

MobileNetV2 기반 LKAS (Pre_Version)

대충 MobilenetV2 기반 LKAS 딥러닝 모델 구현

```
1 import tensorflow as tf
2 from tensorflow.keras.layers import Input, Dense, GlobalAveragePooling2D
3 from tensorflow.keras.models import Model
4 from tensorflow.keras.applications import MobileNetV2
5 def build_rc_car_steering_model(image_shape, num_flags):
6     """
7     이미지와 플래그 입력을 결합하여 RC카 조향을 예측하는 모델을 구성합니다.
8     :param image_shape: 입력 이미지의 크기 (예: (160, 120, 3))
9     :param num_flags: 플래그(수치 입력)의 개수 (예: 3)
10    :return: Keras Model 객체
11    """
12    # 1. 이미지 처리 스트림 (MobileNetV2)
13    # MobileNetV2 기본 모델 로드 (ImageNet 가중치 사용, 상단 FC 층 제외)
14    base_model = MobileNetV2(
15        input_shape=image_shape,
16        include_top=False, # 최종 분류층 제외
17        weights='imagenet' # 사전 학습된 가중치 사용 (전이 학습)
18    )
19    # 기본 모델의 입력
20    image_input = base_model.input
21    # 기본 모델의 출력 특징 추출
22    x = base_model.output
23    # 특징 맵을 1차원 벡터로 변환
24    x = GlobalAveragePooling2D()(x) # (None, 1280) 형태의 벡터가 됨
25    # 2. 플래그 입력 스트림
26    # 플래그(속도, 이진 조향 등 수치 데이터)를 위한 별도의 입력층
27    flag_input = Input(shape=(num_flags,), name='flag_input')
28    # 3. 특징 결합 및 최종 예측
29    # MobileNetV2 특징 벡터와 플래그 입력을 결합
30    combined = Concatenate()([x, flag_input])
31    # 결합된 특징을 처리하기 위한 Dense Layer 추가
32    y = Dense(128, activation='relu')(combined)
33    y = Dense(64, activation='relu')(y)
34    # 최종 조향 각도 출력 (연속 값 예측 - 회귀)
35    # -1.0 ~ +1.0 사이의 조향 각을 예측하기 위해 'tanh' 활성화 함수 사용
36    steering_output = Dense(1, activation='tanh', name='steering_output')(y)
37    # 최종 모델 정의 (두 개의 입력과 하나의 출력)
38    model = Model(inputs=[image_input, flag_input], outputs=steering_output)
39    return model
```

--- 사용 예시 ---

```
1 # 입력 설정
2 IMAGE_H = 120
3 IMAGE_W = 160
4 IMAGE_CHANNELS = 3
5 NUM_FLAGS = 3 # 예: [속도, 가속도, 이진 조향값]
6 # 모델 생성
7 rc_car_model = build_rc_car_steering_model(
8     image_shape=(IMAGE_H, IMAGE_W, IMAGE_CHANNELS),
9     num_flags=NUM_FLAGS
10 )
11 # 모델 컴파일 (회귀 문제: 손실 함수는 MSE, 지표는 MAE/MSE 사용)
12 rc_car_model.compile(
13     optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
14     loss='mse',
15     metrics=['mae'] # Mean Absolute Error (평균 절대 오차)
16 )
17 # 모델 구조 출력
18 print("--- 모델 요약 ---")
```

모델 학습

MobileNetV2 기반의 결합 모델을 학습시키기 위한 **데이터 준비 및 학습 방법**에 대해 설명해 드리겠습니다. 📖

📁 1. 데이터셋 구조 및 준비

이 모델은 두 가지 형태의 입력을 요구하므로, 학습 데이터도 이에 맞춰 구성해야 합니다.

- **입력 1: 이미지 데이터 (Image Data)**

- **형태:** RC카 전방 카메라에서 캡처한 이미지(RGB).
- **전처리:** 모델 입력 크기 (예: 120×160)에 맞게 **크기 조정 (Resizing)**하고, 픽셀 값을 $[0, 255]$ 에서 **$[-1, 1]$ 범위로 정규화**해야 합니다. MobileNetV2는 보통 이 범위의 정규화를 사용합니다.

- **입력 2: 플래그 데이터 (Flag Data)**

- **형태:** RC카의 상태를 나타내는 수치형 데이터 (예: 속도, 이전 조향각, GPS 데이터 등).
- **전처리:** 각 플래그의 특성에 맞게 **정규화**가 필요합니다. 예를 들어, 값이 매우 큰 속도와 작은 조향각이 있다면, 두 데이터 모두 평균 0, 표준편차 1을 갖도록 **Z-score 정규화**를 적용하는 것이 학습에 유리합니다.

- **출력: 조향 값 (Steering Label)**

- **형태:** 예측해야 할 정답 조향 각도 (예: -1.0 (최대 좌회전) $\sim +1.0$ (최대 우회전)).

데이터 로더 (Data Loader) 구성

TensorFlow의 `tf.data.Dataset` 또는 Keras의 `tf.keras.utils.Sequence` 를 사용하여 다음 형태로 데이터를 제공해야 합니다.

```
입력: {'mobilenetv2_input': [Image_Batch], 'flag_input':
      [Flag_Batch]}
```

```
출력 (정답 레이블): {'steering_output': [Steering_Label_Batch]}
```

⚙️ 2. 모델 학습 (Training)

모델을 정의하고 데이터를 준비했다면, 다음 단계로 학습을 진행합니다.

전이 학습 전략 (Transfer Learning)

MobileNetV2는 ImageNet으로 사전 학습되어 있으므로, 초기 레이어의 가중치를 보호하는 것이 좋습니다.

1. **초기 동결 (Freeze):** MobileNetV2의 모든 레이어를 학습 초기에 `trainable=False`로 설정합니다. 이렇게 하면 사전 학습된 특징 추출 능력을 유지하면서, 새로 추가된 Dense Layer (특징 결합 및 예측 부분)만 학습됩니다.
2. **Fine-tuning (선택):** Dense Layer의 학습이 어느 정도 진행된 후, MobileNetV2의 마지막 몇 개의 블록만 `trainable=True`로 풀어줍니다. 이 단계에서는 **매우 작은 학습률(Learning Rate)**을 사용하여 전체 가중치를 미세 조정(Fine-tuning)합니다.

```
1 # 1. 초기 동결
2 rc_car_model.layers[1].trainable = False # MobileNetV2 Base Model을 동결
3 # (참고: MobileNetV2는 rc_car_model.layers[1]에 위치함)
4 # 2. 컴파일 후 학습 (Dense Layer만 학습)
5 rc_car_model.compile(
6     optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
7     loss='mse',
8     metrics=['mae']
9 )
10 # rc_car_model.fit(...)
11 # --- Fine-tuning 시작 ---
12 # 3. MobileNetV2의 일부 레이어만 동결 해제
13 rc_car_model.layers[1].trainable = True
14 # MobileNetV2 내에서도 처음 50개 정도의 레이어만 동결 (예시)
15 for layer in rc_car_model.layers[1].layers[:50]:
16     layer.trainable = False
17 # 4. 매우 작은 학습률로 재컴파일 후 학습
18 rc_car_model.compile(
19     optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5), # 학습률을 1e-5로 설정
20     loss='mse',
21     metrics=['mae']
22 )
23 # rc_car_model.fit(...)
24
```

콜백 (Callbacks) 활용

안정적인 학습을 위해 다음 콜백을 사용하는 것이 좋습니다.

- **ModelCheckpoint**: 가장 성능이 좋은 모델 가중치를 저장합니다.
- **EarlyStopping**: 검증 손실(Validation Loss)이 더 이상 개선되지 않으면 학습을 조기에 중단하여 과적합(Overfitting)을 방지합니다.

🚀 3. 모델 배포 (Deployment)

학습된 모델은 RC카의 임베디드 장치에 배포될 수 있습니다.

- **모델 저장**: `model.save('rc_car_model.h5')`를 사용하여 모델을 저장합니다.
- **TFLite 변환**: MobileNetV2를 사용한 목적과 같이, 모델을 **TensorFlow Lite (TFLite)** 형식으로 변환하여 모바일 또는 임베디드 장치(예: Raspberry Pi, Jetson Nano 등)에서 더 **빠르고 효율적으로 추론**할 수 있도록 합니다.

```
1 # TFLite 변환 예시
```

```

2 converter = tf.lite.TFLiteConverter.from_keras_model(rc_car_model)
3 tflite_model = converter.convert()
4 with open('rc_car_model.tflite', 'wb') as f:
5     f.write(tflite_model)

```

4. 사용

```

1 # 예시 코드 (Keras 모델 로드 및 추론)
2 import numpy as np
3 import tensorflow as tf
4
5 # 1. 모델 로드
6 model = tf.keras.models.load_model('rc_car_model.h5')
7
8 # 2. 전처리된 입력 데이터 (numpy 배열 형태)
9 # image_input_data.shape: (1, 120, 160, 3)
10 # flag_input_data.shape: (1, 3)
11
12 # 3. 예측 실행
13 # 모델이 여러 입력을 받으므로, 입력 데이터를 리스트 또는 딕셔너리로 전달해야 합니다
14 prediction = model.predict(
15     {
16         'mobilenetv2_input': image_input_data, # Keras가 자동으로 처리하는
17         'flag_input': flag_input_data
18     }
19 )
20
21 # 예측된 조향 값 (예: [[0.25]])
22 steering_value = prediction[0][0]

```