최 종 보 고 서

과목명
캡스톤디자인B(001)
담당교수
송형규 교수님
팀명
10조 배추요정
참여인원
18010717 이의석 / 18010716 황용하 18010715 배재광 / 18010713 권오윤
개발주제
AI 이미지 인식을 이용한 배추 병충해 진단 애플리케이션

목차

Ⅰ 작품 개요

1. 배경 및 필요성

건강한 농작물을 재배를 위해서는 토양관리와 수분관리 등 고려할 사항이 많지만 피해를 입는 대표적인 원인으로 병충해가 있다. 병충해에 의한 피해는 경우에 따라 규모가 다양한데, 전염성에 의한 병해는 밭 전체의 피해를 볼 수 있을 만큼 치명적이다. 따라서 병해충에 피해에 대한 대비가 필요하고 병해충 피해가 발생하였을 시에는 즉각적인 대처가 필요하다.

농림수산식품 교육문화정보원의 청년층 귀농 및 귀촌 인구는 2019 년 대비 증가 추이를 보이고 있으며, 농업의 비전 및 사업성을 보고 모이는 양상을 띄고 있다. 또한 주말농장, 도시농장 등 직장을 다니면서 취미의 목적으로 농작물 재배에 도전하는 인구가 늘어나고 있다.

가업을 승계하는 경우 농업에 오래 종사한 부모의 도움으로 농작물 재배를 이끌어 갈 수 있으나, 홀로 귀농을 선택하거나 주말 농장 운영에 도전하는 사람들은 농작물 재배에 대한 지식과 노하우를 알기 어려움이 있고, 즉각적으로 전문가의 도움을 받기가 어려운 상황에 놓인다. 특히 앞서 언급한 병충해의 피해 발생시 즉각적인 판단과 대처가 필요하나, 나타나는 특징이 비슷한 병충해 피해를 전문가의 도움 없이는 진단하기 어려운 상황에 놓이기 쉽다.

2. 구현 목표

10 조의 'AI 를 활용한 농작물 병충해 진단 APP'은 전문가의 즉각적인 도움을 받기 힘든 귀농인, 주말농장 운영인 등을 대상으로 사진 진단 서비스를 제공한다.

병충해에 의한 피해 발생 시, 초보자의 경우 다른 원인에 의한 피해와 구분하기 어려운 상황이 발생하며, 진단을 위해 관련 병충해 사진을 일일이 비교하는 일에서도 어려움이 있다. 그러므로 관련 병충해 사진을 비교하고 병명을 검색하는 수고로움을 줄이는 것에 목표하며, 그 방법으로 스마트폰 카메라 기능을 사용하는 것으로 구상하였다.

'AI 를 활용한 농작물 병충해 진단 APP'은 진단 대상을 스마트폰 카메라로 촬영한 후, 학습된 데이터셋과 비교하여 병명을 진단하도록 한다. 이후 사용자에게 결과값을 제공함과 동시에 해당 질병에 대한 정보와 대처방안을 같이 제공하도록 한다.

진단 가능한 작물은 학습시키는 데이터에 의해 결정되므로 수확량이 많은 작물부터 적은 작물까지 무한히 확장 시킬 수 있으나, 종합 설계 기간내 데이터 수집의 어려움으로 인하여 한가지 작물만을 선택하여 학습시키기로 하고, 해당 작물에 대한 정상 작동 여부를 확인하여 APP 의 병충해 진단 정확도를 테스트 하도록 한다.

3. 기대효과

'AI 를 활용한 농작물 병충해 진단 APP' 사용자들은 스마트폰 카메라를 통한 진단 시스템으로 빠르고 직관적으로 농작물의 병명을 알아낼 수 있고, 제공되는 관련 정보를 이용하여 후속 피해에 빠르게 대처하고 농장 운영 전반의 편리함을 얻게 될 것으로 기대된다.

II 제작 과정

1. 프로젝트 일정

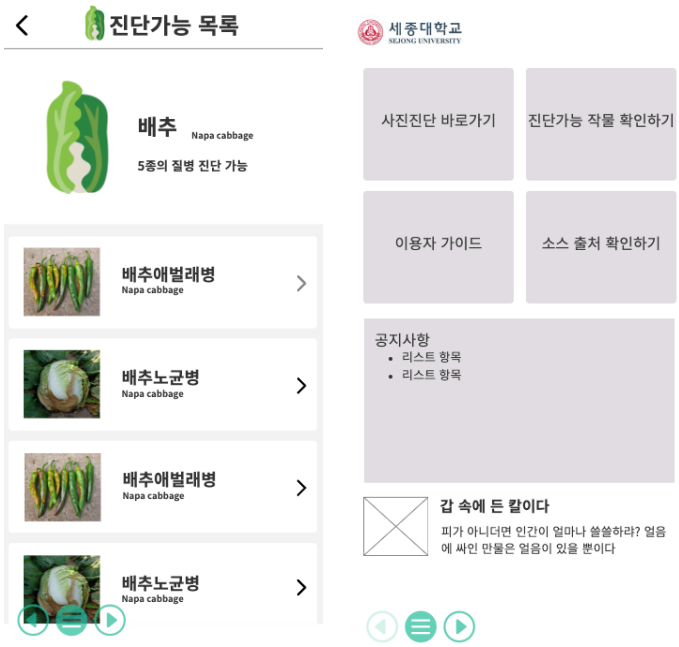
짧은 개발 기간을 고려하여 앱 디자인/모델 훈련/어플리케이션 개발 등 3가지의 역할로 나누어 프로젝트를 진행하였다.

각 주차별 프로젝트 일정은 다음과 같이 진행되었다.

1주차	조 편성	9주차	진단 기능 구현, 모델 테스트
2주차	아이디어 선정	10주차	진단 기능 구현, 모델 테스트
3주차	자료조사	11주차	API 연동, 챗봇 구현
4주차	구현기술 선정	12주차	최종테스트, 발표자료 제작
5주차	앱 디자인, 훈련 데이터 탐색	13주차	최종 테스트, 발표자료 제작
6주차	앱 디자인, 훈련 데이터 탐색	14주차	발표 준비
7주차	앱 디자인, 훈련 데이터 탐색	15주차	발표 준비
8주차	레이아웃 구현, 모델 훈련	16주차	최종 발표

2. 상세 구현 과정

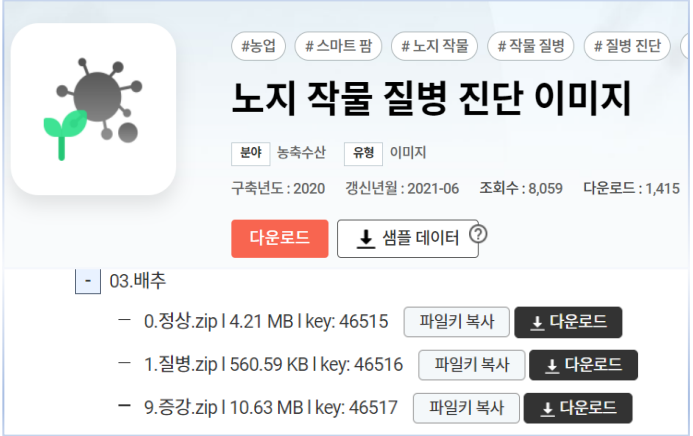
2.1 앱 디자인



UI/UX 기획을 위한 프로토타이핑 툴인 카카오 오븐을 이용하여 앱의 전체적인 레이아웃을 설계했다.

또한 농업 종사자들의 평균 연령대가 높다는 사실을 고려하여 최대한 단순하고 직관적인 디자인을 적용시켰다.

2.2 모델 훈련



모델의 학습 데이터로는 AI HUB에서 제공하는 노지 작물 질병 진단 이미지 데이터 셋을 사용하였다.

전이학습 방식을 지원하는 구글의 티처블 머신으로 모델을 훈련시켰으며

훈련시킨 모델을 모바일 기기에서 직접 구동하게 되면 결과값 추론에 많은 시간과 자원이 소모되므로 본 모델인 Tensorflow를 경량화 시킨 Tensorflow Lite 모델로 변환하는 과정을 거쳤다.

2.3 어플리케이션 개발

인공지능 모델이 병충해를 진단하는 과정은 다음과 같이 구현했다.

<preProcessInput 함수>

```
// #1
TensorImage _preProcessInput(img.Image image) {
  final inputTensor = TensorImage(_model.inputType);
  inputTensor.loadImage(image);
  // #2
  final minLength = min(inputTensor.height, inputTensor.width);
  final cropOp = ResizeWithCropOrPadOp(minLength, minLength);
  // #3
  final shapeLength = _model.inputShape[1];
  final resizeOp = ResizeOp(shapeLength, shapeLength, ResizeMethod.BILINEAR);
  // #4
  final normalizeOp = NormalizeOp(127.5, 127.5);
  // #5
  final imageProcessor = ImageProcessorBuilder()
      .add(cropOp)
      .add(resizeOp)
      .add(normalizeOp)
      .build();
  imageProcessor.process(inputTensor);
  // #6
  return inputTensor;
```

진단 이미지가 입력으로 주어지면 일관성 있는 추론을 위해 입력 이미지를 (224px * 224px) 의 크기로 정규화 한다.

정규화 된 이미지는 (height, width, RGB)의 3D Tensor 데이터로 변환된다.

<predict 함수>

```
List<ClassifierCategory> predict(img.Image image) {
  // Load the image and convert it to TensorImage for TensorFlow Input
  final inputImage = _preProcessInput(image);
  // Define the output buffer
  final outputBuffer = TensorBuffer.createFixedSize(
    _model.outputShape,
    _model.outputType,
  );
  // Run inference
  _model.interpreter.run(inputImage.buffer, outputBuffer.buffer);
  // Post Process the outputBuffer
  final resultCategories = _postProcessOutput(outputBuffer);
  return resultCategories;
```

추론 결과 값을 기록할 변수 outputBuffer를 선언하고

입력 이미지의 픽셀 값과 병충해의 픽셀 값 간의 차이를 계산하여 구한 결과를 outputBuffer에 기록한다.

<_postProcessOutput 함수>

```
List<ClassifierCategory> _postProcessOutput(TensorBuffer outputBuffer) {
  // #1
  final probabilityProcessor = TensorProcessorBuilder().build();
  probabilityProcessor.process(outputBuffer);
  // #2
  final labelledResult = TensorLabel.fromList(_labels, outputBuffer);
  // #3
  final categoryList = <ClassifierCategory>[];
  labelledResult.getMapWithFloatValue().forEach((key, value) {
    // Round off to the second decimal place
    final roundedVal =
        (value * 1000).roundToDouble() / 10;
    final category = ClassifierCategory(key, roundedVal);
    categoryList.add(category);
    debugPrint('label: ${category.label}, score: ${category.score}');
  });
  // #4
  categoryList.sort((a, b) => (b.score > a.score ? 1 : -1));
  return categoryList;
```

픽셀들의 차이 값을 종합하여 각 병충해들의 확률을 산출해내고 (병충해명, 확률)의 (key-value) 데이터 구조로 결과 값을 계산한다.

결과 값의 확률은 소수점 둘째 자리에서 반올림하고 확률이 높은 순서대로 정렬되어 리스트 형태로 반환된다.

<_getDetailedDiseaseInfo 함수>

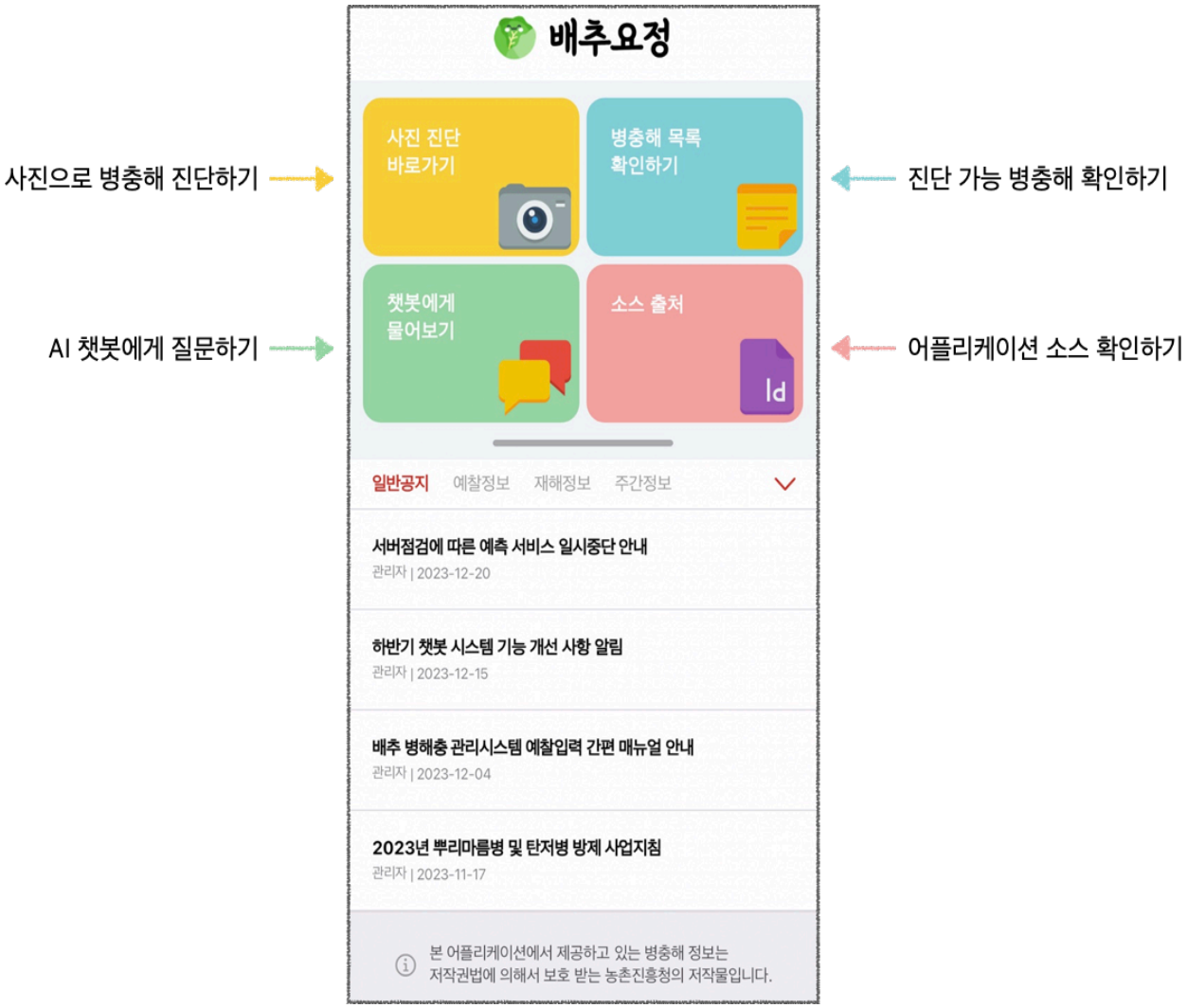
```
Future<DetailedDiseaseInfo> _getDetailedDiseaseInfo(
    DiseaseInfo diseaseInfo) async {
  var uri = Uri.http(requestUrl, pathVariable, <String, String>{
    'apiKey': ncpmsApiKey,
    'serviceCode': detailServiceCode,
    'serviceType': serviceType,
    'sickKey': diseaseInfo.sickKey,
  });
  var response = await http.get(uri);
  var responseBody = utf8.decode(response.bodyBytes);

  var json = jsonDecode(responseBody);
  var info = json['service'];
```

반환된 병충해 목록들의 상세정보를 얻기 위해 국가농작물병해충관리시스템의 URI로 HTTP 요청을 보내고 증상, 발생 환경, 방제 방법 등의 정보를 가져와 사용자에게 보여준다.

III 작품 소개

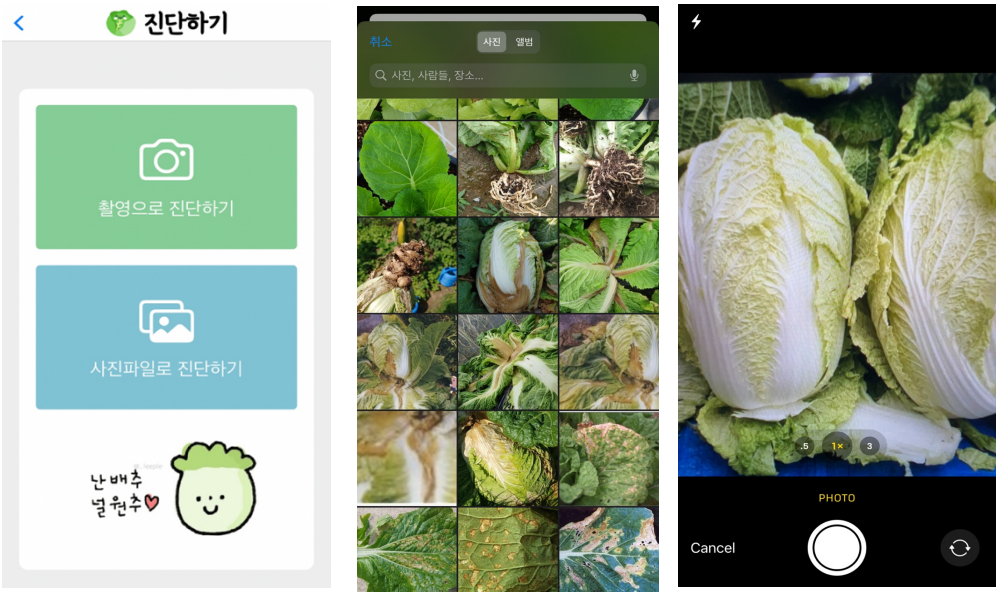
1. 기능 목록



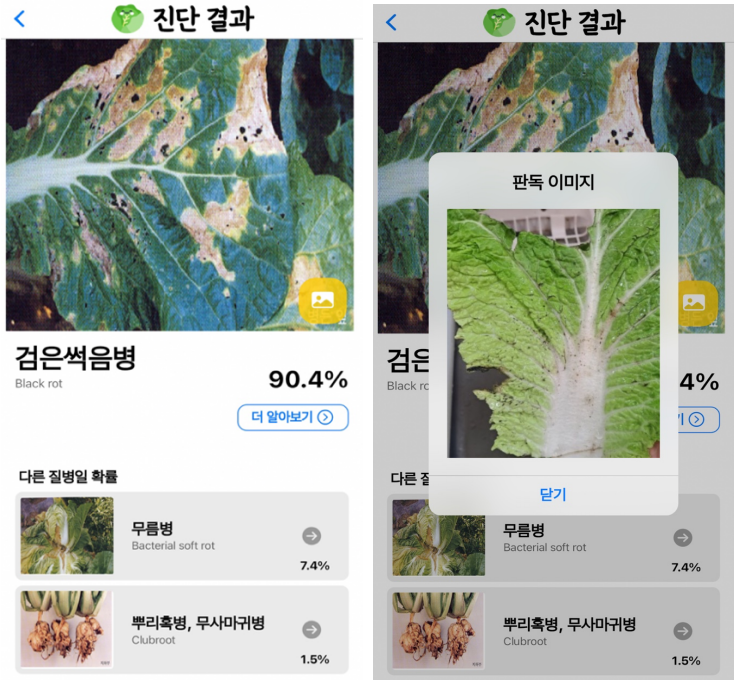
2. 동작 과정

2.1 사진으로 병충해 진단하기

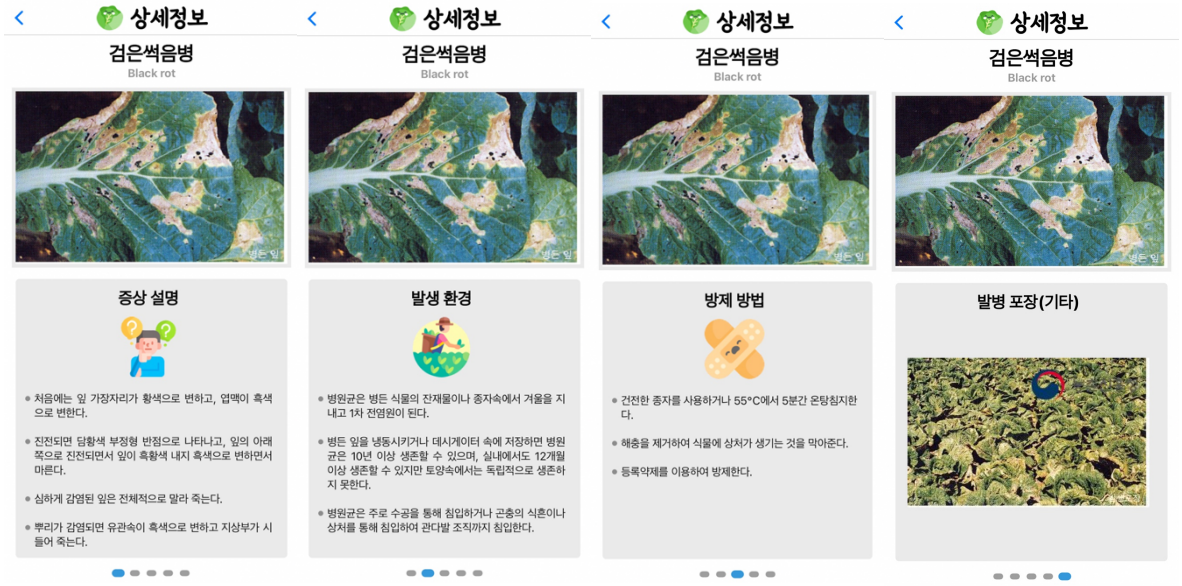
사용자는 진단할 사진을 직접 촬영하거나 앨범에서 선택할 수 있다.



진단 결과 화면에서는 질병의 대표 이미지, 이름, 확률이 표시되며 대표 이미지 우측 하단의 노란 버튼을 누르면 판독 이미지를 확인할 수 있다.

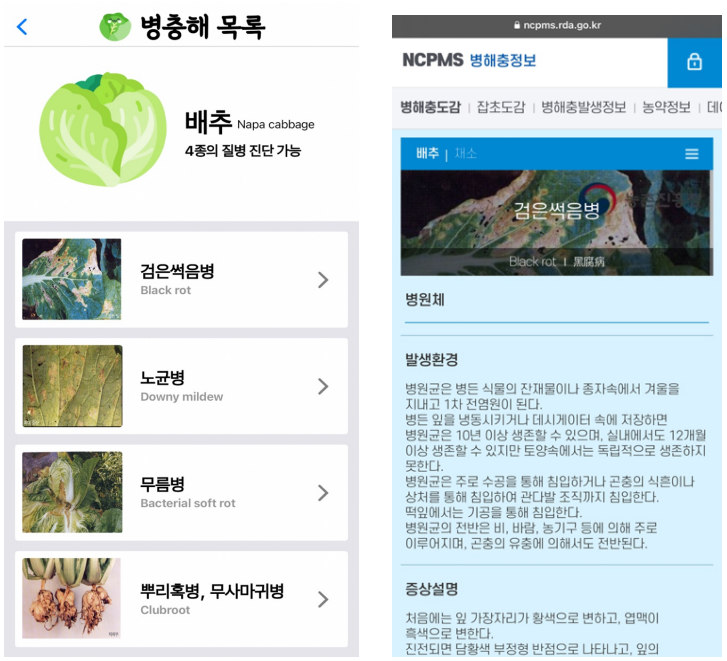


화면의 하단에는 다음으로 감염 확률이 높은 2 개의 결과를 추가로 표시해준다.
더욱 자세한 내용이 궁금한 사용자를 위해 더 알아보기 버튼을 누르면 해당 질병의 증상설명, 발생환경, 방제방법과 추가적인 증상 이미지를 제공해준다.



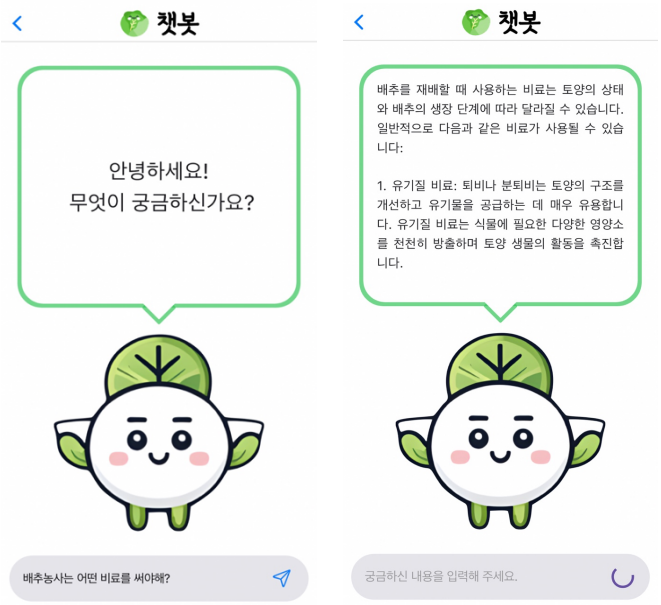
2.2 진단가능 병충해 확인하기

본 어플에서 진단 가능한 병충해 목록을 확인할 수 있고
병명을 누르면 국가농작물병해충관리시스템으로 이동하여 관련정보 확인이 가능하다.



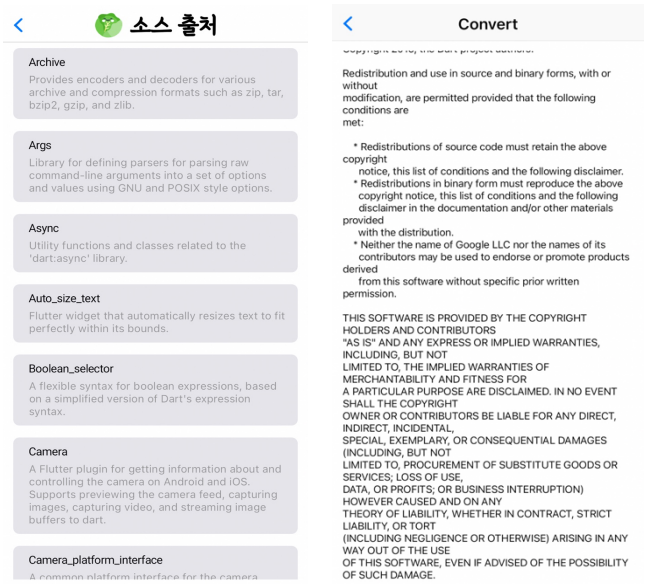
2.3 AI 챗봇에게 질문하기

궁금한 내용을 질문하면 GPT-4 Turbo 기반의 챗봇은 2023년 4월 까지의 데이터를 기반으로 최적의 답변을 생성해준다.



2.4 어플리케이션 소스 확인하기

어플리케이션 개발 과정에서 사용한 패키지들의 라이센스를 준수하기 위해 출처를 명시해주었다.

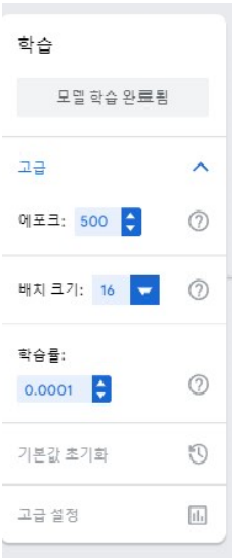


IV 한계점과 개선방안

1. 모델 정확도 개선

1.1 한계점

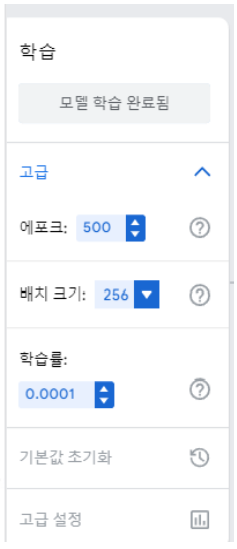
기존의 AI 모델은 노균병 이미지를 무름병으로 오진하는 문제가 있었다.
우리는 해당 문제의 원인을 적절하지 못한 에포크와 배치 크기 때문이라고 판단했다.



<에포크란?>
모델이 데이터 셋을 학습하는 횟수를 의미하며 에포크 값이 지나치게 높으면 과적합 문제가 발생한다.

<배치 크기란?>
학습연산 한번에 들어가는 데이터의 크기를 의미하며 너무 작은 값인 경우 가중치 업데이트가 빈번하게 일어나 훈련이 불안정해진다.

1.2 개선방안



배치 크기를 다양하게 변경해보며 훈련한 결과 배치 크기가 256일때 제대로 된 진단결과를 얻을 수 있었다.

기존의 오진 문제는 배치 크기가 너무 작아 모델이 개별 데이터 포인트에 과도하게 반응하여, 전반적인 패턴을 놓쳤기 때문으로 사료된다.

2. Data Augmentation

2.1 한계점

훈련 데이터 셋을 과도하게 학습하여 훈련에 사용된 이미지에 대해서는 정확한 판단을 내리지만 새로운 이미지에는 부정확한 판단을 내리는 과적합(Overfitting) 문제가 발생했다. 또한 배추 무름병과 뿌리혹병의 경우에는 학습 이미지가 너무 적어 정확도가 떨어지는 문제가 있었다.

2.2 개선방안

기존의 훈련 이미지들을 Cropping, Rotation, Color Shifting하는 Data Augmentation 기법을 사용하여 학습되는 데이터의 양을 증가시켰다.

< OpenCV를 이용한 Data Augmentation>

```
for i in range(1, num_augmented_images):
    change_picture_index = random.randrange(1, total_origin_image_num-1)
    print(change_picture_index)
    print(file_names[change_picture_index])
    file_name = file_names[change_picture_index]

    origin_image_path = 'D:/Disease/' + file_name
    print(origin_image_path)
    image = Image.open(origin_image_path)
    random_augment = random.randrange(1,4)

    if(random_augment == 1):
        #이미지 좌우 반전
        print("invert")
        inverted_image = image.transpose(Image.FLIP_LEFT_RIGHT)
        inverted_image.save(file_path + 'inverted_' + str(augment_cnt) + '.png')

    elif(random_augment == 2):
        #이미지 기울이기
        print("rotate")
        rotated_image = image.rotate(random.randrange(-20, 20))
        rotated_image.save(file_path + 'rotated_' + str(augment_cnt) + '.png')
```

```
elif(random_augment == 2):
    #이미지 기울이기
    print("rotate")
    rotated_image = image.rotate(random.randrange(-20, 20))
    rotated_image.save(file_path + 'rotated_' + str(augment_cnt) + '.png')

elif(random_augment == 3):
    #노이즈 추가하기
    img = cv2.imread(origin_image_path)
    print("noise")
    row,col,ch= img.shape
    mean = 0
    var = 0.1
    sigma = var**0.5
    gauss = np.random.normal(mean,sigma,(row,col,ch))
    gauss = gauss.reshape(row,col,ch)
    noisy_array = img + gauss
    noisy_image = Image.fromarray(np.uint8(noisy_array)).convert('RGB')
    noisy_image.save(file_path + 'noiseAdded_' + str(augment_cnt) + '.png')

augment_cnt += 1
```

< Rotation >

< Color Shifting >

30장의 무름병 이미지를 199장의 이미지로, 40장의 뿌리혹병 이미지를 200장의 이미지로 늘림으로써 과적합 문제와 데이터 부족 문제를 한번에 해결 할 수 있었다.