**IoT시스템설계및실습 : 팀 프로젝트 보고서**

**라즈베리파이를 이용한 자율주행 모델학습**



**4조**

**2020204005 정필규**

**2020204017 고유진**

**2020204037 박희령**

**2020204040 최현준**

**목차**

**<서론>**

**1. 프로젝트 개요**

- 프로젝트의 목적과 배경

- 자율주행 자동차의 중요성 및 활용 분야

**2. 프로젝트 목표 및 범위**

- 프로젝트의 구체적인 목표 - - 프로젝트의 범위 및 한계

**<본론>**

**1. 데이터 수집**

1. 데이터 수집 방법론

- 곡선 위주 트랙

- 혼합 트랙

- 정지선&횡단보도 촬영

ㄴ) 데이터셋 구성

ㄷ) 데이터 전처리 과정

**2. 모델 검토 및 선택 과정**

ㄱ) 기존 모델 검토

- 사전 배포된 NVIDIA 모델 개요

- 해당 모델의 장단점 분석

ㄴ) EfficientNet 모델

**-** EfficientNet 모델 사용 도입 배경

- EfficientNet 모델의 성능 평가 및 한계

**3. MobileNet 모델 선택**

- MobileNet 모델 선택 이유

- MobileNet 모델의 장단점 분석

**4. MobileNet 모델 설계 및 학습**

- MobileNet 모델 구조

- 데이터셋을 활용한 모델 설계 및 학습 과정

- 가중치 파일 생성 과정 및 결과

**5. 자율주행 시스템 구현**

- 자율주행 알고리즘 개요

- 라즈베리파이에 모델 배포 방법

- RC카-라즈베리파이 간의 통신 방법

**6. 횡단보도 및 정지선 인식**

- 학습 데이터 수집 및 처리 방법

- 인식 알고리즘 설계

**7. 테스트 및 결과 분석**

- 자율주행 테스트 방법

- 테스트 결과

- 성공 사례 및 실패 사례 분석

- 문제 해결 방법

**<결론>**

**1. 프로젝트 성과 요약**

- 주요 성과 및 결과 요약

- 자율주행 모델의 성능 평가

**2. 프로젝트의 한계 및 개선 방안**

- 한계점 및 문제점 분석

- 향후 개선 방안 및 발전 가능성

**3. 결론**

- 최종적인 성과 및 의의

- 조원 기여도 평가

**<서론>**

**<서론>**

**1. 프로젝트 개요**

**- 프로젝트의 목적과 배경**

자율주행 자동차는 교통사고 감소, 교통체증 완화, 효율적인 운송수단 등의 이유로 중요한 기술로 부각되고 있다. 본 프로젝트는 자율주행 자동차의 중요성과 이를 다양한 분야에서 활용할 수 있는 가능성을 탐구하는 데 목적이 있다.

**- 자율주행 자동차의 중요성 및 활용 분야**

자율주행 기술은 물류, 대중교통, 개인 차량 등의 다양한 분야에서 활용될 수 있으며, 교통사고를 줄이고, 노약자 및 장애인의 이동성을 향상시키며, 도시 교통 문제를 해결하는 데 중요한 역할을 할 수 있다.

**2. 프로젝트 목표 및 범위**

라즈베리파이로 인공지능 자율 주행 자동차를 만드는 프로젝트이다. 64GB의 메모리 성능을 가졌기에 간단한 모델과 데이터셋이 포화 되지 않으면서 RC카를 자율주행 하도록 해야 한다.

**- 프로젝트의 구체적인 목표**

자율주행을 위한 데이터 수집

적절한 모델 선택 및 학습

자율주행 시스템 구현 및 테스트

**- 프로젝트의 범위 및 한계**

프로젝트의 범위는 데이터 수집부터 모델 학습으로부터 나온 압축된 데이터를 통해, 한계점으로는 데이터 수집의 다양성 부족, 모델의 성능 한계가 있으며 실제 측정 시에는 low-fidelity로 검은색 도화지에 흰색 절연 테이프로 임의의 트랙으로 테스트한다는 점에서 조건의 제한 등이 있다.

**<본론>**

**<본론>**

**1. 데이터 수집**

데이터 수집에 있어서 방향키로 조작하여 다양한 트랙을 다각도로 돌아본다. 이 때 프레임단위로 촬영하여 쌓인 데이터셋으로 학습모델을 선정하여 가중치 파일을 생성한다. 이 때 threshold 값은 150으로 속도는 좌/우회전 직진 동일하게 40으로 주었다.

1. **데이터 수집 방법론**

아래의 트랙대로 주행하여 데이터셋을 쌓을 때 안쪽 경로, 중앙 경로, 바깥 경로로 최대한 다양성을 주며 학습시켰다 .

**- 곡선 위주 트랙**: 곡선 주행 데이터를 수집하여 다양한 주행 상황을 학습한다.

**- 혼합 트랙**: 직선과 곡선이 혼합된 트랙에서 데이터를 수집하여 모델의 적응성을 높인다.

**- 정지선 & 횡단보도 촬영**: 정지선과 횡단보도를 인식하기 위한 데이터를 수집한다. Crosswalk 와 no-corosswalk 분류 한다.

**ㄴ) 데이터셋 구성**

수집된 데이터에서 우선적으로 조향각(streaing angles) 분포를 확인한다. 이 때 찍힌 해당 Angle에 따른 수치를 파악한다. 텍스트, 스크린샷, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명. 라인, 그래프, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**ㄷ) 데이터 전처리 과정**

데이터셋 수집 과정에 있어서 수만장의 이미지파일을 저장하고 학습 시킨다. 데이터의 과대적합이 되지 않도록 train데이터, valid데이터, test데이터로 split한다. 최종적으로 8:1:1 비율로 데이터를 나눈다.

다음 과정에서 train데이터와 valid데이터를 반복학습을 통해 순전파-역전파 과정을 거쳐 손실값을 설정해둔 epoch 횟수만큼 반복한 후 선정된 모델의 최종 성능을 평가하기 위한 test loss 값을 산출할 것이다. 스크린샷, 그래프, 라인, 도표이(가) 표시된 사진

자동 생성된 설명

이 결과로 최적의 파라미터를 산출하기 위해 다음 모델들을 검토하였다.

**2. 모델 검토 및 선택 과정**

**ㄱ) 기존 모델 검토**

**- 사전 배포된 NVIDIA 모델 개요**

NVIDIA 모델은 regression model로 합성곱 CNN 모델을 통해 연속적인 값을 예측하는데 사용된다. 특히 이 프로젝트에 대해서는 연속적인 조향각 분포를 기반으로 예측한다. 입력데이터 즉 threshold값을 통해 트랙 레인만 보이게끔 촬영된 이미지로 조향각 분포를 본다.

손실함수(MSELoss)는 평균 제곱 오차 손실 함수로 예측값과 실제 값 사이의 차이를 제곱하여 평균을 구함으로써 회귀문제에 적합하다. 최종적으로 Adam 옵티마이저를 사용하여 모델 파라미터를 학습한다. 특히 Adam은 학습률을 동적으로 조절하여 효율적인 학습을 가능하게 한다.

텍스트, 스크린샷, 디자인, 영수증이(가) 표시된 사진

자동 생성된 설명

**- 해당 모델의 장단점 분석**

NVIDIA 모델을 사용했을 때 합성곱 층은 이미지의 공간적 특성을 잘 활용하기에 라즈베리파이로 구현하는 자율주행으로서는 큰 장점을 지니고 있다. 특히 배포 모델에는 드롭아웃 층이 사이에 껴있어서 파라미터 수를 신경망보다 적게하여 일반화 성능을 높여줬다. 또한 ELU 활성화 함수 과정을 통해 학습속도가 개선되는 등 다방면에서 장점을 보였다.

하지만 중간 과정에서 많은 기법이 들어간만큼 시간 소요가 컸다. 비용적인 문제도 CoLab에서 반복학습을 시키는 과정에서 많은 컴퓨팅을 소모하여 추가결제를 하는 등의 제약이 있었다. 모델적인 면에서는 좋았으나 간단한 프로젝트로 진행되기에 배포된 NVIDIA 모델이 아닌 타 모델을 기용하여 진행하기로 하였다.

**ㄴ) EfficientNet 모델**

**- EfficientNet 모델 사용 도입 배경**

최초에 EfficientNet 모델을 통해 학습모델을 구축하였다. CNN기반의 모델로 이미지 분류에 최적화된 모델이라 판단하여 기용하였다. 특히 EfficientNet이 이번 프로젝트에 있어서 강점이라고 생각했던 이유 중 하나가 해상도였다. 이미지의 해상도를 증가시킬 때마다 EfficientNet의 주력인 합성곱 계층과 풀링 계층에서 주변 이미지에 대한 패턴과 최대 풀링, 평균 풀링으로 인해 직각과 곡선에 강점을 보일 것으로 예측했다.

**- EfficientNet 모델의 성능 평가 및 한계**

EfficientNet은 효율성과 정확성을 동시에 추구하는 신경망 모델로서, 다양한 크기와 복잡도의 모델을 통해 폭넓은 애플리케이션에 적용할 수 있다는 것이 특징이었다. 그러나 높은 연산 자원 요구 및 복잡한 튜닝 과정 때문에 매번 학습 모델 코드를 수정하는 데 있어서 소요 시간이 누적되는 한계가 있었다. 성능 결과는 좋으나 과정에서 가성비가 좋지 못하다고 판단하였다.

이러한 한계는 라즈베리파이 CPU 및 RAM 사양에 맞게 모델 기용에 대한 차선책을 고려하는 것이 필요했다.

**3. MobileNet 모델 선택**

**- MobileNet 모델 선택 이유**

MobileNet 모델은 경량화된 구조로 이뤄져있다. 특히 NVIDIA 모델은 회귀모델이기에 많은 과정이 내포 되어있다. 그 과정을 줄여주고 분류모델을 사용하며 중간 과정의 채널을 줄여준다는 점이 결정적이었다. 특히 여러 번의 데이터셋 수집과 학습 모델의 많은 반복을 위해서 프로젝트에 있어서 가장 적합한 모델이라고 생각되어 선정하였다.

**- MobileNet 모델의 장단점 분석**

MobileNet 모델의 장점은 경량화와 높은 효율성, 단점은 복잡한 환경에서의 성능 한계입니다. 계산 비용 및 시간적인 면에서 EfficientNet을 뒤로하고 MobileNet을 기용하게 되었다.

MobileNetV2: Inverted Residuals and Linear Bottlenecks

**4. MobileNet 모델 설계 및 학습**

**- MobileNet 모델 구조**

MobileNet은 경량화된 딥러닝 모델로, 주로 모바일 및 임베디드 장치에서의 효율적인 딥러닝 처리를 위해 설계되었습니다. MobileNet의 주요 구조적 특징은 다음과 같습니다.

* **Depthwise Separable Convolutions**: MobileNet의 핵심은 depthwise separable convolution이다. 이는 일반적인 convolution을 두 단계로 나누어 연산량을 줄이는 방식이다.
* **Width Multiplier**: MobileNet은 네트워크 너비를 조절할 수 있는 하이퍼파라미터 α를 도입하여, 모델의 크기와 연산량을 조절한다. α가 작을수록 모델이 더 작아지고 연산량이 감소한다.
* **Resolution Multiplier**: 입력 이미지의 해상도를 조절하는 하이퍼파라미터 ρ를 통해, 모델의 연산량과 메모리 사용량을 더욱 조절할 수 있다.

**- 데이터셋을 활용한 모델 설계 및 학습 과정**

**self.mobilenet = mobilenet\_v2(pretrained=True):**

사전 학습된 MobileNetV2 모델을 로드한다. 이 모델은 ImageNet 데이터셋에서 학습된 가중치를 사용한다.

**self.fc1 = nn.Linear(1000, 256 ) ~:**

MobileNetV2의 출력을 받아서 256개의 노드로 연결하는 완전 연결층(fully connected layer)을 정의합니다. 총 5층으로 이루어져 있다. 여기서 기존 모델은 여러 개의 convolutional layer로 이미지 분류를 한 후 특징을 추출하지만 계산 효율성과 성능 향상을 위해 분류기를 제거하고 특징 추출기만을 사용하였다. 이미지를 1000차원의 벡터로 평탄화한 후 5차원의 Linear 레이어만을 이용하여 최종적으로 차량의 조향각을 예측할 수 있도록 하였다.

**forward 메서드:**

입력 이미지 텐서 x를 받아서 모델을 통해 순전파(forward pass)한다.

**features = self.mobilenet(x):** MobileNetV2 모델을 사용하여 입력 이미지의 특징을 추출한다.

**x = torch.relu(self.fc1(features)) ~ :**

추출된 특징을 첫 번째 완전 연결층에 전달하고, ReLU 활성화 함수를 적용한다. 마찬가지로 5층으로 이뤄져있고 마지막 층에서는 ReLU 활성화 함수를 사용하지 않고 바로 예측값을 반환한다.

**initialize\_model 함수**: 모델을 초기화하고, 옵티마이저를 설정한다.

**model = MobileNetModel().to(device):** MobileNetModel 인스턴스를 생성하고, 지정된 장치(예: GPU)로 모델을 이동시킨다.

**optimizer = optim.Adam(model.parameters(), lr=0.001, weight\_decay=0.001):**

Adam 옵티마이저를 사용하여 모델의 파라미터를 최적화한다. 학습률(lr)은 0.001로 설정하고, 가중치 감쇠(weight\_decay)를 0.001로 설정한다.

**criterion = nn.MSELoss() : 손실함수 정의**

criterion: 평균 제곱 오차(MSELoss)를 손실 함수로 정의한다. 이 손실 함수는 예측된 주행 각도와 실제 주행 각도 간의 차이를 제곱하여 평균을 계산한다.

**- 가중치 파일 생성 과정 및 결과**

Epoch 20/20 (Training): Loss=156

Train Loss: 143.2145,

Validation Loss: 189.9306

스크린샷, 그래프, 라인, 도표이(가) 표시된 사진

자동 생성된 설명

**5. 자율주행 시스템 구현**

**- 자율주행 알고리즘 개요**

**목적**: 원본 이미지를 모델의 입력으로 사용하기 위해 적절히 전처리한다.

**이미지 절반 사용**:

**height, \_, \_ = image.shape**

**image = image[int(height/2):,:,:]**

이미지의 상단 절반은 제거하고 하단 절반만 사용한다. 이는 주로 도로의 하단 부분에 차선이 위치하는 경우가 많고 상단 노이즈를 지우기 위해서다.

**색 공간 변환**:

**image = cv2.cvtColor(image, cv2.COLOR\_BGR2YUV)**

이미지를 BGR 색 공간에서 YUV 색 공간으로 변환했다. YUV 색 공간은 조명 변화에 덜 민감하여 차선 인식에 유리하다.

**크기 조정**:

**image = cv2.resize(image, (200,66))**

이미지를 (200, 66) 크기로 조정한다. 이는 모델의 입력 크기에 맞추기 위함이다.

**가우시안 블러**:

**image = cv2.GaussianBlur(image,(5,5),0)**

가우시안 블러를 적용하여 이미지의 노이즈를 줄였다. (커널 크기는 (5, 5))

**이진화**:

**\_,image = cv2.threshold(image,160,255,cv2.THRESH\_BINARY\_INV)**

이미지를 이진화하여 명확한 경계를 생성한다. 픽셀 값이 160 이상이면 흰색(255)으로, 그렇지 않으면 검정색(0)으로 설정한다. 이진화는 차선과 도로 배경을 구분하는 데 도움된다.

**결론**: 이 두 함수는 자율 주행 차량의 주행 제어 및 이미지 전처리 과정에서 핵심적인 역할을 한다. start\_thread 함수는 주행 상태를 제어하는 역할을 하고, img\_preprocess 함수는 원본 이미지를 모델이 이해할 수 있는 형식으로 전처리한다. 이러한 전처리 과정은 차선 인식의 정확성을 높이는 데 좋은 효과를 보였다.

**- 라즈베리파이에 모델 배포 방법**

사전에 라즈베리파이에 연결된 핫스팟과 host명 및 비밀번호를 저장하였다. 추후 나머지 팀원들의 아이디를 추가하였고 Samba를 통해 4명이 모두 라즈베리파이와 연결할 수 있도록 했다.

**- RC카-라즈베리파이 간의 통신 방법**

VNC를 통해 4명 모두 하나의 가상 프로세스 화면을 통해 라즈베리파이와 동시에 통신할 수 있게 했다.

**6. 횡단보도 및 정지선 인식**

**- 학습 데이터 수집 및 처리 방법**

정지선과 횡단보도를 사전에 제작하여 1만장에 가까운 이미지를 촬영하였다. 이를 따로 학습시켜 횡단보도가 보이면 ‘crosswalk’, 보이지 않으면 ‘no-crosswalk’ 인식하게 하였다.

**- 인식 알고리즘 설계**

사전 데이터 학습으로 ‘crosswalk’가 인식되었다면 자동차를 제어하는 전역변수를 조정하는 thread를 실행하였다. ‘time.sleep()’ 메소드를 통해 지정한 시간 동안 자동차가 정지하도록 전역변수를 조작하였고 이후에는 지정한 시간동안 직진만 하도록 전역변수를 조작하였다.결론적으로 crosswalk를 인식하면 지정한 시간동안 멈춘 후 crosswalk를 지날 때 까지 직진하도록 설계했다.

**7. 테스트 및 결과 분석**

**- 자율주행 테스트 방법**

테스트 당일인 6/7 15시에 공개된 세 트랙을 완주하는 것이다. 완주시간과 트랙 이탈 시간도 같이 체크한다. 1번 트랙은 직각 위주, 2번 트랙은 곡선 위주, 3번 트랙은 운동장형 트랙에서 정지선을 추가하여 정지선에 앞에서 3초간 멈춘 뒤 출발하는 것이 테스트 조건이다. 장소는 광운대학교 새빛관 101호에서 진행하였다.

**- 테스트 결과**

1번 트랙에서1분 중반대의 완주 기록에 비해 트랙 후반부에서 40초가량 트랙을 밟으면서 완주를 하였다. 전체 1/3 가량 통과한 트랙에서 완주시간도 준수했다.

2번 트랙도 실제 도로에서 본 따온 트랙으로 곡선으로 지속된 트랙이다. 사전 자율주행에서 곡선에 약점으로 보였지만 테스트 직전에 속도를 낮춰 안전하게 주행하였고 완주시간과 라인 이탈 시간도 준수했다.

3번 트랙은 일반적인 운동장형 트랙에 정지선과 횡단보도를 추가한 트랙으로 정지선 앞에서 3초간 멈춘 뒤 횡단보도를 통과해야 했다. 전체 팀 중에서도 완주한 두 팀으로 그 중에서도 정지선을 넘지 않고 멈춘 뒤 통과하였다.

**- 성공 사례 및 실패 사례 분석**

우리 팀은 세 트랙 모두 완주하였고 정지선 기준도 완벽하게 정지하였기에 우리팀을 성공 사례로 분석해보겠다. 우리 팀이 성공한 이유는 트랙을 완주하는 것을 핵심 목표로 둔 학습 데이터 확보였다. 자동차가 트랙 밖으로 나가지 않기 위해서는 다양한 각도의 곡선과 직진 데이터가 필요하다. 그중에서도 특히 다양한 경우의 수가 발생하는 곡선에서 양질의 학습 데이터 획득을 위해 노력했다. 실제로 곡선 학습만을 위해 새롭게 제작한 트랙이 3개가 넘었었다. 하지만 우리가 제작한 트랙에서 너무 많이 학습시키면 과적합 문제가 생기기에 학습 데이터의 다양성을 추구하되 너무 많이 학습시키지는 않았다. 그 결과 모든 트랙을 완주하는 모델을 만들 수 있었다.

실패한 팀들의 원인은 크게 두가지로 나눌 수 있다. 첫번째 원인은 모델의 과적합이다. 실패한 팀 중 하나는 12만장의 이미지로 모델을 학습을 시켰다. 해당 팀은 트랙을 잘 돌기로 유명한 팀이었고 실제로 자체 제작한 모든 트랙을 완주할 정도로 좋은 성능을 보였다. 하지만 평가에서는 과적합으로 학습되지 않은 테스트 트랙은 돌지 못하였다.

두번째 원인은 학습데이터의 문제이다. 양질의 학습 데이터는 좌회전, 직진, 우회전의 비율이 비슷해야 한다. 그러기 위해서는 학습 트랙을 설계할 때 직선보다 곡선의 비중이 높아야 한다. 왜냐하면 직진데이터는 좌회전, 우회전을 하는 중간중간에도 쌓이기 때문이다. 추가로 자동차가 어디서 주로 탈선하는 지 생각해봐도 곡선에서 다양한 케이스의 데이터를 확보해야 하는 것은 당연하다. 트랙을 완주하지 못한 팀들은 이런 점들을 고려하지 않은 것으로 판단되고 실제로 성능이 좋지 않은 팀들의 학습 트랙들은 대체로 직선코스의 비중이 높았다.

**- 문제 해결 방법**

우리 팀의 문제점은 트랙의 라인을 밟은 시간이 꽤 존재했다는 것이다. 그 이유는 학습 데이터를 만들 때 자동차가 탈선하지 않게 하기 위해 넣은 아웃코스 데이터의 비중이 높아서 그런 것 같다. 이는 학습 데이터를 만들 때 아웃라인 인코스 주행 데이터의 비율을 줄이고 중앙 주행 데이터의 비율을 늘리면 해결 될 것으로 보인다.

또 다른 문제로는 카메라 노이즈 제어 문제가 있다. 해당 문제로 인해 테스트 3번 트랙의 접힌 부분이 Threshold한 이미지에서 트랙의 라인으로 인식되어 탈선할 뻔하기도 하였다. 이 문제를 해결하기 위한 간단한 방법으로는 빛의 영향을 최소화 하는 필터적용, Adaptive한 Threshold적용, Histogram Equalization이 있고 나아가 Homomorphic 필터링, Retinex 알고리즘 등을 적용할 수 있다.

**<결론>**

**<결론>**

**1. 프로젝트 성과 요약**

**- 주요 성과 및 결과 요약**

모든 트랙을 완주하였고 직각 위주의 트랙인 1번에서는 라인을 밟은 시간이 완주시간의 절반정도 되었다. 2번 트랙에서는 거의 완벽한 주행을 하였고 3번 트랙 또한 마찬가지다.

**- 자율주행 모델의 성능 평가**

이번 테스트를 통해 팀의 MobileNet기반 모델은 트랙을 이탈하지 않았기에 좋은 성능을 보인다 판단할 수 있다. 하지만 라인을 밟는 경우가 많은 것은 고쳐야할 부분이다.

**2. 프로젝트의 한계 및 개선 방안**

**- 한계점 및 문제점 분석**

여실히 드러난 문제점은 학습 데이터였다. 어떤 데이터를 쌓는지에 따라 트랙별 완주도가 달랐기 때문이다. 주행 시 도로 폭이 넓은 트랙도 있을 수 있기에 사전 통제 변인이 주어졌음에도 안심치 못했고 여러 도로에서 학습시켰고 카메라 인식 속도가 있었기에 속도 조절도 필요한 것이 한계점이었다.

**- 향후 개선 방안 및 발전 가능성**

다양한 도로 환경과 조건에서 데이터를 수집하고 데이터 증강 기법을 활용하여 모델의 적응력을 높이며, 고급 네트워크 구조 탐색과 하이퍼파라미터 튜닝을 통해 모델 성능을 최적화하고 싶다. 또한 여건이 된다면 더 좋은 하드웨어에서 GPU/NPU가속을 활용하여 실시간 처리 성능을 향상시켜보고 싶다. 추가로 적응형 제어 알고리즘과 강화 학습을 도입하여 주행 전략을 개선하고, 추후에 발전시킨다면 자율 주차 시스템 자율주행에 완벽을 가하는 방향으로 확장할 수 있을 것 같다.

**3. 결론**

**- 최종적인 성과 및 의의**

최종적으로 평가에서 모든 트랙을 완주한 두개의 팀 중 한 팀이 되었다. 이번 프로젝트를 통해 딥러닝 과정에 대해서 깊이 이해할 수 있었으며, MobileNet 모델을 활용하여 자율 주행 시스템을 성공적으로 구현할 수 있었다. 특히, MobileNet 모델의 경량화된 구조 덕분에 제한된 하드웨어 자원에서도 높은 성능을 발휘할 수 있었고, 실시간 이미지 처리 및 주행 제어를 효율적으로 수행할 수 있었다. 이를 통해 딥러닝 기술이 실제 응용 분야에서 어떻게 활용될 수 있는지 체감할 수 있었으며, 자율 주행 기술의 가능성과 한계에 대해서도 많은 학습을 할 수 있었다. 이번 프로젝트의 성과는 향후 더 발전된 자율 주행 시스템 개발의 기초가 될 것이며, 딥러닝 기술의 응용 가능성을 더욱 확장할 수 있는 계기가 될 것이다.

**- 조원 기여도 평가**

|  |  |  |  |
| --- | --- | --- | --- |
| 학번 | 이름 | 내용 | 기여도 |
| 2020204005 | 정필규 | MobileNet 모델로 데이터셋 학습 | 25 |
| 2020204017 | 고유진 | 데이터셋 학습, 모의 트랙 제작 | 25 |
| 2020204037 | 박희령 | 데이터셋 학습, 주행모델 코드 제작 | 25 |
| 2020204040 | 최현준 | 데이터셋 축적, 보고서 작성 | 25 |