

졸업 작품 최종 보고서

OpenAI 및 머신러닝 기반 멀티모달 레시피 추천
앱

지도 교수 : 박 구 만 교수님

제 출 일 : 2023년 9월 19일

제 작 자 : 22110119 권 영 호

서울과학기술대학교 전자IT미디어공학과

목 차

요약	3
I . 개요	4
1. 제작 배경 2. 목적 3. 기존 기술의 문제점 4. 해결 방안 제시 5. 설계 진행 일정	
II . 본론	6
1. 프로젝트의 구성 도면 2. 식재료 이미지 인식 3. 음성 인식 및 텍스트 처리 4. 레시피 추천	
III . 실험 및 결과 고찰	9
1. 데이터 수집 및 라벨링 2. 다양한 모델 파인 티닝 3. 입력 데이터에 대한 실험 4. TensorFlowLite(TFLite) 5. Flutter	
IV . 결론	35
V . 참고 문헌	36

요 약

제 목 : OpenAI 및 머신러닝 기반 레시피 추천 앱

본 연구에서는 Flutter 프레임워크와 Python언어 그리고 OpenAI의 GPT-3 와 DALL-E를 활용하여 식재료 이미지를 자동으로 분류하거나 STT(Speech-To-Text)기술로 음성을 인식 하여 해당 식재료를 활용할 수 있는 레시피를 추천하는 모바일 앱을 개발하였다.

36가지의 식재료를 대상 데이터를 수집 하였으며, 이를 기반으로 MobileNetV2 인공지능 모델을 학습시켰다. TensorFlowLite를 활용하여 학습된 모델을 모바일에서 이미지 분류가 가능하게 하였다. 휴대폰 카메라와 마이크를 통해 입력한 데이터를 분류 하고, 분류된 식재료를 기반으로 OpenAI의 API를 통해 다양한 레시피가 실시간으로 추천된다.

분류 정확도는 약 97%에 이르렀다. 또한, 사용자 피드백을 통해 레시피의 만족도도 높게 평가받았다. 이러한 기능은 식재료의 효율적인 활용, 낭비 감소 및 자원 절약을 도모할 수 있으며, 사용자가 쉽게 다양한 레시피를 시도해 볼 수 있는 플랫폼을 제공한다.

본 연구를 통해 개발된 앱은 더 나아가 지속 가능한 식문화 및 건강한 식습관 형성에 기여할 것으로 예상된다. 향후 작업에서는 더 많은 식재료를 추가하고, 개인의 질병에 따른 식습관 가이드라인을 제공함과 동시에 다이어트를 위한 칼로리 계산 등의 기능을 만들어 앱의 활용성을 높일 계획이다.

I. 개요

1. 제작 배경

식품 낭비는 전 세계적으로 큰 문제로 대두되어 왔다. 특히 일반 가정에서 식재료를 효율적으로 관리하지 못해 자주 버리는 경우가 많다. 또한, 집에 있는 식재료로 어떤 요리를 할 수 있는지 파악하는 것이 일반적으로 쉽지 않아, 이로 인한 불필요한 식재료 구매가 이루어지곤 한다.

코로나, 우크라이나 사태 이후로 식자재 물가 상승으로 전세계는 유례없는 인플레이션을 겪게 되었으며, 위와 같은 자원의 낭비는 서민들의 주머니 사정에 더욱 직접적인 영향을 주게 되었다.

2. 목적

현대 생활에서 스마트폰은 거의 필수품이 되었고, 다양한 앱을 통해 생활이 편리해지고 있다. 그럼에도 불구하고 식재료 관리와 관련된 효율적인 솔루션이 부족함을 인지하였다. 이러한 배경과 동기를 바탕으로, 식재료를 스마트하게 관리하고 활용할 수 있는 모바일 앱을 제작하려는 목표를 설정하였다.

3. 기존 기술의 한계점

기존에는 식재료 관리를 위한 여러 앱과 솔루션이 있었으나, 대부분은 수동으로 식재료를 입력하고 관리해야 했다. 또한, 실제로 사용자가 가지고 있는 식재료를 기반으로 한 레시피 추천은 제한적이었으며 위 기능을 위해 서버와 데이터베이스의 사용은 필수적이었다. 이러한 수동적인 접근은 사용자로부터 많은 시간과 노력을 요구하는 단점이 있음과 동시에 확장 가능성과 다양성에 상당한 제약을 주었다.

4. 해결 방안 제시

이 문제를 해결하기 위해 이미지 인식 기술과 STT 기술을 활용하여 식재료를 자동으로 분류하는 기능을 앱에 탑재할 계획이다. 이를 통해 사용자는 쉽게 식재료를 앱에 입력 할 수 있다. 또한, OpenAI의 API를 활용하여 식재료 기반의 레시피와 그 예시 사진을 자동으로 추천해 준다. 이렇게 함으로써 사용자의 편의성을 대폭 향상시키고, 실시간으로 개인화된 레시피를 추천하며 식재료 낭비를 줄이는 것이 최종 목표이다.

5. 설계 진행 일정

~2023년 2월 27일 : 프로젝트 계획 및 준비 단계

- + 프로젝트 목표, 범위 결정
- + 필요한 기술 및 리소스 조사
- + 프레임워크 및 라이브러리 선택(Flutter, Python, etc.)

~2023년 3월 13일 : 프로젝트 기반 확립

- + 초기 UI/UX 디자인 및 구현
- + OpenAI와의 연동 가능성 확인
- + AI 모델의 앱에서 구동 가능성 조사
- + 평가지표, 비교 대상 조사

~2023년 3월 27일 : 연구 및 기술 습득

- + 이미지 인식 및 머신러닝에 대한 연구 및 학습

~2023년 4월 17일 : 연구 및 기술 습득

+ ResNet50 모델 활용 가능성 연구

+ 연구 논문 속 코드 분석

~2023년 5월 1일 : 연구 및 기술 습득

+ 여러 AI 모델들에 대한 파인 티닝 및 학습 후 정확도 비교

+ 서버 사용 가능성을 염두한 Django 학습

~2023년 5월 15일 : 입력 이미지 배경에 대한 고찰

+ 실제 실생활 데이터에 대한 선정모델(MobileNetV2)의 정확도 평가

+ 모델의 학습 이미지 배경에 대한 의존도 연구 및 실험

~2023년 6월 27일 : TensorFlowLite(TFLite)

+ 앱에서의 AI활용 방안으로써 TFLite 활용 가능성 검토

+ Flutter의 TFLite 라이브러리 문서 연구 및 구현

~2023년 7월 11일 : TensorFlowLite(TFLite)

+ 앱에서의 TFLite 모델이 정확도 손실을 갖는 문제 원인 분석

+ 문제 해결을 위한 다양한 방법 시도(양자화 인식훈련, 선택적 양자화, 훈련 후 양자화, etc.)

~2023년 7월 25일 : TensorFlowLite(TFLite)

+ 정확도 손실 원인에 대한 심층 검토

+ 앱에서의 TFLite 모델이 정확도 손실 문제 해결

~2023년 8월 8일 : 프로토타입 개발

+ OpenAI와의 연동성 점검 및 수정

+ Image input page 구현

+ STT 라이브러리 연구 분석

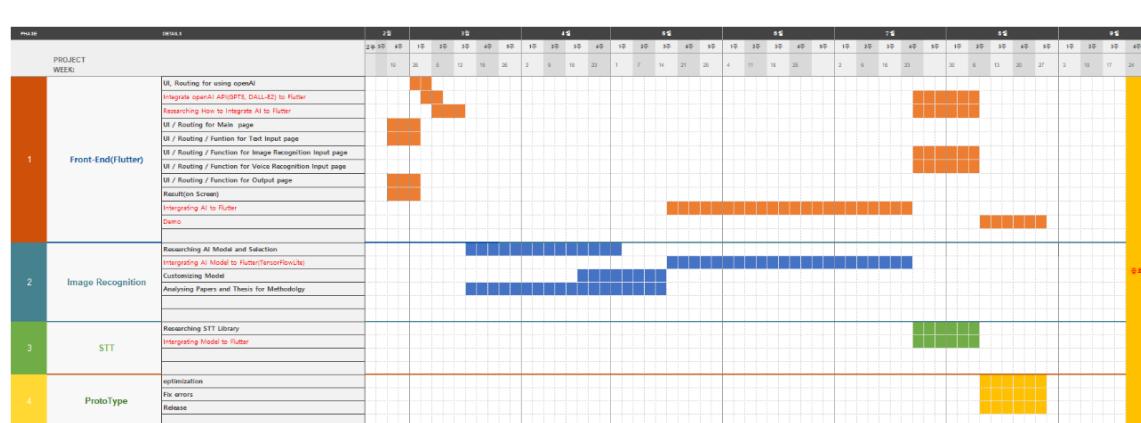
+ Voice input page 구현

~2023년 8월 29일 : 개선 및 최적화

+ 사용자 피드백을 기반으로 한 버그 수정 및 기능 개선

+ 성능 최적화 및 사용자 경험 향상

+ 구현 완료

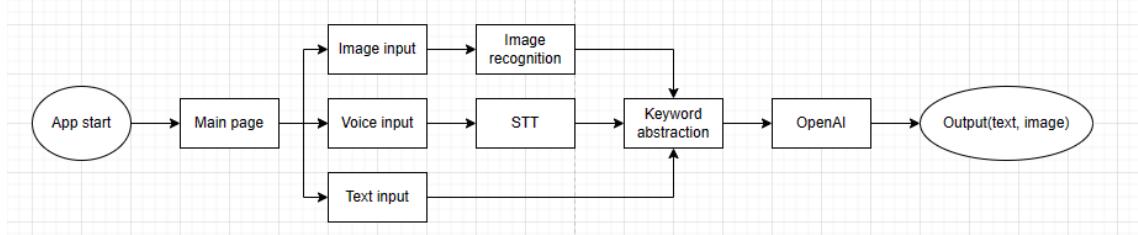


<Fig.1-1. 프로젝트 일정표>

II. 본론

1. 프로젝트의 구성 도면

- 이 프로젝트는 Flutter 앱 프론트엔드를 통해 이미지 인식, 음성인식(STT), 텍스트로 식재료를 입력 받아 OpenAI의 GPT-3, DALL-E를 활용하여 레시피를 제공한다. 이에 대한 Flow chart는 다음과 같다.



<Fig.2-1. Flow chart>

2. 식재료 이미지 인식

(1) 기술 및 선택 이유

- 이미지 인식을 위해 Xception, VGG19, ResNet50, MobileNet, MobileNetV2 모델들을 비교 분석, 관련 논문들을 연구, 정확도 비교를 한 후, MobileNetV2 아키텍처를 선별, 활용하여 식재료를 정확하게 분류할 수 있도록 하였다. MobileNetV2는 높은 정확도와 모바일 환경에서의 빠른 연산 속도가 장점이며, 이는 모바일 앱을 기반으로 둔 이 프로젝트에 적합하다.

(2) 문헌 조사

- 각 논문에서 얻은 주요 통찰과 지식은 프로젝트의 핵심 기술인 이미지 인식에 활용되었다.

[1] “Food Recipe Recommendation Based on Ingredients Detection Using Deep Learning” : 이 논문은 식재료 인식에 사용한 데이터셋의 구성과 라벨링 방법, 머신러닝에 사용할 모델들에 중 ResNet50 대한 유용한 지침을 제공하였다. 또한, 기존 기술의 시스템 구성 요소를 파악하여 이 프로젝트와 비교할 수 있었다.

[2] “Recipe Recommendation Method by Similarity Measure with Food Image Recognition” : 적절한 식재료 검출을 위한 가중치 벡터에 대한 참조가 되었으며, WebGL 기반 환경에서만 동작한다는 기존 시스템의 한계점을 확인 및 차별점으로 두게 되었다.

[3] “Food Ingredients Recognition Through Multi-label Learning” : CNN(Convolutional Neural Networks)을 사용한 특징 추출 및 Multi-label 분류 알고리즘 방법론을 학습하여 이 프로젝트에 적용하는 것을 고려하였다. Food-101, Ingredients101 등의 데이터셋을 확인 하였으며 sigmoid activation, binary cross-entropy loss에 대한 지식을 습득하였다. InceptionV3 모델을 사용한 점을 참고하여 본 연구에서는 Xception을 채택하게 되었다.

[4] “FROM MARKET TO DISH: MULTI-INGREDIENT IMAGE RECOGNITION FOR PERSONALIZED RECIPE RECOMMENDATION” : 이 논문에서는 마트에서 찍은, 실생활에서의 사진들에 대한 인식을 시도한다. 이는 비닐 포장에 의한 빛 반사, 과다 노출, 포장에 의한 객체 가려짐 등 발생 가능한 인식 장애에 대한 해결책으로 SRN(Spatial Regularization Network) 기법을 제시하였고, 이에 대한 고찰은 본 프로젝트에서 생길 비슷한 문제점에 대한 가이드 라인이 되었다.

[5] “AIFood: A Large Scale Food Images Dataset for Ingredient Recognition” : 이 논문에서는 이미지 인식에서 휴대폰 카메라로 촬영시 발생할 수 있는 문제점인 다양한 환경, 색 온도, 밝기 문제들을 데이터 전처리 과정에서 극복할 수 있는 방향성을 제시해 주었다. AWB(Automatic White Balancing), CLAHE(Contrast Limited Adaptive Histogram Equalization)을 통해 너무 밝거나 어두운 사진을 제어한다.

[6] “MobileNetV2: Inverted Residuals and Linear Bottlenecks” : MobileNetV1에서 Depthwise Separable Convolution 개념을 통해 연산량과 모델 사이즈를 줄일 수 있었고 그 결과 모바일 디바이스와 같은 제한된 환경에서도 사용하기에 적합한 뉴럴 네트워크를 제시한 것에 의의가 있었다. MobileNetV2에서는 기존 구조와 더불어 데이터를 고차원에서 저차원으로 손실없이 저장하는 아이디어를 기반으로 Inverted Residual 구조를 사용하였다는 차이점이 있으며, 이러한 네트워크 설계를 통해 효율적임과 동시에 성능을 보존했다. 연구 후 MobileNetV1, MobileNetV2을 이 프로젝트에 사용했다.

[7] “NOISE OR SIGNAL: THE ROLE OF IMAGE BACKGROUNDS IN OBJECT RECOGNITION” : 훈련 데이터의 배경 구성에 대한 고찰에 솔루션이 되었다. ImageNet 데이터를 함축 시킨 데이터를, 그 배경을 Grabcut, Rectangular Bounding Box 메소드를 통해 7개의 방법으로 합성하여 데이터셋을 구성하였다. 이 데이터셋을 통해 다양한 모델들이 이미지 분류를 할 때의 배경 의존도에 대한 실험이 이루어졌다.

[8] ““GrabCut” — Interactive Foreground Extraction using Iterated Graph Cuts” : 위 논문[7]에 대한 이해를 돋기 위하여 연구, 참고 하였다.

(3) 구현 과정

- 현재까지 36개에 대한 야채 및 과일 이미지를 train, validation, test 데이터 순으로 각각 100, 10, 10개씩 수집하고 라벨링을 진행하였다.
- kaggle, AI-Hub, Github, Web Crawling을 통해 데이터를 수집하였다.
- 모델 학습 용 이미지 데이터의 배경구성과, 실제 앱 서비스 사용 시 예상되는 입력 이미지 데이터의 배경 구성의 차이가 분류 적중률에 영향을 끼칠 지에 대한 연구 및 실험을 진행하였다. 배경이 모델의 이미지 인식에 영향을 끼치는 것으로 파악이 되었다. 다양한 배경의 학습 데이터가 획일화 되지 않은 배경의 입력 데이터에 대해 강인하다고 판단되어 위를 바탕으로 데이터 수집하였다.
- 위 데이터를 토대로 Xception, VGG19, ResNet50, MobileNet, MobileNetV2, 모델을 학습, 정확도 비교 끝에 최종적으로 MobileNetV2 모델이 약 97%의 정확도를 달성하였다.

(4) 모바일 환경 적용

- TensorFlowLite(TFLite)를 사용하여 훈련된 모델을 모바일 환경에서 실행 가능한 형태로 변환하였다.
- TFLite는 개발자가 모바일, 내장형기기, IoT기기에서 모델을 실행할 수 있도록

지원하여 기기 내 머신러닝을 사용할 수 있도록 하는 도구 모음이다.

- TFLite는 TFLite Converter와 TFLite Interpreter로 구성되어 있다.

- TFLite Converter는 TensorFlow 모델을 Interpreter가 사용할 수 있도록 최적화하는 역할을 하며 효율적이고, 적은 용량, 성능은 유지하도록 한다.

- TFLite Interpreter은 최적화된 모델을 다양한 하드웨어에서 구동 될 수 있도록 도와주는 역할로 모바일폰, 임베디드 리눅스 디바이스, 마이크로 컨트롤러 등에서 동작한다.

3. 음성 인식 및 텍스트 처리

(1) 기술 및 선택 이유

- 사용자의 음성 명령을 인식하기 위해 STT(Speech-to-Text) 기술을 사용하였다. 이는 사용자에게 다양한 입력 방법을 제공하기 위한 선택이었다.

(2) 구현 과정

- Flutter와 호환되는 STT 라이브러리를 선별하여 앱에 적용하였다.

4. 레시피 추천

(1) 기술 및 선택 이유

- 기존에 존재하였던 레시피 추천 플랫폼들은 서버-데이터베이스(DB) 시스템을 구축하여, DB에 이미 저장된 레시피를 서버 요청에 의해 호출하는 방식이었다. 이는 백엔드 시스템 유지 비용 및 제한적인 레시피 제공이라는 필연적인 문제점을 가지고 있다.

- OpenAI의 GPT-3를 사용하여 인식된 식재료를 기반으로 레시피를 생성하였다. GPT-3는 높은 자연어 처리 성능을 보이므로 이를 활용하기에 충분하다고 판단되며, 제한적이지 않은 다양한 선택지와 보다 창의적인 솔루션을 제공해 줄 것으로 기대된다.

(2) 구현 과정

- 사용자가 제공한 식재료 정보를 GPT-3 API에 전달하여 레시피를 생성하도록 하였다.

5. 시각적 피드백

(1) 기술 및 선택 이유

- 생성된 레시피에 대한 시각적 피드백을 제공하기 위해 DALL-E를 사용하였다.

(2) 구현 과정

- GPT-3에서 생성된 레시피 중 첫번째 레시피를 DALL-E API에 전달하여 생성된 이미지를 받아 화면에 표출하도록 하였다.

III. 실험 및 결과 고찰

1. 데이터 수집 및 라벨링

- 36가지 야채, 과일에 대한 이미지 데이터를 수집하고 라벨링을 진행하였다.
다음은 수집 방식 중 하나였던 Web Crawling에 대한 코드 및 결과에 대한 내용이다.

```
import os
import shutil
from bing_image_downloader.bing_image_downloader import downloader

classes_list = ['apple', 'banana', 'beetroot', 'bell pepper', 'cabbage', 'capsicum', 'carrot', 'cauliflower',
                'chilli pepper', 'corn', 'cucumber', 'eggplant', 'garlic', 'ginger', 'grapes', 'jalapeno',
                'kiwi', 'lemon', 'lettuce', 'mango', 'onion', 'orange', 'paprika',
                'pear', 'peas', 'pineapple', 'pomegranate', 'potato', 'radish', 'soy beans',
                'spinach', 'sweetcorn', 'sweetpotato', 'tomato', 'turnip', 'watermelon']

query_1 = ' as a ingredient'
query_2 = ' with white background'
query_list = [query_1, query_2]

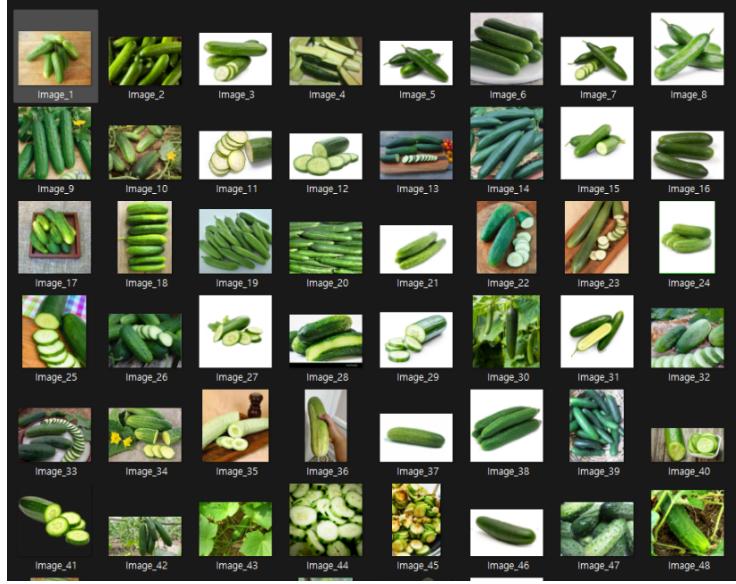
for directory in directory_list:
    if not os.path.isdir(directory):
        os.makedirs(directory)

def dataset_split(query, train_cnt, val_cnt):
    for directory in directory_list:
        if not os.path.isdir(directory + '/' + query):
            os.makedirs(directory + '/' + query)
    cnt = 0
    for file_name in os.listdir(query):
        if cnt < train_cnt:
            print(f'[Train Dataset] {file_name}')
            shutil.move(query + '/' + file_name + query + '/' + file_name)
        elif (train_cnt + val_cnt) > cnt and cnt > val_cnt:
            print(f'[Validation Dataset] {file_name}')
            shutil.move(query + '/' + file_name + query + '/' + file_name)
        else:
            print(f'[Test Dataset] {file_name}')
            shutil.move(query + '/' + file_name + query + '/' + file_name)
        cnt += 1
    shutil.rmtree(query)

    for keyword in classes_list:
        for c_query in query_list:
            query = keyword+c_query
            downloader.download(query, limit=60, output_dir='./', adult_filter_off=True, force_replace=False, timeout=60)
            dataset_split(query, 100, 10)
```

<Fig.3-1. WebCrawler>

- bing_image_downloader 패키지를 이용하였다. 원하는 라벨들이 담긴 classes_list변수를 만들고 추가 문장을 query_list로 만든 다음 이것들을 query로 하여 download함수의 인자 값으로 넣는다. 이때 크롤링한 이미지들을 각각의 폴더에 넣어주기 위해 dataset_split함수를 구현하여 사용하였다.



<Fig.3-2. 데이터셋 모습 예시 (cucumber)>

2. 다양한 모델 파인 티닝

(1) 데이터 전처리

- 수집한 데이터들을 모델에 훈련시킬 수 있도록 전처리 하였다.

```
# GPU checking
device_names = tf.test.gpu_device_name()
device_names

batch_size = 32
#image_size = 256
target_size = (256,256)
input_shape = (256, 256, 3)

#Fetching train data and validation data and processing the data
train_datagen = ImageDataGenerator(rescale = 1.00 / 255.0)
val_datagen = ImageDataGenerator(rescale = 1.00 / 255.0)
test_datagen = ImageDataGenerator(rescale = 1.00 / 255.0)

train_dir = ''
test_dir = ''
val_dir = ''

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size = target_size,
    batch_size = batch_size

)

validation_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size = target_size,
    batch_size = batch_size
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size = target_size,
    batch_size = batch_size
)
```

<Fig.3-3. 데이터 전처리>

(2) 모델 학습

- 모델 학습을 위한 다양한 시도 끝에 다음과 같은 구성으로 파인 티닝을 진행하였다.

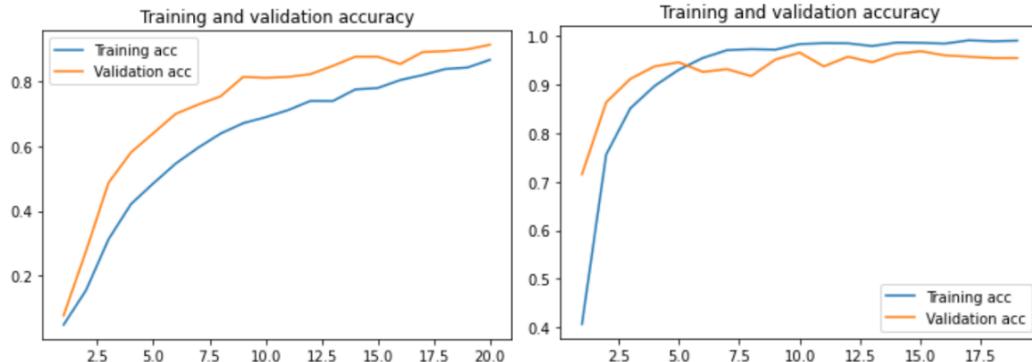
```
# add new classifier layers
flat1 = model_mobilenetV2.layers[-1].output
flat1 = GlobalAveragePooling2D()(flat1)
class1 = layers.Dense(1024, activation='relu')(flat1)
class1 = layers.Dense(512, activation='relu')(class1)
class1 = layers.Dense(256, activation='relu')(class1)
# class1 = layers.Dense(128, activation='relu')(class1)
# class1 = layers.Dense(64, activation='relu')(class1)
output = layers.Dense(36, activation='sigmoid')(class1)
# define new model
modelMobileNetV2 = Model(inputs=model_mobilenetV2.inputs, outputs=output)
modelMobileNetV2.compile(
    loss = "binary_crossentropy",
    optimizer="adam",
    metrics = ["acc"]
)
# summarize
modelMobileNetV2.summary()

#Training the model with train data and judging this training with validation data
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
historyMNV2 = modelMobileNetV2.fit(
    train_generator,
    batch_size=batch_size,
    epochs = 25,
    validation_data = validation_generator,
    callbacks=[es],
    verbose=1)
```

〈Fig.3-4. 모델 학습 코드〉

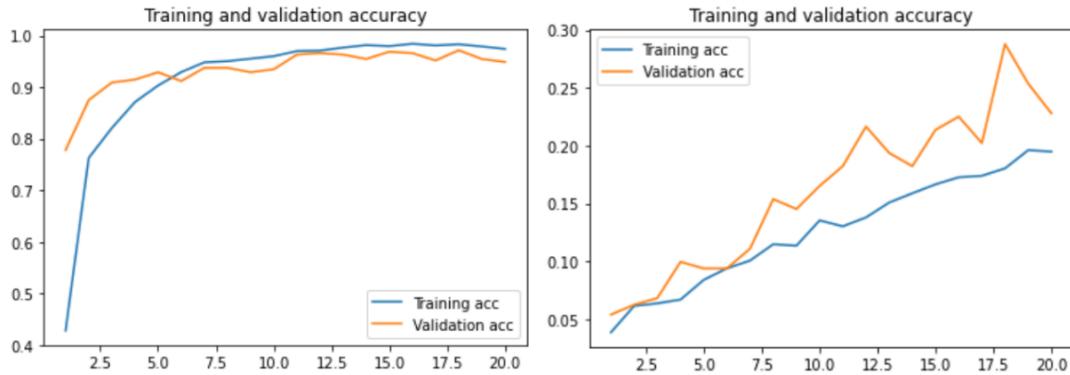
(3) 정확도 비교

- 위의 데이터 및 모델 구조 조건으로 VGG19, MobileNet, Xception, ResNet50, MobileNetV2 모델들의 학습을 하였고 정확도를 지표로 하여 비교하였다. 그 결과 MobileNetV2를 이 프로젝트의 이미지 분류 모델로써 활용하기로 결정하였다.



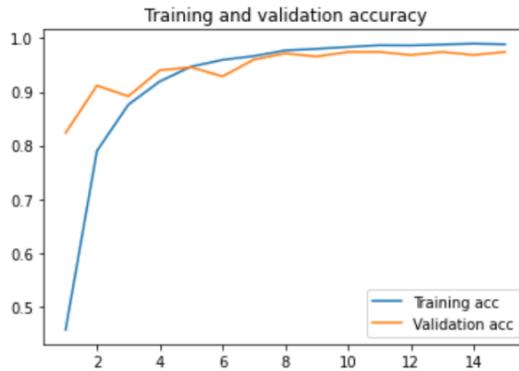
〈Fig.3-5. VGG19 Accuracy〉

〈Fig.3-6. MobileNet Accuracy〉



〈Fig.3-7. Xception Accuracy〉

〈Fig.3-8. ResNet50 Accuracy〉



〈Fig.3-9. MobileNetV2 Accuracy〉

Model		loss	acc
VGG19		0.0161	0.9136
MobileNet		0.0105	0.9554
Xception		0.0087	0.9471
ResNet50		0.0963	0.2284
MobileNetV2		0.0059	0.9747

〈Table.3-1. Models ' Test Accuracy〉

3. 입력 데이터에 대한 실험

(1) 입력 데이터 규격에 대한 고찰

- 이 시스템에 실제로 입력될 데이터는 냉장고에서 꺼낸 직후의 식재료의 모습일 것으로 가정할 수 있다. 그랬을 때 예상되는 모습은 육류, 어류, 햄, 우유, 치즈와 같이 어떠한 포장물이나 용기에 담겨있는 것들, 가공 전 상태의 모습일 것이며 그 배경 적 맥락은 도마위, 식탁위, 부엌 등으로 예상될 것이다.

- 이때 고려할 수 있는 요소들은 다음과 같다.
- + 저장 형태에 따라 달라지는 모습 ex) 썰어둔 감자, 채썬 야채, 다진 고기
- + 지역에 따라 달라지는 식재료 ex) 한국의 참외와 외국의 멜론, 배 등
- + 비슷하게 생긴 식재료들 ex) 돼지고기-소고기, 파프리카-피망 등
- + 애초에 포장 용기에 담긴 것들 ex) 복숭아캔, 우유, 치즈 등 브랜드에 따라 걸 포장지가 달라짐.

- 그에 따른 해결책들은 다음과 같이 정리하였다. 이는 추후 보완해야 할 사항들이다.
 - + 저장 형태에 따라 달라지는 모습 -> 해당 라벨을 따로 만들어 둔다
 - + 지역에 따라 달라지는 식재료 -> 사용자의 피드백을 받는 페이지를 만들어 요청 사항 발생 시 추가한다.
 - + 비슷하게 생긴 식재료들 -> 분류 결과 비슷한 확률의 라벨이 2개 이상 확인될 경우, 사용자가 선택할 수 있도록 하는 페이지를 만든다.
 - + 애초에 포장 용기에 담긴 것들 -> 추후 더 깊이 있는 연구와 사례 조사를 통해 해결한다.

(2) 이미지의 배경에 대한 실험 : 연구 및 분석

- 실제 입력 이미지의 배경 적 맥락은 도마위, 식탁위, 부엌 등으로 예상된다. 따라서 그 배경은 부엌, 식탁의 색으로 주로 사용되는 검정색, 나무색, 하얀색으로 좁힐 수 있을 것이다. 그렇다면 학습 데이터를 위의 가정과 같거나 비슷한 배경을 지닌 데이터로 한정한다면, 실제 시스템 사용에서 더 만족스러운 정확성을 가질 수 있지 않을까 하는 의문이 들었고 이에 대한 연구 및 실험을 진행하였다.
- [7]Kai Xiao et al. 에서 이미지 분류 모델이 이미지 배경에서 오는 signal의 의존적인 정도를 실험하였다. ImageNet dataset class를 단순하게 통합하여 총 9개의 class를 갖는 dataset을 생성 및 이미지 합성을 하였다. 이 데이터를 Rectangular Bounding Box, GrabCut을 통해 7개의 방법으로 합성하여 하나의 이미지에 대해 총 8개의 이미지를 만들었다.



<Fig.3-10. 7 ways in the paper>

- 위 데이터를 활용하여 여러가지 방법론을 통해 학습 및 정확도 평가가 진행되었다. 이에 따른 결론은 다음과 같다.
 - + 모델들은 전혀 다른 배경의 변화에 따라 성능이 좌지우지 될 수 있다.
 - + 그러나 맥락적인 정보의 관점에서 변인을 통제한다면 배경은 여전히 유용할 수 있다.
 - + 분류할 객체가 흐릿하거나, 인식하기 어려운 상황에서 배경이 도움이 될 수 있다.
 - + classifier 향상을 통해 모델이 객체를 더 잘 찾고, 배경의 변화에 강인하게 할 수 있다.
 - + 배경의 변화에 더욱 강인한 모델을 얻기 위해서는, Mixed-Rand dataset을 즉, 배경의 맥락이 객체와 연관 없는 데이터를 이용하는 것을 고려 할 수 있다.
 - 위와 같은 결론을 토대로, 이 프로젝트와 같이 배경을 통제할 수 없는 상황이라면, 훈련 데이터의 배경을 다양하게 구성하는 것이 더 높은 분류 정확도를 갖게 될 것이라고 예상을 하였고 이를 증명하기 위한 실험을 진행하였다.

(3) 이미지의 배경에 대한 실험 : 실험

- 다음과 같은 경우로 나누어 여러 분류 모델들의 정확도 경향성을 파악 및 비교한다. 사용 모델 : VGG19, MobileNetV2, Xception

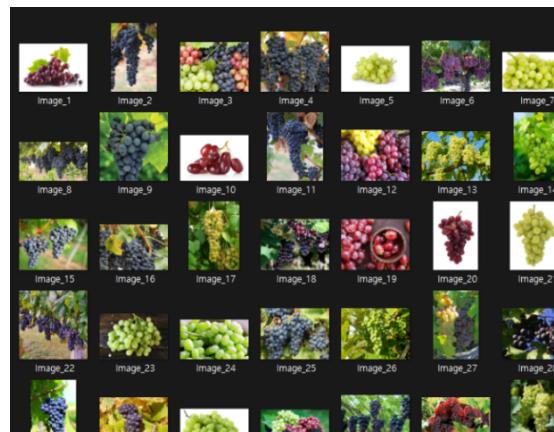
	train	test
case1	Various-BG	Various-BG Sorted-BG
case2	Sorted-BG	Sorted-BG Various-BG

〈Table.3-2. Experiment Table〉

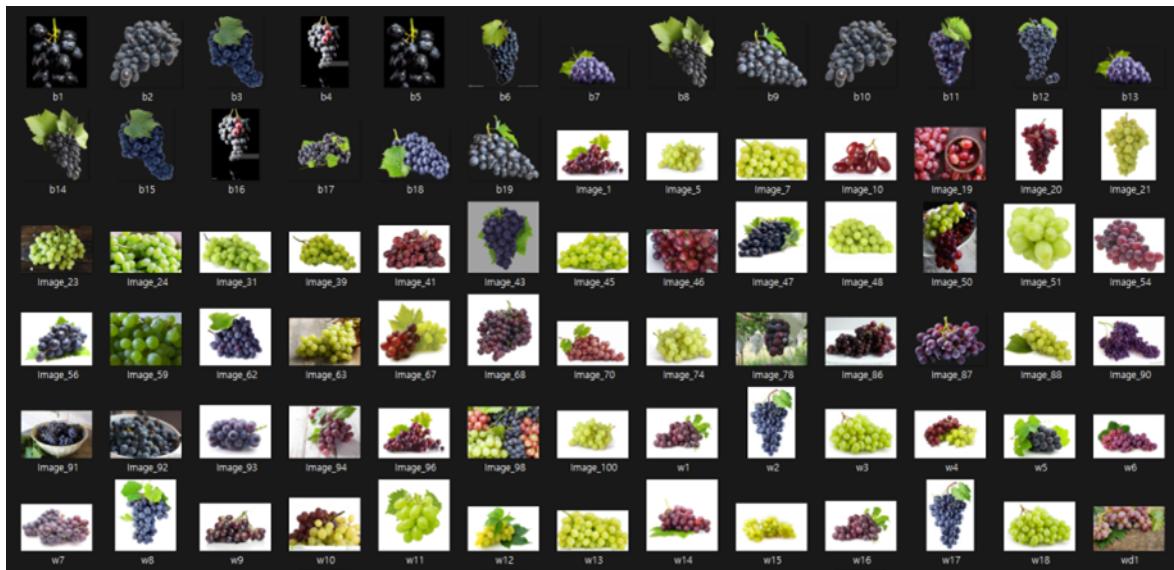
- 데이터셋은 36개중 8개의 라벨을 선별하여 train 50, validation 5, test 5로 구성하였다. 이때 case1의 데이터들은 기존 데이터셋을 사용하였으며 그 배경은 자연, 하얀색 등이다. case2의 데이터들은 기존에 사용하였던 〈Fig.3-1.Web Crawler〉 code를 응용하여 수집하였으며 그 배경은 검은색, 하얀색, 나무색이다.

```
># classes_list = ['apple', 'banana', 'beetroot', 'bell pepper', 'cabbage', 'capsicum', 'carrot', 'cauliflower', ...
# # test_list = ['cucumber']
classes_list=['apple', 'banana', 'grapes', 'kiwi', 'lemon', 'mango', 'orange', 'watermelon']
query_white = ' with white background'
query_black = ' with black background'
query_wooden = ' with wooden background'
query_list = [query_white, query_black, query_wooden]
```

〈Fig.3-11. Crawler 추가된 코드〉

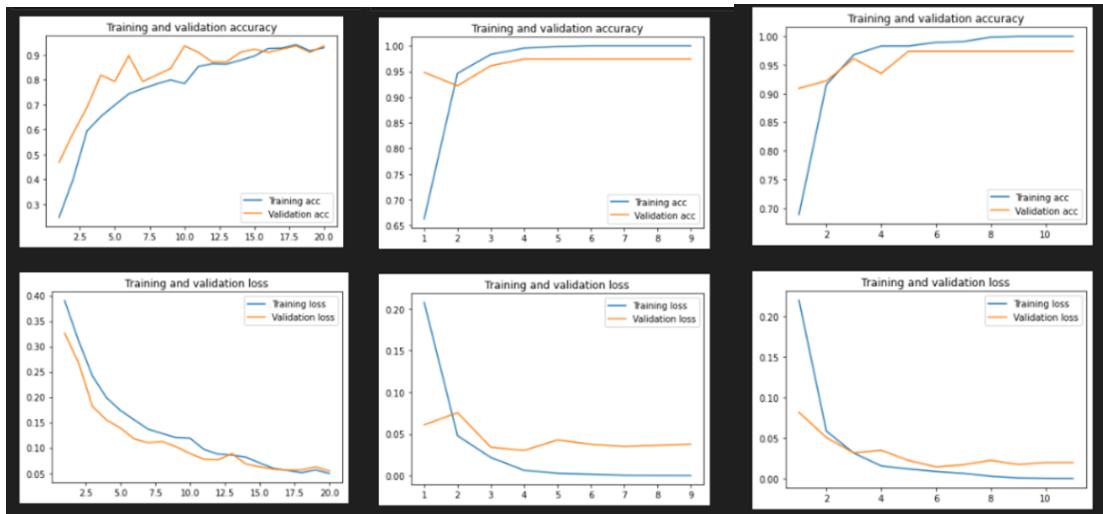


〈Fig.3-12. case1 데이터의 모습 예 (grapes)〉

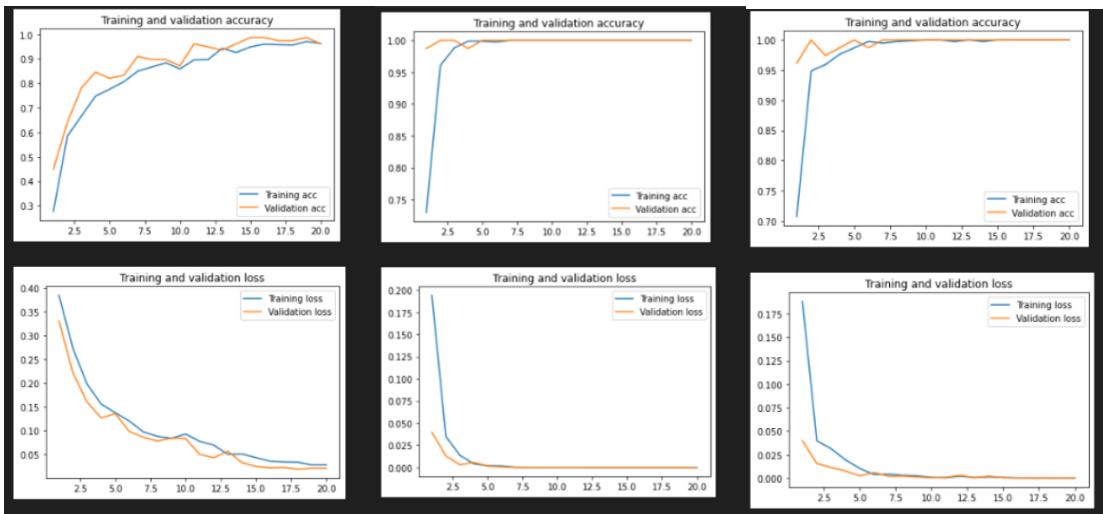


<Fig.3-13. case2 데이터셋의 모습 (grapes)>

- 이러한 데이터셋으로 VGG19, MobileNetV2, Xception 모델들을 학습시켰으며 그 결과들은 다음과 같았다. 그래프 순서는 VGG19, MobileNetV2, Xception 이다.



<Fig.3-14. case1 results>



<Fig.3-15. case2 results>

(4) 결과 비교 및 분석

test model	case1				case2			
	various	sorted	sorted	various	loss	acc	loss	acc
loss	acc	loss	acc	loss	acc	loss	acc	
VGG19	0.0534	0.9367	0.0594	0.9367	0.01	0.9873	0.0848	0.8987
MobileNetV2	0.0368	0.9747	0.0287	0.962	2.38E-05	1	0.0402	0.9747
Xception	0.0192	0.9747	0.0086	1	2.47E-05	1	0.0287	0.9747

<Table.3-3. Experiment Result>

- 다양한 배경으로 학습된 모델들은 그보다 더 단순한 배경에도 정확도를 유지하는 경향성을 보인다. 그러나 그와 대조되는 단순한 배경으로 학습된 모델은 단순한 배경의 테스트 데이터를 완벽에 가깝게 분류해내는 반면, 다양한 배경의 데이터에는 정확도가 떨어지는 경향성을 보여준다.
- 실제 입력될 데이터에 대한 강력한 통제가 이루어 지지 않는 이상 더 다양한 배경의 학습 데이터를 구성하는 것이 바람직하며, 이 프로젝트에서는 훈련 데이터로써 다양한 구성의 배경의 이미지를 사용하는 것이 옳다고 결론 내었다.

4. TensorFlowLite(TFLite)

(1) 개발 진행 순서

- TFLite의 개발 진행 순서는 다음과 같이 정리 된다.
 - + 모델 훈련: TensorFlow를 사용하여 개발 기기나 서버에서 머신러닝 모델을 훈련한다. 이 결과는 일반적으로 모델 형식에 따라 .pb 또는 .h5 파일이 생성된다.
 - + 모델 변환: 모델이 훈련되면 TensorFlow Lite Converter를 사용하여 TFLite 형식으로 변환해야 한다. 이 과정에서는 모델이 장치에서의 속도와 크기를 최적화하는 형식으로 변환된다. 이 과정의 결과는 .tflite 파일이다.
 - + 모델 통합: .tflite 파일이 생성되면 이를 모바일 또는 임베디드 애플리케이션에 통합할 수 있다. TFLite는 Java, Swift, C++, Python 등 다양한 언어에 대한 API를 제공하며, Android 및 iOS와 같은 인기 있는 플랫폼을 지원한다.
 - + 모델 추론: 앱에서는 TFLite 인터프리터를 사용하여 모델을 불러오고, 이를 통해 추론을 실행하고 결과를 해석한다. 모델에 입력하는 데이터는 모델이 예상하는 형식이어야 하며, 출력은 모델이 설계된 바에 따라 해석되어야 한다.
 - + 하드웨어 가속: TFLite는 특정 장치에서 하드웨어 가속을 지원한다. 이는 장치가 머신러닝 작업을 가속화할 수 있는 하드웨어(예: GPU 또는 Neural Processing Unit)를 가지고 있다면, TFLite가 이를 활용해 추론을 더 빠르게 실행할 수 있다는

것을 의미한다.

(2) 개발

- 앞서 훈련된 기존 모델을 변환하여 진행하였다.

```
import tensorflow as tf

# Load the model
model = tf.keras.models.load_model('./models/modelMobileNetV2.h5')

# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model.
with open('model.tflite', 'wb') as f:
    f.write(tflite_model)

[13] ✓ 28.1s
```

... WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_conv
INFO:tensorflow:Assets written to: C:\Users\YOUNGH~1\AppData\Local\Temp\tmp_xd3axr4\assets
INFO:tensorflow:Assets written to: C:\Users\YOUNGH~1\AppData\Local\Temp\tmp_xd3axr4\assets

Python

<Fig.3-16. TFLite Converting>

- 분류에 사용될 라벨들을 txt파일로 만들어 Flutter 내에서 참조할 수 있도록 하였다.

```
classes_list = ['apple', 'banana', 'beetroot', 'bell pepper', 'cabbage', 'capsicum', 'carrot', 'cauli  
'chilli pepper', 'corn', 'cucumber', 'eggplant', 'garlic', 'ginger', 'grapes', 'jalep  
'kiwi', 'lemon', 'lettuce', 'mango', 'onion', 'orange', 'paprika',  
'pear', 'peas', 'pineapple', 'pomegranate', 'potato', 'radish', 'soy beans',  
'spinach', 'sweetcorn', 'sweetpotato', 'tomato', 'turnip', 'watermelon']

def write_data(filename, data):
    with open(filename, 'w') as f:
        for line in data:
            f.write(line + '\n')
    write_data('labels.txt', classes_list)

[1] ✓ 0.0s
```

```
// Load labels from assets
Future<void> loadLabels() async {
    final labelTxt = await rootBundle.loadString(labelsPath);
    labels = labelTxt.split('\n');
}
```

<Fig.3-17. labels to txt>

- 모델 통합: Flutter의 tflite_flutter 라이브러리의 관련 문서 및 google 관련 서적, TensorFlow, stackoverflow 등을 연구 및 참조하여 모델 통합을 진행하였다. 모델 통합의 구동 순서는 다음과 같다.

- + 갤러리, 카메라를 통한 이미지 입력
- + 이미지의 픽셀 읽음 및 디코딩
- + 모델에 입력하기 위한 resize(256, 256)

- + (1, 256, 256, 3) 크기의 matrix 생성
- 이를 구현한 코드는 다음과 같다.

```

Future<void> processImage() async {
  if (imagePath != null) {
    // Read image bytes from file
    final imageData = File(imagePath!).readAsBytesSync(); // List<int>
    // print(imageData.shape);

    // Decode image using package:image.dart (https://pub.dev/)
    image = img.decodeImage(imageData);
    // print(image);
    setState(() {});

    // Resize image for model input (Mobilenet use [256, 256])
    final imageInput = img.copyWithResize(
      image!,
      width: 256,
      height: 256,
    );

    // Get image matrix representation [256, 256, 3]
    final imageMatrix = List.generate(
      imageInput.height,
      (y) => List.generate(
        imageInput.width,
        (x) {
          final pixel = imageInput.getPixel(x, y);
          return [pixel.r, pixel.g, pixel.b];
        }
      )
    );
  }
}

```

<Fig.3-18. TFLite 모델 통합>

- 모델 추론 :변환 시킨 .tflite 파일과 label.txt를 interpreter로써 할당시킨뒤 이를 통해 나온 output 을 표시해보면 다음과 같다. 이때 모든 라벨에 할당된 값 중 가장 큰 값이 할당된 라벨을 추출하기 위해 argmax 함수를 구현 및 적용 하였다.

```

// Run inference
Future<void> runInference(List<List<List<num>>> imageMatrix) async {
  // Set tensor input [1, 256, 256, 3]
  final input = [imageMatrix];

  // Set tensor output [1, 36]
  final output = [List<double>.filled(36, 0)];

  // Run inference
  interpreter.run(input, output);

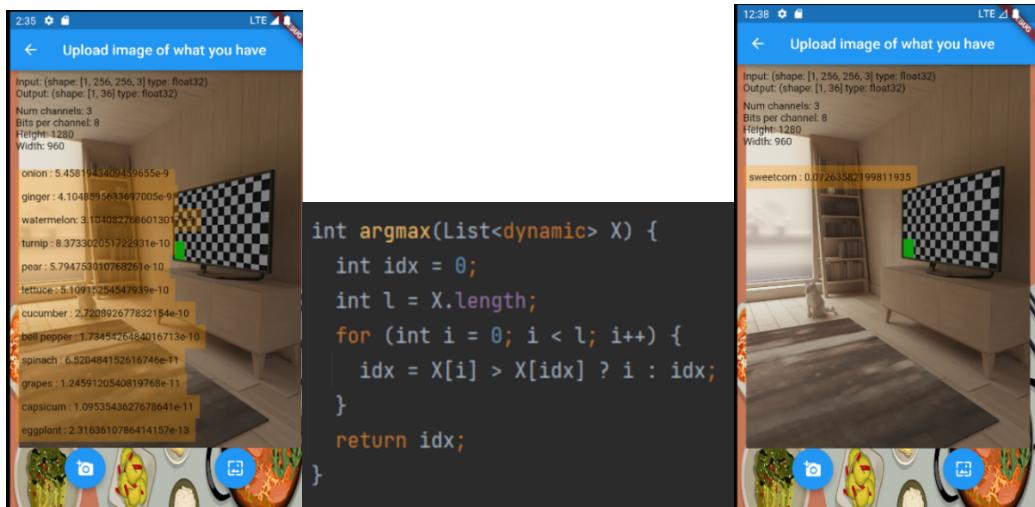
  // Get first output tensor
  // result : [probabilities]
  final result = output.first;

  // argmax result
  // resultIndex : The biggest probability Index
  final resultIndex = argmax(result);

  // label : classified ingredient
  final label = labels[resultIndex];
  showConfirmationDialog(File(imagePath!), label);
  setState(() {});
}

```

<Fig.3-19. interpreter>



<Fig.3-20. argmax함수 적용 전과 후>

- 하드웨어 가속 : 이후 하드웨어 가속은 다음과 같은 코드로 구현하였다.

```
// Load model
Future<void> loadModel() async {
    final options = InterpreterOptions();

    // Use XNNPACK Delegate
    if (Platform.isAndroid) {
        options.addDelegate(XNNPackDelegate());
    }

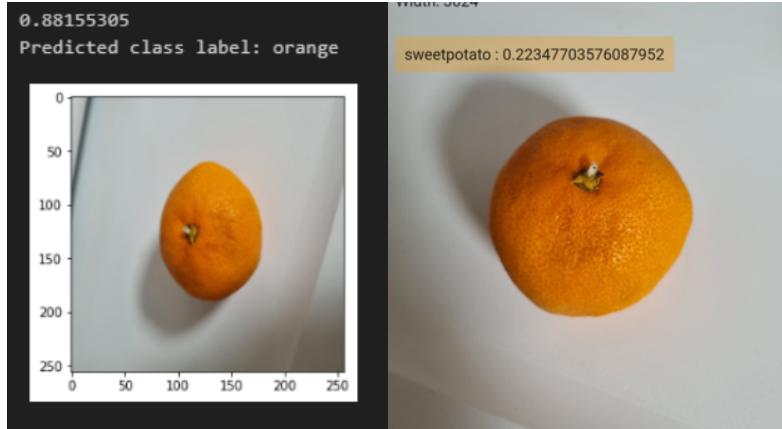
    // Use GPU Delegate
    // doesn't work on emulator
    if (Platform.isAndroid) {
        options.addDelegate(GpuDelegateV2());
    }

    // Use Metal Delegate
    if (Platform.isIOS) {
        options.addDelegate(GpuDelegate());
    }
}
```

<Fig.3-21. 하드웨어 가속>

(3) 결과 확인

- 실제 사진으로 tflite변환 전 모델과 변환 후 모델에서 테스트를 진행하였다. 변환 전에는 잘 인식하던 것과 달리 변환 후 인식률이 저하된 것을 확인할 수 있었다.



<Fig.3-22. 인식률 비교>

- 이와 같은 성능 저하 원인의 경우를 고민해본 결과 총 4가지 가능성이 존재했다.
 - + 앱에서 라벨 순서 오 인식
 - + 앱에서 interpreter 설정 오류
 - + 모델 변환 과정에서 오류
 - + 모델 최적화 과정에서 큰 손실 발생

(4) 정확성 저하 문제 고찰

- 우선 라벨과 interpreter부분의 코드를 점검해 보았지만 이상이 없다고 판단되었다. 그렇다면 모델을 변환하는 과정 또는 최적화 과정에서 손실이 발생했다는 것으로 유추할 수 있다.
 - 모델 변환에서 오류가 발생 했다고 가정하고, 이것을 해결하기 위해 TensorFlowLite 공식 문서를 참고하여 진행하였고 이를 해결하고자 변환 방식에 약간의 변화를 주었다.

Python API

도우미 코드 설치된 TensorFlow 버전을 확인하려면 `print(tf.__version__)`를 실행하고 TensorFlow Lite 변환기 API에 관해 자세히 알아보려면 `print(help(tf.lite.TFLiteConverter))`를 실행합니다.

TensorFlow 2.x를 설치한 경우 옵션에는 다음 두 가지가 있습니다. [TensorFlow 1.x를 설치한 경우 GitHub](#)를 참고하세요.

- `tf.lite.TFLiteConverter` 를 사용하여 TensorFlow 2.x 모델을 변환합니다. TensorFlow 2.x 모델은 저장된 모델 형식을 사용하여 저장되며 상위 수준 `tf.keras.*` API(Keras 모델) 또는 하위 수준 `tf.*` API(여기서 구체적 함수 생성)를 사용하여 생성됩니다. 따라서 다음 세 가지 옵션이 있습니다. 예는 다음 몇 가지 색션을 참고하세요.
- `tf.lite.TFLiteConverter.from_saved_model()` (권장): 저장된 모델을 변환합니다.
- `tf.lite.TFLiteConverter.from_keras_model()`: Keras 모델을 변환합니다.
- `tf.lite.TFLiteConverter.from_concrete_functions()`: concrete 함수를 변환합니다.

<Fig.3-23. 모델 변환의 방법들>

- 기존에 사용했던 방법은 `.from_keras_model()` 함수를 이용하는 것으로 파일 형식이 `.h5` 인 모델을 변환해 주는 방법이었다. `.from_saved_model()` 함수는 `.pb` 또는 `.pbtxt` 형식의 모델을 변환 해주는 차이점이 있으며 이 방법으로 시도해 보았으나 문제해결이 되지 않았다.

- 그렇다면 남은 가능성은 모델 최적화 과정에서 손실이 발생했다는 것인데, 이 모델 최적화 과정은 크게 3가지 방법이 있다.

- + 양자화
- + 잘라내기
- + 클러스터링

- 양자화는 모델의 가중치 및 활성화의 정밀도를 줄여 모델의 크기를 줄이고 저전력 장치에서 성능을 향상, 모델축소, 정확도 감소하게 한다. 양자화는 기본적으로 32bit 부동 소수점 숫자인 모델의 매개변수를 나타내는 데 사용되는 숫자의 정밀도를 줄여서 동작한다. 그 결과 모델 크기가 작아지고 계산 속도가 빨라진다.

- 잘라내기는 예측에 미미한 영향만 미치는 모델 내 매개변수를 제거하는 방식으로 동작한다. 잘라낸 모델은 디스크에서 크기가 같고 런타임 지연 시간이 같지만 더 효과적으로 압축할 수 있다. 따라서 잘라내기는 모델 다운로드 크기를 줄이는 데 유용한 기술이다.

- 클러스터링은 모델에 있는 각 레이어의 가중치를 미리 정의된 수의 클러스터로 그룹화한 다음 각각의 개별 클러스터에 속하는 가중치의 중심 값을 공유하는 방식으로 동작한다. 그 결과 모델의 고유한 가중치 값의 수가 줄어들어 복잡성이 줄어든다. 결과적으로 클러스터링된 모델을 보다 효과적으로 압축하여 잘라내기와 유사한 배포 이점을 제공할 수 있다.

- 양자화에는 2가지 방법이 있다.

- + 훈련 후 양자화
- + 양자화 인식 훈련

- 훈련 후 양자화에는 모델 정확성의 저하 없이 CPU 및 하드웨어 가속기의 지연 시간, 처리, 전력 및 모델 크기를 줄여 주는 일반적인 기술이 포함된다. 이러한 기술은 이미 훈련된 float TensorFlow 모델에서 수행할 수 있으며 TensorFlow Lite 변환 중에 적용할 수 있다. 이러한 기술은 TensorFlow Lite 변환기의 옵션으로 활성화된다.

- 양자화 인식 훈련은 추론 시간 양자화를 에뮬레이트하여 다운스트림 도구가 실제로 양자화된 모델을 생성하는 데 사용할 모델을 생성한다. 양자화된 모델은 낮은 정밀도(예: 32bit 부동 소수점 대신 8bit)를 사용하므로 배포 중에 이점이 있다.

- 훈련 후 양자화에는 또 세가지 방법이 있다.

- + 훈련 후 동적 양자화
- + 훈련 후 전체 정수 양자화
- + 훈련 후 float16 양자화

- 훈련 후 동적 양자화, 훈련 후 float16 양자화를 시도하였으나 정확도 손실은 여전하였다.

```

import tensorflow as tf

# Load the model
model = tf.keras.models.load_model('./models/mobileNetV2.h5')
model_dir = './models/forPBytype/saved_model.pb'

# Convert the model.
# converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter = tf.lite.TFLiteConverter.from_saved_model('./models/forPBytype')
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [tf.float16]
tflite_quant_model = converter.convert()
tflite_model = converter.convert()

# Save the model.
with open('quantized_model_16.tflite', 'wb') as f:
    f.write(tflite_quant_model)

✓ 11.7s

```

<Fig.3-24. 훈련 후 float16 양자화>

- 다른 방법들 중 하나인 양자화 인식 훈련을 진행하였다. 전체 모델에 양자화 인식 훈련을 적용하고 모델 요약에서 이를 확인한다. 이때 모든 레이어 앞에 “quant”가 붙는다.

```

import tensorflow_model_optimization as tfmot

quantize_model = tfmot.quantization.keras.quantize_model

# q_aware stands for quantization aware.
q_aware_model = quantize_model(model)

# 'quantize_model' requires a recompile.
q_aware_model.compile(optimizer='adam',
                      loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                      metrics=['accuracy'])

q_aware_model.summary()
✓ 74s

WARNING:tensorflow:From c:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\autograph\pyct\static_analysis\liver
Instructions for updating:
Lambda functions will be no more assumed to be used in the statement where they are used, or at least in the same block. Model: "model"

Layer (type)          Output Shape         Param #   Connected to
=====
input_5 (Inputlayer) [(None, 256, 256, 3  0
)]                   []
quantize_layer (QuantizeLayer) (None, 256, 256, 3) 3   ['input_5[0][0]']
quant_Conv1 (QuantizeWrapperV2 (None, 128, 128, 32  929
) )                  ['quantize_layer[0][0]']
quant_bn_Conv1 (QuantizeWrap (None, 128, 128, 32  129
perV2) )             ['quant_Conv1[0][0]']
quant_Conv1_relu (QuantizeWrap (None, 128, 128, 32  3
perV2) )             ['quant_bn_Conv1[0][0]']
quant_expanded_conv_depthwise (None, 128, 128, 32  291
(QuantizeWrapperV2) )  ['quant_Conv1_relu[0][0]']
quant_expanded_conv_depthwise_ (None, 128, 128, 32  129
... )                 ['quant_expanded_conv_depthwise[0
...
Total params: 4,255,285
Trainable params: 4,166,884
Non-trainable params: 88,401

```

<Fig.3-25. 양자화 인식 훈련>

- 양자화 인식 훈련된 모델과 기존 모델의 정확도를 비교하였을 때 양자화 인식 훈련된 모델의 정확도가 매우 낮다는 것을 확인 할 수 있었다.

```

q_aware_model.fit(train_generator,
                  batch_size=batch_size,
                  epochs=1,
                  validation_data=validation_generator)
    ✓ 7m 49.2s
c:\ProgramData\Anaconda3\lib\site-packages\PIL\Image.py:98: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  "Palette images with Transparency expressed in bytes should be "
98/98 [=====] - 468s 5s/step - loss: 0.2914 - accuracy: 0.0218 - val_loss: 0.3382 - val_accuracy: 0.0285
<keras.callbacks.History at 0x23d2a2c2cc8>

_, baseline_model_accuracy = model.evaluate(test_generator, verbose=0)
_, q_aware_model_accuracy = q_aware_model.evaluate(test_generator, verbose=0)

print('Baseline test accuracy:', baseline_model_accuracy)
print('Quant test accuracy:', q_aware_model_accuracy)
✓ 53.0s
Baseline test accuracy: 0.9749303460121155
Quant test accuracy: 0.027855154126882553

```

<Fig.3-26. 정확도 비교>

- 이를 해결하기 위하여 양자화 모델 훈련의 epoch수를 늘렸으나 결과는 같았다. 새로운 모델을 제작, 학습 시켜 위의 과정을 다시 한번 시도하여 그럼에도 결과가 같은지 확인하기 위해 다음과 같이 모델을 구성하여 재시도 하였다.

```

from tensorflow import keras
from keras import layers
from keras import models
from keras.layers import Dropout

model2=tf.keras.Sequential()

model2.add(layers.Conv2D(filters = 32, kernel_size=(3, 3), padding = "same", activation='relu', input_shape=input_shape))
model2.add(layers.MaxPooling2D(pool_size=(2, 2)))
# model2.add(Dropout(0.2))

model2.add(layers.Conv2D(64, kernel_size=(3, 3), activation='relu'))
model2.add(layers.MaxPooling2D(pool_size=(2, 2)))
# model2.add(Dropout(0.2))

model2.add(layers.Conv2D(128, kernel_size=(3, 3), activation='relu'))
model2.add(layers.Flatten())

model2.add(layers.Dense(128, activation='relu'))

# model2.add(Dropout(0.5))
model2.add(layers.Dense(36, activation='softmax'))

model2.compile(optimizer= "adam",loss='categorical_crossentropy',metrics=[ 'accuracy'])

history2 = model2.fit(
    train_generator,
    batch_size=batch_size,
    epochs=25,
    validation_data=validation_generator,
    callbacks=[es]
)

```

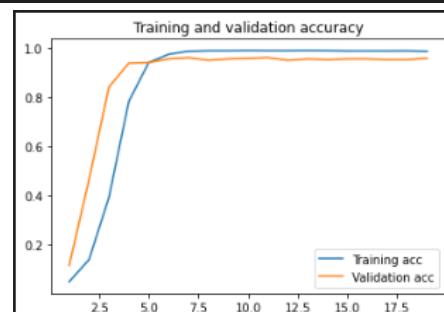


```

result = model2.evaluate(test_generator)

12/12 [=====] - 24s 2s/step - loss: 0.2977 - accuracy: 0.9610

```



<Fig.3-27. 새로운 모델 훈련>

```

_, baseline_model_accuracy2 = model2.evaluate(test_generator, verbose=0)

_, q_aware_model_accuracy2 = q_aware_model2.evaluate(test_generator, verbose=0)

print('new model2 test accuracy:', baseline_model_accuracy2)
print('Quant test accuracy:', q_aware_model_accuracy2)

65]   ✓ 43.8s
.. new model2 test accuracy: 0.961002767086029
      Quant test accuracy: 0.9582172632217407

```

<Fig.3-28. 양자화 인식 훈련 결과>

```

save_dir = ''
q_aware_model2.save(save_dir+'/q_model2.h5')

converter = tf.lite.TFLiteConverter.from_keras_model(q_aware_model2)
# converter.optimizations = [tf.lite.Optimize.DEFAULT]
quantized_tflite_model = converter.convert()

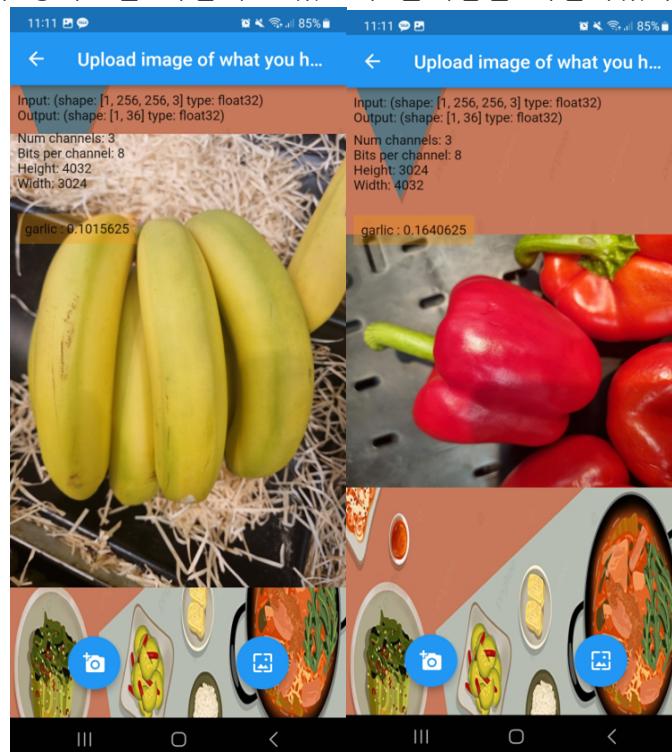
WARNING:absl:Found untraced functions such as _update_step_xla, conv2d_10_layer_call_fn, conv2d_10_layer_c
INFO:tensorflow:Assets written to: C:\Users\YOUNGH\1\AppData\Local\Temp\tmpx9ftx56d\assets
INFO:tensorflow:Assets written to: C:\Users\YOUNGH\1\AppData\Local\Temp\tmpx9ftx56d\assets
c:\ProgramData\Anaconda3\lib\site-packages\tensorflow\lite\python\convert.py:765: UserWarning: Statistics
warnings.warn("Statistics for quantized inputs were expected, but not "

# Save the model.
with open('q_model2.tflite', 'wb') as f:
    f.write(quantized_tflite_model)

```

<Fig.3-29. 새롭게 훈련된 모델 변환>

- 새 모델과 그 모델의 양자화 인식 훈련이 완료된 모델의 정확도가 각각 약 96.1%, 95.8% 정도로 완성되어 곧바로 tflite 형식 파일로 변환 시킨 후 다시 모바일 기기에서의 정확도를 확인해 보았으나 문제점은 여전하였다.



<Fig.3-30. 여전한 문제점들>

- 이쯤에서 성능 저하 문제 해결에 대한 방향성이 옳은지 점검해야 한다고 생각했다.
- 위에서 시도한 모든 모델의 추론 결과가 만족스럽지 않았다. 결국 이 문제의 원인은 둘 중 하나였다.
- + PC단 변환 과정에서 정확도 손실 발생
 - + APP단 통합, 추론 과정에서 정확도 손실 발생
 - PC단 변환 과정에서 정확도 손실 발생한 것이라면 그 지금까지 시도해본 방법과 원인 가능성은 다음과 같다.
- + API Method
 - .from_keras_model()
 - .from_saved_model()
 - + 양자화 인식 훈련
 - 동적 양자화
 - float16 양자화
 - + 훈련 후 양자화
 - + 모델 재 훈련
 - 연산 확인
 - 즉, 모든 가능성에 대한 시도를 해보았고 실패하였다. 그말인 즉슨 문제가 App단에서, 통합, 추론하는 과정에서 발생했다고 추론할 수 있다. 이를 간접적으로 확인하기 위해 다음과 같은 아이디어를 생각해냈다. 우선 PC에서 TFLite interpreter을 구현하여 그 분류 정확도와, App에서의 분류 정확도를 비교하였을 때, PC에서 괜찮은 정확도를 보인다면 그것은 App단에서 발생한 문제라는 것을 추론할 수 있다.
 - 먼저 PC에서 관련 문서를 연구하여 interpreter을 구현하였다.

```

# Select the image for prediction
imgpath_folder = ""
imgpath_folder += "/real_data"
imgpath = imgpath_folder + "/apple2.jpg"
input_image = preprocess_image(imgpath)

# Load TFLite model and allocate tensors.
interpreter = tf.lite.Interpreter(model_path="models/quantized_model.tflite")
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Provide your input to the model.
input_shape = input_details[0]['shape']
print('preprocess:', preprocess_image(imgpath))
input_data = np.array(preprocess_image(imgpath), dtype=np.float32)
print('after:', input_data)
interpreter.set_tensor(input_details[0]['index'], input_data)

# Run the model
interpreter.invoke()

# The function `get_tensor()` returns a copy of the tensor data.
output_data = interpreter.get_tensor(output_details[0]['index'])
print(np.sum(output_data))
print(output_data)
index=np.argmax(output_data)
print(output_data[0][index])
print(index)
print(classes_list[index])

```

<Fig.3-31. PC에서의 interpreter>

- 위 양자화 과정에서 5가지 모델을 완성하였는데 이를 interpreter에 할당하여 그 분류 정확도를 확인하였다. 다음은 5가지 모델들의 목록이다.

- + modelMobileNetV2
- + from_saved_model
- + quanted_model
- + quanted_model_16
- + q_model2

〈Fig.3-32. PC에서의 interpreter 정확도〉

- 이미지 하나하나 입력해 보았을 때 준수한 정확도를 보여주었으나, 좀 더 객관적인 정확도 지표를 얻기 위하여, test dataset을 사용하여 정확도 평가를 진행하였다. 이때 이미지 분류를 직접 만든 interpreter가 수행하게 된다. 따라서 test dataset을 통한 평가가 이루어지게 하기 위해 다음과 같은 순서로 코드 구현하였다.

- + 데이터 준비
 - + tflite 모델을 interpreter로 하여
 - + tensor 할당
 - + for문 통해 각 test img의 label과 interpreter의 예측 label 비교
 - + 일치할 시 correct_predictions+1
 - + 매 반복 시 total_predictions+1
 - + acc = correct / total

```
# batch_size = 32
#image_size = 256
target_size = (256,256)
input_shape = (256, 256, 3)

# Your image data generator
test_datagen = ImageDataGenerator(rescale = 1.00 / 255.0)
test_dir = ''
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size = target_size,
    batch_size = 1,
    class_mode='categorical'
)
print('test_generator image shape: ',test_generator.image_shape)
print('test_generator OK')

# Load TFLite model and allocate tensors.
interpreter = tf.lite.Interpreter(model_path="models/modelMobileNetV2.tflite")
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
print('expected input Shape:', input_details[0]['shape'])
# input details:  [('name': 'serving_default_conv2d_10_input:0', 'index': 0, 'shape': array([1, 36, 36, 36], dtype=int32))
# output details:  [('name': 'StatefulPartitionedCall:0', 'index': 22, 'shape': array([1, 36, 36, 36], dtype=int32))

correct_predictions = 0
total_predictions = 0
```

<Fig.3-33. evaluate code 1>

```

for images, labels in test_generator:

    # reshape images if necessary, convert to correct type
    images = np.array(images, dtype=np.float32)

    interpreter.set_tensor(input_details[0]['index'], images)
    interpreter.invoke()

    tflite_predictions = interpreter.get_tensor(output_details[0]['index'])

    # print('tflite_predictions: ', tflite_predictions)

    # now compare tflite_predictions to labels
    # you would write this comparison code based on your specific use case
    # for a binary classification, it might be:
    tflite_predictions = (tflite_predictions > 0.5).astype(int)
    predicted_class_index = np.argmax(tflite_predictions)
    correct_class_index = np.argmax(labels)

    print('predicted index: ', predicted_class_index)
    print('correct index: ', correct_class_index)

    if(predicted_class_index == correct_class_index):
        correct_predictions += 1

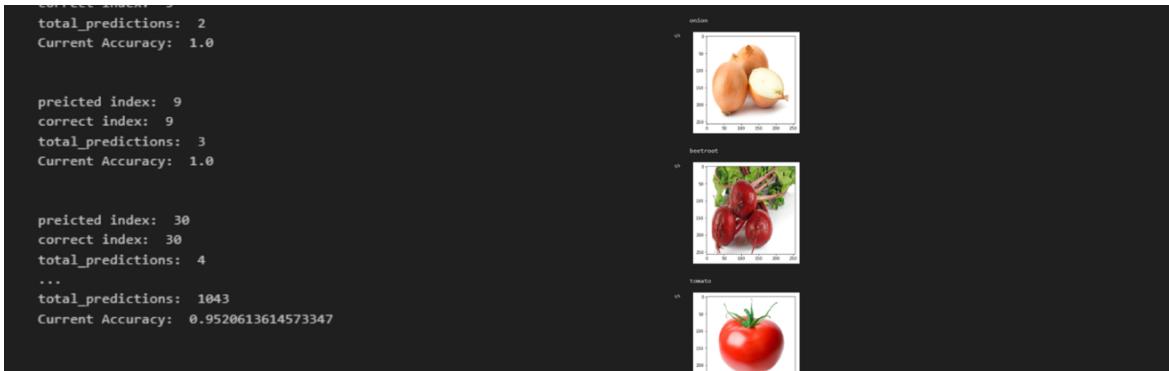
    total_predictions += 1

    # print('correct_predictions: ', correct_predictions)
    print('total_predictions: ', total_predictions)
    print("Current Accuracy: ", correct_predictions / total_predictions)
    print('\n')

print("Total Accuracy: ", correct_predictions / total_predictions)

```

〈Fig.3-34. evaluate code 2〉



〈Fig.3-35. evaluate code result〉

model	modelMobileNetV2	from_saved_model	quantized_model	quantized_model_16	q_model2
acc	0.973	0.974	0.975	0.974	0.952

〈Table.3-4. evaluate code accuracy〉

- PC상에서 5가지 모델 모두 준수한 성능을 보이며 이는 App단에서 정확도 저하가 발생한다는 것을 의미한다. 이를 간단하게 나마 검증하고자 하나의 이미지를 3가지 모델(원형모델, 변형모델, 앱에서의 모델)에 입력해 보았고 더 명료해졌다.

```

1/1 [=====] - 0s 54ms/step
prediction: [[2.6149184e-08 4.8239218e-07 8.9550030e-09 2.4989469e-10 5.3187108e-05
2.0321824e-07 9.9996495e-01 1.599979e-09 6.1415448e-06 1.5557597e-04
4.8147206e-07 3.5986730e-07 4.3291937e-08 4.7835847e-06 4.3771184e-08
1.3904929e-08 3.3312810e-09 2.9000415e-07 3.3391876e-07 8.1320968e-09
1.8294845e-08 7.2989644e-07 5.7922065e-08 1.4205537e-07 1.6540556e-06
1.6730466e-08 2.7335501e-09 4.1839852e-09 7.9194378e-06 1.4657936e-11
3.8952916e-10 6.5868937e-07 3.8923824e-09 2.1105278e-09 6.2242727e-08
2.7731740e-08]]
sum: 1.0001982412624042
6
0.99996495
Predicted class label: carrot

```

<Fig.3-36. acc in pc>



<Fig.3-37. acc in app>

(5) App단 분석 및 문제 해결

- App단에서 모델의 작동 순서는 이러하다.
- + interpreter <- 모델 할당
- + 이미지 입력
- + 이미지 디코딩 및 정규화
- + interpreter.run()
- + 분류 결과
- 이중 문제가 발생할만한 부분은 이미지 디코딩 및 정규화 그리고 interpreter.run() 이다. 해당 부분을 보다 면밀히 분석하였고, 이상한 점을 발견하였다. 바로 이미지 정규화의 pixel크기를 정규화 하지 않은 것이었다. 이를 수정하자 문제점이 곧바로 해결되었다.

```

final imageMatrix = List.generate(
    imageInput.height,
    (y) => List.generate(
        imageInput.width,
        (x) {
            final pixel = imageInput.getPixel(x, y);
            return [pixel.r, pixel.g, pixel.b];
        },
    ), // List.generate
); // List.generate
// print(imageMatrix.shape); // [256, 256, 3]

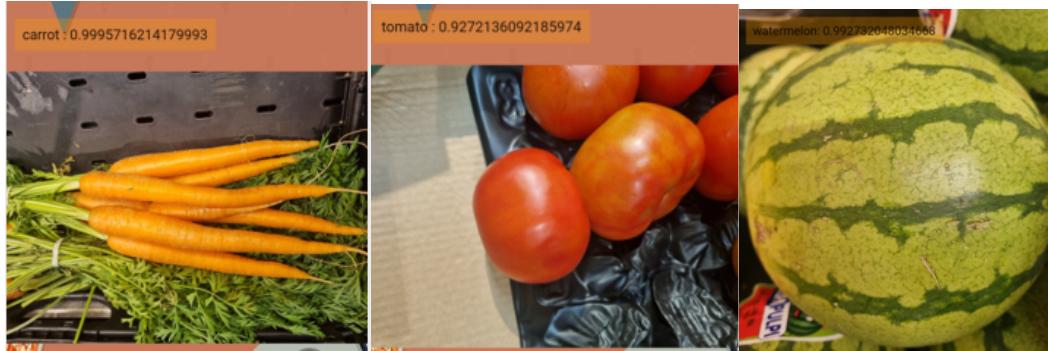
```

```

final imageMatrix = List.generate(
    imageInput.height,
    (y) => List.generate(
        imageInput.width,
        (x) {
            final pixel = imageInput.getPixel(x, y);
            return [pixel.r / 255.0, pixel.g / 255.0, pixel.b / 255.0];
        },
    ), // List.generate
); // List.generate
// print(imageMatrix.shape); // [256, 256, 3]

```

<Fig.3-38. 이미지 정규화 전, 후>



<Fig.3-39. 문제 해결 후 분류 정확도>

5. Flutter

(1) OpenAI API 연동

- OpenAI API reference를 참조하여 구현하였다. GPT-3와 DALL-E를 사용하였다. GPT-3에 원하는 질의를 하기 위해 request body에 prompt문을 작성해야한다. suggest 3 recipes base on those + \$ingredients(분류된 라벨 List)의 요청문을 만들어 사용하였다. 받게 될 response에는 세가지 레시피가 1. 2. 3. 으로 시작하게 된다. 이를 기준으로 response를 자르고 반환하도록 하였다.

```

class ApiService {
    //request for openAI GPT
    static Future<List<String>> postRecipes(List<String> ingredients) async {
        try {
            var response = await http.post(Uri.parse(API_URL),
                headers: {
                    'Authorization': 'Bearer $API_KEY',
                    "Content-Type": "application/json"
                },
                body: jsonEncode({
                    "model": "text-davinci-003",
                    // "model": "gpt-3.5-turbo",
                    "prompt": "suggest 3 recipes base on those:\n\n$ingredients",
                    "temperature": 0.83,
                    "max_tokens": 997,
                    "top_p": 1,
                    "frequency_penalty": 0.8,
                    "presence_penalty": 0
                }));
        }
    }
}

```

<Fig.3-40. API : GPT-3 request>

```

    });
    Map jsonResponse = jsonDecode(response.body);
    String recipes = jsonResponse["choices"][0]["text"];
    String recipe1 = recipes.substring(recipes.indexOf('1. '), recipes.indexOf('2. '));
    String recipe2 = recipes.substring(recipes.indexOf('2. '), recipes.indexOf('3. '));
    String recipe3 = recipes.substring(recipes.indexOf('3. '));
    return [recipe1, recipe2, recipe3];
} catch (error) {
    print("error-postRecipes-response: $error");
    return ['recipe request failed'];
}
}

```

<Fig.3-41. API: GPT-3 response>

- DALL-E의 경우 역시 request body에 prompt문을 사용하여 생성할 이미지를 요청한다.

```

// api request for Image Generation Model
static Future<String> postImage(String recipe1) async {
try {
    var response = await http.post(Uri.parse(API_IMG_URL),
        headers: {
            'Authorization': 'Bearer $API_KEY',
            "Content-Type": "application/json"
        },
        body: jsonEncode({"prompt": recipe1, "n": 1, "size": "256x256"}));
    Map jsonResponse = jsonDecode(response.body);
    // print("getImage: $jsonResponse");
    String img_url = jsonResponse["data"][0]["url"];
    // print("img url: $img_url");
    return img_url;
} catch (error) {
    print("error-postImage-response: $error");
    return 'image request failed';
}
}

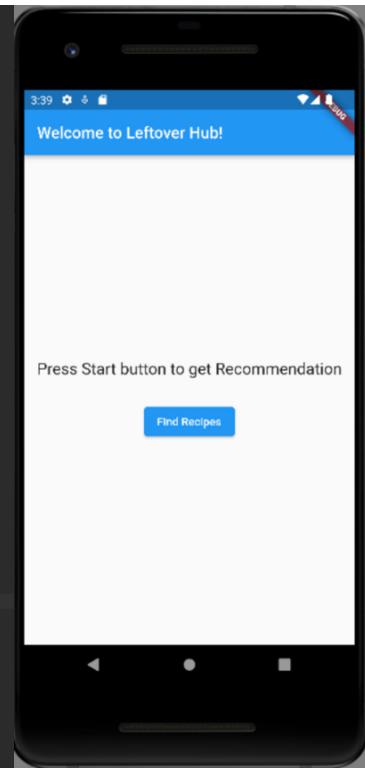
```

<Fig.3-42. API: DALL-E request>

(2) 앱 UI 및 라우팅

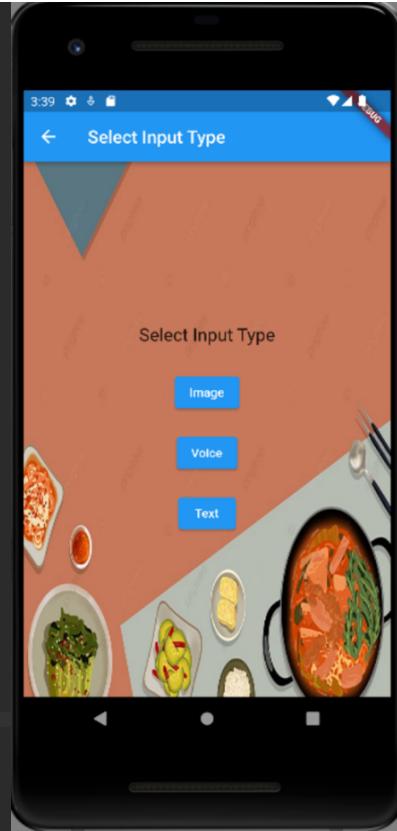
- 앱의 동작 순서는 <Fig.2-1. Flow chart>와 동일하다.

```
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(home: MyHome());
  }
}
class MyHome extends StatelessWidget {
  const MyHome({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Welcome to Leftover Hub!'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text(
              'Press Start button to get Recommendation',
              style: TextStyle(fontSize: 20.0),
              textAlign: TextAlign.center,
            ),
            const SizedBox(height: 30),
            ElevatedButton(
              onPressed: () {
                if (kDebugMode) {
                  print("Find Recipes");
                }
                Navigator.push(context, MaterialPageRoute(builder: (context) => const InputTypePage()));
              },
              child: const Text('Find Recipes'),
            ),
            const SizedBox(height: 30),
          ],
        ),
      ),
    );
  }
}
void main() {
  runApp(
    MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (_) => RecipeProvider()),
      ],
      child: const MyApp(),
    ),
  );
}
```



<Fig.3-43. main page>

```
class InputTypePage extends StatelessWidget {
  const InputTypePage({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Select Input Type'),
      ),
      body: Container(
        decoration: const BoxDecoration(
          image: DecorationImage(
            image: AssetImage('assets/background.jpg'),
            fit: BoxFit.fill),
        ),
        child: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              const Text(
                'Select Input Type',
                style: TextStyle(fontSize: 20.0),
                textAlign: TextAlign.center,
              ),
              const SizedBox(height: 30),
              ElevatedButton(
                onPressed: () {
                  if (kDebugMode) {
                    print("input type : image");
                  }
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => const ImageInputPage(), // MaterialPageRoute
                    ),
                  );
                },
                child: const Text('Image'),
              ),
              const SizedBox(height: 20),
              ElevatedButton(
                onPressed: () {
                  if (kDebugMode) {
                    print("input type : voice");
                  }
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => const VoiceInputPage(), // MaterialPageRoute
                    ),
                  );
                },
                child: const Text('Voice'),
              ),
              const SizedBox(height: 20),
              ElevatedButton(
                onPressed: () {
                  if (kDebugMode) {
                    print("input type : text");
                  }
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => const TextInputPage(), // MaterialPageRoute
                    ),
                  );
                },
                child: const Text('Text'),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```



<Fig.3-44. input_type_page>

```

@Override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Upload image of what you have'),
    ), // AppBar
    body: Container(
      decoration: const BoxDecoration(
        image: AssetImage("assets/background.jpg"), fit: BoxFit.fill),
      child: Column(
        children: [
          const SizedBox(
            height: 20,
          ), // SizedBox
          Expanded(
            child: ListView.builder(
              itemCount: ingredients.length,
              itemBuilder: (BuildContext context, int index) {
                return Container(
                  height: 50,
                  margin: const EdgeInsets.all(2),
                  color: const Color.fromARGB(150, 150, 200, 150),
                  child: Center(
                    child: Text(
                      ingredients[index],
                      style: const TextStyle(fontSize: 18),
                    )), // Text, Center
                ); // Container
              })), // ListView.builder, Expanded
          const SizedBox(
            height: 20,
          ), // SizedBox
          ElevatedButton(
            child: const Text('Search'),
            onPressed: () async {
              if (kDebugMode) {
                print("Search");
              }
              try {
                recipes = await ApiService.postRecipes(ingredients);
                imgUrl = await ApiService.postImage(recipes[0]);
              } catch (error) {
                if (kDebugMode) {
                  print("Error $error");
                }
              }
            },
            Navigator.push(context, MaterialPageRoute(
              builder: (resultContext) => Result(
                recipes: recipes,
                ingredients: ingredients,
                imgUrl: imgUrl,
              )), // Result, MaterialPageRoute
            );
          ), // ElevatedButton
        ],
      ), // Column
    ), // Container
  ); // Scaffold
}

```

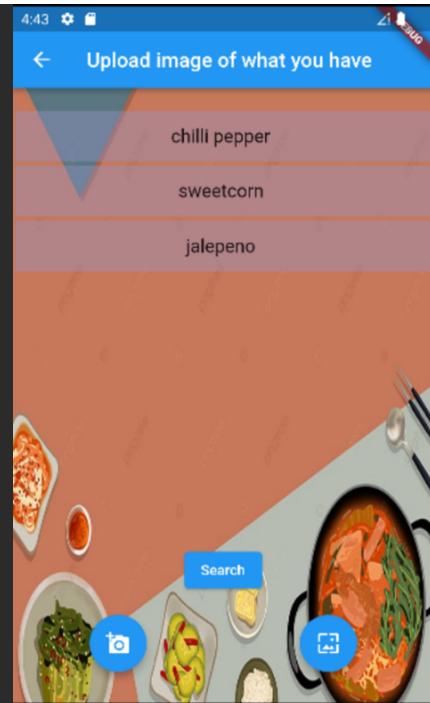
```

), // ElevatedButton
const SizedBox(
  height: 20,
), // SizedBox
Row(
  mainAxisAlignment: MainAxisAlignment.spaceAround,
  children: [
    FloatingActionButton(
      heroTag: 'btnCamera',
      onPressed: () async {
        cleanResult();
        final result = await imgPicker.pickImage(
          source: ImageSource.camera,
        );

        imagePath = result?.path;
        setState(() {});
        processImage();
      },
    ),
    child: const Icon(Icons.add_a_photo),
  ), // FloatingActionButton
    FloatingActionButton(
      heroTag: 'btnGallery',
      onPressed: () async {
        cleanResult();
        final result = await imagePicker.pickImage(
          source: ImageSource.gallery,
        );

        imagePath = result?.path;
        setState(() {});
        processImage();
      },
    ),
    child: const Icon(Icons.wallpaper),
  ), // FloatingActionButton
], // Row
const SizedBox(
  height: 30,
), // SizedBox
), // Column
), // Container
); // Scaffold
}

```



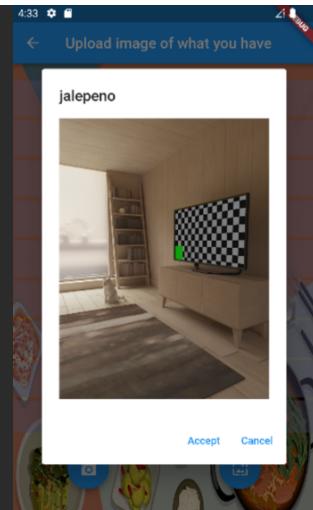
<Fig.3-45. image_input_page>

- 입력된 이미지와 예측된 라벨을 모달창에 띄우고, 이것을 확인 또는 취소한다.

```

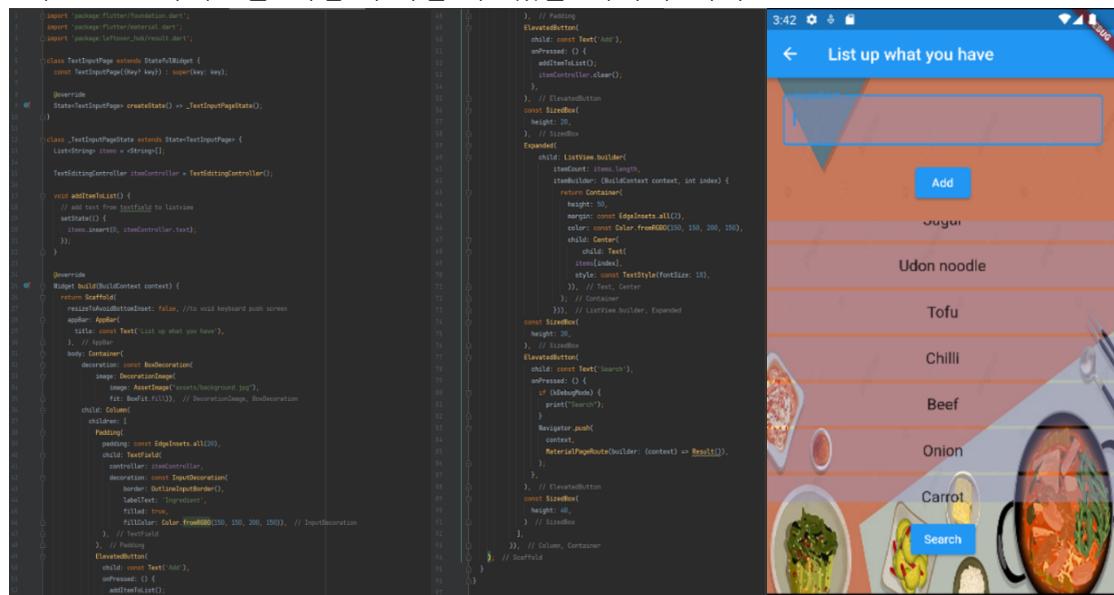
Future showConfirmationDialog(File image, String label) { // Run inference
  return showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text(label),
        content: Image.file(image),
        actions: <Widget>[
          TextButton(
            child: const Text('Accept'),
            onPressed: () {
              setState(() {
                ingredients.insert(0, label);
              });
              Navigator.of(context).pop(true);
            },
          ), // TextButton
          TextButton(
            child: const Text('Cancel'),
            onPressed: () {
              Navigator.of(context).pop(false);
            },
          ), // TextButton
        ], // <Widget>[]
    ); // AlertDialog
  }, // Future
}

```

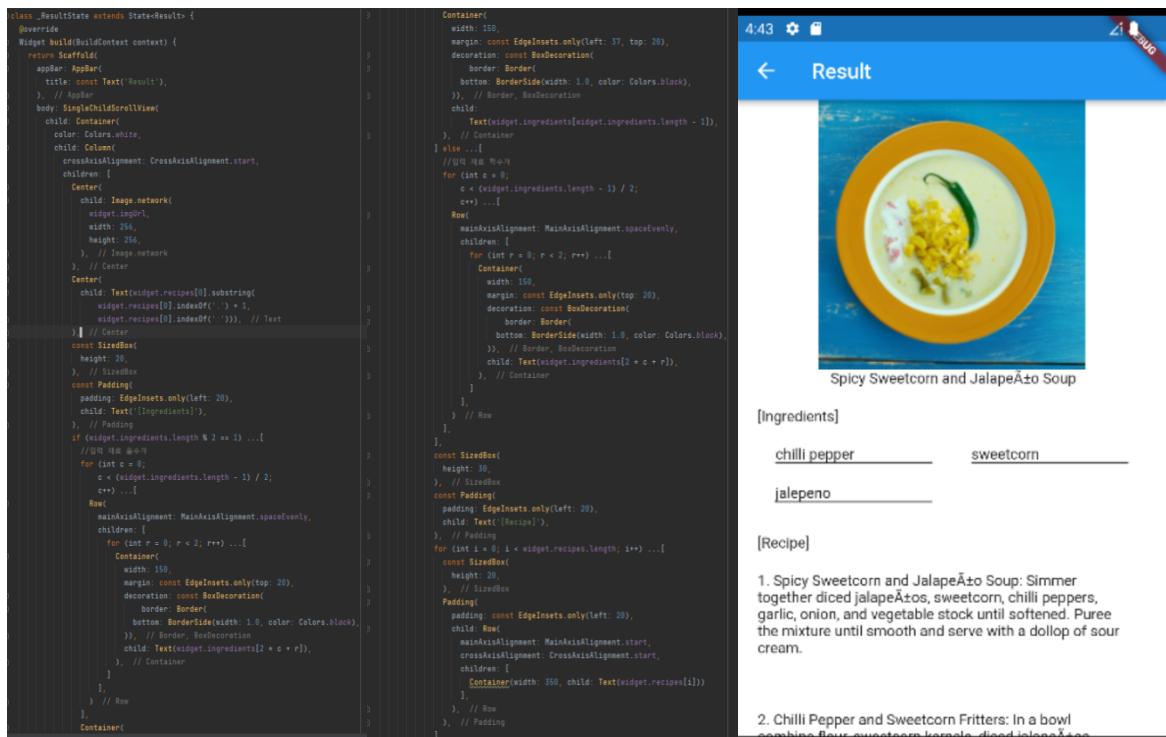


<Fig.3-46. confirmation dialog>

- 텍스트로 식재료를 직접 추가할 수 있는 페이지 이다.



<Fig.3-47. text_input_page>



<Fig.3-48. result>

(3) STT 페이지 구현

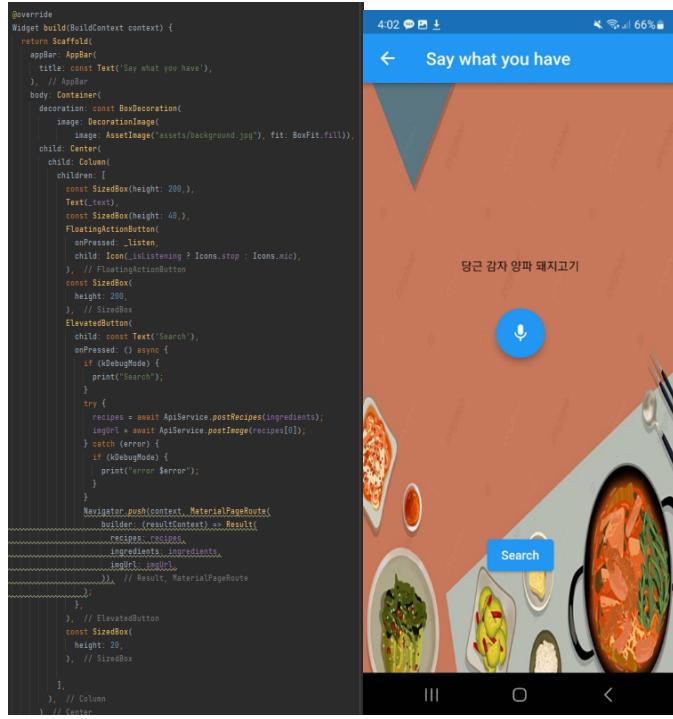
- STT라이브러리인 speech_to_text를 연구 분석하여 구현 하였다. 버튼 클릭, speech.initialize() 함수를 통한 마이크 활성화 여부 체크, 활성화 여부에 따라 적절한 처리를 통해 인식한 음성을 recognizeWords() 함수를 통해 String으로 저장한다. 이를 구현한 코드는 다음과 같다.

```
Text(_text),
const SizedBox(height: 40,),
FloatingActionButton(
 onPressed: _listen,
 child: Icon(_isListening ? Icons.stop : Icons.mic),
), // FloatingActionButton
```

<Fig.3-49. 마이크 버튼 클릭>

```
void _listen() async {
if (!_isListening) {
    bool available = await _speech.initialize(
        onStatus: (val) => print('onStatus: $val'),
        onError: (val) => print('onError: $val'),
    );
    if (available) {
        setState(() => _isListening = true);
        _speech.listen(
            onResult: (val) => setState(() {
                _text = val.recognizedWords;
                // ingredients.add(_text);
                splitTextBySpace();
            }),
            localeId: localeId
        );
    }
} else {
    setState(() => _isListening = false);
    _speech.stop();
}
}
```

<Fig.3-50. 마이크 활성화 여부 체크 및 인식된 음성 표시>



<Fig.3-51. voice_input_page>

IV. 결론

본 프로젝트에서는 식재료 인식을 기반으로 레시피를 추천하는 모바일 앱을 성공적으로 개발하였다. 이를 위해 다양한 기술, 논문, 그리고 도구를 종합적으로 연구하고 적용하였다. 특히, TensorFlow와 MobileNetV2를 활용한 이미지 인식 모델은 약 97%의 높은 정확도를 보였다. 또한, 사용자의 음성을 텍스트로 변환하는 STT 기술, 그리고 OpenAI의 GPT-3와 DALL-E를 이용한 레시피 생성 및 시각적 피드백도 원활하게 구현할 수 있었다.

프로젝트를 진행하면서 겪었던 주요 어려움 중 하나는 TensorFlow Lite를 통해 훈련된 모델을 모바일 앱에 적용하는 과정이었다. 이 문제는 깊은 연구와 여러 번의 시행착오 끝에 해결할 수 있었다.

향후 개선 및 확장 계획으로는 다음과 같은 사항들이 있다:

1. 더 다양한 종류의 식재료를 인식할 수 있도록 모델을 업그레이드 할 계획이다.
2. 사용자의 특정 건강 상태나 질병에 따른 식습관 가이드라인을 제공하는 기능을 추가할 예정이다.
3. 다이어트를 위한 칼로리 계산 기능을 통해 사용자에게 더 많은 정보를 제공할 계획이다.

이 프로젝트를 통해 얻은 경험과 지식은 머신 러닝, 앱 개발, 그리고 사용자 경험 디자인 등 다양한 분야에서의 활용 가능성이 크다. 이러한 성과와 앞으로의 계획을 통해, 본 프로젝트가 사용자에게 실질적인 편의와 가치를 제공할 수 있을 것이라고 확신한다.

V. 참고 문헌

- [1] S. J. Rokon, K. Morol, I. B. Hasan, et al, Food Recipe Recommendation Based on Ingredients Detection Using Deep Learning, ICCA 2022, March 10-12, 2022, Dhaka, Bangladesh
- [2] T. Shota, T. Hyuga, U. Rune and N. Takafumi, Recipe Recommendation Method by Similarity Measure with Food Image Recognition, ICISDM 2022, May 27-29, 2022, Silicon Valley, CA, USA
- [3] M. Bola~nos, A. Ferr`a and P. Radeva, Food Ingredients Recognition Through Multi-label Learning, ICIAP 2017 International Workshops, LNCS 10590, pp. 394-402, 2017.
- [4] L. Zhang, J. Zhao, S. Li, B. Shi and L. Duan, FROM MARKET TO DISH: MULTI-INGREDIENT IMAGE RECOGNITION FOR PERSONALIZED RECIPE RECOMMENDATION, 2019 IEEE International Conference on Multimedia and Expo (ICME)
- [5] G. G. Lee, C. W. Huang, J. H. Chen, S. Y. Chen and H. L. Chen, AIFood: A Large Scale Food Images Dataset for Ingredient Recognition, 2019 IEEE Region 10 Conference (TENCON 2019)
- [6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. C. Chen, MobileNetV2: Inverted Residuals and Linear Bottlenecks, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4510-4520
- [7] K. Xiao, L. Engstrom, A. Ilyas, A. Madry, NOISE OR SIGNAL: THE ROLE OF IMAGE BACKGROUNDS IN OBJECT RECOGNITION, Published as a conference paper at ICLR 2021
- [8] C. Rother, V. Kolmogorov and A. Blake, “GrabCut” — Interactive Foreground Extraction using Iterated Graph Cuts, ACM Transactions on Graphics, Volume 23, Issue 3, pp 309-314