 대한 고찰

포트홀을 검출하기 위해 다양한 방법을 사용하여 도로 위의 위험을 줄이는 연구를 진행하였다.

1.2.1 진동 기반 포트홀 검출 시스템

진동 기반 포트홀 검출 시스템은 Central Server에서 Vehicle Clients로부터 받는 진동 데이터를 비교하여 포트 홀인지 일반 도로인지 판단을 내리는 시스템이다.

Central Server에 포트홀 데이터를 저장하기 위해 3축 가속도 센서와 GPS를 가지고 있는 임베디드 시스템, 예를 들어 핸드폰을 가지고 달리는 차 안에서 도 심, 비포장도로, 철길, 맨홀 뚜껑, 포트홀 등 다양한 도로를 달리면서 도로의 상 태에 따른 가속도 센서의 진동 데이터를 Server에 저장한다. 이때 달리는 자동 차는 택시, 일반 승용차, 트럭, 쓰레기 수거 트럭 등등 다양한 차량 데이터를 수집한다. 이때 임베디드 시스템(핸드폰)을 차 내부에 설치하느냐 사람이 품속에 간직하느냐 차량의 Dashboard에 장착하느냐에 따라 포트홀 검출 기준이 달라 질 수 있으며 기준이 늘어날수록 서버 용량도 커지기 때문에 비용 상의 단점이 발생한다.

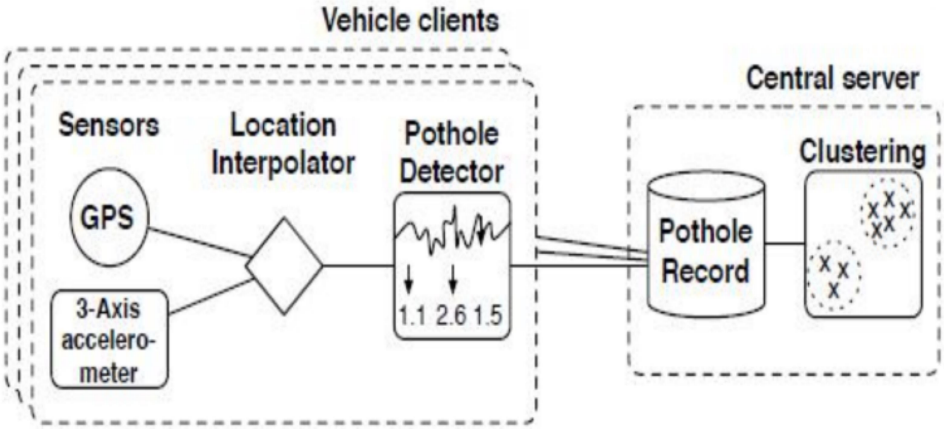

그림 4. 진동 기반 포트홀 검출 시스템

3축 가속도 센서가 장착된 임베디드 시스템을 가지고 일반도로에서 탔을 경우 에 비해 포트홀을 지날 때 확연히 진동 축이 흔들리는 것을 확인할 수 있다. 포 트홀을 지날 때 진동 데이터를 수없이 많이 서버에 저장해 놓고 Vehicle Client에서 보내는 데이터와 비교해서 포트홀의 여부를 판단할 수 있다.

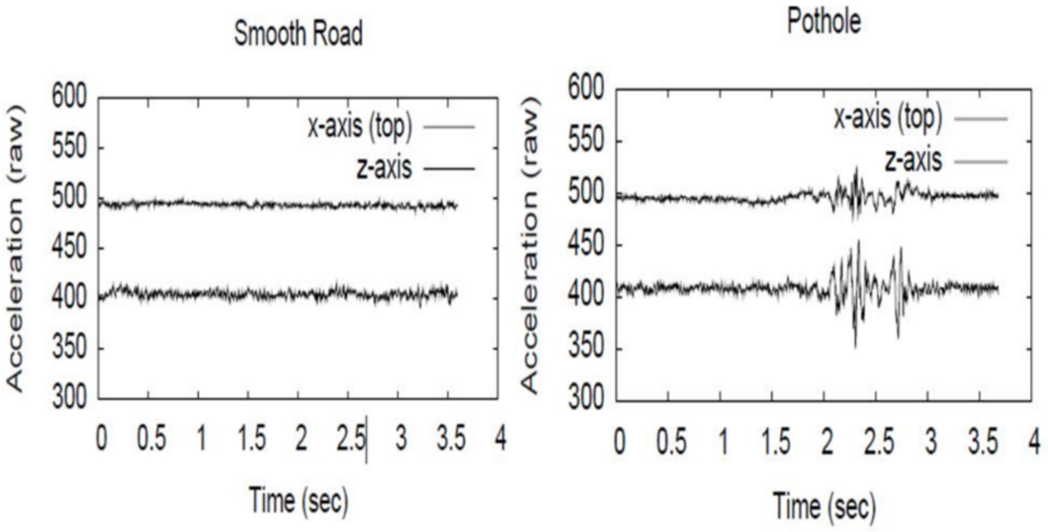

그림 5. 일반도로(좌) 포트홀(우)의 진동 그래프

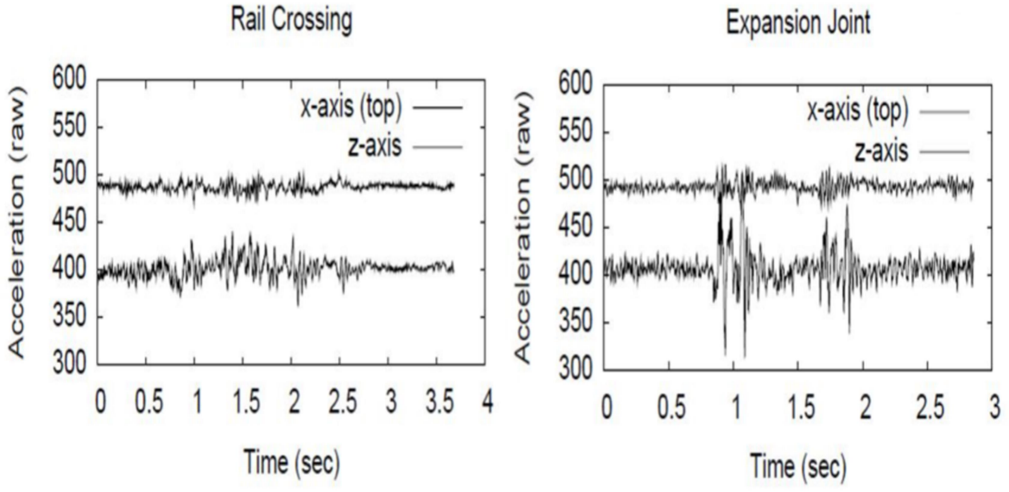

그림 6. 철길 건널목(좌) 신축 이음(우)의 진동 그래프

위에서 본 결과 포트홀과 일반도로 이외의 도로 진동 그래프이다. 철길 건널목과 신축 이음 등을 지날 때 역시 진동 그래프의 변화가 일반도로와는 확연히 차이가 나는 것을 볼 수 있다.

1.2.2 가속도 센서 기반 포트홀 검출 기법

가속도 센서는 3축의 가속도 변화량 데이터를 제공하며, 특히 상하 흔들림의 크기를 나타내는 Z축 가속도 크기 변화는 포트홀 검출 특성상 활용하기 좋은 특징이다. 차량이 이동할 때의 상하 흔들림을 통해 얻는 데이터를 분석하여 진폭, 평균 속도 변화 등의 임계값을 결정하고 이를 초과하는 경우 포트홀로 판단한다. Z축 흔들림 크기, Z축 임계값, 특정 구간 내의 표준 편차를 이용한 포트홀 검출 방법이 발표된 바 있다.

가속도 센서의 장점은 포트홀 검출에서 가격대비 검출 성능이 매우 높다는 점 이다. 일반 단말기에도 내장될 만큼 보편적이고, 단가가 낮으며 처리할 데이터량도 적기 때문에 다른 센서와 함께 활용하는데 부담이 적다[1].

1.2.3 LiDAR를 이용한 3차원 포트홀 검출 시스템

2개의 2D LiDAR 센서를 이용하여 3차원의 포트홀을 검출하기 위한 시스템이다. 다른 시스템에 비해 저렴하고, 전파방향의 영향이 적고, 포트홀의 깊이, 폭 등 포트홀의 구체적인 스펙을 비교적 정확하게 알아낼 수 있는 장점이 있다.

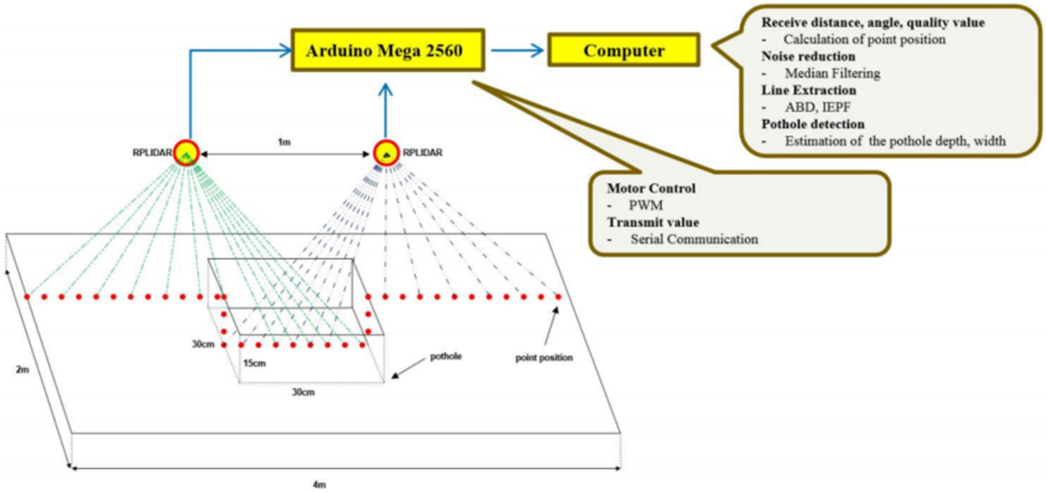

그림 7. 포트홀 검출 시스템 구성도

구성하려고 하는 포트홀 검출 시스템의 블록도는 다음 그림 8과 같다. 기 본적으로 LiDAR는 PWM방식을 이용해 구동된다. LiDAR를 구동시키는데 쓰이는 MCU는 Arduino를 사용했다. LiDAR는 물체와의 거리, 각도, 정확도 등의 정 보를 시리얼통신으로 아두이노로 보낸다. 아두이노는 이 정보를 받아 신뢰할 수 없는 정보 즉 음수 값을 가지는 거리 정보, 음수 값을 가지는 각도 정보, 정확도가 낮아 신뢰할 수 없는 정보를 필터를 통해 걸러낸다. 필터를 거친 정보는 시리 얼 통신으로 PC로 보내지게 되고 MATLAB을 이용해 거리나 각도 정보를 사용 자 필요에 의해 가공하게 된다.

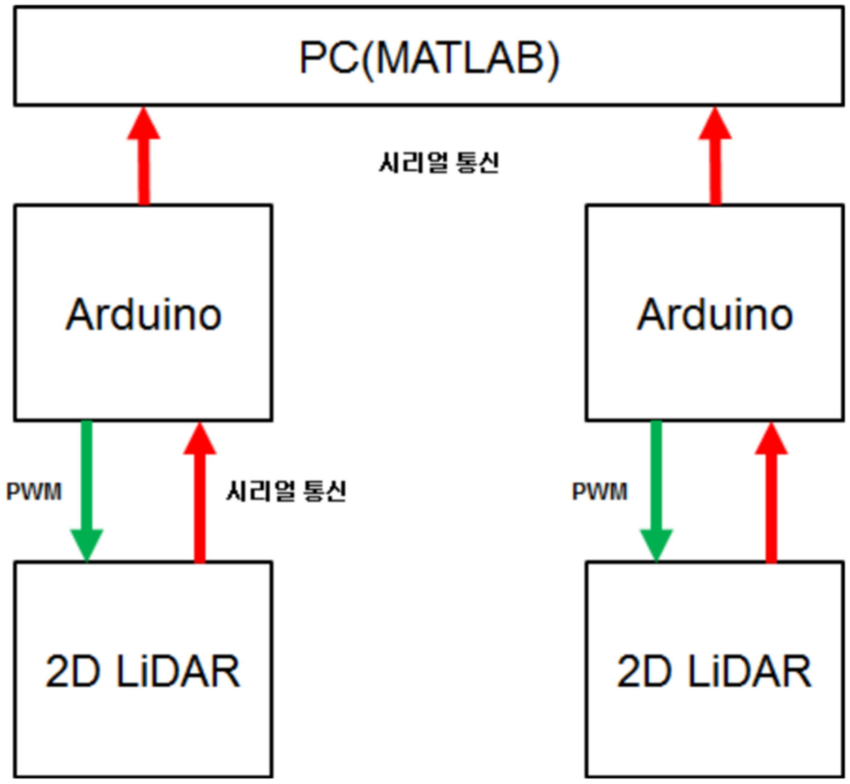

그림 8. 시스템의 블록도

2D LiDAR를 이동체에 달아 이동하며 모은 2차원 데이터를 속도 보정 값을 더해 3차원 데이터로 구축하여 포트홀 검출 및 크기 깊이까지 알아 낼 수 있다[2].

1.3 선행 기술의 문제점 및 현 연구에서의 차별성

위의 기술로도 포트홀을 검출해낼 수 있지만 각각의 시스템에는 단점이 있다. 먼저

진동센서로 포트홀을 검출하고자 할 때 다양한 포트홀의 모양과 깊이로 인해 발생되는 진동 데이터들이 매우 많을 것이며 맨홀이나 다른 이유로 발생 되는 진동을 포트홀로 인식 할 수 있는 단점이 있다. 가속도 센서를 사용하여 포트홀을 검출하고자 할 때 진동센서와의 같은 문제로 상하의 진동이 발생하였을 때 포트홀 외의 원인으로 발생한 진동인지 구분하기 매우 까다롭다. 위와 같은 경우는 다양한 포트홀을 직접 지나다니며 데이터를 계속 축적해야 하며 검출된 데이터가 포트홀에 의해 나타난 것으로 판단하기 어려운 경우가 많다는 단점이 있다. LiDAR를 이용하여 포트홀을 검출 할 때는 설치 각도에 따라 들어오는 데이터의 값이 달라져 고정된 각도에서의 데이터 값을 계속 쌓아야 하며 포트홀 검출 할 때 낮은 속도에서의 제한상황이 있어 차를 이용하여 실시간으로 포트홀을 검출해 내기에는 여러가지 제약 조건들이 있었다.

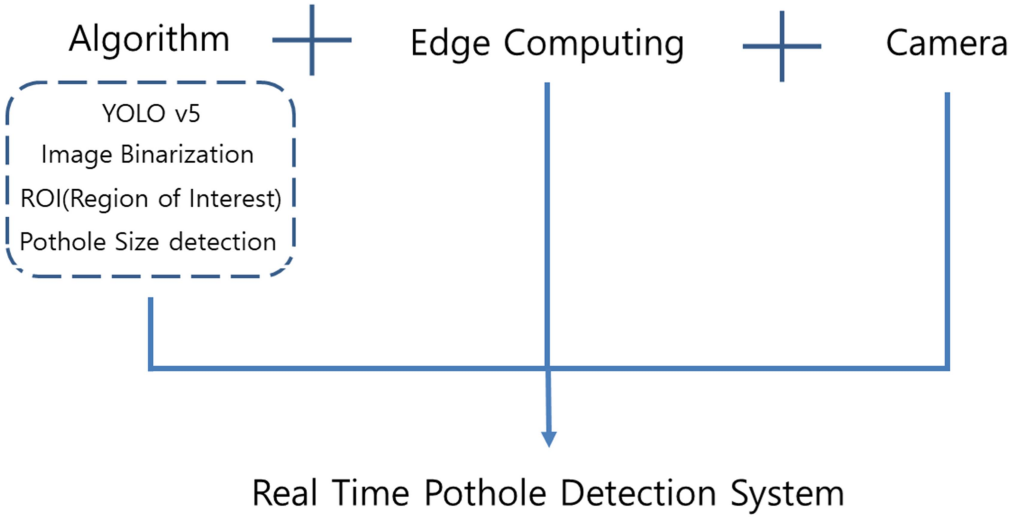

그림 9. 실시간 포트홀 검출 시스템 블록다이어그램

본 논문에서는 포트홀을 검출을 가시적으로 확인할 수 있도록 비전 시스템과 네트워크와 연결하여 클라우드와 서버를 이용하지 않고 엣지 컴퓨팅을 이용하여 어떠한 환경에서도 카메라에서 들어오는 영상 데이터를 연산하여 포트홀을 검출하고자 한다. 검출한 포트홀의 크기를 측정하여 크기에 따라 보수공사의 우선순위를 두어 큰 피해를 막고자 한다. 사용할 엣지 컴퓨터는 NVIDIA에서 출시한 Jetson Xavier를 사용한다. 포트홀 검출에 사용될 알고리즘으로는 실시간 물체 검출에 강한 YOLO를 사용하며 포트홀과 차의 거리를 계산하여 수식을 통해 얻어진 픽셀 하나의 길이를 통해 포트홀의 크기를 측정하는 과정을 진행하고자 한다.

2. 본론

2.1 엣지 컴퓨팅

엣지 컴퓨팅의 기술은 최근 급속히 확산되는 IoT 기기로 인해 기기-서버 간 데이터 통신량이 폭증하면서 클라우드 서버에서의 지연율 발생과 과부하의 위험과 클라우드 서버로 모든 데이터를 전송함에 따른 중요 정보에 대한 침해성 문제도 존재하며 일시적 네트워크 중단 등 기술적 한계를 해결하기 위해 나온 기술이다. 이러한 문제들을 해결하기 위해 데이터가 발생한 현장 또는 근거리에서 실시간으로 데이터를 연산할 수 있도록 엣지 컴퓨팅이라는 새로운 데이터 처리 패러다임이 나타나고 있다. 최근 인공지능(AI) 기술의 발전과 함께 아마존, 마이크로소프트, 구글 등 전 세계 기업들이 엣지 컴퓨팅기술에 대한 연구 개발을 진행하고 있다. 이로 인해 엣지 컴퓨팅의 기술 시장 또한 빠르게 발전하고 있다.

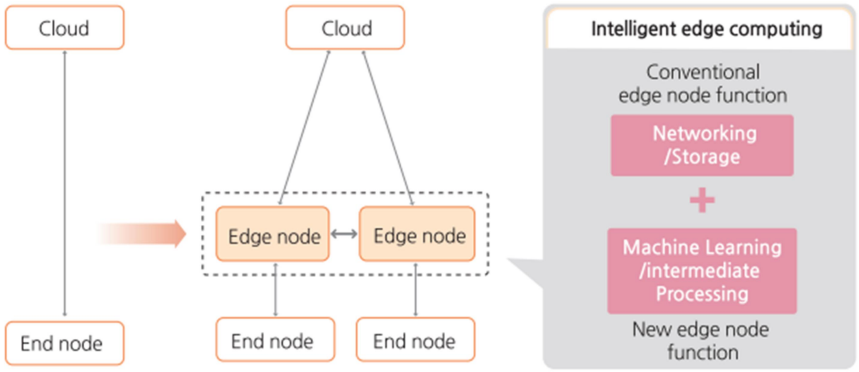

그림 10. 지능형 엣지 컴퓨팅 개념도

엣지 컴퓨팅은 기기와 가까운 네트워크의 가장자리에서 지원하는 컴퓨팅을 의미한다. 클라우드 컴퓨팅이 중앙 집중형으로 물리적으로 떨어져 있는 곳에서 데이터를 처리해주는 것과는 반대로 기기의 주변에서 개별로 데이터를 수집, 분석, 처리를 통해 활용할 수 있는 기술이다. 위의 그림에서 보듯, 지금까지는 엣지에 위치한 장비들이 단순히 데이터 수집 및 전송의 역할만을 수행하였다면 지능형 엣지 컴퓨팅은 에지 장비에서 인공지능 기능을 부여한 데이터 분석을 할 수 있도록 만들어서 클라우드 컴퓨팅의 단점을 보완할 수 있는 차세대 컴퓨팅 기술로 부각되고 있다. 클라우드 컴퓨팅의 경우 수많은 스마트 기기 또는 지능형 기기가 사용됨에

따라 기기에서 발생하는 모든 데이터를 중앙에 집중해서 수집, 처리, 분석하는 것이 현실적이지 않다. 그래서 데이터가 생성된 위치에서 가까운 곳에서 처리할 수 있는 방안이 필요하다. 즉, 네트워크 에지 노드에서 컴퓨팅을 함으로써 실시간에 가깝게 중요한 데이터를 처리하는 방안이 일반적으로 엣지 컴퓨팅이다[3].

2.1.1 엣지 컴퓨팅의 장점

엣지 컴퓨팅의 주요 장점은 네트워크 지연 시간을 극복하는 것이다. IoT 애플리케이션에서 1초 미 만의 응답 시간이 필요한 경우 클라우드에 대한 요청을 기다리는 것은 문제가 될 수 있다. 예를 들어, 안전 제어 시스템에서 사람이 기계에 너무 가까이 있음을 감지하면 그 즉시 기계를 중지시켜야 한다. 이때, 응답 시간이 지연되면 심각한 인명 피해나 기계 손상이 발생할 수 있으므로 센서에 의한 사람 인식 처리와 기계 정지 결정 처리는 네트워크 통신 에 의해 지연되어서는 안 된다. 마찬가지로 자율주행 차량이나 증강현실 애플리케이션은 20ms 미만 의 응답 시간이 필요한데, 클라우드와의 통신으로는 제공할 수 없어서 센서 데이터 처리를 엣지 게이트웨이로 이동함으로써 네트워크 지연을 방지 하고 원하는 응답 시간을 달성할 수 있다.

비용 절감 또한 엣지 컴퓨팅의 주요 장점이다. 센서 및 엑추에이터에 의해 생성되는 대부분의 원격 측정 데이터는 IoT 애플리케이션과 관련이 없어서 엣지 컴퓨팅을 사용하면 데이터를 클라우드로 보내기 전에 필터링하고 처리할 수 있다. 이것은 데이터 전송에 따른 네트워크 비용을 줄이고, 애플리케이션과 관련이 없는 데이터에 대한 클라우드 스토리지 및 처리 비용을 줄이는 것이다.

기업이 성장함에 따라 IT 인프라 요구사항을 항 상 예측할 수는 없으며 전용 데이터 센터를 구축하거나 확장하는 것은 비용이 많이 들기 때문에 컴퓨팅, 스토리지 및 분석 기능이 최종 사용자에게 더 가까이 배치될 수 있는 엣지 컴퓨팅 장치를 사용하면 그 범위와 기능을 빠르고 비용 효율적으로 확장 할 수 있다. 또한, 새로운 장치가 추가될 때마다 네트워크 코어에 상당한 대역폭을 요구하지 않으므로 확장 비용이 줄어들 수 있다. 엣지 컴퓨팅 장치와 엣지 데이터 센터가 최종 사용자에게 더 가깝게 배치되면 먼 위치의 네트워크 문제가 영향을 미칠 가능성이 작다. 즉, 데이터 센터가 중단되는 경우에도 엣지 컴퓨팅 장치는 중요한 처리기능을 기본적으로 수행하기 때문에 자체적으로 계속해서 효과적으로 작동할 수 있다. 또한, 네트워크에 연결된 엣지 컴퓨팅 장치와 엣지 데이터 센터가 많아서 사용자가 필요한 제품 및 정보에 여러 경로를 통해 액세스할 수 있기 때문에, 한 번의 실패로 서비스를 완전히

중단시키기가 어렵다. 따라서 기업은 고객에게 더 빠르고 원활한 서비스를 보장할 수 있다.

엣지 컴퓨팅은 네트워크를 통해 이동해야 하는 데이터의 양을 줄이고, 데이터를 한 곳에 저장시키는 것이 아니라 분산시키기 때문에 보안 측면에서 분명히 장점이 있다. 또한 클라우드에 존재하는 정보는 쉽게 해킹되는 경향이 있지만, 엣지 컴퓨팅은 관련 정보만 클라우드로 전송하므로 이를 방지할 수 있다. 즉, 해커가 클라우드에 침투하더라도 사용자의 모든 정보가 위험에 처한 것은 아니다. 심지어, 때로는 엣지 컴퓨팅에 네트워크 연결이 전혀 필요하지 않다. 따라서 클라우드와 비교하면 엣지 컴퓨팅은 잠재적으로 보안에 대한 위험이 적다[4].

2.1.2 NVIDIA Jetson Xavier

NVIDIA Jetson은 엣지에서 AI를 위한 세계 최고의 플랫폼이다. 새롭고 놀라운 기능을 엣지에 제공하며 제품 개발 기회를 제공하고 있다. 단순한 솔루션 제공을 위한 Jetson Nano부터 최신 AI 모델을 최고급 사용 사례에 적용하는 데 이상적인 Jetson AGX Orin까지 다양한 제품군이 형성되어 있다. Jetson 모듈 및 개발 키트는 통합 소프트웨어 아키텍처에서 지원되어 개발자가 사용한 코드를 다른 Jetson 상·하위 모듈에 호환되어 내가 원하는 환경에 맞춰 하드웨어를 선택할 수 있어 확장성이 좋다. NVIDIA JetPack SDK에는 익숙한 Linux 환경, CUDA-X 가속 라이브러리, API, AI 엣지 애플리케이션 개발용 도구가 포함되어 있다. 또한 스트리밍 영상 분석용 DeepStream, 로보틱스용 NVIDIA Isaac 등 더 높은 수준의 SDK와 플랫폼을 지원한다. 먼저 Jetson Xavier NX에 대해 알아본다[8].

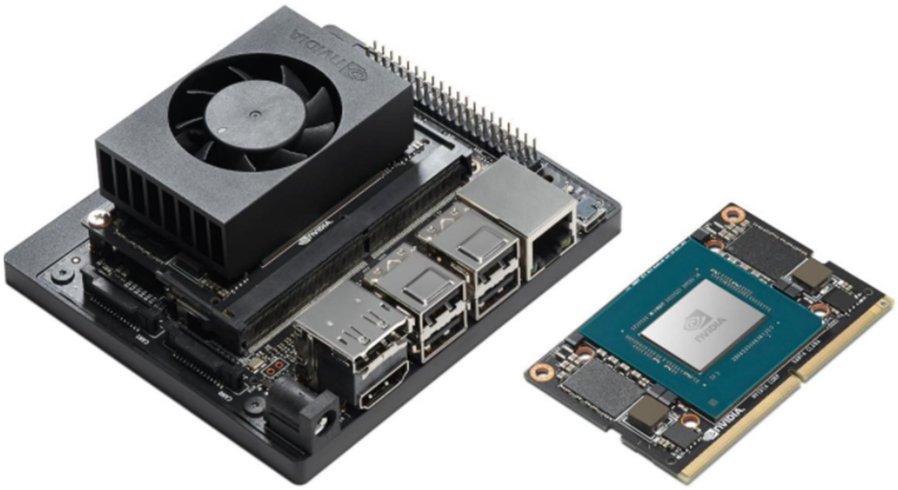

그림 11. NVIDIA NX Jetson Xavier

Jetson NX Xavier는 15W 미만의 전력으로 소형 폼 펙터에서 최대 21 TOPS 컴퓨팅을 제공하며 엣지 AI 장치 및 자율 기계에 서버 수준 성능과 클라우드 네이티브 워크플로를 제공한다. Jetson NX Xavier를 사용하면 놀라운 AI 기반 애플리케이션을 만들고 심층신경망(DNN)모델과 인기 있는 기계 학습 프레임워크를 현장에서 신속하게 배포할 수 있다. NVIDIA Jetpack 4.4 Developer Preview의 초기 소프트웨어 지원에는 CUDA Toolkit 10.2 및 cuDNN 8.0 TensorRT 7.1 및 DeepStream 5.0의 미리보기 릴리스와 함께 기계 학습 및 사전 훈련된 DNN 모델을 위한 새로운 Docker 컨테이너가 포함된다.

Jetson Xavier는 여러 복잡한 모델과 여러 고화질 센서 스트림을 병렬로 실행할 수 있는 NVIDIA의 획기적인 Xavier SoC를 기반으로 한다. 여기에는 48개의 Tensor 코어가 있는 통합 NVIDIA Deep Learning Accelerator 엔진, Seven-way VLIW 비전 가속기, 6 코어 NVIDIA Carmel 64비트 ARM v8.2 CPU, 8GB 128비트 LPDDR4x 같은 기능이 포함된다. 다음으로 Jetson AGX Xavier에 대해 알아보며 NX와의 성능 비교 또한 알아본다.

그림 12. NVIDIA Jetson AGX Xavier

Jetson AGX Xavier는 NX와 마찬가지로 엣지에 배포할 수 있는 컴퓨팅 밀도, 에너지 효율성 및 AI 추론 기능에 대한 새로운 기준을 설정하여 엔드 투 엔드 자율 기능을 찾춘 지능형 엣지 컴퓨팅을 말한다. 64개의 Tensor 코어, 8코어 NVIDIA Carmel ARM v8.2 64비트 CPU, 16GB 256비트 LPDDR4x, 듀얼 NVIDIA Deep Learning Accelerator를 포함한 통합 512 코어 NVIDIA Volta GPU로 구성 된다. Jetson AGX Xavier용 NVIDIA의 JetPack SDK4.1.1에는 CUDA 10.0, cuDNN 7.3 및 TensorRT 5.0이 포함된다. 이를 통해 로봇, 비전, 의료 기기에 결합하여 AI를 가속화 할 수 있다[9].

Jetson AGX Xavier는 고정 기능 CNN(Convolutional Neural Networks)의 추론을 오프로드하는 2개의 NVIDIA DLA(Deep Learning Accelerator) 엔진이 특징이다. 이러한 엔진은 에너지 효율성을 개선하고 GPU를 확보하여 사용자가 구현한 복잡한 네트워크와 동적 작업을 실행할 수 있다.

그림 13은 Jetson NX Xavier와 AGX를 비교하기 위해 비전 DNN을 NVIDIA NVDLA(Deep Learn Accelerator) 엔진과 GPU는 모두 INT8 정밀도로 동시에 실행한 결과이다.

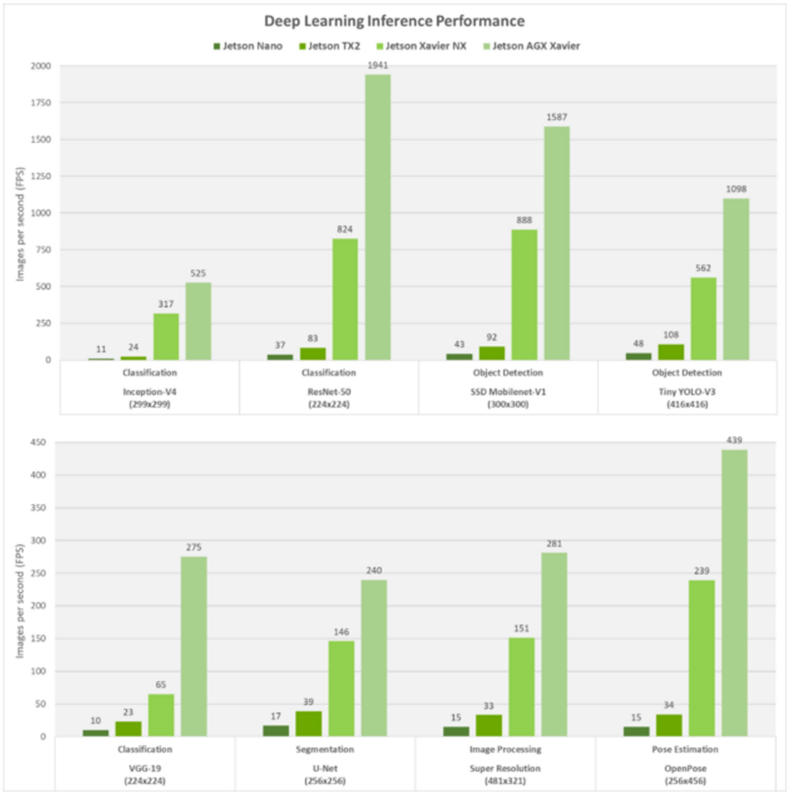

그림 13. Jetson NX, AGX Xavier TensorRT를 사용한 비전 기반 DNN 성능

결과를 보면 Jetson NX Xavier에 비해 Jetson AGX Xavier가 성능적으로 월등히 좋은 것을 알 수 있다. Classification과 Object Detection이 2배 가까이 성능 차이가 난다. 다양한 또한 BERT에 대한 결과 또한 확인해 볼 수 있는데 BERT는 QA, 의도분류, 감정 분석, 번역, 자동 완성 등 여러 NLP 작업 전반에 걸쳐 성공적인 애플리케이션이다.

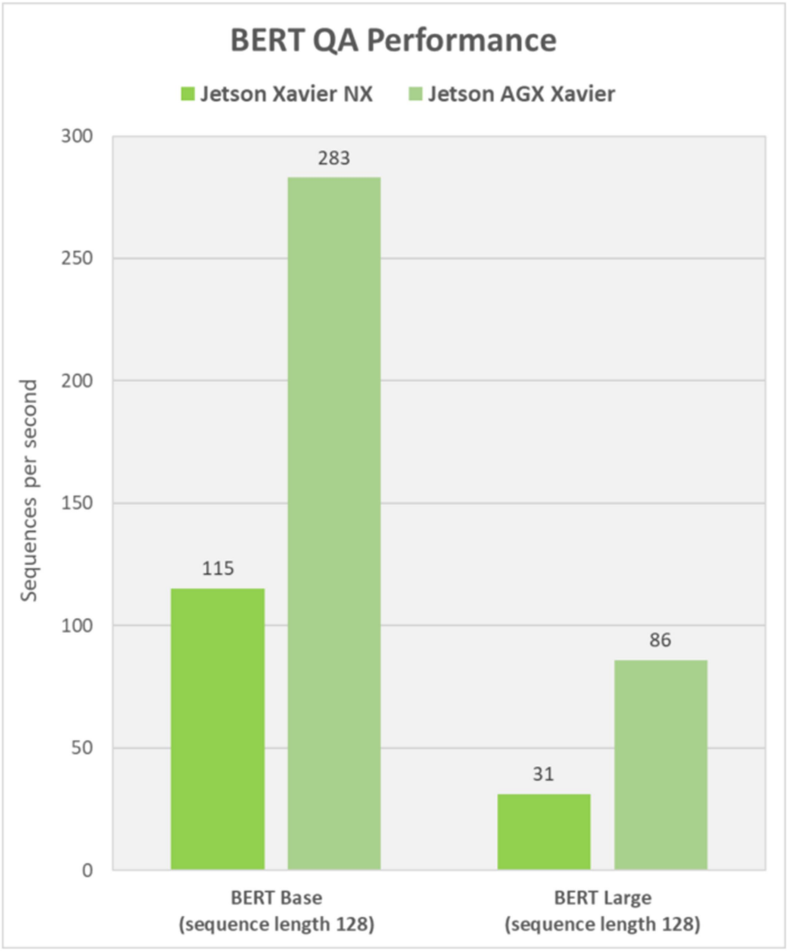

그림 14. BERT Large 런타임 성능

NLP 작업에 50ms 대기 시간 임계값을 사용하여 질문 응답 시 BERT Base 및 BERT Large의 런타임 성능을 보여주는데 이 작업의 결과 Jetson Xavier NX와 AGX이 BERT Base에서는 2배 이상의 성능 차이를 보여주며 BERT Large에서는 3배에 가까운 성능 차이를 볼 수 있다. 이러한 성능차이로 인해 본 논문에서는 Jetson Xavier AGX를 사용하여 테스트를 진행하고자 한다.

2.2 YOLO 알고리즘

앞서 나온 Real Time Object Detection의 포문을 연 알고리즘은 Region Proposal과 Classification을 나눠 Object Detection을 수행하는 2 Step 접근 방식이지만 YOLO는

한번에 Object Detection을 수행한다. 다음은 이러한 구조의 간단한 예시이다[6].

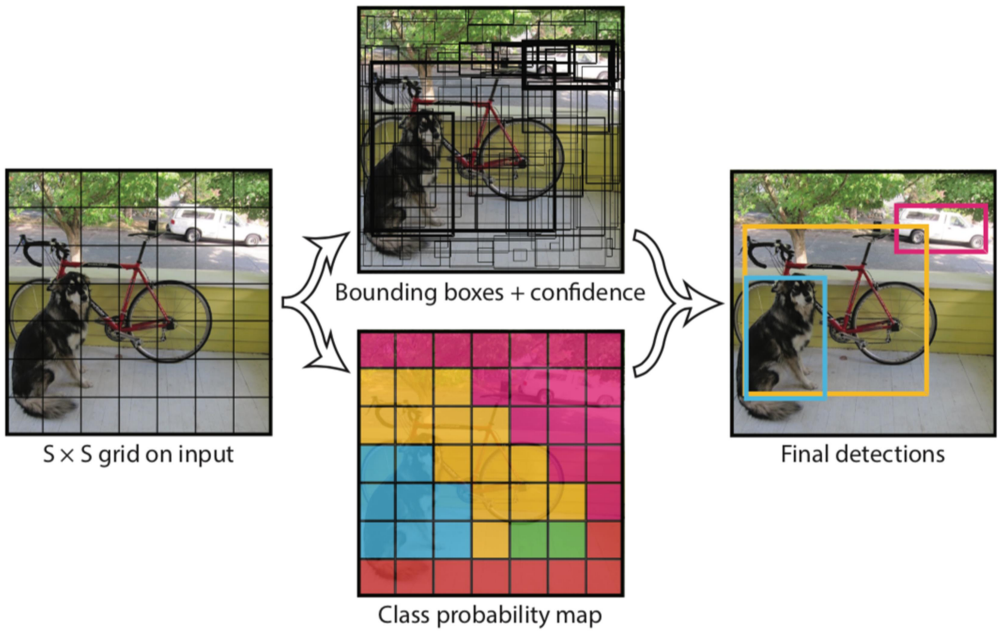

그림15. YOLO Unified Detection

먼저 입력 이미지를 S×S 그리드 영역으로 나눈다. 이 때, 실제 입력 이미지를 나누는 것이 아니다. 아래에 이 부분의 내용을 세부적으로 다루도록 한다. 이미지에서 각 그리드 영역에서 물체가 있을 만한 곳에 해당하는 N개의 Bounding Box를 예측한다. 이는 (x, y, w, h)로 나타내어 지는데 (x, y)는 Bounding Box의 중심점 좌표이며 w, h는 넓이와 높이이다. 이 다음과정으로는 해당 박스의 Confidence를 계산하여 신뢰도를 확인한다. 이는 해당 그리드에 물체가 있을 확률 Pr(Object)와 예측한 박스와 Ground Truth 박스와의 겹치는 영역을 비율을 나타내는 IoU를 곱해서 계산한다.

$$\Pr(Object) * IOU_{pred}^{truth} \quad (1)$$

그 다음, 각각의 그리드마다 C개의 클래스에 대하여 해당 클래스일 확률을 계산하며 수식은 아래와 같다. 이 때, 특이한 점은 기존의 Object Detection에서는 항상 클래스 수 +1 (배경)을 집어넣어 분류 하는데, YOLO는 그렇지 않다는 것이다.

$$\Pr(Class_i|Object) \quad (2)$$

이렇게 YOLO는 입력 이미지를 그리드로 나누고, 각 그리드 별로 바운딩 박스와 분류를 동시에 수행한다. 구체적인 네트워크 디자인은 다음과 같으며 이를 통해 더 자세한 분석을 할 수 있다.

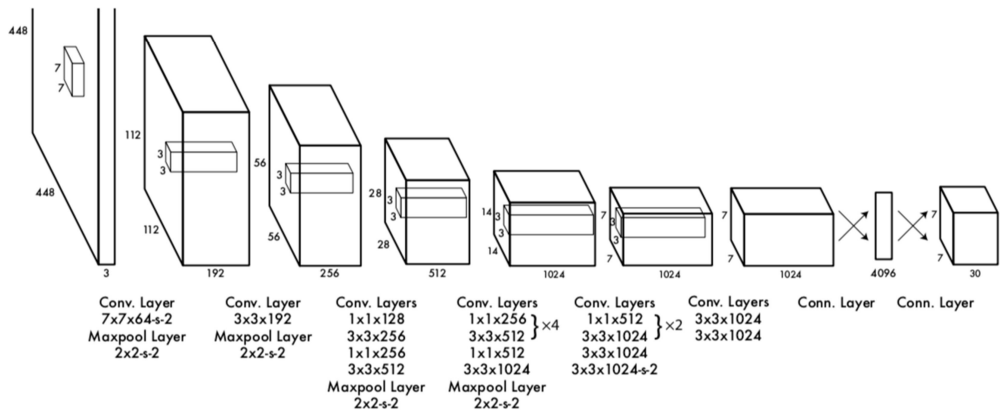

그림 16. YOLO Network(1)

상기 그림은 논문에서 가져온 네트워크의 그림이다. 이 구조는 GoogleNet의 아키텍처로부터 영감을 받아 Inception 블록 대신 단순한 컨볼루션으로 네트워크를 구성한 것이다. 224×224 크기의 Image Net Classification으로 Pretrain 시킨 것이며, 이후엔 입력 이미지로 448×448 크기 이미지를 입력으로 받는다. 이러한 과정을 거쳐 앞쪽 20개의 컨볼루션 레이어는 고정하고, 뒷 단의 4개 레이어만 Object Detection Task에 맞게 학습시킨다. 그림 17은 더 직관적인 그림이다.

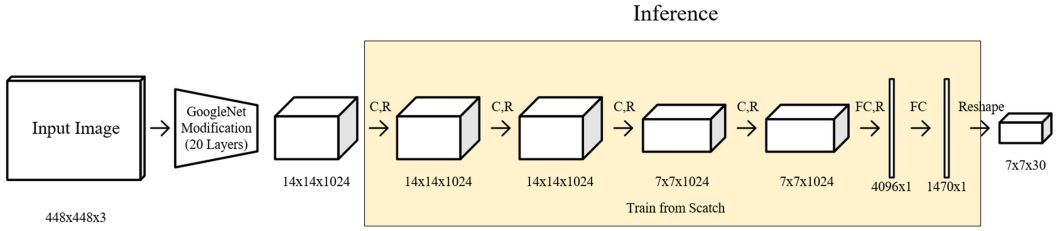

그림 17. YOLO Network(2)

상기 정리한 것들을 종합하여 네트워크의 출력인 7×7×30 피쳐맵에 대해 분석한다. 이 안에는 앞서 언급된 그리드 별 바운딩 박스와 신뢰도 지수, 그리고 각 클래스 별 예측값들이 포함되어 있다.

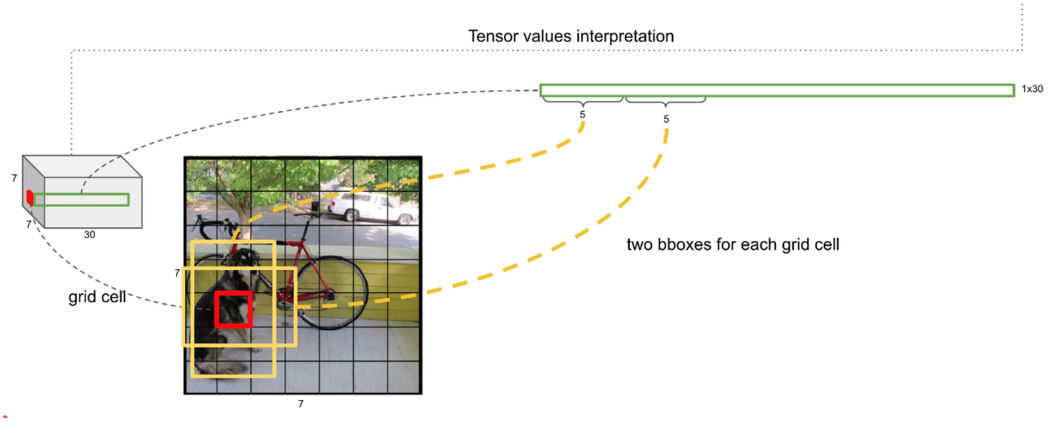

그림 18. Process for Getting Result of YOLO(1)

우선 7×7은 그리드를 의미하며, 각각의 인덱스는 총 30차원의 백터 값을 가진다. 그림 18에서 볼 수 있듯이 7×7 그리드 가운데 하나의 인덱스에 붉은 색 박스가 있는 것을 볼 수 있다. 앞서 하나의 인덱스에서 B개의 Bounding Box를 추측하며, YOLO에서는 이를 2로 설정하였다. 30차원 벡터 가운데 앞의 10개의 수는 바로 이 두 개의 박스를 의미한다. 하나의 박스는 중심점 x와 y, 너비와 높이 w, h 그리고 신뢰도 지수 C를 포함해 총 다섯 차원의 벡터로 나타낼 수 있으며, 두 개 박스는 10차원 벡터에 해당한다. 그 다음 오는 20차원 벡터는 해당 인덱스가 특정 클래스일 확률 값들이며, 여기서는 클래스가 20인 데이터 셋을 사용하였기 때문에 20차원 벡터로 표현된다. 앞서 박스의 신뢰도과 클래스별 확률 값을 산출해내는 수식에 대한 정리를 했으며, 이 둘을 곱해주면 해당 박스가 특정 클래스일 확률 값이 된다. 이를 인덱스 i의 모든 B개 바운딩 박스에 적용하고 이를 다시 S×S 인덱스에 적용하면 다음과 같은 결과를 얻을 수 있다.

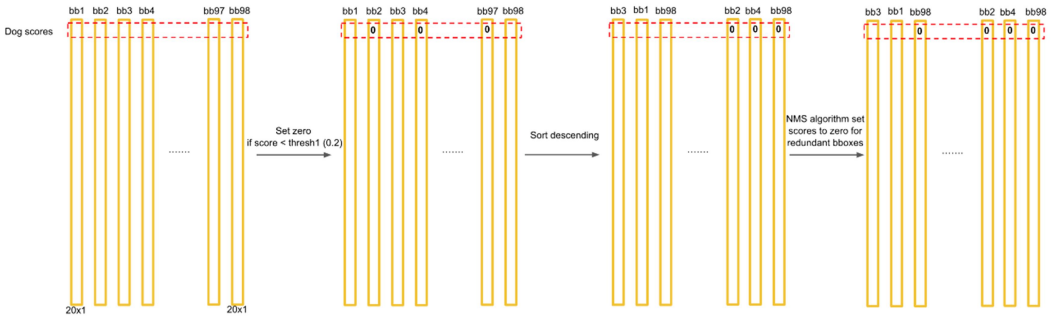

그림 19. Process for Getting Result of YOLO(2)

앞서 언급한 일련의 과정을 통해 구한 벡터들을 모두 취합하여 나란히 세우면 가장 위 차원부터 각 클래스 별 전체 바운딩 박스에서의 확률 값을 산출해낼 수 있다.
다음은 YOLO의 손실함수에 대한 설명이다. 수식은 다음과 같다.

$$
\begin{aligned}
&\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{obj}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right] \\
&\quad+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{obj}\left[\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right] \\
&\quad+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{noobj}\left(C_i-\hat{C}_i\right)^2 \\
&\quad+\sum_{i=0}^{S^2}\mathbb{I}_{ij}^{obj}\sum_{c\in classes}\left(p_i(c)-\hat{p}_i(c)\right)^2
\end{aligned}
\tag{3}
$$

이 때, $\mathbb{I}_{ij}^{obj}$은 Object가 등장하는 I 인덱스의 j번째 바운딩 박스가 최종 예측을 낸 것을 의미한다.

$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{obj}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right]$$

$$+\ \lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{obj}\left[\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right] \qquad (4)$$

$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2$$

이 수식은 상기 손실 함수의 앞 단이다. 최종 프레딕션에 포함된 바운딩 박스를 찾아내어 x, y좌표와 w, h 값, C 값을 통해 예측 값과 Ground Truth 값의 차를 구해 모두 더해준다. 이 때, x, y, C 값은 단순 차이 이며 w, h는 비율 값이기 때문에 제곱근을 적용해 차이를 구해준 차이점이 있다. 앞에 붙은 람다는 물체가 있을 때의 오차와 없을 때의 오차 사이의 비율을 맞춰주기 위한 상수이다.

$$\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{noobj}\left(C_i-\hat{C}_i\right)^2+\sum_{j=0}^{S^2}\mathbb{I}_{ij}^{obj}\sum_{c\in classes}\left(p_i(c)-\hat{p}_i(c)\right)^2 \qquad (5)$$

위 수식은 상기 손실함수의 뒷 단이다. 이 부분은 찾아야 하지만 찾아내지 못한 물체들에 대한 패널티를 매기는 수식이다. $\mathbb{I}_{ij}^{noobj}$는 물체가 없다고 판단된 i인덱스의 j번째 바운딩 박스가 실제로는 Ground Truth와 IoU가 가장 높은 인덱스를 의미한다.
최종적으로 모든 물체가 있다고 판단된 인덱스 i들에 대해 모든 클래스 들에 대한 예측 값과 실제 값의 차이를 구해 더한다.
이를 기반으로 하여 나온 Yolo 시리즈 중 최신 버전인 Yolo V4와 Yolo V5에 대해서 알아보도록 한다.

2.2.1 YOLO v4

YOLO v4는 최근에 발표되었다. YOLO v3에서 바뀐 점은 백본 네트워크가 Darknet-53에서 CSPDarknet-53로 변경되었다는 점이다. YOLO v4를 발표한 논문은 기존의 완성된 Yolo에 현존하는 최신 기술들을 접목하여 학습시켰을 때의 실험 결과를

리뷰한 논문이다. 이에 다양한 실험 결과를 거쳐 CSPDarknet-53가 가장 성능이 좋은 것을 알 수 있었고, 본 논문에서는 이를 적용하여 문제를 해결한다[7].

CSPDarknet-53는 DarkNet-53를 사용하는 Object Detection을 위한 컨벌루션 신경망 혹은 백본 네트워크이다. CSPNet 전략을 사용해 기본 계층의 피쳐맵을 두 부분으로 분할한 후 Cross-Stage Hierachy를 통해 병합한다. 분할 및 병합하는 전략을 사용하면 네트워크를 통해 더 많은 Gradient Flow가 가능하다는 특징을 갖는다.

그 외에는 트레이닝에서 Bag of Freebies(이하 BoF)을 사용하였으며 기존의 Spatial Attention Module(이하 SAM)과 Path Aggregation Network(이하 PANet)을 수정해서 적용했다. 수정된 SAM과 PAN의 구조는 그림 20에 나와있다. 그리고 최근 딥러닝의 정확도를 비약적으로 높여주는 것으로 매우 많이 알려진 Data Augmentation기술 역시 접목되어 있다.

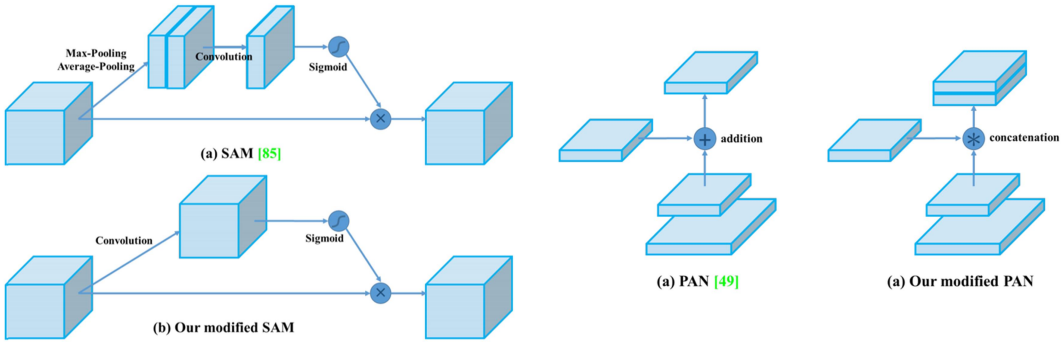

그림 20. Yolo V4를 위해 개선된 SAM과 PANet

PANet이 공개된 시기에는 Object Detection의 기술력이 매우 높다. 이 전에는 Bounding Box의 물체를 구별하는 것까지 였다면, 최근 들어서 해당 물체에 정확한 Mask까지 씌워주는 Instance Segmentation까지 수행되어야 한다. 이 때 PANet는 좋은 성능을 보이는데 성공을 하게 된다. 이 기술에 대한 설명은 다음과 같다. PANet는 YOLO v4의 Neck에서 SAM과 함께 사용되었으며 주로 모델에 통합되어 공간 정보를 보존하여 인스턴스 분할 프로세스를 향상시킨다. PANet이 YOLO v4에서 인스턴스 분할 과정에 선택된 이유는 공간 정보를 보존할 수 있는 기능이 있어 Mask Formation을 위한 픽셀의 적절한 위치 선정에 도움을 줄 수 있기 때문이다. 다음의 그림은 PANet의 아키텍쳐를 나타낸 그림이다.

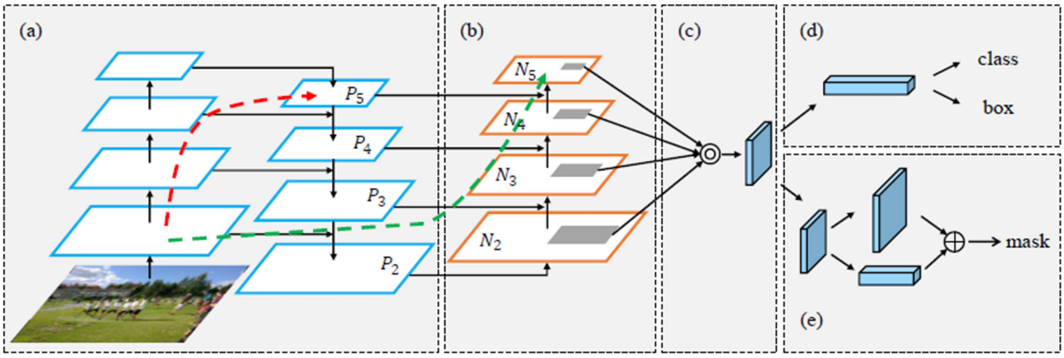

그림 21. PANet의 아키텍쳐

이미지는 신경망의 다양한 Layer들을 통과하면서 특징의 복잡성은 상승하면서 이미지의 공간적 분해능은 감소한다. 이로 인해, 픽셀 수준이 높은 수준의 특징에서 정확하게 판별될 수 없게 된다. YOLO v3에서 사용되었던 FPN은 Top-Down 경로를 사용해 정확한 위치 정보와 의미론적으로 풍부한 특징들을 결합하고 추출해낸다. 그러나 큰 객체에 대한 Mask를 제작하는 경우, 이러한 기술은 공간적 정보가 수백개의 레이어를 전파해야할 필요가 있을 정도로 지나치게 길어진다. 이에 반해 PANet은 상기 그림의 (a)에서 볼 수 있듯, FPN에서의 Top-Down 경로에 추가적인 Bottom-Up 경로를 수행한다. 이는 Bottom Layer들로부터 Top Layer들까지의 측면 연결부를 깨끗하게 하여 해당 경로를 단축시키는데 도움을 준다. 이를 Shortcut Connection이라고 하며, 약 10개의 Layer들 정도 밖에 되지 않는다.

이전까지 사용되던 Mask R-CNN과 같은 기술들은 Mask Prediction을 위해 Single Stage를 이용해 추출한 특징들을 사용했다. ROI가 큰 경우 ROI Align Pooling을 사용하여 상위 수준에서 특징을 추출했다. 이는 꽤 정확하지만 10 픽셀 정도의 차이를 가진 두 개의 예측이 두 개의 다른 계층에 할당될 수 있다. 그러나 이것은 실제로는 매우 유사한 제안이기 때문에, 원치 않는 결과를 초래할 수 있다. 이 문제를 방지하기 위해 PANet은 모든 계층의 기능을 사용하고 네트워크에서 어떤 계층이 유용한지 결정할 수 있도록 한다. 각 피쳐맵에서 ROI Align 작업을 수행하여 개체의 특징을 추출한다. 다음으로 요소별 Max Fusion 작업이 수행되어 네트워크가 새 기능을 조정할 수 있다.

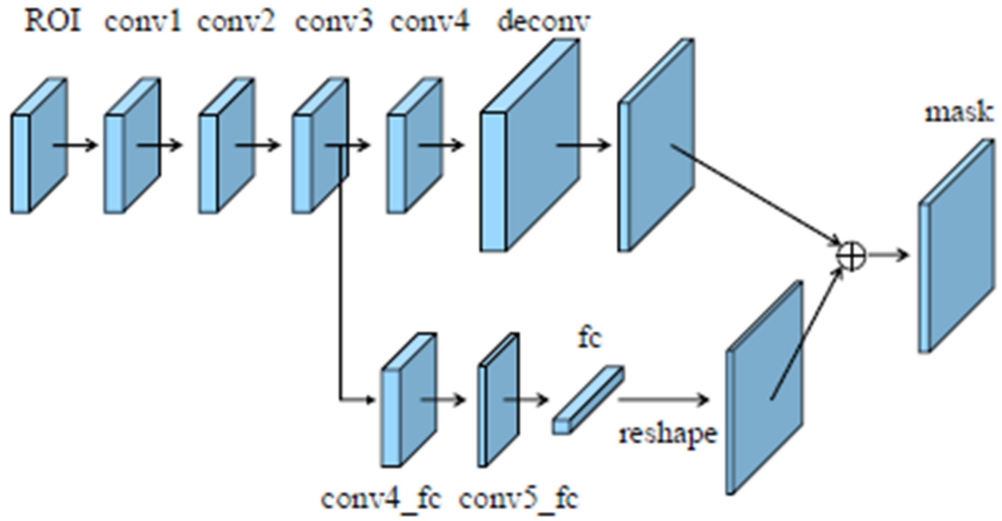

그림 22. Fully-Connected Fusion에서의 Mask Prediction Branch

Mask R-CNN에서는 공간정보를 보존하고 네트워크의 매개변수 수를 줄이기 때문에 Fully-Connected Layer 대신 Fully Convolutional Network(FCN)가 사용된다. 그러나 모든 공간 위치에 대해 매개변수가 공유되기 때문에 모델은 예측을 위해 픽셀 위치를 사용하는 방법을 실제로 배우지 않는다. 기본적으로 이미지 상단에는 하늘을, 하단에는 도로를 보여준다. 이에 반해 Fully-Connected Layer들은 위치에 예민하며 다른 공간적 위치들에 적용될 수 있다. PANet는 더 높은 Mask Prediction 정확도를 제공하기 위해 두 레이어들로부터 얻은 정보들을 사용한다.

2.2.2 YOLO v5

최종적으로 YOLO v5는 2020년 6월에 출시하여 YOLO v4에 달라진 것은 Pytorch, K mean anchors, genetic learning algorithms이 대표적인 큰 특징이라 볼 수 있다. YOLO v4와 같은 CSPNet 기반의 백본을 설계하여 사용했지만 Darknet이 아닌 PyTorch로 구현하였다. 이로 인해 Pytorch 환경에서만이 아닌 Tensorflow 기반으로도 작동이 가능하다. 두번째로는 K mean anchors와 genetic learning algorithms을 적용하여 사물 인식 또는 Image detecting 속도가 전 버전 보다 개선이 되었다. 사용자 지정 구성에 어려움이 없다면 Darknet의 YOLO v4를 사용하는 것이 가장 정확하지만 실시간에 가까운 물체 감지를 프로젝트에 신속하게 통합하기 위해서 YOLO v5 알고리즘을 사용하는 것이 좋다.

2.3 포트홀 검출 과정에서 연산량을 줄이기 위한 방법

2.3.1 이미지 이진화 기법 적용

이미지 이진화는 이미지의 픽셀 값을 0 또는 255로 모두 변환하는 것이다. 픽셀값을 0과 255만으로 바꾸기 위해 thresh라는 임계값을 먼저 정해야 한다. 임계값보다 큰 픽셀은 모두 0 그렇지 않으면 255로 변환하는 과정이 필요하다. 임계값 설정은 자동으로 설정해주는 Otus's method 알고리즘을 이용한다[10].

Othus's method 알고리즘의 기본적인 전제는 이미지가 bimodal distribution 이라는 것이다. 즉, 이미지 픽셀값들의 히스토그램은 허로 다른 두 개의 픽셀값에서 최대값이 나타나는 double peak 분포를 가진다는 것이다. 왼쪽 클래스의 피크는 배경값, 오른쪽 클래스의 피크는 물체의 값일 것이다. 이 두 분포를 가장 잘 나누는 픽셀값을 찾는 것이 이 알고리즘의 핵심이다. 이 전제에 의거해 다음의 수식을 세울 수 있다.

$$\sigma_T^2 = \sigma_\omega^2(t) + \sigma_b^2(t) \tag{6}$$

이것의 의미는 전체 분포의 분산 σ_T는 두 클래스 내의 (within-class) 분산(σ_ω)과 두 클래스 사이의 (between-class) 분산(σ_b)으로 이루어져 있다는 것을 뜻한다.

$$\sigma_\omega^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_b^2(t) \tag{7}$$

두 클래스 사이의 분산 (between-class variance)은 다음과 같이 정의된다.

$$\sigma_b^2(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_2(t)]^2 \tag{8}$$

이때, 두 클래스의 경계값은 σ_ω을 최소화 시킴과 동시에 σ_b를 최대화시키는 값으로 정해진다. 따라서 두 클래스 분포 중 하나를 선택해서 두 클래스를 가장 잘 구분하는 최적의 thresh 값을 찾아내는 것이다. 주로 두 번째 Between-class 분포를 이용한다.

2.3.2 ROI(Region Of Interest) 설정

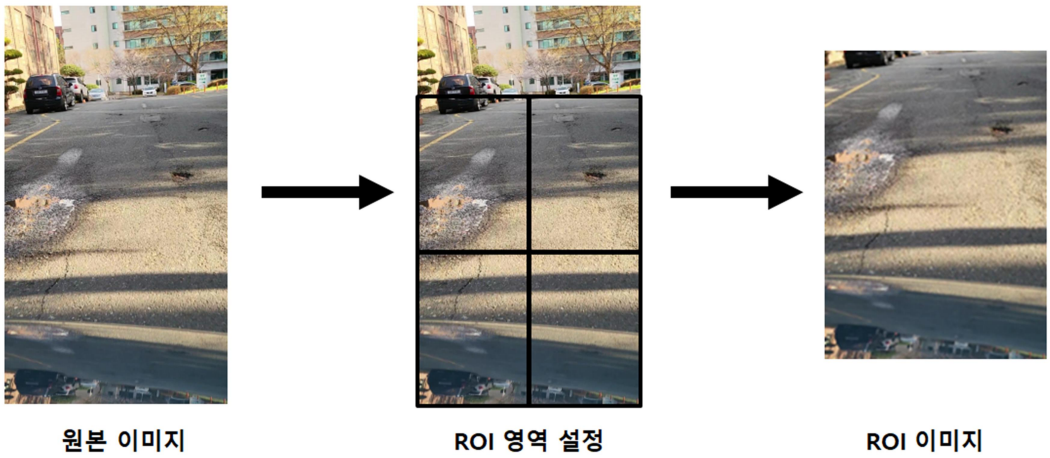

그림 23. ROI 영역 설정 적용

ROI는 이미지나 영상에서 내가 관심있는 부분을 뜻한다. 관심영역을 지정하는 것은 불필요한 영역에 대한 이미지 처리를 방지할 수 있으므로 연산량을 줄여 부하를 덜어주는 역할을 한다. 차량이 이동하는 도로에 포트홀을 검출하기 때문에 도로가 아닌 정면부의 배경이나 하늘 부분을 잘라내도록 한다.

2.4 포트홀 크기 검출

포트홀 크기를 검출하기 전 먼저 카메라 렌즈의 왜곡을 알아야 한다. 카메라는 렌즈 중앙을 벗어난 곳에 화각으로 인한 왜곡이 생긴다. 이러한 이유로 화면에서의 들어오는 영역이 가장자리로 갈수록 늘어나 보이거나 휘어져 보일 수 있다. 이런 문제로 인해 카메라 영역에서의 픽셀 값의 보정이 필요하다. 픽셀의 위치 마다 어느 정도의 보정이 필요한지에 대한 수식을 찾기 위해 선형 회귀 분석을 이용하였다.

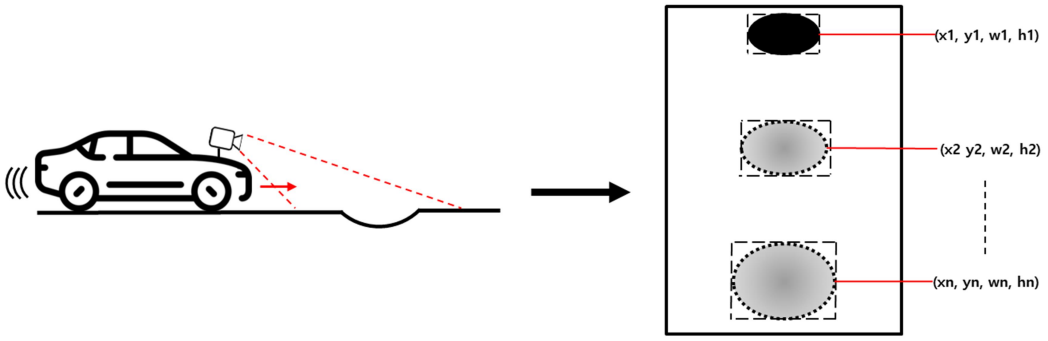

그림 24. 포트홀 크기 측정 방안

포트홀 크기 검출을 위해 차에 카메라를 일정한 각도로 고정하여 포트홀이 있는 도로를 촬영을 한다. 촬영한 영상에는 포트홀을 찾을 수 있는데 여기서 포트홀이 화면 다양한 영역에 다 나오게 촬영을 하는 것이 중요하다. 이렇게 촬영을 한 포트홀 영상에서 얻을 수 있는 것은 x_1, y_1, w_1, h_1의 정보를 얻을 수 있다. x_1와 y_1은 검출된 포트홀의 중점 좌표값이다. w_1, h_1은 화면에의 포트홀 가로, 세로 길이이다.

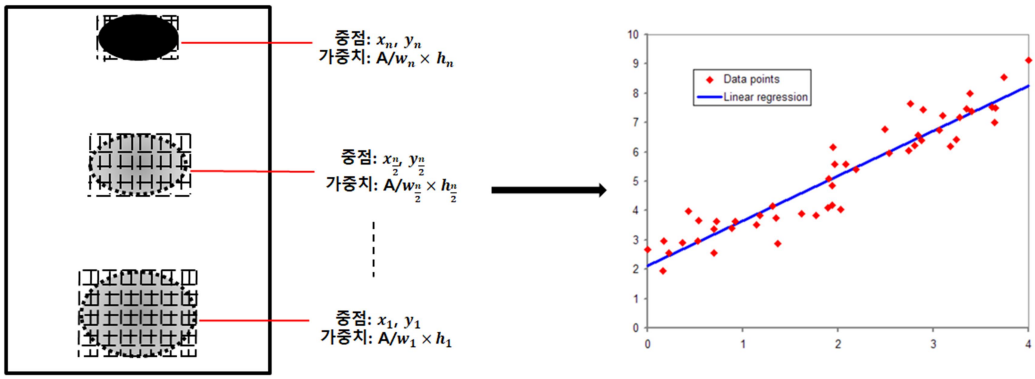

그림 25. 포트홀 크기 데이터

위에서 얻은 화면상의 포트홀 정보를 이용하여 이미지에서 포트홀 크기를 검출해 보고자 한다. 중점좌표 x_n와 y_n에서 구해지는 포트홀 픽셀의 넓이 $w_n \times h_n$을 구해 실제 포트홀의 넓이인 A 에 나눠준다. 나눠준 값인 $A/w_n \times h_n$ 을 이용하여 중점 좌표값에 일치하는 다양한 포트홀이 검출이 되었을 때 $A/w_n \times h_n$을 이용해 곱해준다면 이미지에서의 포트홀 넓이가 아닌 실제 포트홀의 넓이를 구할 수 있게

된다. 포트홀의 중점 마다 곱해져야 하는 값은 비선형으로 나타나는데 이것을 선형 회귀 분석을 이용하여 실제 포트홀 크기인 예측값을 알아내고자 한다.

3. 실험

3.1 실험 장비

본 연구의 실험을 하기 위해 필요한 장비로 도로위의 포트홀을 촬영하기 위한 Canon사의 EOS 750D을 사용하였고, 카메라로 받은 영상 데이터에서 포트홀 검출과 크기를 측정하기 위해 NVIDIA사의 Jetson AGX Xavier를 사용하였다.

그림 26. Jetson AGX Xavier

Jetson AGX Xavier	
GPU	512-core Volta GPU with Tensor Cores
CPU	8-core ARM v8.2 64-bit CPU, 8MB L2 + 4MB L3
Memory	32GB 256-Bit LPDDR4x | 137GB/s
Storage	32GB eMMC 5.1
DL Accelerator	(2x) NVDLA Engines
Vision Accelerator	7-way VLIW Vision Processor
Encoder/Decoder	(2x) 4Kp60 | HEVC/(2x) 4Kp60 | 12-Bit Support
Size	105 mm x 105 mm x 65 mm
Deployment	Module (Jetson AGX Xavier)
USB-C	2x USB 3.1, DP (Optional), PD (Optional) Close-System Debug and

	Flashing Support on 1 Port
Camera Connector	(16x) CSI-2 Lanes
40-Pin Header	UART + SPI + CAN + I2C + I2S + DMIC + GPIOs
HD Audio Header	High-Definition Audio
eSATAp + USB3.0 Type A	SATA Through PCIe x1 Bridge (PD + Data for 2.5-inch SATA) + USB 3.0
HDMI Type A	HDMI 2.0

표 1. Jetson AGX Xavier

그림 27. Canon EOS 750D

Canon EOS 750D	
형식	디지털, 일안 반사식, AF/AE 카메라
이미지 센서 크기	약 22.3 x 14.9 mm
이미지 유효 화소수	약 2,420만 화소
이미지 화면 비율	3:2
레코딩 이미지 형식	JPEG, RAW
레코딩 화면 비율	3:2, 4:3, 16:9, 1:1
동영상 기록 형식	MP4, MPEG-4 AVC/H. 264
동영상 기록 크기	FHD, HD, VGA
크기	약 131.9 x 100.7 x 77.8 mm
무게	약 555g
작동 온도 범위	0 °C – 40 °C
작동 습도	85% 이하

표 2. EOS 750D

3.2 YOLO Custom Training Set 만들기

본 논문에서는 포트홀(Pothole)을 만들 대상으로 한다.

3.2.1 포트홀 데이터 수집

포트홀 데이터를 수집하기 위해 울산에서 포트홀이 많이 있는 길을 찾아 카메라를 이용하여 포트홀 데이터를 수집한다. 영상으로 모은 포트홀의 이미지를 저장한다.

그림 28. 도로 위 포트홀 데이터 수집

3.2.2 트레이닝 이미지 라벨링

저장한 이미지를 불러와 관심 대상의 영역을 설정하고 라벨링한다.

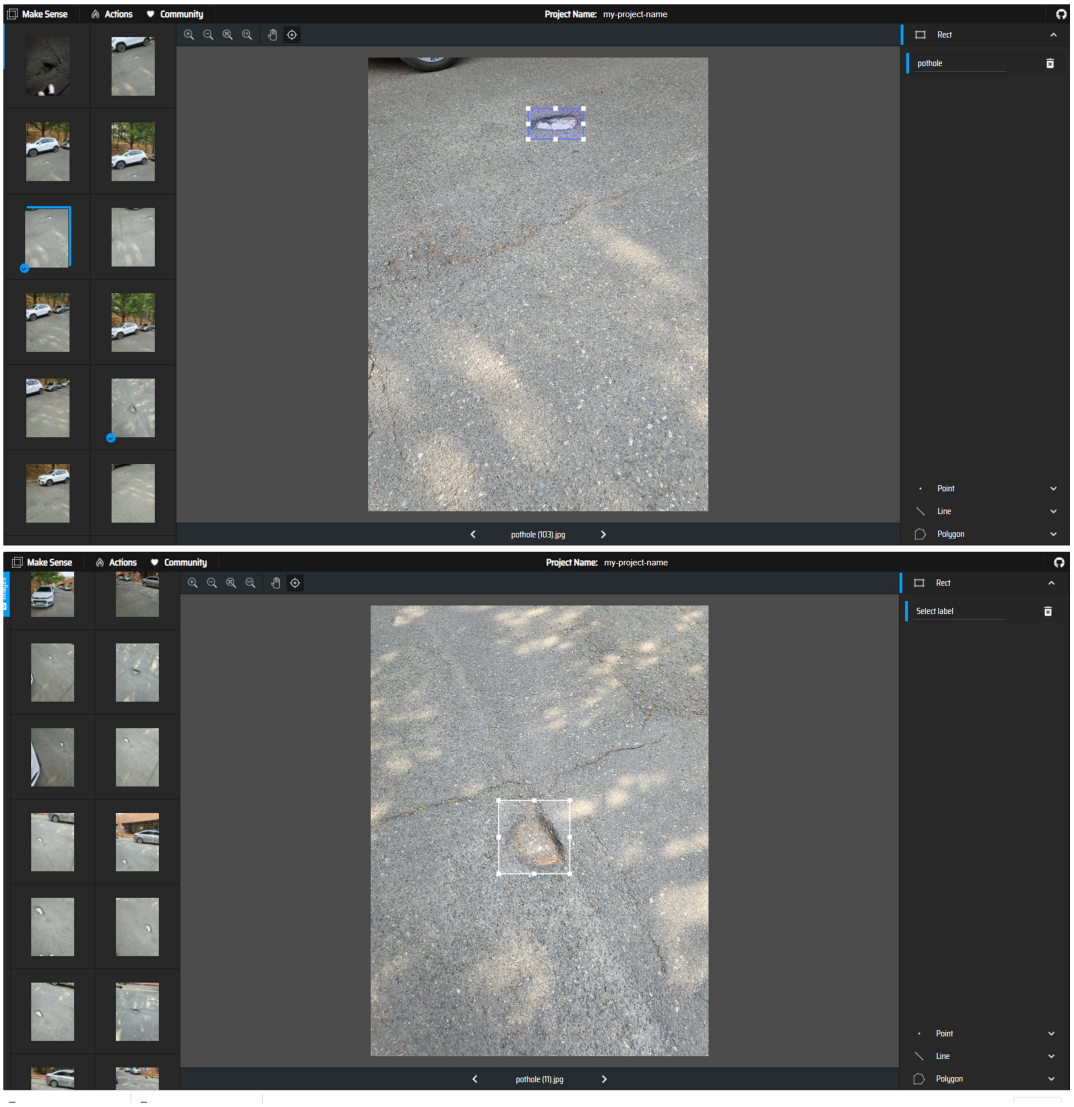

그림 29. Image Labeling

본 논문에서 관심대상인 포트홀은 다음과 같은 데이터로 표현된다.

0 0.257937 0.087500 0.246032 0.165000

0 0.690476 0.127500 0.253968 0.245000

다음 데이터에서

0 0.257937 0.087500 0.246032 0.165000

맨 앞의 데이터는 종류(0번은 Pothole.)를 뜻하고 그 뒤의 데이터는 이미지에서

관심역역의 x, y, width, height를 뜻한다.

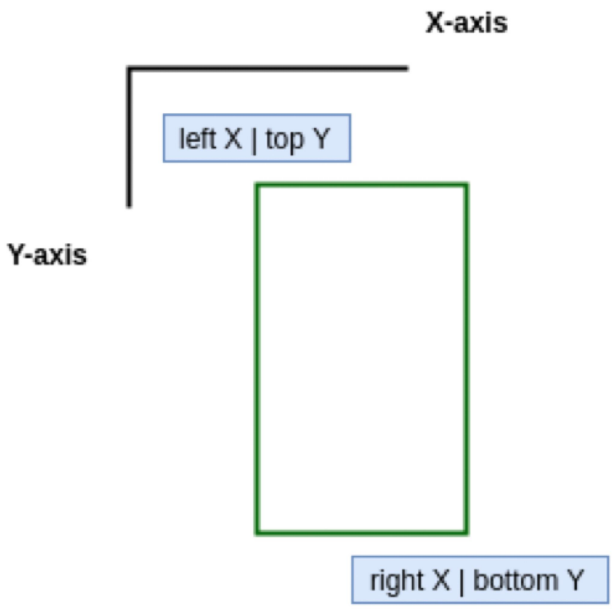

그림 30. 라벨링 데이터 설명

3.2.3 이미지 트레이닝

학습 명령을 통해 학습 프로세스 전체에서 모델이 학습이 되며 평균 손실 대 반복 차트를 볼 수 있다. 모델의 정확도를 위해 2미만의 손실 값을 가질 때까지 학습한다.

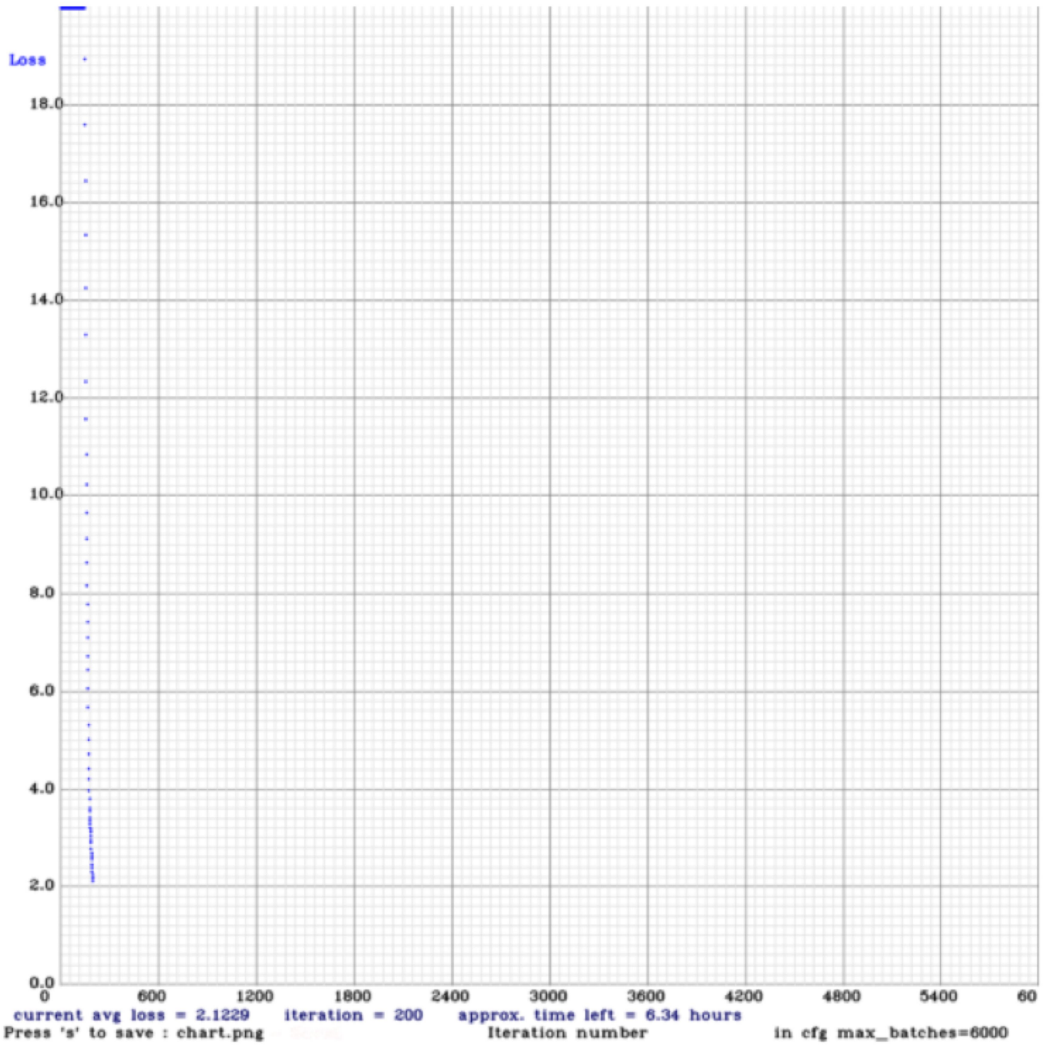

그림 31. 학습과정과 그에 따른 정확도 증가 및 손실 함수(1)

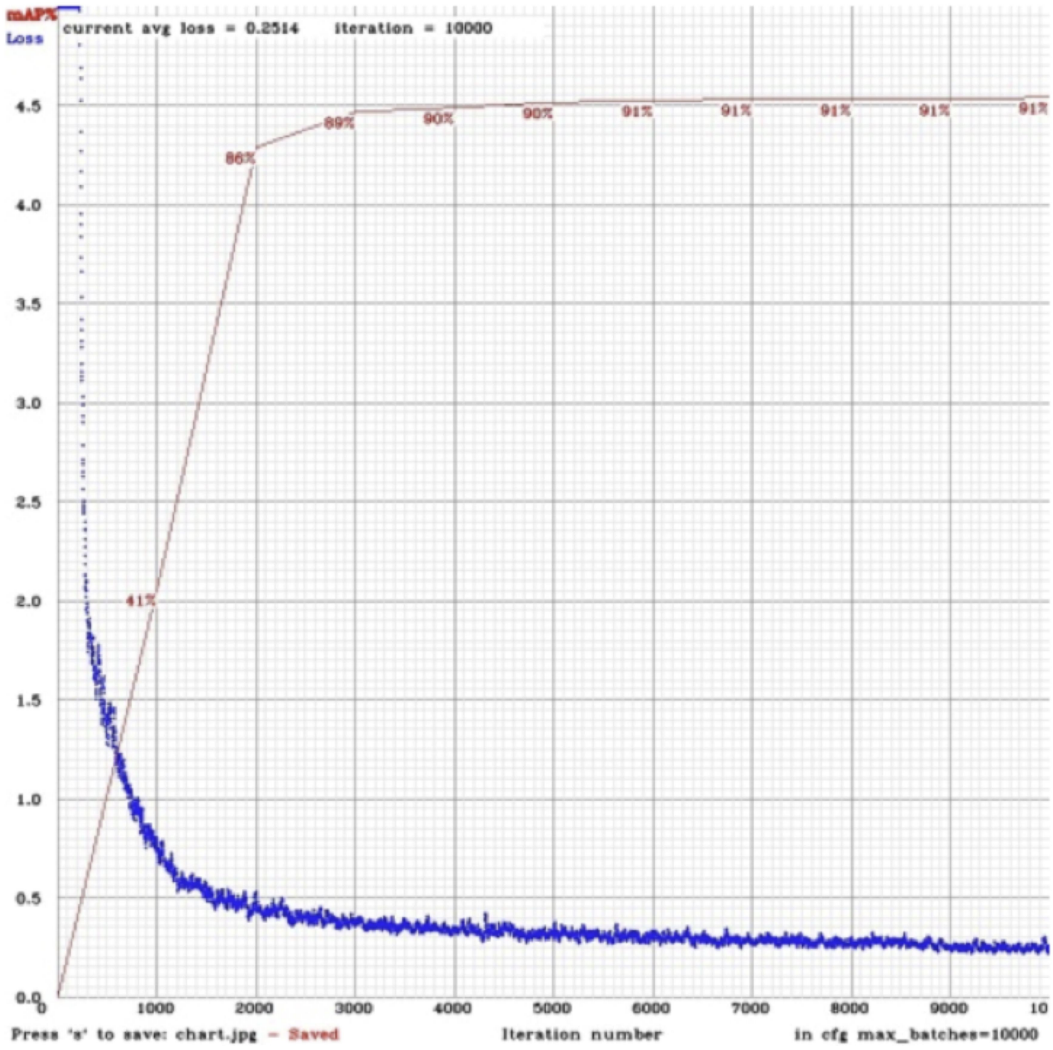

그림 32. 학습과정과 그에 따른 정확도 증가 및 손실 함수(2)

그림 32에서 학습과정과 그에 따른 정확도 증가 그리고 손실 함수 값을 확인 할 수 있다. 이때 손실 값이 감소 하지 않으면 학습을 중단하고 수정 후 다시 수행한다.

3.3 포트홀 검출 성능 비교

그림 33. YOLO v4를 이용하여 포트홀 검출

그림 34. YOLO v5를 이용하여 포트홀 검출

	YOLO v4		YOLO v5	
	FPS	Detection rate	FPS	Detection rate
영상1	14.20	0.65	16.80	0.73

영상2	14.70	0.69	17.00	0.77
영상3	14.30	0.75	16.90	0.78
영상4	14.50	0.71	16.50	0.75
영상5	13.90	0.67	16.10	0.74
영상6	14.10	0.69	16.00	0.70
영상7	14.80	0.72	16.30	0.73
영상8	14.70	0.71	16.10	0.71
영상9	15.10	0.76	17.50	0.80
영상10	15.00	0.77	17.00	0.83
영상11	14.30	0.72	16.90	0.77
영상12	14.50	0.71	17.00	0.79
영상13	14.20	0.68	16.50	0.75
영상14	13.80	0.65	16.30	0.77
영상15	14.00	0.66	16.60	0.71
영상16	14.20	0.71	16.90	0.75
영상17	14.90	0.70	17.30	0.76
영상18	14.50	0.78	16.90	0.80
영상19	14.00	0.77	16.50	0.81
영상20	14.80	0.79	17.20	0,84

표 3. YOLO v4와 v5 FPS 및 검출율

YOLO v4와 YOLO v5를 비교하였을 때 FPS와 검출율 높은 YOLO v5를 물체 검출 알고리즘으로 사용하기로 하였다.

3.4 포트홀 크기 측정 수식

그림 35. 실제 포트홀 크기

실험을 위해 포트홀을 선정후 가로와 세로 길이를 측정하였다. 실제 포트홀의 넓이는 $800cm^2$이다.

그림 36. 포트홀 x_1, y_1, w_1, h_1 데이터 검출

먼저 선형 회귀 분석을 하기 위해 x_1, y_1, w_1, h_1 데이터를 추출한다. 다양한 영역에서의 포트홀의 데이터를 수집하기 위해 카메라의 다양한 위치에서 포트홀이 잡힐 수 있도록 촬영을 한 것을 이용하여 x_1, y_1, w_1, h_1 데이터를 추출하여 Excel 파일에 저장한다. Excel에 모은 데이터 베이스를 이용하여 선형 회귀 분석을 통해 이미지 상의 포트홀 크기를 실제 포트홀 크기와 같게 구할 수 있도록 한다. 이 과정을 Colab을 이용하여 구현한다.

그림 37. 포트홀 크기 검출

데이터를 러닝하여 선형 회귀 분석을 통해 포트홀의 크기를 검출해낸 결과 실제 포트홀 $800cm^2$과 매우 유사한 수치가 검출되는 것을 확인 할 수 있었다.

3.5 포트홀 검출 시 연산량을 줄이기 위한 방안

3.5.1 연산량을 줄이기 위한 ROI 영역 설정

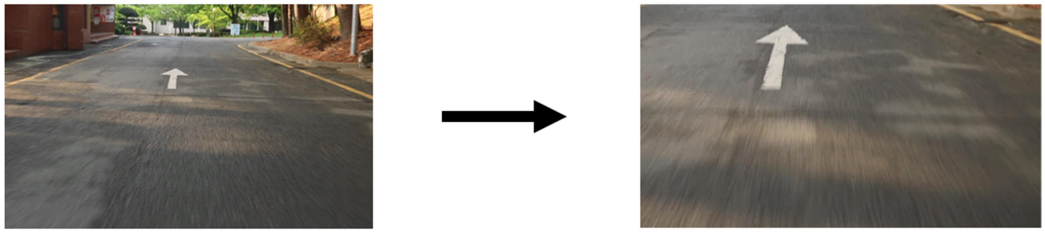

그림 38. ROI 영역 적용

그림 39. ROI 영역 적용 전 FPS

그림 40. ROI 영역 적용 후 FPS

같은 구간에서 영역에서 카메라에 들어오는 이미지 데이터를 가지고 ROI 적용하였을 때 FPS가 4.5가 올라갔다. 크게 상승하진 않았지만 영역 설정을 더 좁게 가져가면 추후에 더 올라갈 수 있을 것이라 판단하여 ROI 영역 설정을 적용하기로 하였다.

3.5.2 연산량을 줄이기 위한 이미지 이진화

그림 41. 이진화 적용 이미지

위의 그림은 영상을 이진화 처리하여 나오는 영상이다.

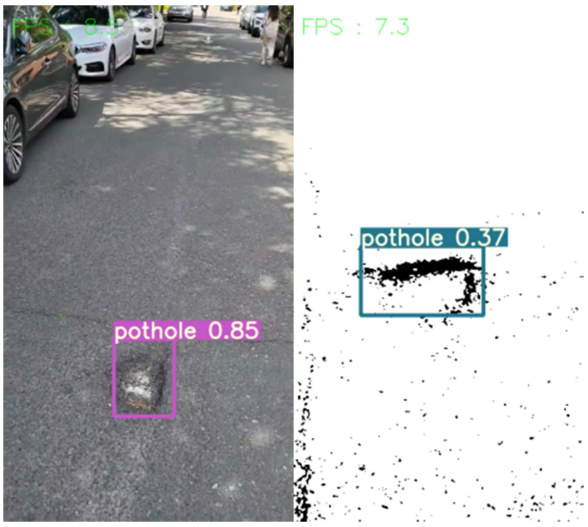

그림 42. YOLO v5 와 이미지 이진화 적용한 YOLO v5 포트홀 검출

이미지 이진화를 한 경우 검출율과 프레임 레이트가 떨어진 것을 볼 수 있었다. 이진화를 하게 되면 도로의 노면 상태에 따라 크랙과 점들이 많이 들어와 포트홀 검출에 적합하지 않다고 판단한다.

3.6 속도에 따른 포트홀 검출 및 크기 측정

그림 43. 10km/h 에서 포트홀 검출 크기 측정

	Detection rate	Size	FPS
영상 1	0.84	799.51cm^2	19.40
영상 2	0.87	798.80cm^2	19.70
영상 3	0.70	801.00cm^2	19.90
영상 4	0.80	800.99cm^2	20.20
영상 5	0.82	799.86cm^2	19.70
영상 6	0.42	800.47cm^2	20.00
영상 7	0.67	799.18cm^2	20.40
영상 8	0.68	797.53cm^2	19.80
영상 9	0.63	800.22cm^2	20.40
영상 10	0.65	799.60cm^2	19.50

표 4. 10km/h 검출율 및 크기 검출

그림 44. 20km/h 에서 포트홀 검출

	Detection rate	Size	FPS
영상 1	0.42	800.32cm^2	20.10
영상 2	0.56	802.00cm^2	19.60
영상 3	0.70	801.87cm^2	20.00
영상 4	0.55	800.76cm^2	20.30
영상 5	0.71	800.99cm^2	20.40
영상 6	0.52	801.10cm^2	19.80
영상 7	0.51	799.98cm^2	19.70
영상 8	0.68	802.27cm^2	19.50
영상 9	0.74	801.06cm^2	19.90
영상 10	0.56	800.30cm^2	20.00

표 5. 20km/h 검출율 및 크기 검출

그림 45. 30km/h 에서 포트홀 검출

	Detection rate	Size	FPS
영상 1	0.32	809.64cm^2	19.20
영상 2	0.50	810.76cm^2	19.30
영상 3	0.58	806.87cm^2	19.70
영상 4	0.56	802.20cm^2	19.60
영상 5	0.42	815.56cm^2	20.00
영상 6	0.45	811.41cm^2	19.90
영상 7	0.51	810.80cm^2	20.10
영상 8	0.73	812.73cm^2	20.10
영상 9	0.42	812.00cm^2	19.80
영상 10	0.67	809.21cm^2	19.40

표 6. 30km/h 검출율 및 크기 검출

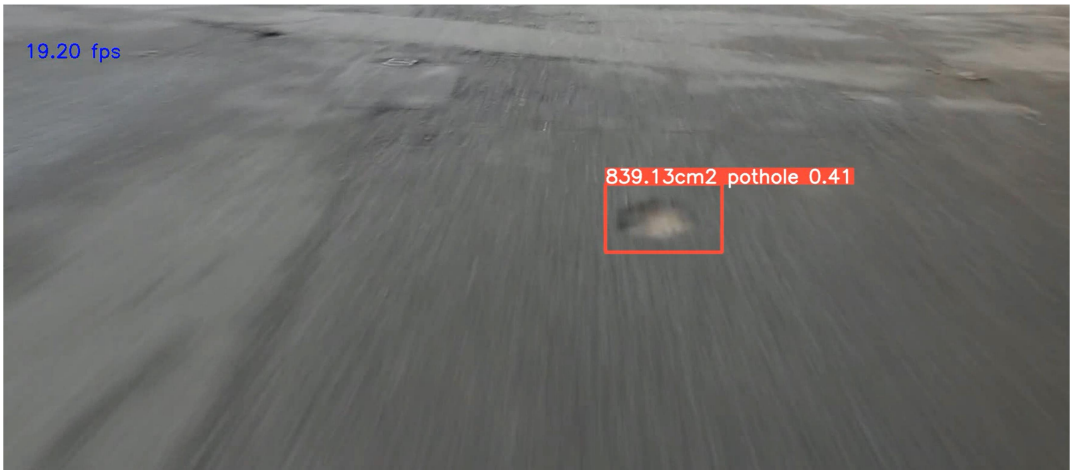

그림 46. 40km/h 에서 포트홀 검출

	Detection rate	Size	FPS
영상 1	0.41	839.13cm^2	19.20
영상 2	0.58	833.55cm^2	19.90
영상 3	0.52	843.54cm^2	19.60
영상 4	0.28	837.89cm^2	20.30
영상 5	0.33	832.78cm^2	19.50
영상 6	0.41	836.90cm^2	19.80
영상 7	0.56	839.70cm^2	19.80
영상 8	0.36	840.34cm^2	20.10
영상 9	0.47	847.68cm^2	20.00
영상 10	0.41	837.00cm^2	19.60

표 7. 40km/h 검출율 및 크기 검출

그림 47. 50km/h 에서 포트홀 검출

	Detection rate	Size	FPS
영상 1	0.76	840.22cm^2	20.40
영상 2	0.38	842.99cm^2	19.70
영상 3	0.37	849.91cm^2	19.80
영상 4	0.50	837.33cm^2	19.20
영상 5	0.27	839.43cm^2	19.60
영상 6	0.22	845.27cm^2	19.80
영상 7	0.39	839.83cm^2	19.30
영상 8	0.30	848.47cm^2	19.30
영상 9	0.47	840.39cm^2	20.10
영상 10	0.49	843.81cm^2	19.80

표 8. 50km/h 검출율 및 크기 검출

	Detection rate	Size 오차율	FPS
10km/h	0.708	0.1%	19.9
20km/h	0.595	0.13%	19.93
30km/h	0.516	1.26%	19.71
40km/h	0.433	4.98%	19.78
50km/h	0.415	5.35%	19.7

표 9. 속도별 평균 검출율, 크기 오차율, FPS 수치

본 실험은 포트홀이 있는 구간에서 속도를 10km.h에서부터 50km/h까지 10번씩 주행하여 얻은 영상을 가지고 검출율을 확인했다. 실험 결과 10 ~ 20km/h에서 검출율이 높았으며 30 ~ 50km/h에서 0.2이상 떨어진 것을 볼 수 있었다. FPS 수치는 거의 동일 했으며 포트홀 크기 검출 부분에서는 10 ~ 20km/h에서는 실제 포트홀 크기와 유사하게 나왔지만 30km/h부터 10 cm^2 오차가 생겼으며 40 ~ 50km/h 30~45cm^2 정도의 오차가 생겼습니다. 속도에 따른 평균 오차율은 10km/h에서 0.1%, 20km/h에서 0.13%, 30km/h에서 1.26%, 40km/h에서 4.98%, 50km/h에서 5.35%로 나타난다. 속력이 높아질 때마다 포트홀을 검출하는 구간이 짧아졌으며 인식률 또한 떨어졌고 크기 검출에 대한 오차율도 높아졌다.

4. 연구 결론

4.1 연구 고찰

본 논문에서 포트홀 검출과 포트홀 크기를 측정하는 알고리즘에 대해 연구하였다. 연산량을 낮추기 위해 이미지 이진화와 ROI 영역 설정을 하고 YOLO v5를 이용하여 먼저 포트홀을 검출해 낸 뒤 검출 된 위치 좌표값을 이용해 수식에 값을 대입하여 포트홀의 크기를 실시간으로 볼 수 있으며 실제 도로 주행 환경에서 적용 가능 한 것을 볼 수 있었다. 포트홀 검출에 관한 연구는 이미 많은 연구가 이뤄졌지만 실제 주행 환경에서 적용하기에는 많은 어려움이 있었다. 실시간 검출 및 포트홀 크기를 측정을 실제 주행 환경에서 엣지 컴퓨팅(Edge Computing)을 이용하여 구현한 것도 상당히 유의미하다. 본 연구의 실험을 진행하기 위해 포트홀 검출을 할 때 YOLO 버전을 비교하여 성능의 차이를 확인하였다. 수치를 비교하였을 때YOLO v4에 비해 v5는 FPS가 약 15% 성능이 증가하였고, 검출율은 약 1% 미비하게 증가한 것을 확인할 수 있었다. FPS를 증가시키기 위해 ROI 영역을 설정하여 진행한 결과 FPS는 27% 성능이 증가한 것을 알 수 있었다. 시스템을 종합하여 속도별 측정을 한 결과 10km/h와 20km/h에서 포트홀 크기 검출에 대한 오차율이 약 0.1% 였지만 30km/h에서 1.26%, 40km/h에서 4.98%, 50km/h에서 5.35%로 속도가 높아질수록 오차율이 커졌다.

본 연구의 실험을 진행하였을 때 포트홀을 검출하는 부분에서는 문제가 없었지만 크기 검출에서는 속도가 높아질수록 오차율이 증가되었다. 추후 연구에 이를 해결하기 위해 선형 회귀 분석을 하기 전, 속도별 좌표값 데이터를 추가하여 테스트를 진행하고자 한다.

4.2 향후 연구

본 논문에서는 실시간 포트홀 인식 및 크기를 알아내기 위한 실험을 진행하였다. 이를 통해 엣지 컴퓨팅 장치에서 포트홀 인식 및 크기 검출 기술을 구현해냈다. GPS 데이터를 추가하여 실제 차가 달리는 환경에서 포트홀 인식과 크기 검출을 할 때 위치 정보를 나타내어 어느 위치에 포트홀 검출과 크기 데이터를 저장할 수 있도록 한다.

본 과제에서 활용한 Jetson Xavier AGX 의 상위버전인 Orin 이 최근 출시되었다.

그림 48. AGX Xavier 와 AGX Orin Xavier 성능 비교표

AGX Orin 은 기존의 AGX 성능에 3 배 이상의 효과를 낼 수 있다. 이를 적용하는 것이 지금 AGX 를 사용하는 것보다 실시간 환경에 더 적합하다 생각되어 AGX Orin 으로 하드웨어를 바꿔 향후 연구에 적용하고자 한다.

[참고문헌]

[1] 송백기 “포트홀 검출을 위한 신경망 하드웨어 구조” 경희대학교 석사 학위논문, 2017.8

[2} 김정주 “2D LiDAR기반 3차원 포트홀 검출 시스템 구현” 전남대학교 석사 학위논문, 2017.2

[3] 김귀훈, 유태완, 홍정하, 김민석, 홍용근 “지능형 에지 컴퓨팅 기술 및 표준화 동향” 표준·시험인증 정책 기술 동향 2019.2

[4] 홍정하, 이강찬, 이승윤 “엣지 컴퓨팅 기술 동향” 한국전자통신연구원 전자통신동향분석 35권 제 6호 2020.12

[5] 김준현 “고속도로 위 지뢰 ‘포트홀’ 올해 상반기에만 1천900백건” 국토일보 2020.10

[6] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi “You Only Look Once Unified, Real-Time Object Detection” University of Washington, Allen Institute for AI, Facebook AI Research May 2016

[7] Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao “YOLOv4 Optimal Speed and Accuracy of Object Detection” University of Washington, April 2020

[8] NVIDIA, “Bringing Cloud-Native Agility to Edge AI Devices with the NVIDIA Jetson Xavier NX Developer Kit”, https://developer.nvidia.com/blog/bringing-cloud-native-agility-to-edge-ai-with-jetson-xavier-nx/

[9] NVIDIA, “NVIDIA Jetson AGX Xavier Delivers 32 TeraOps for New Era of AI in Robotics”, https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/?ncid=so-fac-mdjngxxrmllhml-69163

[10] Voyager, [OpenCV] 이미지 이진화(Image binarization)를 이용한 image

segmentation (Python), https://gmnam.tistory.com/263