IoT 시스템 설계 및 실습

라즈베리파이를 이용한 자율주행 자동차 프로젝트 보고서

3조
문혜진: 2022204020
박제인: 2021204068
손아현: 2022204062
최수현: 2022204041

목차

Ⅰ. 초록

본 보고서는 라즈베리 파이를 활용한 자율주행 자동차 프로젝트에 대해 다룬다. 본 프로젝트의 주요 목표는 자율주행 자동차가 다양한 트랙에서 안정적으로 주행할 수 있도록 하는 것이다. 이를 위해 학습 데이터의 취득, 모델 학습, 주행 테스트 및 성능 평가를 포함한 다양한 기술적 요소를 활용하였다. 본 보고서에서는 학습 데이터 취득 방법, 사용한 학습 모델 및 학습 방법, 학습 및 주행 결과에 대한 분석, 조원의 역할 등을 상세히 기술하였다.

학습 데이터 취득 방법은 프로젝트의 성공을 위해 필수적인 첫 단계였다. 자율주행 자동차에 장착된 카메라와 다양한 센서를 사용하여 데이터를 수집하였다. 수집된 데이터는 트랙의 다양한 구간에서 자동차의 위치, 방향, 속도 등을 포함하였으며, 이를 바탕으로 모델을 학습시켰다. 데이터 수집 과정에서는 특히 곡선 구간, 직선 구간, 교차로 등 다양한 환경을 반영하여 수집 범위를 넓혔다.

학습 모델로는 MobileNet 모델을 사용하였다. MobileNet은 경량화된 신경망 구조로, 딥러닝 분야에서 널리 사용되는 모델이다. 이 모델은 Depthwise Separable Convolutions를 사용해 연산량을 줄이면서도 높은 성능을 유지할 수 있도록 설계되었으며, 자율주행 자동차의 조향각 예측에 적합한 성능을 보였다. 학습 과정에서는 수집된 데이터를 전처리하고, 적절한 하이퍼파라미터를 조정하여 모델의 정확도를 최대화하였다. 학습은 GPU를 활용한 데스크탑 환경에서 진행되었으며, 학습이 완료된 모델은 라즈베리 파이에 배포되어 실시간 주행에 사용되었다.

학습 및 주행 결과는 전반적으로 긍정적이었다. 학습된 MobileNet 모델은 트랙의 대부분 구간에서 안정적인 주행 성능을 보여주었다. 훈련 과정에서 주요 지표인 Mean Squared Error (MSE) 손실이 점진적으로 감소하여 모델의 안정성을 확인할 수 있었다. 조기 종료 콜백을 통해 22번째 에폭(Epoch)에서 훈련이 종료되었으며, 최종 검증 손실은 58.0534로 나타났다. 훈련 데이터셋에 대한 손실 값은 지속적으로 감소하였고, 검증 데이터셋에 대한 최종 손실 값도 58.0534로 나타났다. 그러나 일부 복잡한 곡선 구간에서는 예측 오차가 발생하여 트랙의 경계선을 밟는 경우도 있었다. 이는 향후 개선의 여지가 있는 부분으로, 추가적인 데이터 수집과 모델 개선을 통해 극복할 수 있을 것으로 기대된다.

조원의 역할에 대해서도 상세히 기술하였다. 조장은 프로젝트의 총괄 및 일정 관리를 담당하였고, 데이터 엔지니어는 데이터 수집 및 라벨링, 전처리를 책임졌다. 모델 엔지니어는 딥러닝 모델의 설계와 학습, 평가를 맡았으며, 하드웨어 엔지니어는 라즈베리 파이의 설정 및 자율주행 자동차의 조립과 테스트를 진행하였다. 테스트 엔지니어는 모델 배포 및 실시간 주행 테스트를 수행하고, 주행 결과를 분석하여 개선 방안을 제시하였다.

본 프로젝트를 통해 라즈베리 파이를 활용한 자율주행 자동차 제작과 주행 테스트를 성공적으로 수행할 수 있었다. 프로젝트 수행 과정에서 얻은 경험과 지식을 바탕으로 자율주행 기술의 기초적인 구현과 실제 환경에서의 테스트를 통해 자율주행

시스템의 가능성과 한계를 탐구하였다. 향후 추가적인 데이터 수집과 모델 개선을 통해 더 안정적이고 효율적인 자율주행 시스템을 개발할 수 있을 것으로 기대된다. 각 조원의 역할 분담과 협업을 통해 프로젝트를 원활히 진행할 수 있었으며, 이를 통해 얻은 경험과 지식을 바탕으로 더욱 발전된 프로젝트를 수행할 수 있을 것이다.

Ⅱ. 프로젝트 개요

2.1 프로젝트 수행 단계

프로젝트 수행을 위해 다음과 같이 8단계를 거쳤다.

2.1.1 자율주행 자동차 제작

자율주행 자동차의 제작은 프로젝트의 첫 번째 단계로, 하드웨어와 소프트웨어의 조합이 필요하였다. 주요 작업은 다음과 같다.

- 하드웨어 구성: 라즈베리 파이를 중심으로 카메라 모듈, 초음파 센서, 모터 컨트롤러 및 배터리 등의 부품을 조립하였다. 하드웨어 엔지니어는 각 부품의 연결과 작동을 확인하여 기본적인 자율주행 기능을 구현하였다.
- 소프트웨어 설치: 라즈베리 파이에 Raspbian OS를 설치하고, 필요한 라이브러리와 패키지 (OpenCV, TensorFlow 등)를 설치하였다. 하드웨어와 소프트웨어의 통합을 통해 기본적인 주행 및 데이터 수집 기능을 확보하였다.

2.1.2 자료조사

프로젝트의 방향성과 기술적 접근 방식을 정하기 위해 다양한 자료조사를 수행하였다. 주요 조사 내용은 다음과 같다:

- 논문 조사 : 최신 자율주행 기술과 관련된 학술 논문을 검토하여, 자율주행 알고리즘과 학습 모델의 최신 동향을 파악하였다.
- GitHub 및 블로그 : 자율주행 관련 오픈소스 프로젝트를 참고하여 실무에서 사용되는 코드와 구현 방법을 학습하였다.
- 책 : 딥러닝과 자율주행에 관련된 책을 통해 이론적 배경을 보강하였다.

2.1.3 모델 결정

조사한 자료를 바탕으로 최적의 학습 모델을 선택하였다. 다양한 모델 중에서 MobileNet 모델을 선정하였다. 선택 이유는 다음과 같다:

- 성능: MobileNet은 Depthwise Separable Convolutions를 사용하여 연산량을 줄이면서도 높은 성능을 유지하며, 실시간으로 많은 연산을 처리해야 하는 상황에 효율적인 작동 가능하다. 또한 이미지 인식 작업 성능이 뛰어나다.
- 적용성: MobileNet은 경량화된 신경망 구조로, 이미지 인식 작업에서 뛰어난 성능을 보여, 자율주행 자동차의 조향각 예측에 적합하다.

2.1.4 학습 모델 코드 작성

모델 엔지니어는 자율주행자동차의 차선 인식 및 횡단보도 정지선 감지를 위한 MobileNet 및 MobileNetV2 기반의 학습 모델 코드를 작성하였다. 주요 작업은 다음과 같다.

① 차선 인식 모델

- 데이터 준비: 두 개의 ZIP 파일에서 차선 인식 데이터를 추출하고 병합하였다. 손상된 이미지를 식별하여 제거하고, 이미지 경로와 조향 각도를 추출하여 데이터프레임을 생성하였다.
- 이미지 전처리: 수집된 이미지 데이터를 모델에 맞게 전처리하였다. 이미지 크기를 `(70, 220)`으로 조정하고, 정규화하여 학습 효율을 높였다.
- 모델 구성: MobileNet 모델을 정의하고, 손실 함수로 MSE를, 최적화 알고리즘으로 Adam Optimizer를 사용하여 모델을 구성하였다.
- 학습 및 검증: 전처리된 데이터를 이용하여 모델을 10 에포크 동안 학습하고, 검증 데이터셋을 통해 모델의 성능을 평가하였다. 최종 모델은 `mobilenet_model.h5` 파일로 저장되었다.

② 횡단보도 정지선 감지 모델

- 데이터 준비: ZIP 파일에서 정지선 감지 데이터를 추출하고 병합하였다. 손상된 이미지를 제거하고, 이미지 경로와 `stopline` 여부를 라벨링하여 데이터프레임을 생성하였다.
- 이미지 전처리: 수집된 이미지 데이터를 전처리하여 모델에 맞게 변환하였다. 이미지 크기를 조정하고 정규화하였다.
- 모델 구성: MobileNetV2 모델을 정의하고, 손실 함수로 categorical crossentropy를, 최적화 알고리즘으로 Adam Optimizer를 사용하여 모델을 구성하였다.
- 학습 및 검증: 전처리된 데이터를 이용하여 모델을 학습하고, 테스트 데이터셋을 통해 모델의 성능을 평가하였다. 최종 모델은 `stopline_detection_model.h5` 파일로 저장되었다.

2.1.5 트랙 제작

모델의 성능을 테스트하기 위한 트랙을 제작하였다. 다양한 주행 상황을 반영할 수 있도록 곡선 구간, 직선 구간, 교차로 등을 포함하였다. 트랙 제작 시 고려한 사항은 다음과 같다:

- 다양한 주행 조건: 트랙의 복잡성을 높여 다양한 주행 조건에서 모델의 성능을 평가할 수 있도록 하였다.
- 안전성: 테스트 중 발생할 수 있는 사고를 방지하기 위해 다른 조원들의 경우 트랙밖에서 주행 중인 자동차를 주시하여 즉각적으로 반응할 수 있게 준비하였다.

2.1.6 학습 데이터 수집

수집된 데이터를 바탕으로 자율주행 모델을 학습시켰다. 주요 단계는 다음과 같다:

- 데이터 수집: 수동으로 자동차를 조종하여 트랙을 주행하면서 카메라와 센서를 통해 데이터를 수집하였다.
- 라벨링: 수집된 데이터에 조향각 라벨을 추가하여 학습에 필요한 형태로 전처리하였다.
- 데이터 보강: 다양한 주행 조건을 반영하기 위해 조명, 날씨 등의 변화를 시뮬레이션하여 데이터를 보강하였다.

2.1.7 배포 및 주행 테스트

학습된 모델을 라즈베리 파이에 배포하고 실시간 주행 테스트를 수행하였다. 주요 작업은 다음과 같다:

- 모델 배포 : 학습이 완료된 MobileNet 모델을 라즈베리 파이에 배포하여 실시간 주행이 가능하도록 설정하였다.
- 실시간 테스트 : 트랙에서 자율주행 테스트를 수행하고, 주행 성능을 평가하였다. 주행 결과를 분석하여 성능을 측정하였다.

2.1.8 결과 분석 및 반복 작업

초기 주행 결과가 기대에 미치지 못하는 경우, 문제를 분석하고 개선 작업을 반복하였다. 주요 과정은 다음과 같다:

- 결과 분석 : 주행 결과를 분석하여 모델의 오차를 파악하고, 원인을 규명하였다.
- 모델 코드 수정 : 분석 결과를 바탕으로 학습 모델 코드를 수정하고, 필요한 경우 하이퍼파라미터를 조정하였다.
- 추가 데이터 수집 : 부족한 데이터를 보강하기 위해 새로운 데이터 수집을 수행하였다.
- 새로운 트랙 제작 : 성능이 부족한 구간에 대해 새로운 트랙을 제작하여,

모델이 다양한 조건을 학습할 수 있도록 하였다.
- 반복 학습 및 테스트 : 수정된 모델을 다시 학습시키고, 새로운 트랙에서 주행 테스트를 반복하였다.

이와 같은 과정을 통해 지속적으로 모델을 개선하여, 최종적으로 자율주행 자동차가 다양한 트랙에서 안정적으로 주행할 수 있도록 하였다.

2.2 학습 데이터 취득 방법

학습 데이터 취득은 자율주행 자동차 프로젝트의 핵심 단계로, 다양한 주행 조건을 반영한 데이터를 수집하는 것이 중요하다. 본 프로젝트에서는 다양한 형태의 트랙에서 데이터를 수집하여 모델 학습에 사용하였다. 특히, 모든 트랙에 대해 트랙의 정중앙에서 출발하는 경우와 정중앙이 아닌 곳에서 출발하는 경우를 모두 포함하였으며, 출발 방향과 반대 방향으로 출발하는 경우도 학습하였다. 또한, 정지선과 횡단보도를 추가하여 횡단보도를 인식하고 정지하도록 학습하였다. 학습 장소는 새빛관 1층, 참빛관 지하, 야외에서 진행되었는데, 이는 빛에 의한 방해를 받는 환경을 미리 대비하기 위함이다.

2.2.1 트랙의 정중앙(중심)에서 출발하여 직사각형 트랙 학습

직사각형 트랙에서 자율주행 자동차를 트랙의 정가운데에서 출발시켜 데이터를 수집하였다. 이는 차량이 이상적인 경로를 따라 주행하는 경우를 학습시키기 위함이다. 직사각형 트랙의 각 변에서 데이터를 수집하여 다양한 주행 조건을 반영하였다.

2.2.2 트랙의 정중앙이 아닌 주변에서 출발하게 하여 직사각형 트랙 학습

트랙의 주변에서 출발하여 직사각형 트랙을 주행하는 데이터를 수집하였다. 이는 자동차가 최적의 경로를 찾아가도록 학습시키기 위함이다. 출발 위치를 변형하여 수집된 데이터는 모델이 다양한 출발 조건에서도 안정적으로 주행할 수 있도록 돕는다.

2.2.3 트랙의 정중앙과 정중앙이 아닌 곳에서 출발하여 타원 트랙 학습

타원형 트랙에서 트랙의 중심과 중심이 아닌 곳에서 출발하여 주행 데이터를 수집하였다. 곡선 주행 시 차량의 안정성을 높이고, 곡선에서의 주행 경로를 학습시키기 위해 타원형 트랙을 선택하였다.

2.2.4 트랙의 정중앙과 정중앙이 아닌 곳에서 출발하여 D자 트랙 학습

D자 형태의 트랙에서도 중심과 비중심 위치에서 출발하여 데이터를 수집하였다. 이는 복합적인 곡선과 직선 구간을 포함하는 D자 트랙에서 주행 안정성을 높이기 위함이다.

2.2.5 트랙의 정중앙과 정중앙이 아닌 곳에서 출발하여 Z자 트랙 학습

Z자 형태의 트랙에서 다양한 출발 위치에서 데이터를 수집하였다. Z자 트랙은 급격한 방향 전환이 필요한 구간이 많아, 이러한 구간에서의 주행 능력을 향상시키기 위해 학습 데이터를 수집하였다.

2.2.6 트랙의 정중앙과 정중앙이 아닌 곳에서 출발하여 S자 트랙 학습

S자 형태의 트랙에서 중심과 비중심 위치에서 출발하여 데이터를 수집하였다. S자 트랙은 연속된 곡선 주행을 포함하고 있어, 곡선 주행 시 안정성을 높이는 학습 데이터를 제공하였다.

2.2.7 트랙의 정중앙과 정중앙이 아닌 곳에서 출발하여 직각, 곡선 등의 코스가 섞인 트랙 학습

직각과 곡선 구간이 혼합된 복합 트랙에서 다양한 출발 위치에서 데이터를 수집하였다. 이러한 트랙은 현실 세계의 복잡한 도로 상황을 반영하여, 모델이 다양한 주행 조건을 학습할 수 있도록 한다.

2.2.8 트랙의 정중앙과 정중앙이 아닌 곳에서 출발하여 직각, 곡선 등의 코스가 섞였는데 각이 다른 트랙 학습

각도가 다른 직각과 곡선 구간이 혼합된 트랙에서 중심과 비중심 위치에서 출발하여 데이터를 수집하였다. 이는 모델이 다양한 각도에서의 주행 경로를 학습하도록 돕는다.

2.2.9 모든 트랙에 테이프 사용하여 정지선과 횡단보도 만들어 학습

모든 트랙에 테이프를 사용하여 정지선과 횡단보도를 만들어 학습 데이터를 수집하였다. 자율주행 자동차가 횡단보도를 인식하고 정지할 수 있도록 학습시키기 위함이다. 정지선과 횡단보도에서의 데이터를 수집하여, 자율주행 자동차가 교차로와 보행자 보호 구간에서 안전하게 정지하도록 하였다.

이와 같은 다양한 트랙 조건과 출발 위치에서 수집된 데이터는 모델이 다양한 주행 환경에서 안정적으로 작동할 수 있도록 학습하는 데 중요한 역할을 하였다.

2.3 사용한 학습 모델 및 학습 방법

2.3.1 MobileNetV2 모델 사용

MobileNet은 경량화된 딥러닝 모델로서, MobileNetV1, MobileNetV2, MobileNetV3 등 다양한 버전이 존재한다. 본 프로젝트에서는 MobileNetV2를 사용하였으며, 이 모델은 효율적인 연산과 높은 정확도를 동시에 제공하는 특징이 있다. MobileNetV2는 Depthwise Separable Convolution을 사용하여 연산량을 크게 줄였으며, 이로 인해 실시간 자율주행 시스템에 적합하다. 또한, Inverted Residual 구조를 도입하여 네트워크의 성능을 더욱 향상시켰다. 이 모델은 ImageNet 데이터셋에서

사전 학습된 가중치를 사용하여 초기화되었으며, Transfer Learning 기법을 통해 우리의 자율주행 데이터셋에 맞게 미세 조정되었다.

MobileNetV2의 핵심 아이디어 중 하나는 기존의 표준 합성곱 층을 두 단계로 분리하는 것이다. 첫 번째 단계는 각 채널에 대해 독립적으로 공간적 필터링을 수행하는 Depthwise Convolution이며, 두 번째 단계는 Pointwise Convolution으로 1x1 합성곱을 통해 채널 간의 결합을 수행한다. 이를 통해 모델의 복잡도를 크게 줄이면서도 성능 저하를 최소화할 수 있다.

또한, MobileNetV2는 Inverted Residuals와 Linear Bottlenecks라는 두 가지 중요한 개념을 도입하였다. Inverted Residual 구조에서는 입력과 출력의 차원을 확장한 후 축소하는 방식을 사용하여 정보의 손실을 방지하고, Linear Bottleneck 층은 활성화 함수 적용 후의 비선형 왜곡을 줄여준다.

모델 학습 과정에서는 Adam 옵티마이저를 사용하였고, 학습률은 0.001로 설정하였다. 학습 데이터는 자율주행 자동차의 카메라로부터 수집된 이미지들로 구성되었으며, 데이터 증강 기법을 적용하여 다양한 주행 환경에서도 견고한 성능을 발휘하도록 하였다. 데이터 증강 기법으로는 회전, 이동, 확대, 축소 등이 사용되었으며, 이를 통해 모델이 다양한 환경에서도 높은 성능을 유지할 수 있도록 하였다.

2.3.2 학습 코드

모델 학습

1) 필요한 라이브러리 import

- 이 섹션에서는 차선 인식을 위한 프로젝트에 필요한 다양한 라이브러리를 불러온다. 주요 라이브러리로는 os, fnmatch, Counter, zipfile, shutil, numpy, pandas, matplotlib.pyplot, PIL.Image, tensorflow 등이 있다.

```
import os
import fnmatch
from collections import Counter
import zipfile
import shutil
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
import pandas as pd

import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, GlobalAveragePooling2D
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.model_selection import train_test_split
```

2) 데이터 준비

이 단계에서는 차선 인식을 위한 데이터 파일을 압축 해제 후 데이터를 병합하였다. 데이터의 전처리 및 정리를 담당하며, 초기 1회만 실행하는 부분이다. 파일 압축 해제 및 데이터 병합 코드는 두 개의 ZIP 파일을 해제하고 데이터를 하나의 디렉토리에 병합하는 작업을 수행한다. 먼저, `resnet1.zip` 파일을 `./data` 폴더에 압축 해제한다. 다음으로, `resnet2.zip` 파일을 임시 디렉토리에 압축 해제한 후, 임시 디렉토리에서 압축 해제된 모든 파일을 대상 폴더(`./data/resnet1`)로 이동한다. 이동 과정에서 파일 이름이 중복될 경우, `_1`, `_2` 등 숫자를 붙여 이름을 변경하여 중복을 방지한다. 모든 파일 이동이 완료되면 임시 디렉토리를 삭제하고 작업 완료 메시지를 출력한다. 이 코드는 ZIP 파일의 데이터를 효율적으로 병합하여 관리할 수 있도록 한다.

- 파일 압축 해제

```
zip_file = zipfile.ZipFile('./resnet1.zip')
zip_file.extractall(path="./data")
```

- 다른 파일의 압축 해제 및 데이터 병합

```
# 압축 파일과 대상 디렉토리 경로 설정
zip_file_path = './resnet2.zip'#바꾸기
target_folder_path = './data/resnet1'

# 임시 디렉토리 생성
temp_folder_path = './dd'
os.makedirs(temp_folder_path, exist_ok=True)

# 압축 파일 풀기
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(temp_folder_path)

# 'Video_b3' 폴더 경로 설정
video_folder_path = os.path.join(temp_folder_path, 'resnet2')#바꾸기

# 'Video_b3' 폴더 내부의 파일들을 대상 디렉토리로 복사
for root, _, files in os.walk(video_folder_path):
    for file in files:
        source_file_path = os.path.join(root, file)
        target_file_path = os.path.join(target_folder_path, file)

        # 중복되는 파일이 있을 경우 이름 변경
        if os.path.exists(target_file_path):
            base, extension = os.path.splitext(file)
            counter = 1
            new_file_name = f"{base}_{counter}{extension}"
            new_target_file_path = os.path.join(target_folder_path, new_file_name)
            while os.path.exists(new_target_file_path):
                counter += 1
                new_file_name = f"{base}_{counter}{extension}"
                new_target_file_path = os.path.join(target_folder_path, new_file_name)
            target_file_path = new_target_file_path

        shutil.move(source_file_path, target_file_path)

# 임시 디렉토리 삭제
shutil.rmtree(temp_folder_path)

print("파일 병합이 완료되었습니다.")
```

- 아래 코드는 손상된 이미지를 삭제하는 작업을 수행한다. `./data/resnet1` 디렉토리 내의 모든 파일을 검사하며, 파일명이 ``.png``, ``.jpg``, ``.jpeg``로 끝나는 경우 이미지를 열어서 손상 여부를 확인한다. 이미지가 손상되었을 경우, 해당 파일을 삭제한다. 이 과정에서 이미지를 열다가 `IOError`나 `SyntaxError`가 발생하면, 해당 파일이 손상된 것으로 간주하고 삭제한다. 손상된 파일이 삭제되었음을 알리는 메시지를 출력한 후, 정상적인 파일은 그대로 둔다. 이 코드는 디렉토리 내 손상된 이미지 파일을 효율적으로 식별하고 제거하였다.

- 손상된 이미지 삭제

```
folder_path = './data/resnet1'

# 폴더 내의 모든 파일을 순회
for filename in os.listdir(folder_path):
    if filename.endswith(('.png', '.jpg', '.jpeg')):  # 이미지 파일 확장자를 여기에 추가
        try:
            img = Image.open(os.path.join(folder_path, filename))  # 이미지 열기 시도
            img.verify()  # 이미지가 손상되었는지 확인
        except (IOError, SyntaxError) as e:
            print(f'{filename} is corrupted and will be deleted.')  # 손상된 이미지 파일명 출력
            os.remove(os.path.join(folder_path, filename))  # 손상된 이미지 삭제
```

3) 데이터셋 로드 및 전처리

이 섹션에서는 준비된 데이터를 로드하고 필요한 전처리 작업을 수행한다. 이미지 데이터를 불러오고, 차선 인식을 위한 적절한 형식으로 데이터를 변환한다.

- 데이터의 분포 확인: `./data/resnet1` 폴더 내에 있는 이미지 파일들을 인덱스에 따라 분류하고 그 개수를 확인하는 작업을 수행한다. 먼저 `Counter` 객체를 사용하여 인덱스별 이미지 개수를 집계한다. 폴더 내의 모든 파일을 순회하며, 파일명이 ``.png``, ``.jpg``, ``.jpeg``로 끝나는 경우 해당 파일의 인덱스를 파일명에서 추출한다. 추출된 인덱스는 `filename[12:15]` 위치에 있으며, 이를 `index_counter`에 추가하여 개수를 증가시킨다. 마지막으로, 인덱스와 그에 해당하는 이미지 파일의 개수를 출력한다. 이 코드는 `090`

(go), `135` (right), `045` (left), `180` (stop) 인덱스를 기준으로 이미지 파일의 분포를 확인할 수 있게 한다.

- 아래 코드는 `./data/resnet1` 디렉토리 내의 이미지 파일들을 읽고, 이미지 경로와 해당 조향 각도를 리스트에 저장한다. 먼저 디렉토리 내의 파일 리스트를 얻고, 각 파일에 대해 `.png` 형식의 파일을 찾아 경로를 설정한다. 파일을 열어 성공적으로 로드되면, 해당 파일의 경로를 `image_paths` 리스트에 추가하고, 파일명에서 조향 각도를 추출하여 `steering_angles` 리스트에 추가한다. 만약 파일 열기에 실패하면 예외 처리를 통해 해당 파일을 무시한다. 이후, 조향 각도가 0이 아닌 이미지들만 필터링하여 리스트를 업데이트한다. 특정 인덱스의 이미지를 표시하고 경로와 각도를 출력한다. 마지막으로, 이미지 경로와 조향 각도를 포함하는 데이터프레임을 생성한다.

4) 이미지 전처리 함수

위 코드는 `preprocess_image`라는 함수로, 주어진 이미지 경로를 입력받아 이미지를 전처리한다. 함수는 `load_img`를 사용해 지정된 크기(70x220)로 이미지를 불러온 후, `img_to_array`를 사용해 이미지를 배열로 변환한다. 변환된 이미지를 255.0으로 나누어 정규화하여 [0, 1] 범위로 변환한 후, 전처리된 이미지를 반환한다.

```
def preprocess_image(img_path):
    img = load_img(img_path, target_size=(70, 220))
    img = img_to_array(img)
    img = img / 255.0  # 정규화하여 [0, 1] 범위로 변환
    return img
```

5) 학습/검증 데이터 분리

데이터프레임 `df`에서 이미지 경로와 각도를 배열로 변환하고 이를 학습용과 테스트용 데이터로 분할한다. `preprocess_image' 함수를 사용하여 `df["ImagePath"]`에 있는 각 경로의 이미지를 전처리하여 배열로 변환한 후, `numpy` 배열로 저장한다. `df["Angle"]`에 있는 각도 값을 `numpy` 배열로 변환한다. 그런 다음, `train_test_split` 함수를 사용하여 이미지를 학습용과 테스트용 데이터로 분할하며, 테스트 데이터 비율은 20%, 랜덤 시드는 42로 설정한다.

```
images = np.array([preprocess_image(path) for path in df['ImagePath']])
angles = np.array(df['Angle'])

X_train, X_val, y_train, y_val = train_test_split(images, angles, test_size=0.2, random_state=42)
```

6) 모델 선언

여기에서는 MobileNet 모델을 사용하여 차선 인식을 위한 신경망을 정의한다. TensorFlow와 Keras 라이브러리를 사용하여 모델을 구축하였다. 코드는 딥러닝 모델을 정의하고 학습을 시작한다. 먼저 `Sequential` 모델을 생성하고, 합성곱 레이어와 풀링 레이어로 구성된 여러 층을 추가한다. 첫 번째 층은 입력 모양이 `(70, 220, 3)`인 `Conv2D` 레이어로, 24개의 필터와 `(5, 5)`의 커널 크기를 갖는다. 다음으로, `MaxPooling2D` 레이어와 `Dropout` 레이어를 추가하여 과적합을 방지한다. 그런 다음, 또 다른 `Conv2D` 레이어와 풀링, 드롭아웃 레이어를 추가한다. 마지막으로, `Flatten` 레이어를 사용하여 1차원 배열로 변환한 후, 완전 연결층(`Dense` 레이어)을 추가한다. 모델을 컴파일할 때, `adam` 옵티마이저와 `mean_squared_error` 손실 함수를 사용한다. 학습 데이터(`X_train`, `y_train`)로 모델을 10 에포크 동안 학습시키며, 배치 크기는 32로 설정한다. 학습 과정은 `history` 객체에 저장된다.

7) 학습 및 손실 계산

모델의 조기 종료를 설정하고 학습을 수행한다. `EarlyStopping` 콜백을 사용하여 검증 손실(`val_loss`)이 개선되지 않으면 학습을 조기에 종료하도록 한다. `patience` 매개변수를 3으로 설정하여, 연속 3회 에포크 동안 개선이 없으면 종료하며, `restore_best_weights=True`로 설정하여 최상의 가중치를 복원한다. 모델을 `adam` 옵티마이저와 `mean_squared_error` 손실 함수를 사용해 컴파일한 후, 학습 데이터를 사용하여 최대 30 에포크 동안 학습한다. 이 과정에서 검증 데이터(`X_val`, `y_val`)를 사용하여 검증 손실을 모니터링한다. 배치 크기는 32로 설정하고, 조기 종료 콜백을 사용하여 학습을 진행한다. 학습이 완료된 후, 테스트 데이터를 사용하여 모델을 평가하고 검증 손실을 출력한다. 마지막으로 학습된 모델을

`mobilenet_model.h5` 파일로 저장한다.

- 최대 epoch를 30으로 설정, 일정 횟수동안 손실이 일정하다면 조기종료 되도록

```
from tensorflow.keras.callbacks import EarlyStopping

# 조기 종료 콜백 설정
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# 모델 컴파일
model.compile(optimizer='adam', loss='mean_squared_error')

# 모델 훈련
history = model.fit(
    X_train, y_train,
    epochs=30,
    validation_data=(X_val, y_val),
    callbacks=[early_stopping],
    batch_size=32
)

# 모델 평가
loss = model.evaluate(X_val, y_val)
print(f"Validation Loss: {loss}")

# 모델 저장
model.save('./mobileNet_model.h5')
```

8) 손실 및 데이터 예측 시각화

history.history['loss']와 history.history['val_loss']를 사용하여 학습 손실과 검증 손실 값을 각각 가져와 플롯으로 그린다. plt.plot을 사용해 학습 손실과 검증 손실 곡선을 그리고, plt.legend를 통해 두 곡선의 범례를 추가한다. 마지막으로 plt.show를 사용하여 손실 곡선을 화면에 표시한다. 이후 저장된 모델을 로드하고 예시 이미지에 대한 실제 값과 예측값을 비교한다. 먼저, `tf.keras.models.load_model`을 사용하여 `./mobileNet_model.h5` 파일에서 모델을 로드한다. 데이터프레임 `df`에서 무작위로 4개의 인덱스를 선택하여 해당 이미지들을 전처리한 후 배열로 변환한다. 선택된 이미지들의 실제 각도 값을 `df`에서 가져온다. 로드된 모델을 사용하여 전처리된 이미지들에 대해 예측을 수행한다. 마지막으로, 각 이미지에 대해 실제 각도 값과 예측된 각도 값을 제목으로 설정하고 이미지를 출력한다. 이미지는 `load_img`를 사용하여 `(70, 220)` 크기로 로드되며, `imshow`와 `plt.title`을 사용하여 제목과 함께 화면에 표시된다.

```
# 모델 로드
model = tf.keras.models.load_model('./mobileNet_model.h5')

# 예시 이미지 4개에 대한 실제 값과 예측값 보여주기
example_indices = np.random.choice(len(df), 4, replace=False)
example_images = [preprocess_image(df['ImagePath'].iloc[idx]) for idx in example_indices]
example_images_array = np.array(example_images)
example_angles = df['Angle'].iloc[example_indices].values

# 예측 수행
predicted_angles = model.predict(example_images_array)

# 실제 값과 예측값 출력
for i in range(4):
    plt.figure()
    plt.imshow(load_img(df['ImagePath'].iloc[example_indices[i]], target_size=(70, 220)))
    plt.title(f"Actual: {example_angles[i]}, Predicted: {predicted_angles[i][0]:.2f}")
    plt.axis('off')
    plt.show()
```

횡단보도 정지선 3초간 정지 학습

1) 필요한 라이브러리 import

파일 작업, 데이터 처리, 이미지 처리, 및 시각화를 위해 필요한 여러 라이브러리를 임포트하였다. os와 fnmatch는 파일 및 디렉토리 작업을 위해 사용되며, Counter는 데이터의 빈도수를 계산하는 데 사용된다. zipfile과 shutil은 파일 압축 해제와 파일 및 디렉토리 관리를 위해 사용된다. numpy와 pandas는 데이터 배열 및 데이터프레임 처리를 위해 사용되며, matplotlib.pyplot은 데이터 시각화를 위해 사용된다. 마지막으로, PIL.Image는 이미지 파일을 열고 처리하는 데 사용된다.

2) 데이터 준비

앞선 모델 학습 코드 때 사용한 것과 동일하다. `resnet2.zip` 파일을 압축 해제하고, 그 내용물을 `./data/VideoS3` 디렉토리에 병합하는 작업을 수행한다. 먼저 `zip_file_path`와 `target_folder_path`를 설정한 후, 임시 디렉토리 `./dd`를 생성한다. `zipfile.ZipFile`을 사용하여 압축 파일을 임시 디렉토리에 풀고, `video_folder_path`를 설정하여 임시 디렉토리 내의 파일들을 대상으로 복사한다. 파일을 복사할 때, 대상 디렉토리에 중복되는 파일이 있으면 `_1`, `_2` 등의 숫자를 추가하여 이름을 변경한다. 모든 파일을 복사한 후, 임시 디렉토리를 삭제하고 작업 완료 메시지를 출력한다.

3) 손상된 이미지 삭제

- 앞선 모델 학습 코드와 동일하기에 설명은 생략하였다.

4) 데이터의 분포 확인

아래 코드는 `./data/VideoS3` 디렉토리 내의 이미지 파일들을 `go`, `right`, `left`, `stop` 네 가지 카테고리로 분류하여 각 카테고리의 파일 개수를 계산한다. `Counter` 객체를 사용하여 각 카테고리의 파일 개수를 집계하고, 디렉토리 내의 파일 리스트를 얻어 파일명을 순회한다. 파일명이 ``.png``, ``.jpg``, ``.jpeg``로 끝나는 경우 파일명에서 인덱스를 추출하여 카운터에 추가한다. 추출된 인덱스는 파일명에서 12번째부터 15번째 문자까지를 사용하며, 각 인덱스에 해당하는 파일 개수를 증가시킨다. 마지막으로, 각 인덱스별 파일 개수를 출력한다.

- go/right/left/stop 별 이미지 데이터의 개수 확인
  - index: 090 ➡ go
  - index: 135 ➡ right
  - index: 045 ➡ left
  - index: 180 ➡ stop

```
folder_path = './data/VideoS3'

# 이미지 파일 인덱스 집계를 위한 빈 Counter 객체 생성
index_counter = Counter()

# 삭제된 이미지를 제외한 폴더 내의 모든 파일을 순회
for filename in os.listdir(folder_path):
    if filename.endswith(('.png', '.jpg', '.jpeg')):  # 이미지 파일 형식을 여기에 추가
        index = filename[12:15]  # 파일명에서 인덱스 추출

        index_counter[index] += 1  # 추출된 인덱스에 대한 카운트 증가

# 인덱스별 이미지 개수 출력
for index, count in index_counter.items():
    print(f'Index: {index}, Count: {count}')
```

5) 데이터 로드 및 특정 인덱스 이미지의 경로와 각도 출력

`./data/VideoS3` 디렉토리 내의 이미지 파일들을 `stop` 데이터와 `go/right/left` 데이터로 분류하고, `stopline` 여부를 기준으로 라벨링한다. 먼저, 파일 리스트를 얻고, 이미지 경로와 `stopline` 여부를 저장할 리스트를 초기화한다. 파일명을 순회하면서 파일명이 `.png` 형식에 맞는 경우 이미지를 열어 성공적으로 로드되면 해당 파일의 경로를 리스트에 추가하고, 파일명에서 인덱스를 추출하여 `stopline` 여부를 결정한다. 인덱스가 `180`이면 `stopline`을 1로 설정하고, 그렇지 않으면 0으로 설정하여 `stopline_yesno` 리스트에 추가한다. 예외 발생 시 파일을 무시한다. 이후, 조향 각도가 0이 아닌 이미지를 필터링하여 리스트를 업데이트하고, 특정 인덱스의 이미지를 표시하며 경로와 각도를 출력한다. 마지막으로, 이미지 경로와 `stopline` 여부를 포함하는 데이터프레임을 생성한다.

- stop데이터 ➡ stopline = 1으로 라벨링
- go/right/left데이터 ➡ stopline = 0으로 라벨링

```
data_dir = './data/VideoS3'
file_list = os.listdir(data_dir)
image_paths = [] # 이미지 경로를 저장하는 변수
stopline_yesno = [] # 이미지 라벨을 저장하는 변수
pattern = "*.png" # .png 확장자 가진 파일만 취급

for filename in file_list:
    if fnmatch.fnmatch(filename, pattern):
        file_path = os.path.join(data_dir, filename)
        try:
            # 이미지 파일을 열어보려고 시도
            with Image.open(file_path) as img:
                # 파일이 성공적으로 열렸다면 리스트에 추가
                image_paths.append(file_path)
                if filename[12:15]=='180':
                    stopline = 1
                else:
                    stopline = 0
                stopline_yesno.append(stopline)
        except:
            # 파일 열기에 실패한 경우, 예외가 발생하고 해당 파일은 무시
            print(f"Failed to open {file_path}")


# 이후 과정은 동일하게 진행
image_index = 20
plt.imshow(Image.open(image_paths[image_index]))
print("image_path: %s" % image_paths[image_index])
print("steering_Angle: %d" % stopline_yesno[image_index])

df = pd.DataFrame()
df['ImagePath'] = image_paths
df['Stopline'] = stopline_yesno
```

6) stop과 non-stop 데이터의 분포 확인

이미지 경로와 `stopline` 여부를 포함하는 데이터프레임을 생성하고, `stopline` 값에 따른 분포를 확인한다. 먼저, `image_paths`와 `stopline_yesno` 리스트를 사용하여 데이터프레임을 생성한다. 그런 다음, `value_counts` 메서드를 사용하여 `stopline` 컬럼의 각 값에 대한 빈도를 계산하고, 이를 `distribution` 변수에 저장한다. 마지막으로, 분포 결과를 출력하여 `stopline` 여부에 따른 데이터의 개수를 확인한다.

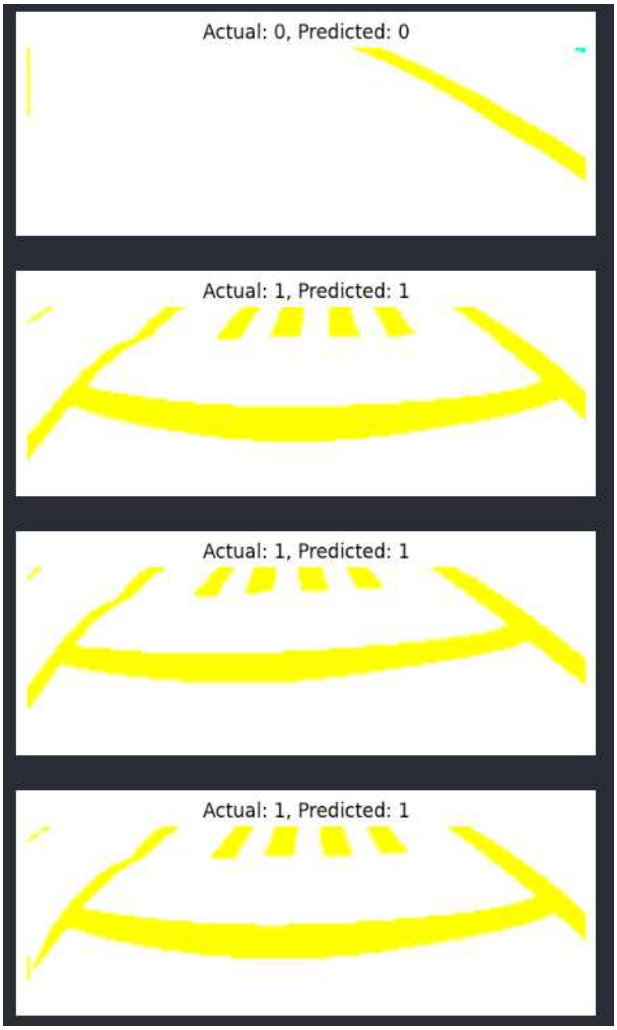
```
# 데이터프레임 생성
df = pd.DataFrame({
    'ImagePath': image_paths,
    'Stopline': stopline_yesno
})

# stopline 값에 따른 분포 확인
distribution = df['Stopline'].value_counts()

print(distribution)
```

7) 전처리,데이터 분리,모델 선언,모델 학습,평가,저장

이미지 데이터를 로딩 및 전처리하고, MobileNetV2 모델을 사용하여 `stopline` 감지 모델을 학습 및 평가한다. 먼저, 필요한 라이브러리를 임포트한 후, 이미지를 로딩하고 전처리하는 `preprocess_image` 함수를 정의한다. 데이터프레임 `df`에서 이미지 경로를 받아 전처리된 이미지를 `numpy` 배열로 변환하고, 라벨을 추출한다. 데이터를 학습용과 테스트용으로 분리한 후, 라벨을 원-핫 인코딩한다. MobileNetV2 모델을 로드하여 기본 모델의 출력을 가져오고, 글로벌 평균 풀링층과 완전 연결층을 추가하여 최종 예측층을 정의한다. 기본 모델의 층들은 학습되지 않도록 설정하고, Adam 옵티마이저와 `categorical_crossentropy` 손실 함수를 사용하여 모델을 컴파일한다. 이미지 데이터 증강을 설정하고, 학습 데이터를 증강하여 모델을 학습시킨다. 학습이 완료된 후 테스트 데이터로 모델을 평가하고, 정확도를 출력한다. 마지막으로 학습된 모델을 `stopline_detection_model.h5` 파일로 저장하고, 저장 완료 메시지를 출력한다.

8) 데이터 예측 시각화

여기서는 저장된 `stopline_detection_model.h5` 모델을 로드하고, 예시 이미지 4개에 대한 실제 값과 예측값을 비교한다. 먼저, `tensorflow`와 `PIL` 라이브러리를 사용하여 모델을 로드하고 이미지를 처리한다. 데이터프레임 `df`에서 무작위로 4개의 인덱스를 선택하여 해당 이미지들을 전처리한 후 배열로 변환한다. 선택된 이미지들의 실제 `stopline` 값을 `df`에서 가져온다. 로드된 모델을 사용하여 전처리된 이미지들에 대해 예측을 수행하고, 각 이미지에 대해 실제 `stopline` 값과 예측된 `stopline` 값을 비교하여 출력한다. 이미지는 `Image.open`을 사용하여 열고, 모델 입력 크기와 일치하도록 크기를 조정한 후, `plt.imshow`와 `plt.title`을 사용하여 실제 값과 예측값을 제목으로 설정하여 화면에 표시한다. 예측된 라벨은 원-핫 인코딩된 예측값에서 가장 높은 확률을 가진 인덱스를 선택하여 결정한다. 아래는 실행 결과이다.



2.4 모델 배포 및 주행 테스트

2.4.1. 모델 배포 준비

모델을 라즈베리 파이에 배포하는 단계는 프로젝트의 중요한 부분 중 하나이다. 아래는 이 단계를 자세히 설명한 것이다.

1) 모델 준비

MobileNet 모델을 사용하여 학습이 완료된 모델을 '.h5' 형식으로 저장하여 다운로드한다. '.h5' 파일 형식은 HDF5(Hierarchical Data Format) 기반의 파일 형식으로, 대용량 데이터를 효율적으로 저장하고 관리할 수 있다.

2) 라즈베리 파이에 모델 업로드

Samba를 이용하여 모델 파일을 라즈베리 파이로 전송하는 단계이다. 파일 탐색기를 사용하여 모델 파일을 라즈베리 파이로 이동시킨다. 이렇게 하면 네트워크 상에서 쉽게 파일을 전송하고 관리할 수 있다.

3) 모델 실행 환경 설정

모델을 라즈베리 파이에서 실행하기 위해서 TensorFlow 라이브러리를 설치해야 한다. 현재 파이썬 버전과 아키텍처를 맞춰 적절한 TensorFlow 버전을 선택하여 'whl' 파일을 설치한다. 이 외에도, 모델 실행에 필요한 추가 라이브러리나 종속성을 설치한다.

2.4.2. 실시간 실행 및 모니터링

실시간 실행 및 모니터링을 구현할 때, 다양한 요소를 고려해야 한다. 이러한 요소들 중 가장 중요한 것은 데이터의 실시간 처리와 모델의 예측 정확도이다. 이를 위해, 먼저 학습된 두 개의 모델을 로드하고, 실시간으로 데이터를 입력받아 처리할 수 있는 환경을 설정한다. 이후에는 입력된 데이터를 모델에 전달하여 예측을 수행하고, 그 결과에 따른 주행이 이뤄지도록 한다. 이 모든 과정을 순차적으로 구현하는 코드를 파이썬 스크립트로 작성하여 라즈베리파이에 업로드했다. 각 단계별로 중요한 부분들은 다음과 같다.

1) 필요한 라이브러리 및 모듈 import

- 'threading': 타이머와 스레딩 기능
- 'time': 시간 지연 기능
- 'cv2': OpenCV 라이브러리로, 이미지 및 비디오 처리를 위한 함수들을 포함
- 'RPi.GPIO': 라즈베리 파이의 GPIO 핀을 제어하기 위한 라이브러리
- 'numpy': 수치 계산을 위한 라이브러리로, 배열 및 행렬 연산을 지원
- 'tensorflow.keras.models'의 'load_model' 함수: 사전 학습된 딥러닝 모델을 로드하는 데 사용
- 'tensorflow.keras.preprocessing.image'의 'img_to_array' 함수: 이미지를 배열로 변환하는 데 사용

2) GPIO 핀 설정 및 모터 제어 함수 생성

Raspberry Pi 기반 자율주행차의 GPIO 핀 설정 및 모터 제어 기능 구성을 담당한다. 좌우 모터의 GPIO 핀을 초기화하고 방향을 설정하며 모터의 움직임(전방, 정지, 우회전, 좌회전)을 제어하는 기능을 정의한다.

- 왼쪽 모터는 'PWMA'(18), 'AIN1'(22), 'AIN2'(27) 핀에 연결되고, 오른쪽 모터는 'PWMB'(23), 'BIN1'(25), 'BIN2'(24) 핀에 연결된다.
- 모터를 전진, 정지, 우회전, 좌회전시키는 함수 'motor_go' , 'motor_stop', 'motor_right', 'motor_left' 를 정의한다.
- GPIO 핀 설정을 위해 'GPIO.setwarnings(False)'로 경고를 무시하고, 'GPIO.setmode(GPIO.BCM)'으로 핀 번호를 BCM 모드로 설정한다.
- 각 핀을 출력 모드로 설정한 후, 왼쪽 모터는 'PWMA' 핀을 통해, 오른쪽 모터는 'PWMB' 핀을 통해 PWM 신호를 받도록 설정한다. PWM 주파수는 100Hz로 설정되며, 초기 듀티 사이클은 0으로 시작한다.

3) 이미지 전처리 함수 생성

- 전처리 함수들은 주어진 카메라 캡처 이미지를 차선 및 정지선 검출을 위해 처리하는 과정을 담당한다. 이 함수들은 입력 이미지를 YUV 색상 공간으로 변환하고, 이후 리사이징, 가우시안 블러 처리, 이진화 등의 단계를 거쳐 최종적으로 모델의 입력으로 사용될 형태로 변환한다.
- 두 함수가 분리된 이유는 각각의 전처리 과정이 차선 및 정지선을 감지하는 데 필요한 특정한 처리를 포함하고 있기 때문이다. 이러한 세밀한 처리 과정은 모델의 예측 정확도를 향상시키는 데 기여하며, 각각의 함수를 통해 얻어지는 전처리된 이미지를 화면에 표시하여 시각적으로 확인할 수 있는 장점도 있다.

```python
# 이미지 전처리 함수 (차선 검출용)
def img_preprocess(image):
    height, _, _ = image.shape  # 이미지의 높이를 구하기
    image = image[int(height / 2):, :, :]  # 이미지의 하단 절반을 잘라냄
    image = cv2.cvtColor(image, cv2.COLOR_BGR2YUV)  # 이미지를 YUV 색상 공간으로 변환
    image = cv2.resize(image, (220, 70))  # 이미지 크기 변경
    image = cv2.GaussianBlur(image, (5, 5), 0)  # 가우시안 블러 적용
    _, image = cv2.threshold(image, 140, 255, cv2.THRESH_BINARY_INV)  # 이진화 수행
    return image

# 이미지 전처리 함수 (정지선 검출용)
def stopline_img_preprocess(image):
    height, _, _ = image.shape  # 이미지의 높이를 구하기
    image = image[int(height / 2):, :, :]  # 이미지의 하단 절반을 잘라냄
    image = cv2.resize(image, (70, 220))  # 이미지 크기 변경
    image = cv2.GaussianBlur(image, (5, 5), 0)  # 가우시안 블러 적용
    _, image = cv2.threshold(image, 140, 255, cv2.THRESH_BINARY_INV)  # 이진화 수행
    image = img_to_array(image)  # 이미지를 배열로 변환
    image = image / 255.0  # 이미지 정규화
    image = np.expand_dims(image, axis=0)  # 배치 차원 추가
    return image
```

4) 정지선 검출 플래그 생성

- 정지 가능 여부를 판단하는 데 사용되는 변수 'stopline_flag'를 초기값으로 'False'로 설정한다. 'stopline_flag'가 'False'일 경우 정지가 가능한 상태를 의미하며, 'True'일 경우 정지가 불가능한 상태를 나타낸다. 즉, 정지선을 감지하더라도 해당 변수가 'True'이면 모터가 정지하지 않는다.
- 'reset_stopline_flag' 함수는 'stopline_flag'를 'False'로 리셋하는 역할을 한다. 이 함수 내에서 'global' 키워드는 함수 외부에 정의된 'stopline_flag' 변수를 수정해야 함을 나타낸다.
- 이후 '프레임 처리 함수' 코드에서는 정지 이후의 10초 동안은 정지가 불가능하도록 'stopline_flag'를 'True'로 설정한다. 이는 횡단보도를 건너는 상태를 정지선으로 인식할 수 있기 때문에, 정지한 이후의 10초를 정지 불가능한 상태로 설정해두는 것이다. 10초가 지나면 다시 정지선의 검출을 수행할 수 있도록 'reset_stopline_flag' 함수를 이용하여 'stopline_flag'를 'False'로 설정한다.

```python
stopline_flag = False  # 정지선 감지 플래그 초기화

# 정지선 감지 플래그 재설정 함수
def reset_stopline_flag():
    global stopline_flag
    stopline_flag = False
```

5) 프레임 캡처 함수

- 카메라를 초기화하고 해상도를 설정하며 자동차의 상태 및 프레임 처리와 관련된 변수를 초기화한다. 이를 통해 시스템이 안정적으로 작동할 수 있는 기초를 마련한다.
- 스레드 동기화를 위한 'Lock' 객체를 생성하고, 카메라에서 프레임을 캡처하는 기능을 정의한다. 'capture_frames' 기능은 별도의 스레드에서 실행되도록 설계되어, 메인 스레드가 프레임을 처리하는 동안 카메라에서 프레임을 계속 캡처할 수 있다. 'Lock' 객체는 프레임 변수가 여러 스레드에 의해 동시에 액세스되지 않도록 하여 데이터 손상이나 레이스 상황을 방지한다. 이를 통해 안정적인 데이터 처리를 보장한다.

6) 프레임 처리 함수

자율 주행 자동차 시스템에서 중요한 역할을 수행하는 프레임 처리 및 제어 기능을 정의한다. 주요 기능은 카메라로부터 캡처된 프레임을 처리하고 미리 학습된 TensorFlow/Keras 모델을 사용하여 차선의 조향 각도를 예측하고, 정지선을 감지하여 필요에 따라 차량을 정지시키는 것이다. 다음은 이 코드에서 주요 기능들에 대한 상세 설명이다.

① 학습모델 불러오기:

- 차선을 인식하여 조향각을 예측하는 모델과 정지선을 인식하는 모델을 각각 불러온다.

```
119     model_path = './model/lane_navigation_model.h5'  # 차선 인식 모델 경로
120     stopline_model_path = './model/stopline_model.h5'  # 정지선 인식 모델 경로
121
122     model = load_model(model_path)  # 차선 인식 모델 로드
123     stopline_model = load_model(stopline_model_path)  # 정지선 인식 모델 로드
```

② 프레임 캡처 루프:

- 프레임을 지속적으로 캡처하고 처리하기 위해 'while True' 루프를 사용하여 프레임을 지속적으로 처리한다.
- 'with lock' 구문으로 다중 스레드 환경에서 프레임에 안전하게 접근하고 업데이트되도록 한다.
- 키보드 인터럽트가 발생하면 코드가 루프를 종료하고, 'cv2.destroyAllWindows'를 사용하여 모든 OpenCV 창을 정리한다.

③ 이미지 전처리 및 시각화:

- 멀티스레딩 환경에서 프레임을 안전하게 처리하기 위해 락을 사용하고, 유효한 프레임만을 전처리하여 차선 및 정지선 검출을 위한 준비를 한다.
- 차선 탐색을 위해 이미지 데이터를 전처리하기 위한 'img_preprocess' 와 정지선 검출을 위해 전처리하기 위한 'stopline_img_preprocess' 함수를 이용하여 프레임 전처리를 진행한다.
- 전처리된 프레임은 'cv2.imshow'를 사용하여 시각적으로 표시된다. 이는 전처리된 이미지가 실제로 어떻게 보이는지를 확인하기 위한 목적으로 사용된다.

```
127 with lock:
128     if frame is None:
129         continue
130     preprocessed = img_preprocess(frame)  # 차선 검출을 위해 프레임 전처리
131     stopline_X = stopline_img_preprocess(frame)  # 정지선 검출을 위해 프레임 전처리
132
133 cv2.imshow('pre', preprocessed)  # 전처리된 이미지를 화면에 표시
```

④ 모델 예측 및 제어:

- 전처리된 프레임은 'model.predict' 및 'stopline_model.predict' 함수를 사용하여 차선 내비게이션 및 정지선 검출 모델에 전달된다. 이를 통해 실시간으로 차선의 모양을 분석하고, 적절한 주행 경로를 결정할 수 있다.
- 조향각 예측 모델을 이용하여 도로의 생김새에 따른 조향각을 실시간으로 불러온다. 이 조향각 정보를 바탕으로 차량의 주행 방향을 조정한다.
- 정지선 검출 모델을 이용하여 정지선이 발견되면 1을 리턴한다. 이를 통해 차량이 정지선 앞에서 멈출 수 있도록 제어한다.

```
135 # 조향 각도 예측
136 preprocessed = img_to_array(preprocessed)
137 preprocessed = preprocessed / 255.0
138 X = np.asarray([preprocessed])
139 prediction = model.predict(X)
140 steering_angle = prediction[0][0]   # 예측된 조향각
141 print("Predicted angle:", steering_angle)
142
143 # 정지선 검출
144 stopline_prediction = stopline_model.predict(stopline_X)
145 stopline_detected = np.argmax(stopline_prediction[0])   # 정지선 검출시 1, 미검출시 0
146 global stopline_flag    # 함수 밖에서 정의한 정지선 검출 플래그 불러오기
```

⑤ 정지선 감지 및 제어 플래그 생성:

- 정지 가능한 상태('stopline_detected'가 1), 'stopline_flag'가 'False'일 때 실행된다.
- 해당 경우에는 조향각의 예측값이 나오더라도 모터를 정지시키고 3초 동안 대기한다.
- 그런 다음, 'stopline_flag'를 10초 동안만 'True'로 설정하여 정지선으로 인식하더라도 10초간은 정지 불가능한 상태로 만든다.
- 10초 후에는 'stopline_flag'를 다시 'False'로 설정하여 정지선을 검출하면 다시 정지할 수 있도록 설정한다.

```
148 # 정지 가능 상태 & 정지선이 검출된 경우
149 if stopline_detected and not stopline_flag:
150     print("Stopline detected, stopping for 3 seconds")
151     motor_stop()
152     time.sleep(3)  # 3초 동안 정지
153     stopline_flag = True
154
155     # stopline_flag를 True로 설정한 후 10초 후에 다시 False로 설정
156     # 10초 동안만 정지 불가능한 상태로 만드는 것
157     threading.Timer(10, reset_stopline_flag).start()
158     continue
```

⑥ 차량 상태 및 사용자 입력 처리:

- 'carState' 변수 및 사용자의 키보드 입력을 처리하여 차량의 상태를 변경하고, 예측된 조향각에 따라 직진, 우회전 또는 좌회전을 결정한다.
- 해당 하드웨어의 경우 오른쪽 2개의 모터 힘이 약해서 좌회전을 할 때 속도를 내지 못하는 경향이 있었다. 이는 좌회전의 경우의 speedSet을 우회전보다 증가시켜줌으로 해결했다.

이 코드는 자율 주행 RC 자동차 시스템에서 실시간 이미지 처리와 모터 제어를 통해 차선 탐색 및 정지선 감지를 수행하는 데 중점을 둔다. 다양한 기술적 요소들이 통합되어 자율 주행 기능을 구현하는 데 필수적인 기능을 제공한다.

⑦ 메인 함수

- 두 개의 분리된 스레드를 생성하는데, 하나는 카메라로부터 프레임을 캡처하기 위한 것이고 다른 하나는 프레임을 처리하기

위한 것이다.
- 캡처 스레드는 'capture_frames' 기능을 실행하는데, 이는 카메라로부터 프레임을 지속적으로 읽고 프레임 변수를 최신 프레임으로 업데이트한다.
- 프로세스 스레드는 'process_frames' 기능을 실행하는데, 이는 처리된 프레임을 기반으로 차선 검출, 정지선 검출, 로봇의 움직임 제어 등의 다양한 작업을 수행한다.
- 두 스레드를 생성하고 시작한 후, 메인 함수는 두 스레드가 'join' 방법을 사용하여 끝날 때까지 기다린다.
- 두 스레드가 모두 끝나면, 메인 함수는 'GPIO.cleanup()' 함수를 실행하여 프로그램에 의해 사용되었던 임의의 GPIO 리소스를 해제한다. 이것은 프로그램이 종료되기 전에 제대로 정리되는 것을 보장한다.
- 전체적으로 주요 기능은 캡처 및 처리 스레드를 관리하고 필요한 리소스를 정리함으로써 프로그램의 실행을 조정하는 역할을 한다.

```
190 # 메인 함수
191 def main():
192     capture_thread = threading.Thread(target=capture_frames)  # 프레임 캡처용 스레드 생성
193     process_thread = threading.Thread(target=process_frames)  # 프레임 처리용 스레드 생성
194 
195     capture_thread.start()  # 프레임 캡처 스레드 시작
196     process_thread.start()  # 프레임 처리 스레드 시작
197 
198     capture_thread.join()  # 프레임 캡처 스레드가 종료될 때까지 대기
199     process_thread.join()  # 프레임 처리 스레드가 종료될 때까지 대기
200 
201 if __name__ == '__main__':
202     main()  # 메인 함수 실행
203     GPIO.cleanup()  # GPIO 리소스 정리
```

2.4.3. 주행 테스트

주행 테스트 단계에서는 실제 환경에서 모델이 제대로 작동하는지 확인한다. 다음과 같은 절차를 따른다.

1) 라즈베리 파이를 장착한 차량을 테스트 트랙이나 실제 도로 환경에서 주행시킨다.

모델이 다양한 도로 조건과 시나리오에서 어떻게 반응하는지 관찰한다.

2) 모델이 실시간으로 데이터를 처리하고 주행 경로를 예측하여 차량을 제어하는지 확인한다.

차량의 센서 데이터와 카메라 영상을 실시간으로 분석하여 주행 경로를 예측하고, 이에 따라 차량 제어 명령을 내리는 과정을 모니터링한다.

3) 주행 중 발생하는 문제를 기록하고, 모델의 성능을 평가하여 필요한 경우 모델을 재학습하거나 파라미터를 조정한다.

- 모델 변경: Yolov5, AlexNet, ResNet18, ResNet50, MobileNet 모델을 사용하여 학습시켜보았다.
- 데이터 변경: 데이터셋 크기를 5000장, 10000장, 15000장, 50000장, 70000장으로 변화시키며 학습을 진행하였다. 다양한 데이터셋 크기를 테스트하여 모델의 학습 성능과 일반화 능력을 평가하였다. 최종적으로는 조향각 예측 모델 10,216장, 정지선 검출 모델 3,572장으로 데이터를 학습하였다.
- 에폭 변경: 학습 에폭을 10, 30, 50, 100으로 설정하여 성능을 평가하였다. 최종적으로 조향각 예측 모델 22, 정지선 검출 모델 20으로 데이터를 학습하였다.

4) 테스트 결과를 분석하여 최종적으로 모델의 안정성과 효율성을 검증한다.

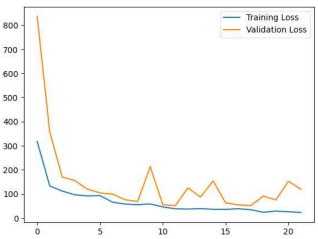
주행 테스트에서 수집된 데이터를 분석하여 모델의 성능을 정량적으로 평가하고, 필요한 경우 추가적인 조정을 진행한다. 모델의 안정성, 효율성, 실시간 처리 능력 등을 종합적으로 검토하여 최종 모델을 확정한다. Yolov5, AlexNet, ResNet18 모델은 선형회귀를 진행하기에 적합하지 않은 모델이어서 예측치를 0, 1, 2의 클래스로 나오게끔 했었다. 하지만 이 때문인지 정확도가 떨어져 ResNet50 모델로 선형회귀를 진행했다. 선형 회귀나 예측 부분에서는 좋은 성능을 보였지만, 라즈베리파이의 하드웨어가 감당하기엔 부피가 큰 모델이었다. 카메라 버벅임이 심해 주행을 할 수 없는 상황이었다. 이에 모델의 안정성, 효율성, 실시간 처리 능력 등을 종합적으로 검토하여 최종 모델을 MobileNet로 선정하게 되었다.

Ⅲ. 학습 및 주행 결과에 대한 논의

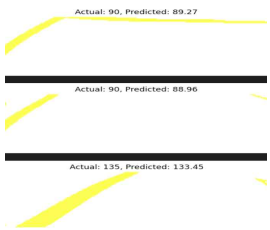
3.1 학습 및 주행 결과

3.1.1. 학습 결과

모델의 학습 손실 및 데이터 예측 결과는 다음과 같이 나왔다.



손실 시각화



모델 예측 결과

3.1.2. 주행결과

본 프로젝트 결과는 다음과 같다. 총 3가지 트랙에서 주행하였으며 모두 성공적으로 완주하였다.

1) 트랙1

트랙1의 경우 주 관심 부분은 직진과 V자 커브이다. 트랙1 완주를 성공적으로 수행 하였다. 하지만 커브를 돌고 난 후 주행선을 밟는 상황이 주로 보여 아쉬웠다.

2) 트랙2

트랙2의 경우는 S자 코너를 활용한 코너였다. 트랙2 역시 성공적으로 완주 하였으나 커브를 돌고 난 뒤 주행을 할 때 중심에서 치우쳐 선을 밟은 경우가 또 발생하였다.

3) 트랙3

트랙3의 경우 타원형 트랙에 횡단보도가 추가된 트랙이다. 트랙

3 역시 성공적으로 완주 하였다. 하지만 총 2개의 횡단보도 중 1의 횡단보도 때 정지선 앞에서가 아닌 정지선에 딱 맞추어 정지하였다.

3.2 학습 및 주행 결과에 대한 논의

모든 트랙을 성공적으로 완주하였음에 큰 의의가 있었지만, 다음 2가지의 상황에 대한 아쉬운 점이 존재하였다.

1) 커브를 돌고 난 뒤 주행 시 선을 밟게 되는 상황
① 문제점: 커브를 돈 후, 직선 구간으로 진입할 때 차량이 차선을 정확히 유지하지 못하고 선을 밟는 경우
② 원인 분석:
- 커브 구간에서의 속도 조절 및 방향 전환이 부정확하여, 직선 구간에서도 주행 궤적이 흔들리는 현상이 발생한 것 같다.
- 차량의 회전 각도와 속도 조절이 미세하게 어긋나면서 차선 유지에 어려움이 발생한 것 같다.
③ 개선 방안:
- 새로운 학습 데이터 추가 획득
- 알고리즘 개선 : 커브 구간에서의 속도와 회전 각도를 더 정밀하게 제어할 수 있도록 알고리즘을 개선할 수 있을 것이다. 커브를 지난 후 차선 복귀를 위한 보정 알고리즘을 추가하여 차량이 신속하게 올바른 궤적으로 돌아올 수 있도록 한다.
- 고도화된 실시간 차선 인식 기술을 적용하여 차량이 차선을 벗어나는 경우를 즉각적으로 감지하고 수정할 수 있도록 한다.

2) 횡단보도 정지 때 정지선에 딱 맞추어 정지되는 상황
① 문제점: 횡단보도 앞에서 차량이 정지선에 정확하게 맞추어 정지하지 못하는 경우
② 원인 분석:
- 정지선 인식 후 제동을 시작하는 시점과 제동 강도가 정확하지 않아 정지 위치가 부정확하게 될 수 있다.
- 차량의 속도와 거리 데이터를 바탕으로 제동을 시작하는 시점에 미세한 오차가 발생할 수 있다.
③ 개선 방안:
- 새로운 학습 데이터 추가 획득

커브 후 차선 유지와 횡단보도 정지선 준수 문제를 개선하는 것은 자율 주행의 성능을 높이기 위해 해결해야 할 앞으로의 주요 과제이다. 이러한 문제를 해결하기 위해서는 지속적인 학습 데이터 보강과 알고리즘 개선이 필요하며, 이를 통해 자율 주행의 안전성과 신뢰성을 더욱 높일 수 있을 것이라 기대된다.

Ⅳ. 조원의 역할 및 기여도

4.1 조원의 역할

4.1 조원의 역할
조원의 역할에는 크게 자료조사, 트랙 제작, 모델 설계 및 학습, 모델 평가, 주행 테스트 코드 작성, 보고서 작성이 있다.

4.2 조원의 기여도

조원 기여도의 경우 문혜진(25%), 박제인(25%), 손아현(25%), 최수현(25%)이고 구체적인 역할 분담은 아래와 같다.

- 문혜진: 자료조사, 트랙 제작, 모델 설계 및 학습, 모델 평가, 주행 테스트 코드 작성, 보고서 작성
- 박제인: 자료조사, 트랙 제작, 모델 설계 및 학습, 모델 평가, 주행 테스트 코드 작성, 보고서 작성
- 최수현: 자료조사, 트랙 제작, 모델 설계 및 학습, 모델 평가, 주행 테스트 코드 작성, 보고서 작성
- 손아현: 자료조사, 트랙 제작, 모델 설계 및 학습, 모델 평가, 주행 테스트 코드 작성, 보고서 작성

Ⅴ. 참고 문헌

• 김예지, 이년용, 남혜원, 김현웅, 고윤석. (2021). 블루투스 무선통신과 라즈베리파이를 이용한 자율주행 알고리즘에 대한 연구. 한국전자통신학회 논문지, 16(4), 689-698.

• 김여경, 정윤서, 황소영. (2020-08-28). 라즈베리파이와 아두이노 및 OpenCV를 활용한 공유형 자율주행차 모델 설계. 한국정보통신학회 여성 ICT 학술대회 논문집, 서울.

• 이성진, 최준형, 최병윤. (2021-10-28). 라즈베리파이와 OpenCV를 활용한 선형 검출 알고리즘 구현. 한국정보통신학회 종합학술대회 논문집, 전북.