Pusan National University

Computer Science and Engineering

Technical Report 2020-38

차량 운행 상태 분석을 위한 Sim2Real 딥러닝 기반 차량 동작 인식 시스템 설계 및 구현

여기에 팀 이름 입력

201524582 정희석

201524473 방형진

201524527 이석준

지도교수 백윤주

목 차

1. 서론

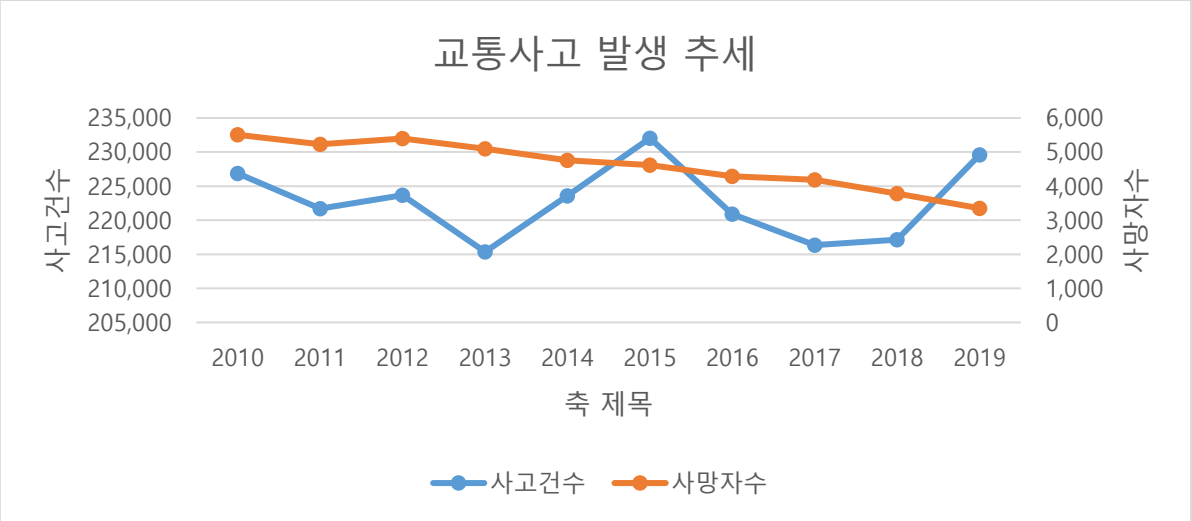
1.1. 연구 배경

대한민국의 차량보유 대수는 2020년 기준 2409만대가 넘으며[1] 이는 거의 인구 2.14명 당 한 대 수준으로 거의 가구당 1대의 자동차가 있는 수준이다. 거기에 더해 현대에 들어 운전면허 시험 과목의 간소화 등 여러 이유로 점점 운전 인구가 늘어가는 감에 따라 교통사고의 발생 건 수도 점점 늘어나고 최근 10년 동안 매년 20만 건이 넘는 교통사고가 발생하고 있다(표 1, 그림 1).

교통사고 추세

분석지표	구분	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
사고건수	전체	226,878	221,711	223,656	215,354	223,552	232,035	220,917	216,335	217,148	229,600
	1일평균	621.6	607.4	611.1	590.0	612.5	635.7	603.6	592.7	594.9	629.0
	인구10만명당	459.2	445.4	447.3	428.8	443.3	458.4	434.9	420.5	420.5	444.0
	자동차1만대당	105.8	101.2	99.0	93.0	93.7	93.7	86.4	85.9	80.4	83.5
	1000세대당	11.4	11.1	11.1	10.5	10.8	11.0	10.4	10.0	9.9	10.2
사망자수	전체	5,505	5,229	5,392	5,092	4,762	4,621	4,292	4,185	3,781	3,349
	1일평균	15.1	14.3	14.7	14.0	13.0	12.7	11.7	11.5	10.4	9.2
	인구10만명당	11.1	10.5	10.8	10.1	9.4	9.1	8.4	8.1	7.3	6.5
	자동차1만대당	2.6	2.4	2.4	2.2	2.0	1.9	1.7	1.7	1.4	1.2
	1000세대당	0.3	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.1
부상자수	전체	352,458	341,391	344,565	328,711	337,497	350,400	331,720	322,829	323,037	341,712
	1일평균	965.6	935.3	941.4	900.6	924.6	960.0	906.3	884.5	885.0	936.2
	인구10만명당	713.3	685.8	689.1	654.5	669.3	692.3	653.0	627.5	625.6	660.8
	자동차1만대당	164.3	155.8	152.4	142.0	141.5	141.5	129.7	128.1	119.5	124.3
	1000세대당	17.7	17.0	17.0	16.1	16.3	16.7	15.6	14.9	14.7	15.2

(표 1. 2010~2019 교통사고 발생 현황, 도로교통공사(TAAS))



(그림 1. 2010~2019 교통사고 발생 현황, 도로교통공사(TAAS))

표 1과 그림 1에서 보다시피 20만 건 이상의 교통사고가 발생하면 그 중에는 차 vs 차, 차량 단독, 차 vs 사람 사고의 유형의 교통사고가 발생하는데 교통사고 처리 과정에서 어떠한 이유로 사고가 발생했는지 정확하게 알아내는 것이 사후 대처와 유사 사고 예방을 위해 필수적이다.

1.2. 기존 문제점

기존의 경우 차량 vs 차량 교통사고가 발생하면 사고 당사자들의 의견과 해당 차량과 주변 차량의 블랙박스 영상, 주변 CCTV등 시각적 데이터를 토대로 교통사고에서의 과실을 가린다. 이 때 당사자들의 기억이 정확하지 않거나 블랙박스가 촬영 각도 등의 문제로 제대로 된 영상을 확보하지 못했을 경우 사고의 과실을 정확하게 측정하지 못하는 문제가 생긴다. 그리고 차량의 결함에 의한 사고는 시각적 데이터로만 가지고는 판별하기 어려운 문제도 있다.

1.3. 연구 목표

기존 연구들은 실제 차량 데이터를 수집해서 해당 데이터를 기반으로 모델을 작성, 모델을 학습하고 자동차 상태를 예측하는 연구가 주로 진행되고 있다. 기존 연구의 예시들 중 부산대학교 임베디드 시스템 연구실에서 2019년 진행한 실시간 운전자 행동 분석을 위한 차량 상태 인식 온-디바이스 딥 러닝 시스템의 설계 및 구현과 같은 연구에서는 실제 차량의 센서 데이터를 수집, 분석하여 차량 상태 인식을 진행하였다[2]. 우리 팀은 차량에 내장된 각종 센서에서 데이터들을 취득하여 해당 차량의 주행 상태를 예측하는 딥 러닝 모델을 제작한다. 단, 기존 연구들과 다르게 모델의 학습에는 실제 차량 데이터가 아닌 실제와 유사한 차량 시뮬레이션 프로그램의 차량 센서 데이터를 사용하고 여기에 Sim2Real 기술을 적용하여 실제 차량의 센서데이터와 비슷하게 가공한 후, 실제 테스트에는 실제 차량 데이터를 사용한다.

2. 과제 배경 지식

2.1. TensorFlow

TensorFlow은 'Google'사에서 개발한 머신러닝(딥러닝)용 라이브러리이다. 하이 레벨 언어인 Python등의 언어를 지원하며 많은 양의 참고자료와 간단한 사용법으로 머신러닝 처음부터 로우 레벨로 구현하거나 신경망 구조 자체를 변경하는 것이 아닌 간단한 수준의 머신러닝에 적합하다. 물론 이를 이용해서 전문가 수준의 복잡한 딥러닝도 가능하다.

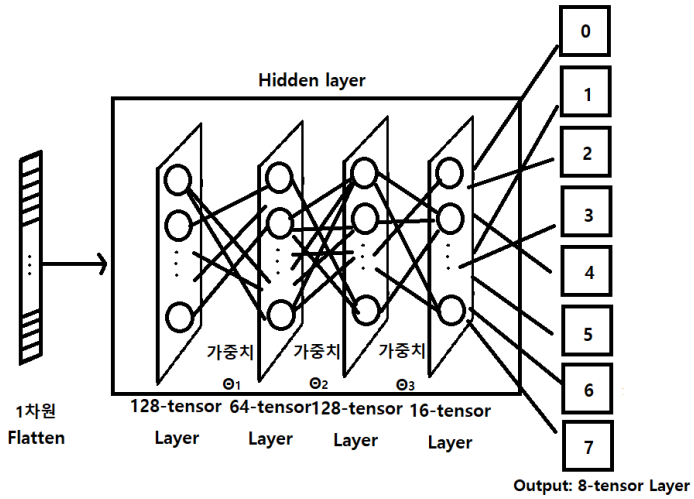
(1) Keras module

Keras 모듈은 TensorFlow에서 모델 제작과 학습을 간편하게 할 수 있도록 도와주는 모듈이다. 해당 모듈은 Python을 기본으로 사용하고 있으며 딥러닝에서 신경망을 구축, 학습, 예측에 사용되는 클래스를 제공하기 때문에 이 클래스들을 받아와서 설계할 수 있고 또한 자신만의 모델을 커스터마이징 할 수 있는 장점이 있다.

(2) Neural Network

- DNN

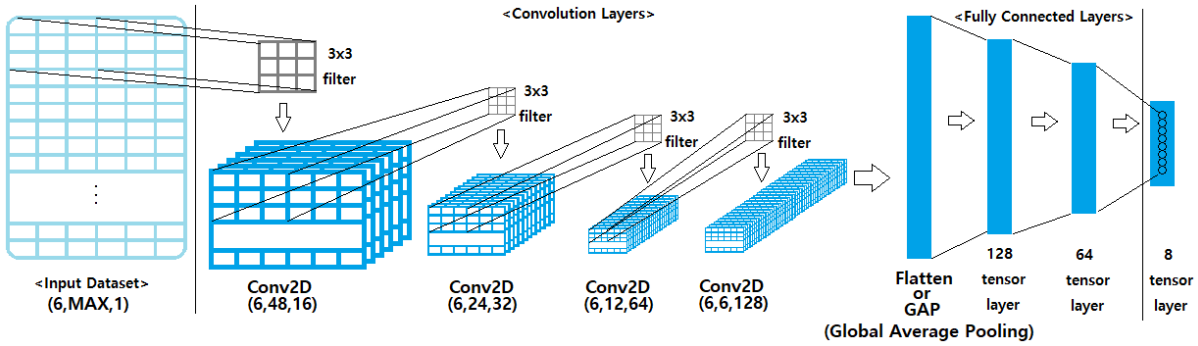
Deep Neural Network(Artificial Neural Network)으로 두뇌의 신경망을 모방하여 각 층에 두뇌의 신경인 뉴런의 역할을 하는 Tensor를 배치, 각 tensor에 데이터가 들어가서 다음 층으로 넘어 갈 때 가중치를 가지고 다음 층으로 진행, 해당 층이 2개 이상이면 Deep Neural Network로 칭해지며 이 층들을 Fully Connected Layers라 한다. 이와 같은 과정을 통해 결과를 예측하게 된다.



(그림 2. DNN모델 예시)

- CNN

Convolution Neural network으로 2차원의 데이터를 여러 채널로 나눠서 각 채널에 합성 곱 필터를 이용하여 feature map으로 만들며 이 과정을 Convolution Layers라고 한다. 이 feature map을 1차원의 데이터로 바꿔서 DNN의 Fully Connected Layers에 넣어서 예측하게 된다.



(그림 3. CNN모델 예시)

2.2. Sim2Real

Sim2Real은 현실이 아닌 시뮬레이션 프로그램 등에서 얻어낸 가상의 데이터를 일련의 후처리 과정을 거쳐 현실과 비슷하게 가공하여 학습 등에서 마치 실제 데이터를 사용하는 것과 비슷한 효과를 가지게 하는 기술이다. 여기에는 다음과 같은 3가지의 방식이 있다.

(1) System Identification

가상의 데이터를 추출해내는 프로그램 자체를 개량하여 처음부터 현실과 매우 유사한 테스트 데이터를 만들어 내는 것이다. 이는 그렇게 해서 추출해낸 데이터 자체에 별다른 후처리를 하지 않아도, 모델 개량이 간단하다는 장점이 있다. 이 경우 시뮬레이션 프로그램 자체를 좀 더 사실적이고 많은 데이터를 제공하는 프로그램으로 대체하거나 시뮬레이션에서 데이터를 추출하는 프로그램을 수정하여야 한다.

(2) Domain Randomization

일정 개수의 데이터를 추출해 낸 뒤, 그 데이터를 알고리즘을 통해 랜덤하게 조정된 수많은 데이터를 자동으로 생성하고 이렇게 생성된 데이터를 후처리를 통해서 모델에 학습시키는 방식이다. 이 경우 데이터를 직접 추출하는 시간이 크게 단축된다는 장점이 있으나, 추출해낸 데이터를 Randomizing하기 위한 알고리즘을 따로 제작하는 데에 시간이 소비된다.

(3) Data Adaptation

가장 간단하게 시뮬레이션에서 많은 수의 데이터를 직접 추출해 내고 이후에 후처리를 통해서 실제와 비슷하게 만들어 낸 후, 데이터를 학습시켜서 모델의 정확도를 높이는 것이다.

이번 프로젝트의 경우 시뮬레이션 프로그램으로는 상용화된 EuroTruckSimulator2를 사용하고, 데이터 추출 프로그램도 이미 제작이 완료된 프로그램을 사용하기 때문에 우리가 변경할 수 있는 부분이 매우 제한적이다. 따라서 System Identification을 제외한 두 개의 방식 중에서 선택하기로 하였고, 그 중 시뮬레이션 프로그램이 이미 갖춰진 상황에서 별다른 처리 없이 데이터의 개수로 학습 정확도를 높일 수 있는 Data Adaptation을 사용하였다.

2.3. OBD(On Board Diagnostics)

온보드 진단기는 차량의 상태를 진단하고 결과를 알려주는 장치이다. 자동차에는 다양한 센서들이 탑재되어 있는데 이 센서들은 ECU(Electronic Control Unit)에 의해 정밀 제어된다. 이때 OBD는 이 차량 내의 ECU에서 각종 센서들의 값을 받아서 차량의 상태를 진단하는 시스템이다[3]. 현재 OBD-II의 표준 규격이 사용되고 있으며 헤드업 디스플레이가 발전하면서 OBD에 대한 존재가 알려지고 있다. 또한 아이나비, 루카스 등 블랙박스에 OBD-II 연결을 지원하여 영상에 차량의 상태를 같이 기록하는 블랙박스도 출시가 되어 차량 센서 데이터에 접근성이 좋아지고 있다.

3. 연구 내용

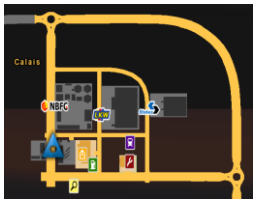
3.1. 시뮬레이터 데이터 수집

차량 시뮬레이션을 사용하여 모델 학습을 위한 가상의 차량 센서데이터를 수집하고, 해당 데이터를 실제 데이터와 비슷하게 정규화 과정을 거친다. 본 프로젝트의 학습데이터 수집을 위한 시뮬레이션 프로그램은 'SCS 소프트웨어'사의 'Euro Truck Simulator 2'가 사용되었고 보다 실제와 비슷한 환경을 위해 'Logitech'사의 'G29' 레이싱 휠을 사용하여 데이터를 수집하였다(그림 2).

(그림 2. 시뮬레이션 디바이스 사진)

3.1.1. 대상 설정

시뮬레이션 사용에 앞서 학습에 사용할 데이터의 종류와 해당 데이터를 모으기에 적합한 주행 코스를 선택한다. 본 프로젝트에서는 차량의 주행 상태를 직진, 좌커브, 우커브, 좌회전, 우회전, 좌측 차선변경, 우측 차선변경, 정지 이렇게 7가지로 나누었고 이에 대한 학습데이터를 수집하기 위해 시뮬레이션 프로그램 상에서 Calais의 시내 코스를 주행 코스로 사용하였다(그림 3).



(그림 3. 시뮬레이션 주행 코스)

해당 주행 데이터는 Ets2SdkClient 프로그램을 사용하여 dat파일과 동영상 파일로 추출한 뒤, 해당 파일을 csv파일로 변환하였다. dat추출 프로그램의 경우 Ets2SdkClient 유저 플러그인으로 임베디드 연구실에서 센서 데이터 수집용으로 수정한 프로그램을 사용하였고, csv변환 프로그램의 경우 Python을 사용하여 연구실에서 센서 데이터 변환용으로 구현해서 사용했던 프로그램을 수정하여 사용하였다.

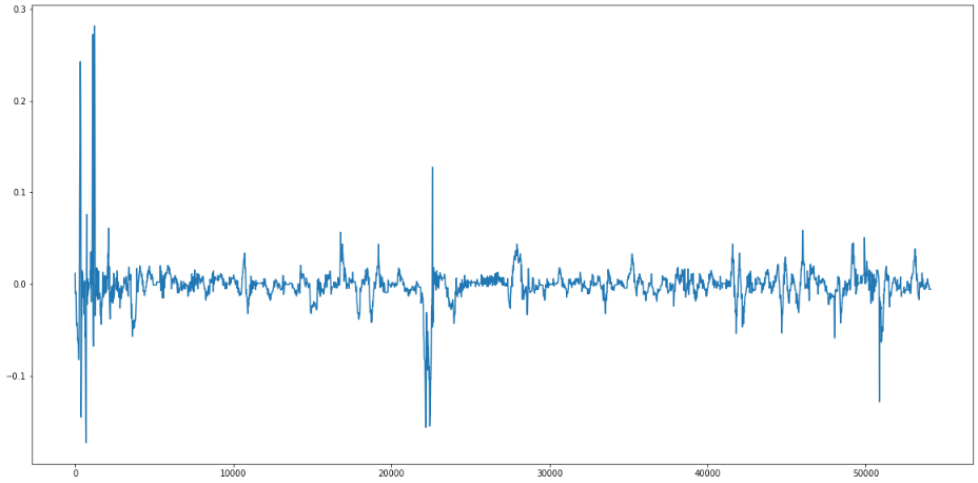
3.1.2. 사용 센서 데이터 선정

시뮬레이터 차량에서 나오는 데이터들을 분석해서 가장 실제 데이터와 근접하고 값의 편차가 크지 않은 데이터를 선정하기 위해 데이터 값의 범위를 확인했다.

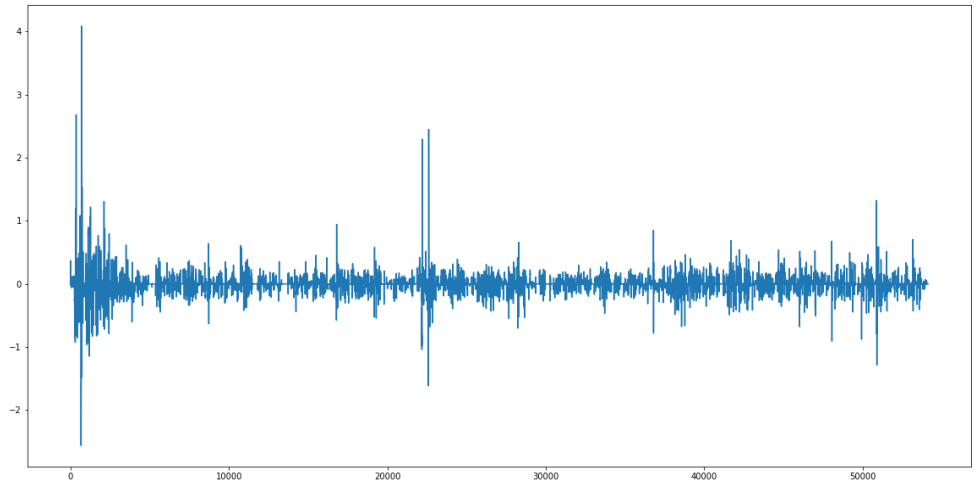
(1) 시뮬레이터 차량

시뮬레이터 프로그램에서의 데이터는 20종류의 데이터가 나오는데 이 중에서 운행 상태에 관련되는 데이터는 시뮬레이터 조향 핸들, 가속 페달, 브레이크 페달 위치 값, 게임 내 차량 속도, 좌우 가속도 값, 차량 회전 값을 추출하여 사용하였다.

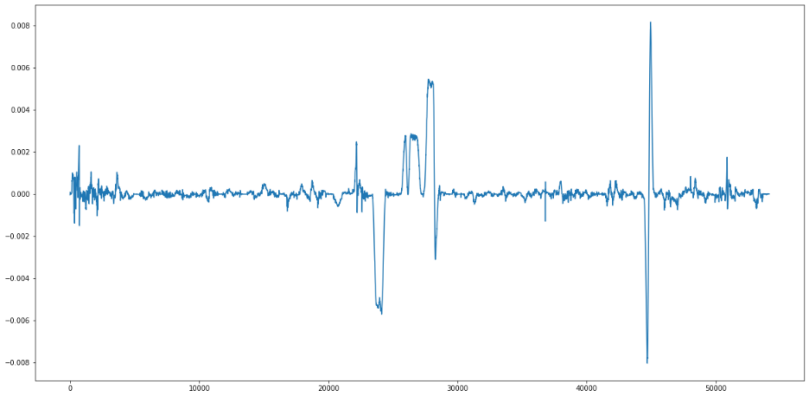
이 데이터를 확인하기 위해서 각 column별 데이터 값을 확인해 보았다.



(그림 4. 핸들 데이터 값 그래프)



(그림 5. 좌우 가속도 값 그래프)



(그림 6. 각속도(yaw rate) 값 그래프)

처음에는 모든 데이터를 다 쓰려고 시도했으나 데이터 분석과 진행 과정에서 계속 사용할 센서 데이터를 바꿔가면서 진행하였다. 데이터를 살펴보면 시뮬레이터 데이터는 상당히 불안정한 것을 확인할 수 있다. (그림 4, 5, 6) 특히 시뮬레이터 센서 데이터에서 노면 상태에 따라 센서 데이터 값이 상당히 실제에서 나오기 힘든 수치가 나오는 것을 확인할 수 있었다.

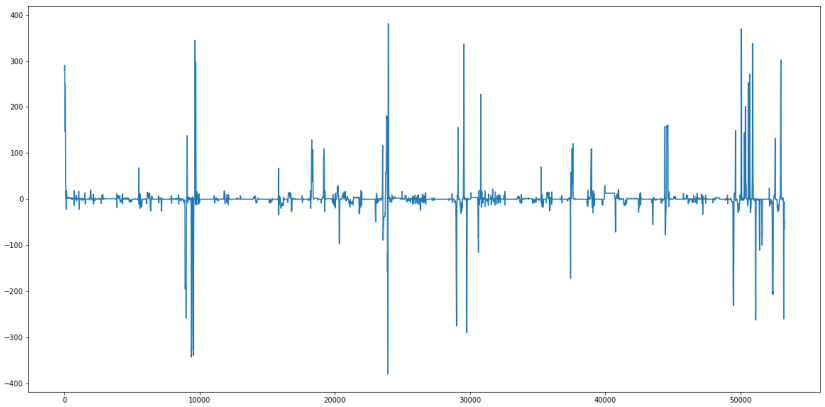
센서종류	Velocity (km/h)	Lateral Acceleration(g)	Yaw Rate (°)	Steering Wheel (°)	Accel Pad (%)	Brake Pad (%)
기본 값	[0.0:255.0]	[-100.0:100.0]	[-1:1]	[-1.0:1.0]	[0.0:1.0]	[0.0:1.0]

(그림 7. 시뮬레이터 센서 데이터 값 범위)

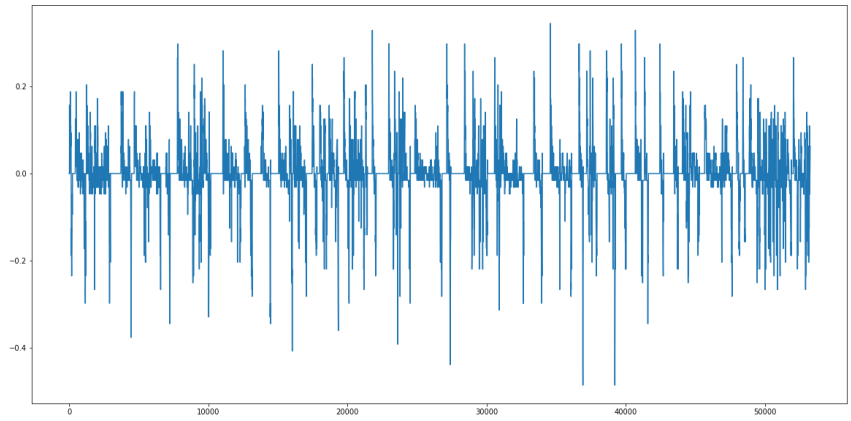
데이터의 범위를 센서별로 정리해본 결과 범위가 나왔지만 값의 편차가 큰 것을 확인할 수 있었다(그림 7).

(2) 실제 차량

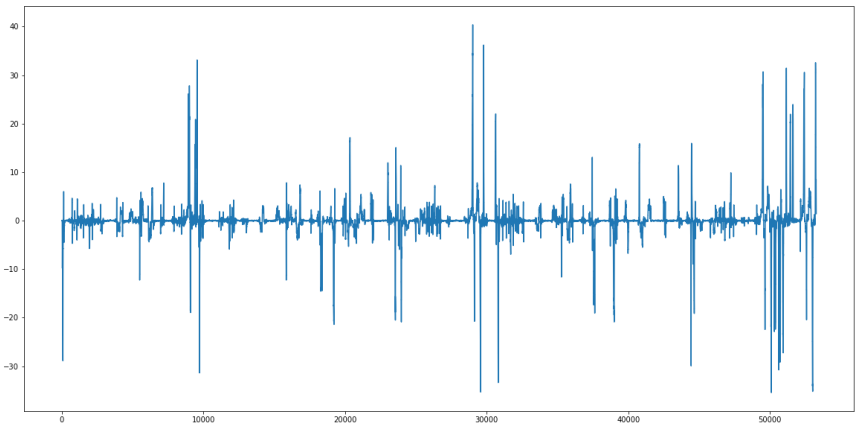
실제 차량에서는 조향 토크 센서, 브레이크 페달 위치 센서, 톤 휠 센서, 가속 페달 위치 센서, 좌우 가속도 센서, 각속도 센서의 데이터를 사용하였다.



(그림 8. 핸들 데이터 값 그래프)



(그림 9. 좌우 가속도 값 그래프)



(그림 10. 각속도(yaw rate) 값 그래프)

실제 데이터를 살펴보면 실제 데이터의 범위를 대략적으로 확인이 가능했다. 이를 기반으로 시뮬레이터 데이터의 정규화를 진행할 계획이다. (그림 9, 10, 11)

센서 종류	SWA (°)	Brake (%)	Speed (km/h)	Accelerator Position (%)	Lateral Acc. (g)	Yaw Rate (°)
센서 값	[-540:540]	[0:255]	[0:255]	[0:100]	[-1.99:2]	[-128:128]

(그림 11. 실제 센서 데이터 값 범위)

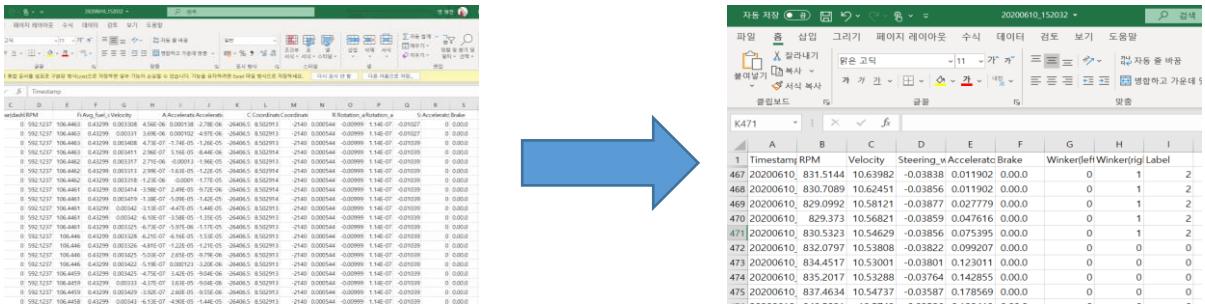
센서 데이터의 범위를 정리하면 그림 11과 같이 표로 정리할 수 있었으며 이를 기반으로 1차 정규화를 진행하게 된다.

3.1.3. 데이터 수집

7월, 8월 약 2개월에 걸쳐 약 40만 개의 데이터를 수집 및 선별하였고, 수집된 데이터는 학습모델에 원활한 적용을 위해 데이터 수정작업을 거친 다음 게임 내의 녹화 영상을 사용해서 상기한 차량 주행 상태에 맞도록 라벨링 작업을 진행하였다.

이 과정에서 확인한 내용 중 하나는 시뮬레이터에서는 데이터가 30~32Hz로 출력되고 실제 차량에서는 데이터를 10Hz로 출력해서 데이터의 개수가 달랐다. 이는 데이터 변환 과정에서 학습용 데이터는 시뮬레이터 데이터를 3개 단위로 1개씩 뽑아내는 방법으로 10Hz데이터 3개로 나누도록 하여 해결하였다.

라벨은 [0: 직진, 1: 좌커브, 2: 우커브, 3: 좌회전, 4: 우회전, 5: 좌차선, 6: 우차선, 7: 정지]의 8개의 라벨로 정해서 각 row가 어떤 상태의 센서데이터 인지를 Label column을 추가하여 표기하였다.



(그림 12. 데이터 선별 및 라벨링)

3.1.4. 데이터 분석

수정된 데이터들은 실제 차량 데이터와 유사하게 1차 정규화를 적용하였다. 정규화를 위해 각각의 데이터 간의 차이점을 확인하였고, 주로 시뮬레이션 데이터를 수정하는 방향으로 진행되었다.

ALL	Velocity	Accel_X	Rotate_Z	Steering	Accel	Brake
Average	56.07	0.00	-0.07	0.32	18.44	2.08
MAX	115.00	1.99	44.99	365.00	100.00	100.00
95%	93.00	0.52	6.54	79.00	60.00	19.00
90%	86.00	0.28	3.85	24.00	50.00	1.00
85%	82.00	0.16	2.57	15.00	44.00	0.00
Q1(75%)	74.00	0.06	1.19	8.00	33.00	0.00
MEDIAN	58.00	0.00	0.00	0.00	11.00	0.00
Q3(25%)	41.00	-0.06	-1.26	-8.00	0.00	0.00
15%	32.00	-0.16	-2.77	-16.00	0.00	0.00
10%	23.00	-0.28	-4.08	-27.00	0.00	0.00
5%	0.00	-0.50	-6.89	-83.00	0.00	0.00
MIN	0.00	-1.99	-44.99	-239.00	0.00	0.00

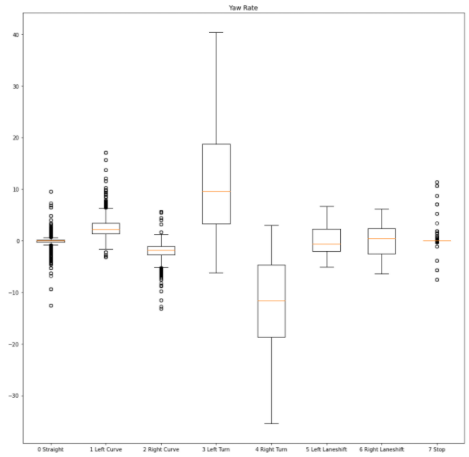
ALL	Velocity	lateral acc	yaw rate	Steering	Accel	Brake
Average	28.35	0.00	0.03	1.23	5.30	5.94
MAX	72.00	0.35	40.38	381.00	54.12	96.00
95%	63.00	0.11	3.25	13.00	23.53	22.00
90%	58.00	0.06	1.56	5.00	19.22	20.00
85%	56.00	0.05	0.56	3.00	16.08	17.00
Q1(75%)	49.00	0.02	0.13	1.00	10.20	13.00
MEDIAN	32.00	0.00	0.00	0.00	0.00	0.00
Q3(25%)	0.00	-0.02	-0.19	0.00	0.00	0.00
15%	0.00	-0.02	-0.50	-1.00	0.00	0.00
10%	0.00	-0.03	-1.06	-5.00	0.00	0.00
5%	0.00	-0.14	-2.75	-13.00	0.00	0.00
MIN	0.00	-0.49	-35.44	-380.00	0.00	0.00

(그림 13. 시뮬레이터 데이터와 실제 차량데이터의 분포)

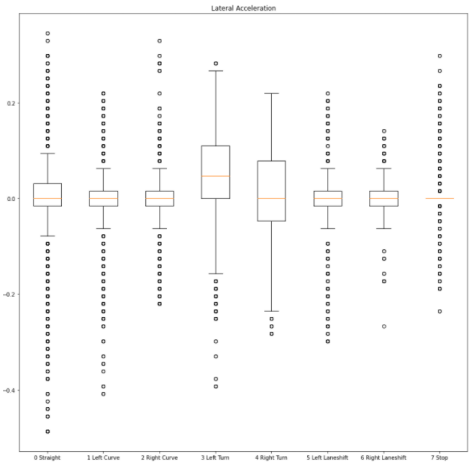
학습용 시뮬레이션 데이터와 실제 차량 데이터를 Lateral Acceleration이 약 4배, Yaw Rate는 2배정도 차이가 나는 것을 확인할 수 있었다(그림 13).

3.1.5. 데이터 정규화

다음은 시뮬레이션 데이터의 정규화를 위해 실제 차량 센서데이터의 Yaw Rate와 Steering Lateral 수치를 종합하여 시각화 한 것이다(그림 14, 15).



(그림 14. 실제 차량 센서데이터 Yaw Rate 범위)



(그림 15. 실제 차량 센서데이터 Lateral Acceleration 범위)

센서종류	Velocity (km/h)	Lateral Acceleration(g)	Yaw Rate (°)	Steering Wheel (°)	Accel Pad (%)	Brake Pad (%)
기본 값	[0.0:255.0]	[-100.0:100.0]	[-1:1]	[-1.0:1.0]	[0.0:1.0]	[0.0:1.0]
1차 정규화(추출)	[0:255]	[-100.0:100.0]	[-1000:1000]	[-450:450]	[0:100]	[0:100]
범위 제한	[0:120]	[-2.0:2.0]	[-45.0:45.0]	[-270:270]	[0:80]	[0:50]
2차 정규화(모델)	[0:120]	[-0.5:0.5]	[-22.5:22.5]	[-270:270]	[0:80]	[0:50]

*Yaw Rate, Steering Wheel: 실제와 시뮬레이터 값의 부호가 반대

(그림 16. 시뮬레이터 데이터 정규화 과정)

데이터 정규화는 총 3번에 걸쳐 진행이 되는데 이 과정이 모두 모델에 데이터가 학습되기 전에 이루어진다(그림 16).

먼저 앞에서 분석한 데이터의 값 분포와 값 범위를 통해 추출되는 데이터의 값을 분석(그림 14,15), 1차로 데이터 추출 및 변환 프로그램에서 실제데이터와 비슷한 값을 출력하도록 값을 증폭한다.

이후 분석한 실제 데이터와의 차이를 통해 비정상적인 값을 가려내기 위해 데이터 값의 범위를 제한한다.

마지막으로 분석한 실제 데이터의 분포를 확인해서 시뮬레이터 데이터를 실제 데이터의 분포와 비슷하도록 변환하여 정규화를 마무리하고 이 결과를 모델에 학습시킨다.

3.2. 분석 모델링

상기한 데이터 학습과 실제 차량 데이터 테스트를 위해 딥러닝 모델을 제작하였다. 먼저 간단한 DNN모델과 CNN모델을 모두 설계해서 테스트를 해보았고 가장 괜찮았던 CNN모델을 채택하게 되었다.

3.2.1. 최종 CNN 모델

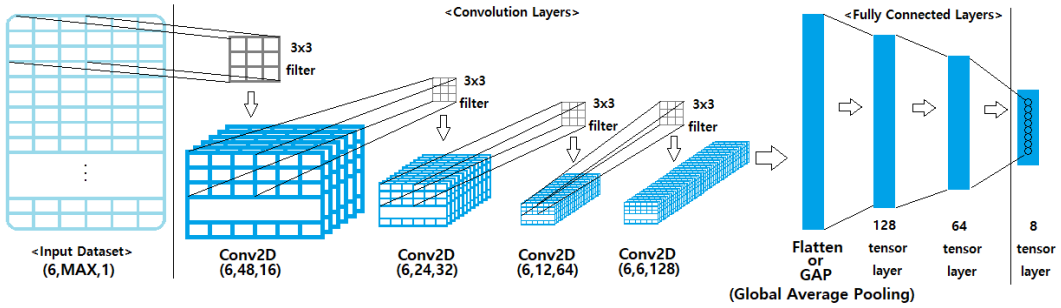
모델은 Python을 사용하여 Google Colab에서 제작되었다. 해당 모델에는 CNN-GAP와 CNN-Flatten 두가지의 모델이 병렬적으로 사용되었다(그림 17,18,19).

```
cnn_model_main = keras.Sequential([
    keras.layers.Conv2D(16,3,padding='same',activation='relu',input_shape = (6,WINDOW_SIZE,1)),
    keras.layers.Conv2D(32,3,strides=(1,2),padding='same',activation='relu'),
    keras.layers.Conv2D(32,3,padding='same',activation='relu'),
    keras.layers.Conv2D(64,3,strides=(1,2),padding='same',activation='relu'),
    keras.layers.Conv2D(64,3,padding='same',activation='relu'),
    keras.layers.Conv2D(128,3,strides=(1,2),padding='same',activation='relu'),
    keras.layers.Conv2D(128,3,activation='relu',padding='same'),
    keras.layers.Dropout(0.1),
    keras.layers.GlobalAveragePooling2D(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(8, activation='softmax')  #stop 빼는 경우
])
```

(그림 17. CNN-GAP 모델)

```
cnn_model_sub = keras.Sequential([
    keras.layers.Conv2D(16,3,padding='same',activation='relu',input_shape = (6,WINDOW_SIZE,1)),
    keras.layers.Conv2D(32,3,strides=(1,2),padding='same',activation='relu'),
    keras.layers.Conv2D(32,3,padding='same',activation='relu'),
    keras.layers.Conv2D(64,3,strides=(1,2),padding='same',activation='relu'),
    keras.layers.Conv2D(64,3,padding='same',activation='relu'),
    keras.layers.Conv2D(128,3,strides=(1,2),padding='same',activation='relu'),
    keras.layers.Conv2D(128,3,activation='relu',padding='same'),
    keras.layers.Dropout(0.2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(8, activation='softmax')  #stop 빼는 경우
])
```

(그림 18. CNN-Flatten 모델)



(그림 19. CNN 모델 구조)

센서 데이터는 numpy.nparray형으로 총 6개의 column의 데이터를 가지고 있고 데이터의 WINDOW SIZE는 4.8초 분량인 48개를 한 개의 segment로 하며 Convolution 2D Layer를 추가하는데 필터의 개수를 16,32,64,128개로 하며 same옵션으로 차원의 수가 유지되도록 하며 Stride (1,2) 옵션을 통해 column개수는 고정하고 Segment의 개수를 절반씩 줄이도록 하였다. Convolution Layer의 마지막에는 10%의 확률로 dropout을 하는 layer를 추가하고 모든 값을 1차원 신경층에 담는 Flatten layer와 각 filter의 평균 값을 1차원 신경층으로 만드는 Global Average Pooling Layer로 가지는 두개의 모델을 만들었으며 1차원 행렬을 각각 128, 64개의 tensor를 가지는 Neural Layer를 통과하고 마지막 8개의 상태를 나타내는 Layer에 도착함으로써 8가지 상태를 표현하게 설계하였다.

```
[[95  0  0  0  1  0  0  0]
 [29 61  0  4  0  3  0  0]
 [29  0 46  0 20  1  1  0]
 [ 8  0  0 87  2  0  0  0]
 [ 1  0  0  0 97  0  0  0]
 [27  8  0  4  5 53  0  0]
 [23  0 12  0  7  1 54  0]
 [ 1  0  0  0  8  0  0 89]]

[[96  0  0  0  0  0  0  0]
 [32 57  0  4  0  5  0  0]
 [34  0 41  0 20  1  1  0]
 [10  1  0 86  0  0  0  2]
 [ 6  0  0  0 93  0  0  0]
 [29  3  0  4  0 60  0  0]
 [23  0 10  0  7  1 56  0]
 [ 1  0  0  0  0  0  0 98]]

[[96  0  0  0  1  0  0  0]
 [46 43  0  3  0  6  0  0]
 [42  0 34  0 18  1  2  0]
 [16  0  0 79  2  0  0  0]
 [ 1  0  0  0 97  0  0  0]
 [37  6  0  1  5 47  0  0]
 [38  0  4  0  4  0 52  0]
 [ 1  0  0  0  8  0  0 89]]
```

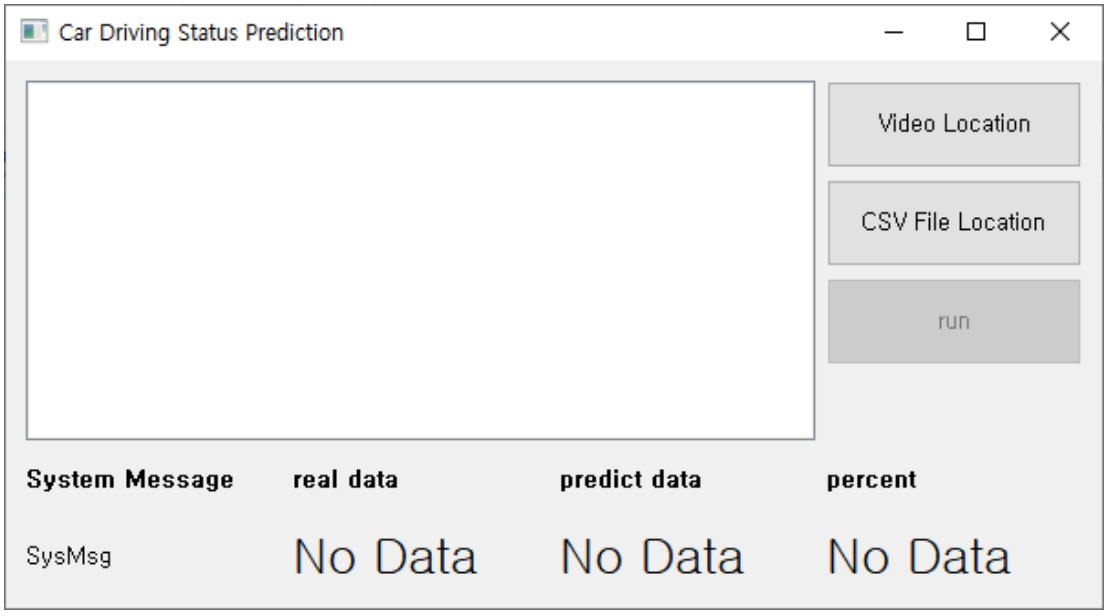
(그림 20 Sim2Real 라벨 별 예측 정확도(좌: Merge, 중간: GAP, 우: Flatten))

각 Model의 Predict를 진행하고 모델의 Predict 정확도를 확인하였다(그림 20). 정확도를 기반으로 두 모델에서 공통적으로 예측한 결과와 0,5,6,7의 상태인 직진, 좌/우차선, 정지 상태는 GAP Layer를 가지는 모델에서 1,2,3,4의 상태인 좌/우커브, 좌/우회전 상태는 Flatten Layer를 가지는 모델에서 가지고 와서 두 모델의 결과를 merge하여 최종 결과로써 출력한다.

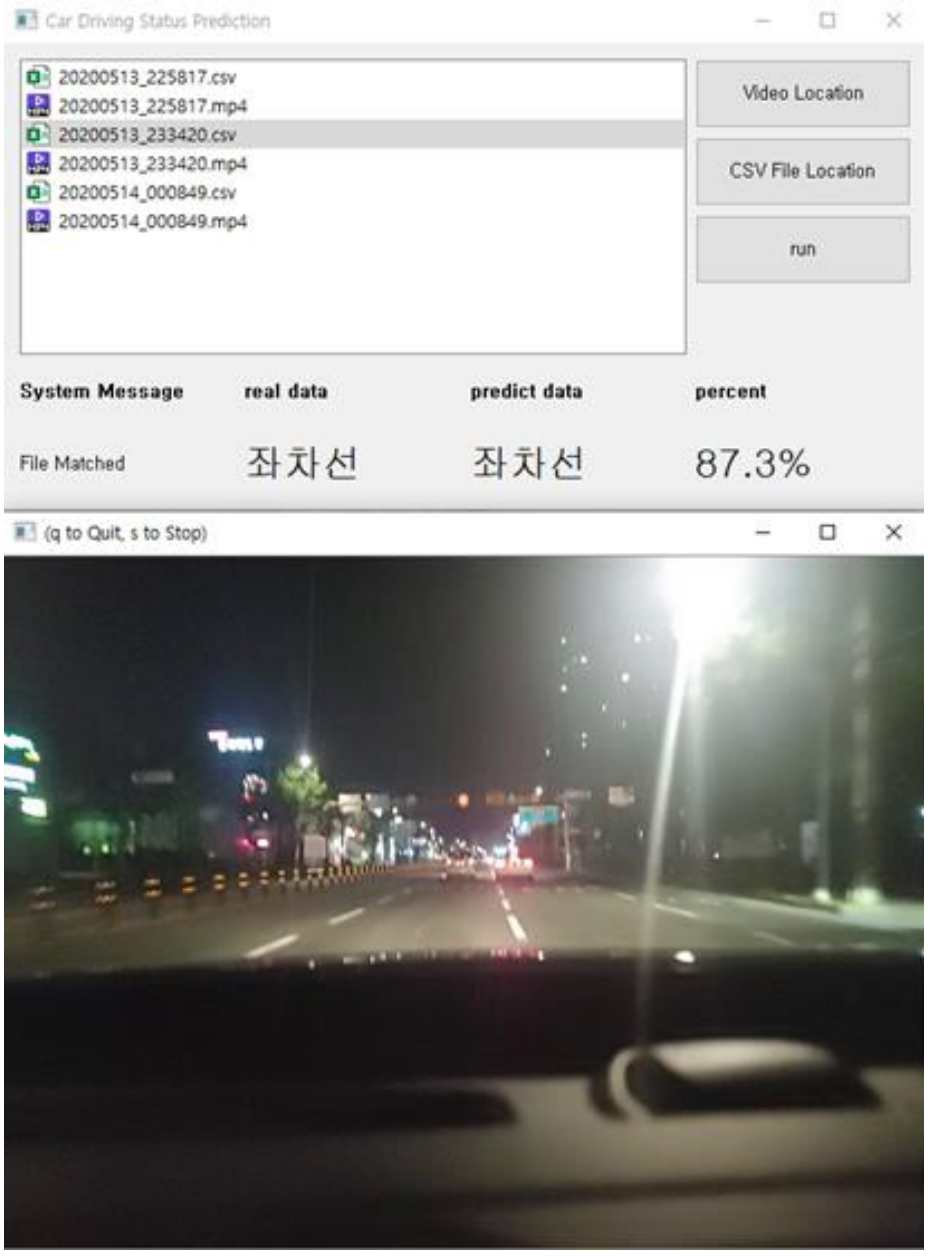
3.3. UI 제작

마지막으로 상기한 모델을 좀 더 직관적이고 편하게 사용하기위해 GUI기반의 프로그램을 제작하였다. UI 프로그램은 간단하게 예측 결과와 실제 결과가 얼마나 일치하는지를 직관적으로 보여주기 위해 제작되었다. 여기에는 PyQt5 모듈을 사용했으며 tensorflow의 keras 모듈을 import하여 load model을 통해 학습된 CNN모델을 이식하였다. 실제 라벨링 데이터와 예측된 라벨링 데이터 간의 정확도를 나타내고, 라벨 데이터의 column을 영상과 매칭시켜서 영상의 재생에 따른 해당 영상의 장면에서의 운전 데이터를 연속적으로 나타내어준다.

3.3.1. UI 구성 요소



(그림 21. UI 프로그램 이미지)



(그림 22. UI 프로그램 실행 화면)

UI가 제공해야 할 기본적인 부분을 포함한다. 영상과 csv파일에 대한 경로를 설정할 수 있고, 해당 경로에서 파일을 선택하여 해당 파일에 대한 영상 파일을 매칭시킨다. 선택된 파일을 내부에 이식된 모델을 통해 분석 및 예측을 실행하고, 실제 라벨 데이터와 얼마나 일치하는지를 정확도를 통해 나타낸다. 분석이 완료되면 영상을 재생하고 해당 영상 시간에 해당하는 라벨 데이터를 연속적으로 프로그램 상에 나타낸다.

3.3.2. UI 기능

기본적인 UI의 틀은 pyqt5를 통해서 제작되었다. QGridLayout을 통해 격자식으로 각 버튼과 label을 배치하였고, 각 버튼이 클릭되면 연결된 함수가 실행된다. 영상에 대한 파일의 경로를 설정하는 버튼과 csv파일에 대한 경로를 설정하는 버튼을 통해서 디렉토리 경로를 설정하고 따로 저장한다. csv파일에 대한 경로가 설정되면 list에 해당 경로에 존재하는 파일을 모두 나타내고 csv파일이 클릭되면 설정된 영상 경로 내에 해당 파일과 같은 이름의 파일이 존재하는지 확인하고 존재한다면 run 버튼을 활성화한다.

run버튼을 클릭하게 되면 csv파일 경로로부터 pandas의 read_csv를 통해서 파일을 읽어오게 되고 해당 데이터의 가장 첫번째 column을 확인한다. Simulation data와 real data의 차이는 Timestamp 데이터의 여부인데, simulation data는 첫번째 column에 Timestamp값을 가진다. 이를 통해서 simulation data와 real data를 구분한다. Simulation data의 경우에는 값의 범위를 제한해 줄 필요가 있고, 정규화 과정이 필요하게 된다. 또한 real data에서는 쓰이지 않는 몇 개의 column에 대해서도 처리해 줄 필요가 있다.

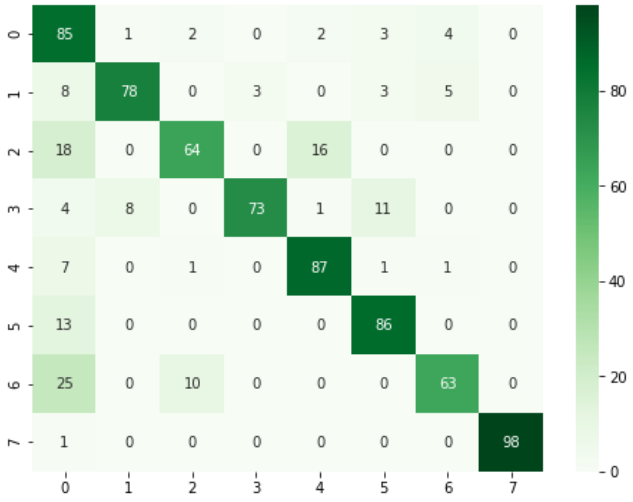
위와 같은 전 처리과정이 완료되면 keras의 load_model을 통해서 load된 모델을 통해서 데이터의 예측을 수행한다. 예측된 데이터는 list형식으로 저장되며, 해당 list와 실제 label 데이터를 비교해서 정확도를 확인한다.

이후 opencv모듈을 통해서 영상의 경로로부터 영상을 불러와서 재생할 준비를 한다. 영상의 처리는 frame을 읽어서 연속적으로 출력해 주는 방식으로 차량 센서 데이터에 대한 값은 영상의 3 frame당 하나의 데이터를 가지는데, 영상과 데이터를 프레임에 맞도록 영상이 3 frame 재생된 다음 label데이터를 갱신해준다. 이때 simulation data의 영상과 real data의 영상은 해상도가 다르므로 이에 대한 구분을 해준다.

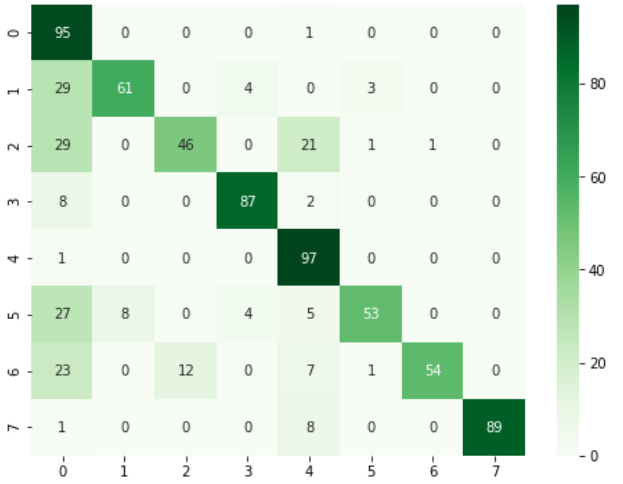
Opencv의 기능을 통해 특정 key를 입력 받으면 특정 동작을 하도록 구현해 두었다. Q를 입력하면 영상을 중지하고 나가게 된다. S를 입력하게 되면 영상을 일시정지 할 수 다. 이후 아무 key나 입력하면 다시 영상이 재생된다.

4. 연구 결과 분석 및 평가

본 프로젝트의 모델의 각 데이터에 대한 예측 결과는 다음과 같다.



(그림 23 Sim2Sim 테스트 결과)



(그림 24. Sim2Real 테스트 결과)

테스트 정확도의 경우 Sim2Sim 과 Sim2Real 양쪽 모두 커브와 차선 변경에서 낮은 정확도를 보였는데 이는 회전에 비해 핸들링 수치나 차량 속도의 변화량이 그리 크지 않기 때문이라 생각된다.

기본적으로 예측이 잘 안되는 경우는 센서 데이터 값이 비슷한 경우인 직진-커브, 커브-차선변경의 경우, 도로의 지면 상태의 불량으로 인한 순간 데이터 값 변화에 의한 잘못된 예측을 하는 경우가 존재했다. 차량 동작을 인식하는 데에 0.5 초 내외의 오차가 존재했다.

1	Pred_Direction	REAL_Direction	1	Pred_Direction	REAL_Direction
464	0	0	527	3	3
465	0	0	528	3	3
466	0	0	529	3	3
467	0	0	530	4	3
468	0	0	531	4	3
469	3	0	532	4	3
470	3	0	533	4	3
471	3	0	534	3	3
472	3	0	535	0	3
473	3	3	536	0	3
474	3	3	537	0	3
475	3	3	538	0	3
476	3	3	539	0	3
477	3	3	540	0	0
478	3	3	541	0	0
479	3	3	542	0	0
480	3	3	543	0	0
481	3	3	544	0	0
482	3	3	545	0	0
483	3	3	546	0	0
484	3	3	547	0	0

(그림 25. 차량 동작 변화 때의 빠른 예측)

좌회전과 우회전과 같이 비교적 변화량이 큰 동작의 경우 동작의 도입부와 끝부분에서 0.5 초의 예측 오류가 있다(그림 25). 이 부분은 차량 상태를 어떻게 판단하는 지에 따라 맞을 수도 있고 틀릴 수도 있는 경우이라 생각한다.

1	Pred_Direction	REAL_Direction	1	Pred_Direction	REAL_Direction
1232	0	0	17023	0	6
1233	0	0	17024	0	6
1234	1	0	17025	0	6
1235	1	5	17026	2	6
1236	1	5	17027	2	6
1237	5	5	17028	2	6
1238	1	5	17029	2	6
1239	1	5	17030	0	6
1240	1	5	17031	0	6
1241	1	5	17032	2	6
1242	1	5	17033	2	6
1243	1	5	17034	2	6
1244	1	5	17035	2	6
1245	5	5	17036	2	6
1246	5	5	17037	2	6
1247	5	5	17038	2	6
1248	1	5	17039	0	6
1249	5	5	17040	0	6
1250	5	5	17041	0	6
1251	5	5	17042	6	6
1252	5	5	17043	6	6

(그림 26. 차선 변경 때 비슷한 데이터 혼동)

좌측 차선변경과 우측 차선변경의 경우 데이터의 변화량이 크지가 않아서 비슷한 데이터인 직진, 커브의 경우와 혼동되기 쉽다(그림 26). 이는 커브와 직진의 경우에서도 확인 가능한 경우이다.

1	Velocity	Accel_X	Rotate_Z	Steering_w	Accelerate	Brake	Pred_Direction	REAL_Direction
1595	55	-0.01569	-0.6875	0	0	0	0	0
1596	55	-0.01569	0.3125	0	0	0	0	0
1597	54	-0.01569	0.3125	0	0	0	0	0
1598	54	-0.01569	0.125	-2	0	0	0	0
1599	54	-0.01569	0.125	-2	0	0	5	0
1600	54	-0.01569	0.125	-2	0	0	5	0
1601	54	-0.01569	1.75	-5	0	0	5	0
1602	54	-0.01569	1.75	-5	0	0	5	0
1603	54	-0.01569	1.9375	-6	8.235294	0	5	0
1604	54	-0.01569	1.9375	-6	8.235294	0	0	0
1605	54	0	2.1875	-5	4.705883	0	5	0
1606	54	0	2.1875	-5	4.705883	0	0	0
1607	54	0	1.125	-2	0	0	5	0
1608	54	0	1.125	-2	0	0	0	0
1609	54	0	1.125	-2	0	0	5	0
1610	54	0	0.625	0	0	0	0	0
1611	54	0	0.625	0	0	0	0	0
1612	54	0	0.1875	0	0	0	5	0
1613	54	0	0.1875	0	0	0	5	0
1614	54	-0.01569	0.1875	0	0	0	5	0
1615	54	-0.01569	0.1875	0	0	0	5	0
1616	54	-0.01569	0.8125	0	0	0	5	0
1617	54	-0.01569	0.8125	0	0	0	5	0
1618	54	-0.01569	0.8125	0	0	0	0	0
1619	54	-0.01569	0.5625	0	0	0	0	0
1620	54	-0.01569	0.5625	0	0	0	0	0
1621	54	-0.01569	-0.1875	0	0	0	5	0
1622	54	-0.01569	-0.1875	0	0	0	0	0
1623	54	-0.03137	-0.0625	0	0	0	0	0
1624	54	-0.03137	-0.0625	0	0	0	0	0
1625	54	-0.01569	0.375	0	1.960784	0	5	0

(그림 27. 센서 데이터의 미세한 변화로 인한 혼동)

예측 상황에 직진을 차선변경으로 혼동한 이유가 차량 균형을 위해서 조향 핸들을 조작해서 센서 값의 변화가 발생했기 때문에 이런 혼동이 발생했다고 예측할 수 있다.

5. 팀원 별 역할 분담

(1) 정희석

역할: 딥러닝 모델 설계

- 데이터 정규화 진행
- Google Colab과 Tensorflow을 사용해서 딥러닝 모델 설계 & 개량

(2) 이석준

역할: UI 개발

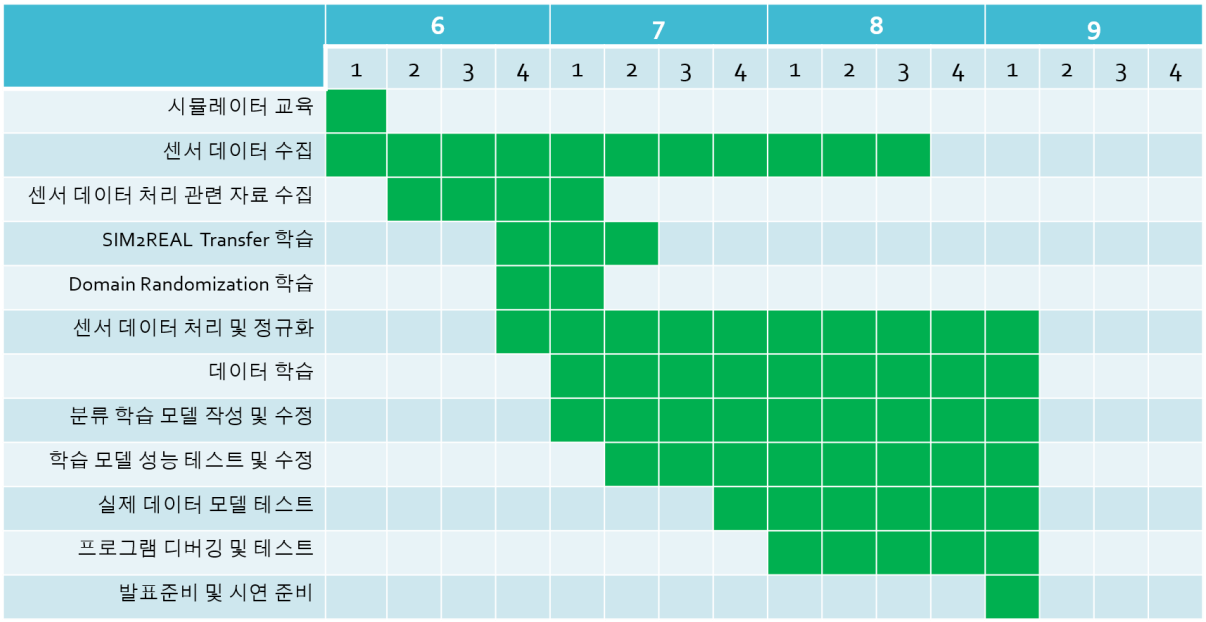
- Python의 PyQt5 모듈을 사용하여 GUI 프로그램 개발
- 시연 프로그램에 프로젝트 예측 모델 이식

(3) 방형진

역할: 학습 데이터 처리

- 시뮬레이션 프로그램에서 획득한 센서데이터를 CSV파일로 변환하는 프로그램 수정
- 센서 데이터 특징 분석

6. 개발 일정



7. 결론 및 향후 연구 방향

본 프로젝트에서 우리는 상기한 바와 같이 실제 데이터를 사용하지 않고도 시뮬레이션 데이터를 통하여 의미 있는 수준의 결과를 이끌어 낼 수 있음을 보일 수 있었다. 이를 통해 차량 주행 예측 모델이 서론에서 언급한 교통사고시의 과실 여부를 알아내거나 해당 데이터를 기반으로 안전 주행 점수를 매겨 운전자 보험료나 과태료를 측정할 수 있을 것이다. 향후 연구로는 실제 차량을 통해 데이터를 뽑기 힘든 교통 사고나 음주운전 같은 데이터를 시뮬레이션을 통해 안전하게 수집하고 이를 해당 기능을 위한 모델 학습에 이용할 수 있도록 하는 방향으로 심화하여 연구할 수 있을 것이다.

8. 참고 문헌

[1] Ministry of Land, Infrastructure and Transport (2020, July Updated), 2020/07 Total Registered Motor Vehicles.xlsx Available: http://stat.molit.go.kr/portal/cate/statFileView.do?hRsId=58&hFormId=5 (downloaded 2020, Sep. 07)

[2] Taegu Kim, Beomjun Kim, Yongsu Jeon, Jaebong Lim, Hyunwook Lee, Yunju Baek (2020). Design and Implementation of the Vehicle State Recognition On-Device Deep Learning System for Real-Time Driver Behavior Analysis. Proceedings of Symposium of the Korean Institute of communications and Information Sciences, 2020.2, 538- 539(2 pages)

[3] Wikipedia (2020, Sep 02 Updated), 2020/07 On-board diagnostics Available: https://en.wikipedia.org/wiki/On-board_diagnostics (downloaded 2020, Sep. 07)