

< CNN 을 활용한 토마토 잎 질병 이미지 분류>

1. 서론
2. 데이터셋 설명
3. 모델 설명
4. 실험 방법
5. 결과 및 분석
6. 결론
7. 참고 문헌 및 부록

1. 서론

1.1 문제 정의

토마토는 전 세계적으로 중요한 농산물 중 하나이나, 곰팡이·바이러스·박테리아 감염으로 인해 심각한 피해를 입고 있다. 토마토 질병으로 인한 손실은 전체 작물의 20~30%에 달한다. 현재 농가는 육안 검사와 수동 분석을 통해 질병을 진단하고 있어 비효율적이며 감별 오류의 위험이 있다. 조기 진단 실패 시 감염 확산과 방제 비용 증가로 이어질 수 있다. 본 연구에서는 CNN 모델을 활용해 정상 잎과 감염 잎을 자동 분류함으로써 예방 조치의 가능성을 높이고자 한다.

1.2 연구의 중요성

본 연구는 CNN 기반 이미지 분류 모델을 통해 토마토 질병을 자동 진단하며, 농업 생산성 향상 및 환경 보호에 기여할 수 있다. 데이터 기반 질병 예측 모델을 통해 기후 변화에 따른 질병 발생률을 정량적으로 분석하고, 농업 생산성 저하 시기를 사전에 예측할 수 있다. 이는 작물 보호 전략 수립과 경제적 손실 최소화에 도움이 되며, 스마트팜과 연계한 실시간 질병 경보 IoT 시스템으로 확장 가능하다. 또한 조기 감지와 최적 방제 시기 예측을 통해 농약 사용을 줄이고 방제 비용을 절감하며 친환경 농업 실현 가능성을 제공한다.

2. 데이터셋 설명

우선 분석을 위해 Kaggle (<https://www.kaggle.com/datasets/cookiefinder/tomato-disease-multiple-sources/code>) 의 데이터를 활용했다. Tomato Disease Source 데이터셋은 토마토 잎에 발생하는 10 가지 질병과 정상 잎을 포함한 총 11 개의 클래스의 이미지 데이터셋이다. 이 데이터셋은 모델의 일반화 성능을 높이는 데 유리하며, 각 이미지에는 정확한 질병 라벨이 지정되어 있어 딥러닝 기반 CNN 모델을 활용한 자동 질병 감지 연구에 적합하다.

이 데이터셋을 통해 모델 생성 시 질병 오분류를 방지하여 농업 종사자들이 신속하게 질병을 탐지하고 대응할 수 있도록 돋는 의사결정 지원 시스템 구축에 활용될 것이다.

해당 자료는 Kaggle의 api를 통해 내려 받을 수 있으며, 아래와 같다.

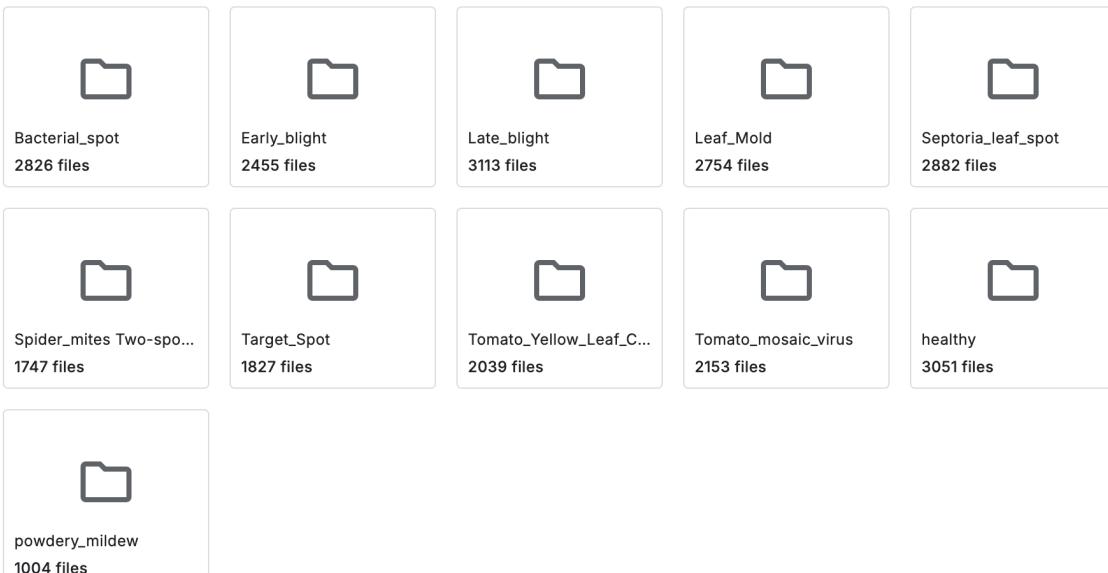
train (11 directories)

⤵ ⤶ ⤷

About this directory

 Suggest Edits

Folders for training. 25851 images



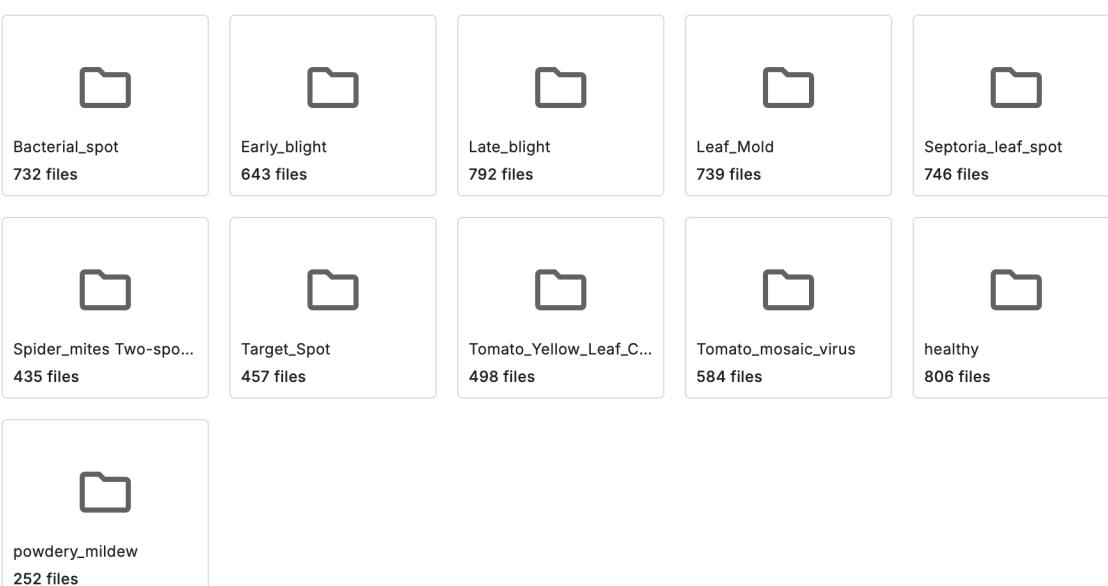
valid (11 directories)

⤵ ⤶ ⤷

About this directory

 Suggest Edits

Folder for validation and testing. 6684 images



3. 모델 설명

이번 프로젝트에서 토마토 질병 데이터셋을 학습시키기 위해 선택한 사전 훈련 모델은 DenseNet121이다. DenseNet121은 2017년 Huang et al.에 의해 개발된 모델로, CNN 아키텍처에서 효율적인 특성 재사용과 정보 흐름 최적화를 목표로 설계되었다. 이 모델은 밀집 연결(Dense Connectivity)을 활용하여, 네트워크의 깊이가 증가하더라도 기울기 소실(Vanishing Gradient) 문제를 완화하고, 파라미터 수를 줄이며, 학습 성능을 향상시키는 것이 특징이다.

DenseNet121을 선택한 이유는 다음과 같다.

첫째, 우수한 성능을 제공한다. DenseNet은 기존 CNN 모델과 비교했을 때, 적은 수의 파라미터로도 높은 정확도를 달성할 수 있으며, 특히 이미지 분류 작업에서 뛰어난 성능을 기록한 바 있다. 따라서, 토마토 질병 데이터셋에서도 높은 분류 정확도를 기대할 수 있다.

둘째, 효율적인 특성 학습이 가능하다. DenseNet의 핵심 개념인 밀집 연결 구조는 모든 이전 층의 출력을 현재 층의 입력으로 사용하여, 특성 재사용(feature reuse)을 극대화하고, 정보 흐름(information flow)을 최적화할 수 있다. 이를 통해 모델이 데이터의 중요한 패턴을 효과적으로 학습할 것으로 판단하였다.

셋째, 사전 훈련된 가중치를 활용할 수 있다. DenseNet121은 ImageNet 데이터셋으로 사전 학습된 가중치를 제공하며, 이를 기반으로 전이 학습(Transfer Learning)이 가능하다. 이는 토마토 질병 데이터셋과 같이 데이터량이 비교적 적은 경우에도 효율적인 학습과 높은 성능을 보장할 수 있음을 의미한다.

DenseNet121은 메모리 사용량이 많고 연산량이 높은 단점이 있다. 그러나, GPU 설정을 최적화하고, 필요에 따라 모델의 일부 층을 동결하거나 입력 크기를 조정하는 등의 방법을 통해 이를 보완할 수 있다. 이러한 점을 고려할 때, DenseNet121은 높은 성능, 효율적인 특성 학습, 전이 학습 지원 등의 강점을 바탕으로 본 연구에서 최적의 모델로 선정되었다.

4. 실험 방법

[1] 필요한 라이브러리 설치

```
import os  
import tensorflow as tf  
import matplotlib.pyplot as plt
```

```

import matplotlib.image as mpimg
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense,
Flatten, Dropout, BatchNormalization
from tensorflow.keras.applications import ResNet50
from tensorflow.keras import layers, models
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.densenet import preprocess_input
import seaborn as sns
from sklearn.metrics import confusion_matrix
import numpy as np

```

[2] 데이터셋 다운로드 및 준비

```

# Kaggle API 설치
# !pip install kaggle

# kaggle.json 파일 업로드
from google.colab import files
files.upload() # kaggle.json 파일을 업로드하세요.

# Kaggle 폴더 생성 및 권한 설정
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# 데이터셋 다운로드
!kaggle datasets download -d cookiefinder/tomato-disease-multiple-
sources

# 압축 해제
!unzip tomato-disease-multiple-sources.zip -d ./tomato_disease_data

```

[3] EDA 데이터셋 시각화

아래는 데이터셋을 시각화한 코드 및 결과이다. 토마토 잎 질병의 이미지를 표현한다.

a. 특정 클래스 이미지 시각화 (Bacterial_spot)

```

# 1. 특정 클래스 이미지 시각화
# 디렉토리 경로 설정 (원하는 클래스 폴더로)
path = "/content/tomato_disease_data/train/Bacterial_spot"

# 폴더 내 이미지 파일 목록
image_files = [f for f in os.listdir(path) if
os.path.isfile(os.path.join(path, f))]

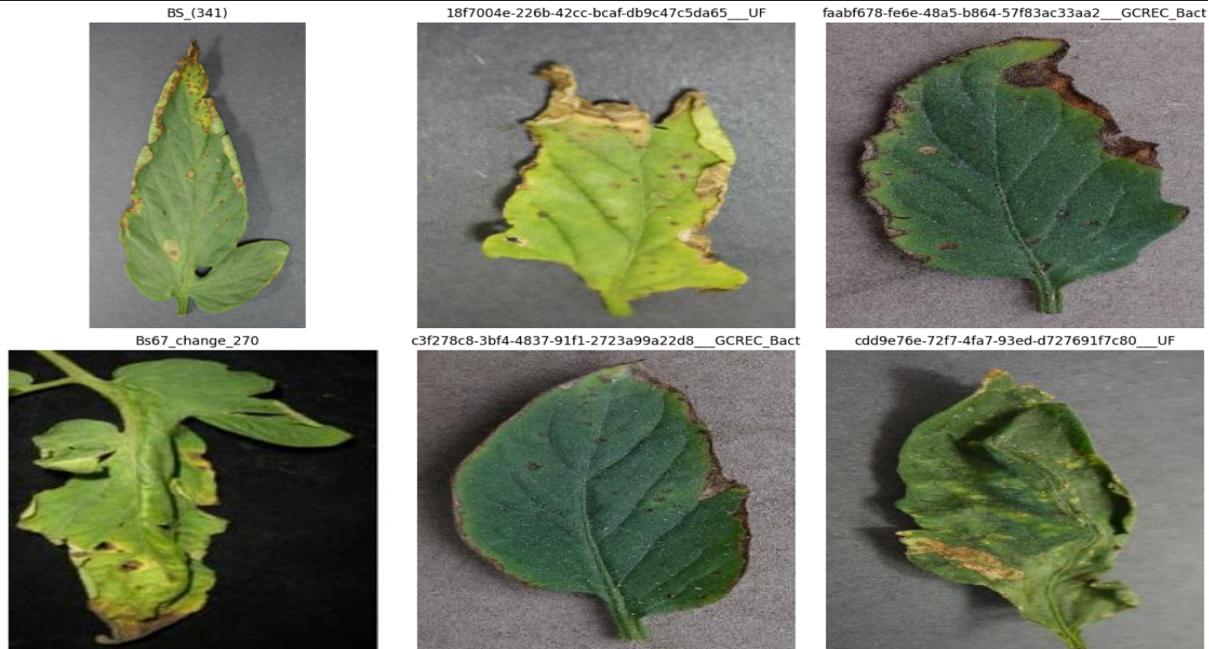
```

```
# 2행 3열의 서브플롯 (총 6장 이미지)
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

for i in range(6):
    image_file = image_files[i]
    label = image_file.split('.')[0] # 확장자 제거
    img_path = os.path.join(path, image_file)
    img = mpimg.imread(img_path)

    ax = axs[i // 3, i % 3]
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(label)

plt.tight_layout()
plt.show()
```



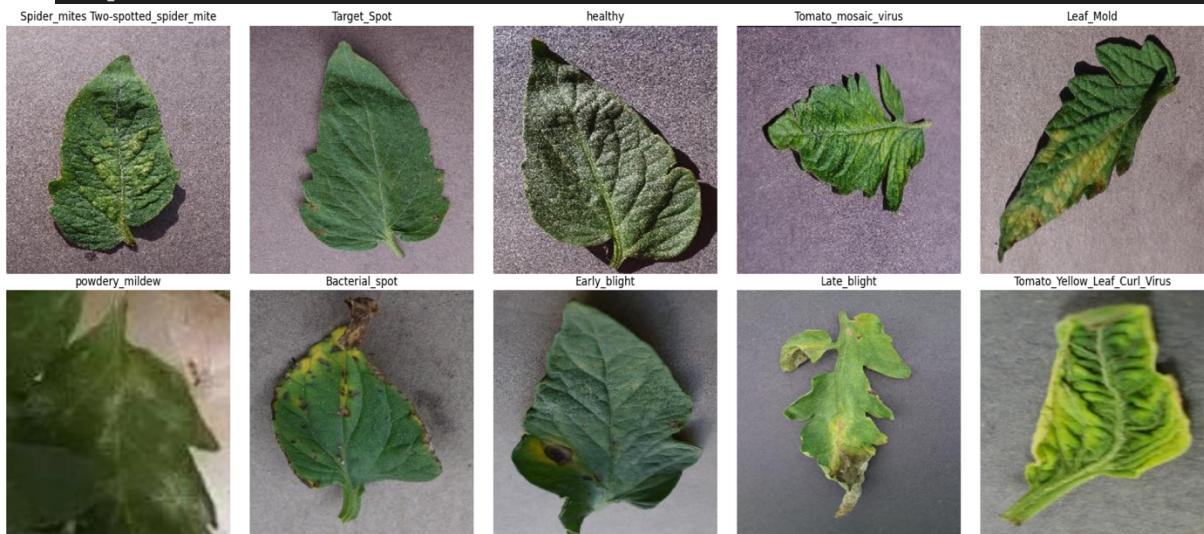
b. 여러 클래스에서 하나씩 샘플 보기

```
c. # 2. 여러 클래스에서 하나씩 샘플 보기
d. base_dir = "/content/tomato_disease_data/train"
e. class_names = os.listdir(base_dir)
f.
g. # 각 클래스에서 무작위 이미지 하나씩 추출
h. fig, axs = plt.subplots(2, 5, figsize=(20, 8))
i.
j. for i, class_name in enumerate(class_names[:10]): # 최대 10 개
    클래스만 표시
k.     class_path = os.path.join(base_dir, class_name)
l.     image_files = os.listdir(class_path)
m.     image_file = random.choice(image_files)
```

```

n.
o.     img_path = os.path.join(class_path, image_file)
p.     img = mpimg.imread(img_path)
q.
r.     ax = axs[i // 5, i % 5]
s.     ax.imshow(img)
t.     ax.axis('off')
u.     ax.set_title(class_name)
v.
w. plt.tight_layout()
x. plt.show()

```



[4] 손상된 파일 삭제하기

```

def find_and_delete_corrupted_images(directory):
    corrupted_files = []
    total_checked = 0

    for root, dirs, files in os.walk(directory):
        for fname in files:
            fpath = os.path.join(root, fname)
            try:
                with Image.open(fpath) as img:
                    img.verify() # 실제 이미지 파일이 맞는지 확인
            total_checked += 1
        except (IOError, SyntaxError):
            corrupted_files.append(fpath)
            os.remove(fpath) # 손상된 파일 삭제
            print(f"Deleted corrupted file: {fpath}")

    print(f"\n검사한 총 이미지 수: {total_checked + len(corrupted_files)}")
    print(f"삭제한 손상 파일 수: {len(corrupted_files)}")
    if not corrupted_files:
        print("손상된 파일이 없습니다.")

```

```

검사한 총 이미지 수: 25851
삭제한 손상 파일 수: 0
손상된 파일이 없습니다.
Deleted corrupted file: ./tomato_disease_data/valid/healthy/HL_(336).png

검사한 총 이미지 수: 6684
삭제한 손상 파일 수: 1

```

[5] 데이터 증강 적용 전 데이터셋 현황

```

def count_images_in_dir(data_path):
    class_counts = {}
    for class_name in os.listdir(data_path):
        class_path = os.path.join(data_path, class_name)
        if os.path.isdir(class_path):
            count = len([
                f for f in os.listdir(class_path)
                if os.path.isfile(os.path.join(class_path, f))
            ])
            class_counts[class_name] = count
    return class_counts

train_dir = './tomato_disease_data/train'
valid_dir = './tomato_disease_data/valid'

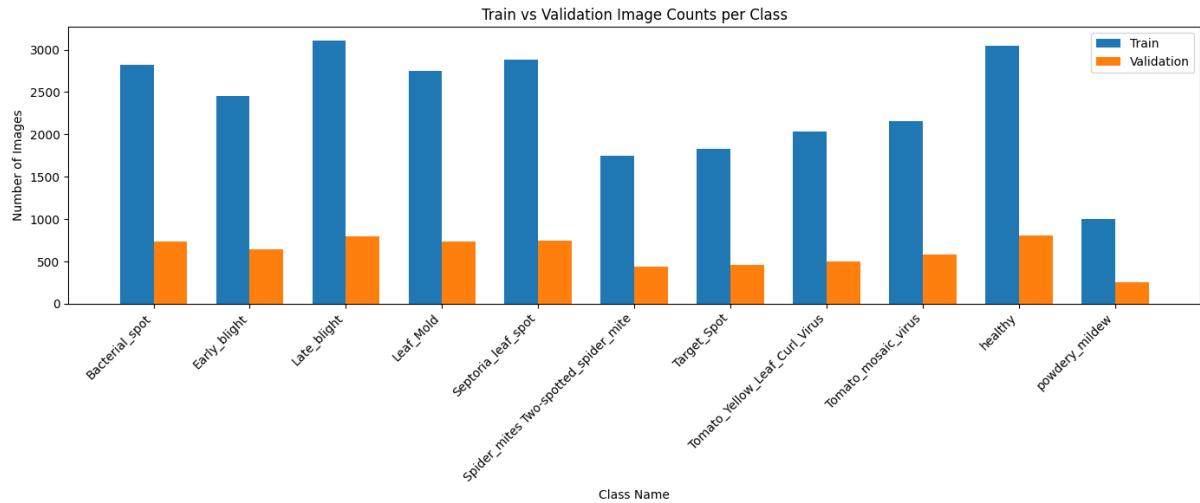
train_counts = count_images_in_dir(train_dir)
valid_counts = count_images_in_dir(valid_dir)

# 막대그래프 (train vs valid)
labels = sorted(set(train_counts.keys()) | set(valid_counts.keys()))
train_values = [train_counts.get(label, 0) for label in labels]
valid_values = [valid_counts.get(label, 0) for label in labels]

x = range(len(labels))
width = 0.35

plt.figure(figsize=(14, 6))
plt.bar(x, train_values, width=width, label='Train')
plt.bar([i + width for i in x], valid_values, width=width,
label='Validation')
plt.xticks([i + width/2 for i in x], labels, rotation=45, ha='right')
plt.title('Train vs Validation Image Counts per Class')
plt.xlabel('Class Name')
plt.ylabel('Number of Images')
plt.legend()
plt.tight_layout()
plt.show()

```



데이터셋의 시각화 결과를 보면 알 수 있듯이 powdery_mildew의 수가 상대적으로 적다.

[6] 데이터증강 정도 설정하기

ImageDataGenerator를 통해 회전, 이동, 확대, 밝기 변화, 수평 뒤집기 등 기본적인 이미지 증강이 설정되어 있으며, 학습 속도를 고려하여 이미지 크기는 (160, 160)으로 설정했다.

```
# 경로 및 설정
train_dir = './tomato_disease_data/train'
img_size = (160, 160)

# 클래스 이름 확인 (소문자/대소문자/띄어쓰기 등 정확히)
class_aug_map = {
    'powdery_mildew': 2
}

default_num_augmented = 0 # 타겟 클래스가 아닌 경우 증강 없음

# 기본 증강 정의 (모든 증강은 basic_aug 사용)
basic_aug = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=10,
    width_shift_range=0.05,
    height_shift_range=0.05,
    zoom_range=0.1,
    horizontal_flip=True,
    brightness_range=[0.95, 1.05],
    fill_mode='nearest'
)
```

[7] 데이터증강 적용하기

상대적으로 데이터셋이 작은 'powdery_mildew' 클래스에만 2 배의 증강을 적용하고 나머지 클래스는 증강하지 않는다.

```
# 결과 저장 리스트
images = []
labels = []

# 클래스 이름 및 인덱스 매핑
class_names = sorted([d for d in os.listdir(train_dir) if
os.path.isdir(os.path.join(train_dir, d))])
class_indices = {name: idx for idx, name in enumerate(class_names)}

# 클래스별 이미지 처리 및 증강
for class_name in class_names:
    class_path = os.path.join(train_dir, class_name)
    label_idx = class_indices[class_name]
    num_augmented = class_aug_map.get(class_name,
default_num_augmented)

    for img_name in os.listdir(class_path):
        img_path = os.path.join(class_path, img_name)

        try:
            # 이미지 로드 및 전처리
            img = load_img(img_path, target_size=img_size)
            x = img_to_array(img)
            x = preprocess_input(x)

            # 원본 이미지 저장
            images.append(x)
            labels.append(label_idx)

            # 증강 이미지 생성 및 저장
            for _ in range(num_augmented):
                aug_img = basic_aug.random_transform(x)
                images.append(aug_img)
                labels.append(label_idx)

        except Exception as e:
            print(f"[오류] {img_path} - {e}")
            continue

# 배열로 변환
X_train = np.array(images)
y_train = to_categorical(labels, num_classes=len(class_names))

print(f"총 train 이미지 수: {X_train.shape[0]}")
```

```

print(f"총 train 클래스 수: {len(class_names)}")
print(f"X_train shape: {X_train.shape}")
print(f"y_train shape: {y_train.shape}")

# 검증 데이터 제너레이터
val_datagen =
ImageDataGenerator(preprocessing_function=preprocess_input)
validation_generator = val_datagen.flow_from_directory(
    './tomato_disease_data/valid',
    target_size=img_size,
    batch_size=64,
    class_mode='categorical',
    shuffle=False
)
총 train 이미지 수: 27859
총 train 클래스 수: 11
X_train shape: (27859, 160, 160, 3)
y_train shape: (27859, 11)
Found 6683 images belonging to 11 classes.

```

[8] 데이터증강 적용 후 데이터셋 현황

```

def plot_class_distribution(y_data, class_names, title='Train Data
Class Distribution'):
    # 클래스 인덱스 추출 (one-hot 인코딩 → 정수 인덱스)
    label_indices = np.argmax(y_data, axis=1)

    # 클래스별 개수 계산
    unique, counts = np.unique(label_indices, return_counts=True)

    # 시각화
    plt.figure(figsize=(12, 6))
    plt.bar([class_names[i] for i in unique], counts, color='skyblue')
    plt.title(title)
    plt.xlabel('Class Name')
    plt.ylabel('Number of Images')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

plot_class_distribution(y_train, class_names)

```



[9] 클래스 가중치

아래 코드는 다중 분류 문제에서 클래스 불균형을 완화하기 위해 compute_class_weight 함수를 사용하여 클래스 가중치를 계산하고, 이를 딕셔너리 형태로 변환한 것이다

```
# y_train은 원-핫 인코딩이므로 argmax로 레이블 추출
y_labels = np.argmax(y_train, axis=1)

# 클래스 가중치 계산
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_labels),
    y=y_labels
)

# 딕셔너리 형태로 변환
class_weights_dict = {i : w for i, w in enumerate(class_weights)}
print("클래스 가중치:", class_weights_dict)

클래스 가중치: {0: np.float64(0.8961912114778356), 1:
np.float64(1.0316237733753009), 2: np.float64(0.8135677364716877), 3:
np.float64(0.9196210470720274), 4: np.float64(0.8787773642041512), 5:
np.float64(1.4497059894884738), 6: np.float64(1.3862268000199034), 7:
np.float64(1.2420972847652592), 8: np.float64(1.1763290123717434), 9:
np.float64(0.8301004141712106), 10: np.float64(0.8408487263068937)}
```

[10] 모델 생성 및 파인튜닝

본 연구에서는 DenseNet121의 사전 학습된 가중치를 활용한 전이 학습과 파인튜닝을 적용하여 토마토 질병 분류 모델을 개발하였다.

a. 사전 훈련된 CNN 불러오기 (DenseNet121)

```
# 사전 학습된 CNN 불러오기
base_model = DenseNet121(weights='imagenet', include_top=False,
input_shape=(160, 160, 3))

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5
29084464/29084464 1s 0us/step
```

b. 파인튜닝 (일부 층은 학습 가능하게 설정, 나머지는 동결)

초기에는 DenseNet121의 사전 학습된 가중치를 고정하여 기본적인 특징을 유지하고, 추가된 분류기만 학습하도록 설정하였다.

```
# 모든 레이어를 일단 학습 가능하게 설정 (fine-tuning 준비)
base_model.trainable = True
for layer in base_model.layers[:-30]: # 마지막 30개만 fine-tuning
    layer.trainable = False # 학습되지 않도록 동결
```

c. 모델 구성

```
# 모델 구성
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    BatchNormalization(),
    Dense(256, activation='relu'),
    Dropout(0.35),
    BatchNormalization(),
    Dense(120, activation='relu'),
    Dense(11, activation='softmax')
])
```

d. 모델 컴파일

모델을 학습시키기 위해 Adam 옵티마이저(학습률 0.0001)와 categorical_crossentropy 손실 함수를 설정하였다.

```
# 모델 컴파일
model.compile(optimizer=Adam(learning_rate=1e-5),
              loss='categorical_crossentropy',
```

e. 모델 학습

모델이 과적합되지 않도록 Early Stopping 기법을 적용하여 검증 손실이 일정 횟수 동안 감소하지 않으면 학습을 중단하도록 설정하였다.

EarlyStopping 콜백은 검증 손실(val_loss)을 기준으로 3 에폭 동안 개선이 없으면 학습을 중단하며, 가장 성능이 좋았던 시점의 가중치를 복원한다.

```
# 모델 학습
early_stopping = EarlyStopping(
    monitor='val_loss', # 검증 손실 기준으로 모니터링
    patience=3, # 3 에폭 연속으로 개선이 없으면 중단
    restore_best_weights=True # 가장 성능 좋았던 시점의 가중치 복원
)
# validation_generator 정의
val_datagen =
ImageDataGenerator(preprocessing_function=preprocess_input)
```

```

validation_generator = val_datagen.flow_from_directory(
    './tomato_disease_data/valid',
    target_size=(160, 160),
    batch_size=64,
    class_mode='categorical',
    shuffle=False
)

history = model.fit(X_train, y_train, epochs=100,
validation_data=validation_generator, callbacks=[early_stopping])

```

```

Epoch 9/100
871/871 21s 25ms/step - accuracy: 0.8248 - loss: 0.5683 - val_accuracy: 0.8689 - val_loss: 0.4284
Epoch 10/100
871/871 22s 26ms/step - accuracy: 0.8401 - loss: 0.5161 - val_accuracy: 0.8815 - val_loss: 0.3889
Epoch 11/100
871/871 22s 25ms/step - accuracy: 0.8542 - loss: 0.4687 - val_accuracy: 0.8896 - val_loss: 0.3569
Epoch 12/100
871/871 22s 26ms/step - accuracy: 0.8671 - loss: 0.4240 - val_accuracy: 0.8985 - val_loss: 0.3279
Epoch 13/100
871/871 23s 27ms/step - accuracy: 0.8795 - loss: 0.3856 - val_accuracy: 0.9048 - val_loss: 0.3045
Epoch 14/100
871/871 22s 25ms/step - accuracy: 0.8830 - loss: 0.3675 - val_accuracy: 0.9131 - val_loss: 0.2853
Epoch 15/100
871/871 22s 25ms/step - accuracy: 0.8945 - loss: 0.3371 - val_accuracy: 0.9195 - val_loss: 0.2657
Epoch 16/100
871/871 22s 25ms/step - accuracy: 0.9057 - loss: 0.3028 - val_accuracy: 0.9208 - val_loss: 0.2510
Epoch 17/100
871/871 22s 25ms/step - accuracy: 0.9135 - loss: 0.2770 - val_accuracy: 0.9267 - val_loss: 0.2359
Epoch 18/100
871/871 22s 25ms/step - accuracy: 0.9175 - loss: 0.2630 - val_accuracy: 0.9304 - val_loss: 0.2220
Epoch 19/100
871/871 22s 25ms/step - accuracy: 0.9238 - loss: 0.2496 - val_accuracy: 0.9345 - val_loss: 0.2110
Epoch 20/100
871/871 22s 25ms/step - accuracy: 0.9317 - loss: 0.2196 - val_accuracy: 0.9382 - val_loss: 0.2005
Epoch 21/100
871/871 22s 25ms/step - accuracy: 0.9344 - loss: 0.2133 - val_accuracy: 0.9404 - val_loss: 0.1898
Epoch 22/100
871/871 22s 25ms/step - accuracy: 0.9411 - loss: 0.1887 - val_accuracy: 0.9422 - val_loss: 0.1834
Epoch 23/100
871/871 22s 26ms/step - accuracy: 0.9477 - loss: 0.1777 - val_accuracy: 0.9446 - val_loss: 0.1762
Epoch 24/100
871/871 22s 25ms/step - accuracy: 0.9485 - loss: 0.1677 - val_accuracy: 0.9466 - val_loss: 0.1699
Epoch 25/100
871/871 22s 25ms/step - accuracy: 0.9536 - loss: 0.1556 - val_accuracy: 0.9481 - val_loss: 0.1635
Epoch 26/100
871/871 23s 26ms/step - accuracy: 0.9595 - loss: 0.1370 - val_accuracy: 0.9481 - val_loss: 0.1591
Epoch 27/100
871/871 22s 25ms/step - accuracy: 0.9594 - loss: 0.1329 - val_accuracy: 0.9517 - val_loss: 0.1531
Epoch 28/100
871/871 22s 25ms/step - accuracy: 0.9665 - loss: 0.1179 - val_accuracy: 0.9536 - val_loss: 0.1486
Epoch 29/100
871/871 22s 25ms/step - accuracy: 0.9676 - loss: 0.1100 - val_accuracy: 0.9538 - val_loss: 0.1458
Epoch 30/100
871/871 22s 25ms/step - accuracy: 0.9682 - loss: 0.1078 - val_accuracy: 0.9559 - val_loss: 0.1426
Epoch 31/100
871/871 23s 26ms/step - accuracy: 0.9701 - loss: 0.1022 - val_accuracy: 0.9584 - val_loss: 0.1398
Epoch 32/100

```

5. 결과 및 분석

[1] 모델 평가 및 결과 시각화

a. 모델 평가

```

# 모델 평가 (Loss & Accuracy 출력)
evaluation = model.evaluate(validation_generator)
print(f"검증 손실: {evaluation[0]:.4f}")

```

```

print(f"검증 정확도: {evaluation[1] * 100:.2f}%)")
105/105 - 15s 140ms/step - accuracy: 0.9623 - loss: 0.1399
검증 손실: 0.1173
검증 정확도: 96.59%

```

b. 시각화 (그래프)

```

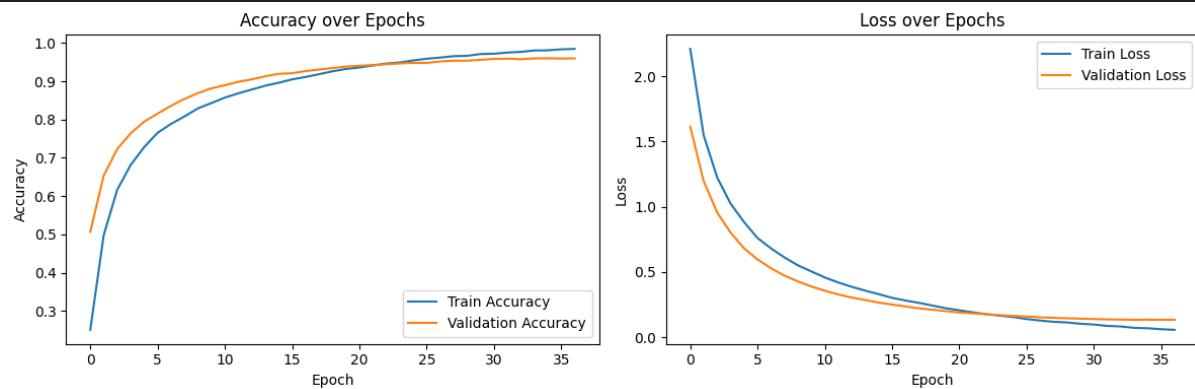
# 학습 과정 시각화
plt.figure(figsize=(12, 4))

# 정확도(Accuracy) 그래프
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy')

# 손실(Loss) 그래프
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')

plt.show()

```



모델의 학습 결과, 훈련 및 검증 정확도는 에폭이 증가함에 따라 꾸준히 상승하였으며, 최종적으로 약 96% 이상의 높은 정확도를 달성하였다. 손실 값 역시 감소 추세를 보이며 과적합 없이 안정적인 학습이 이루어졌음을 확인할 수 있다.

c. 시각화 (흔동행렬)

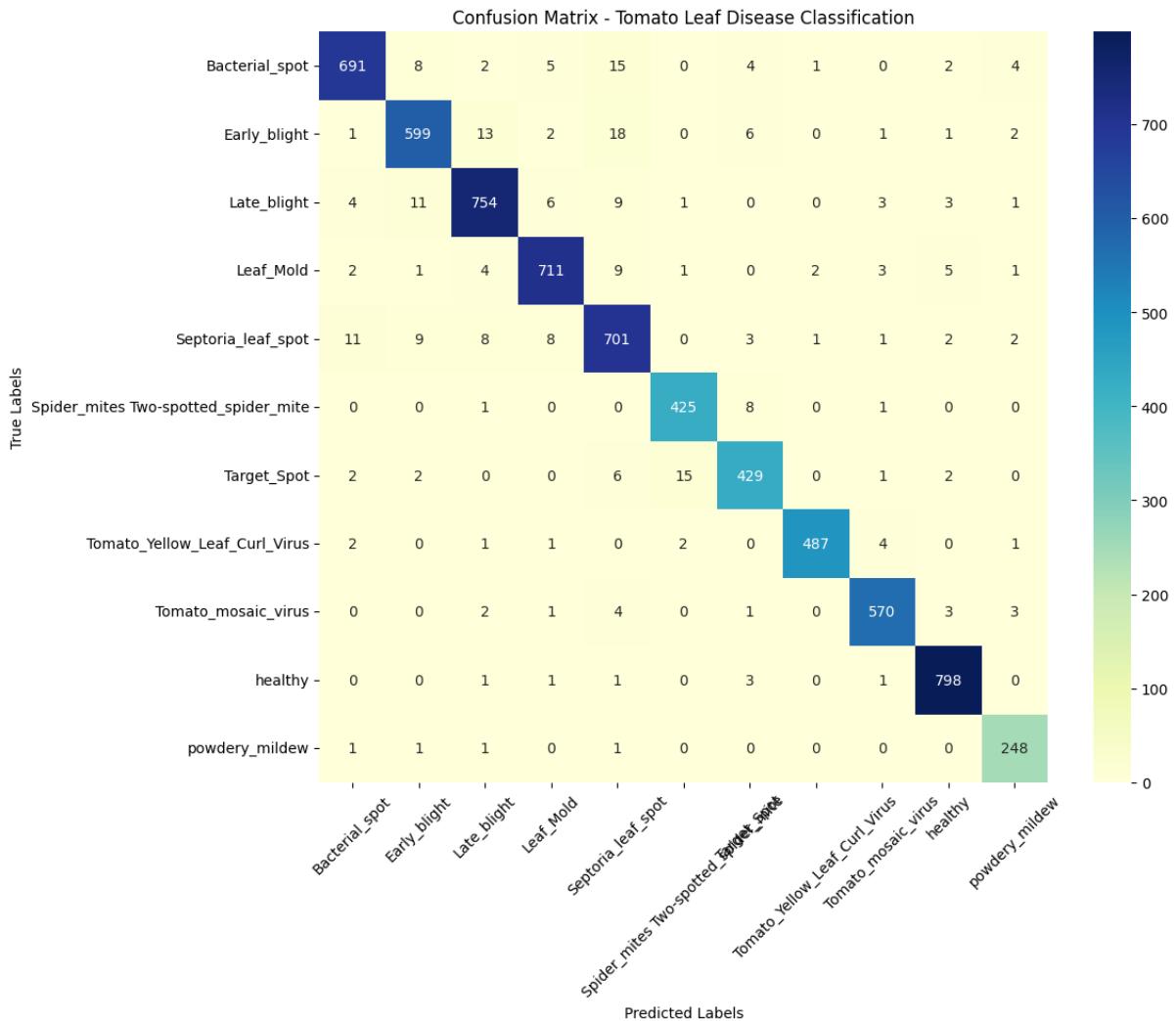
```
# 검증 데이터셋 예측
y_pred_probs = model.predict(validation_generator)
y_pred = np.argmax(y_pred_probs, axis=1)

# 실제 라벨
y_true = validation_generator.classes

# 클래스 이름 추출
class_names = list(validation_generator.class_indices.keys())

# 혼동 행렬 계산
cm = confusion_matrix(y_true, y_pred)

# 시각화
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Tomato Leaf Disease Classification')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
```



흔동 행렬에서는 대부분의 클래스에서 높은 정확도를 나타냈음을 알 수 있다.

d. 시각화 (흔동행렬 - 정규화)

```
# 예측값 및 실제값
y_pred_probs = model.predict(validation_generator)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = validation_generator.classes
class_names = list(validation_generator.class_indices.keys())

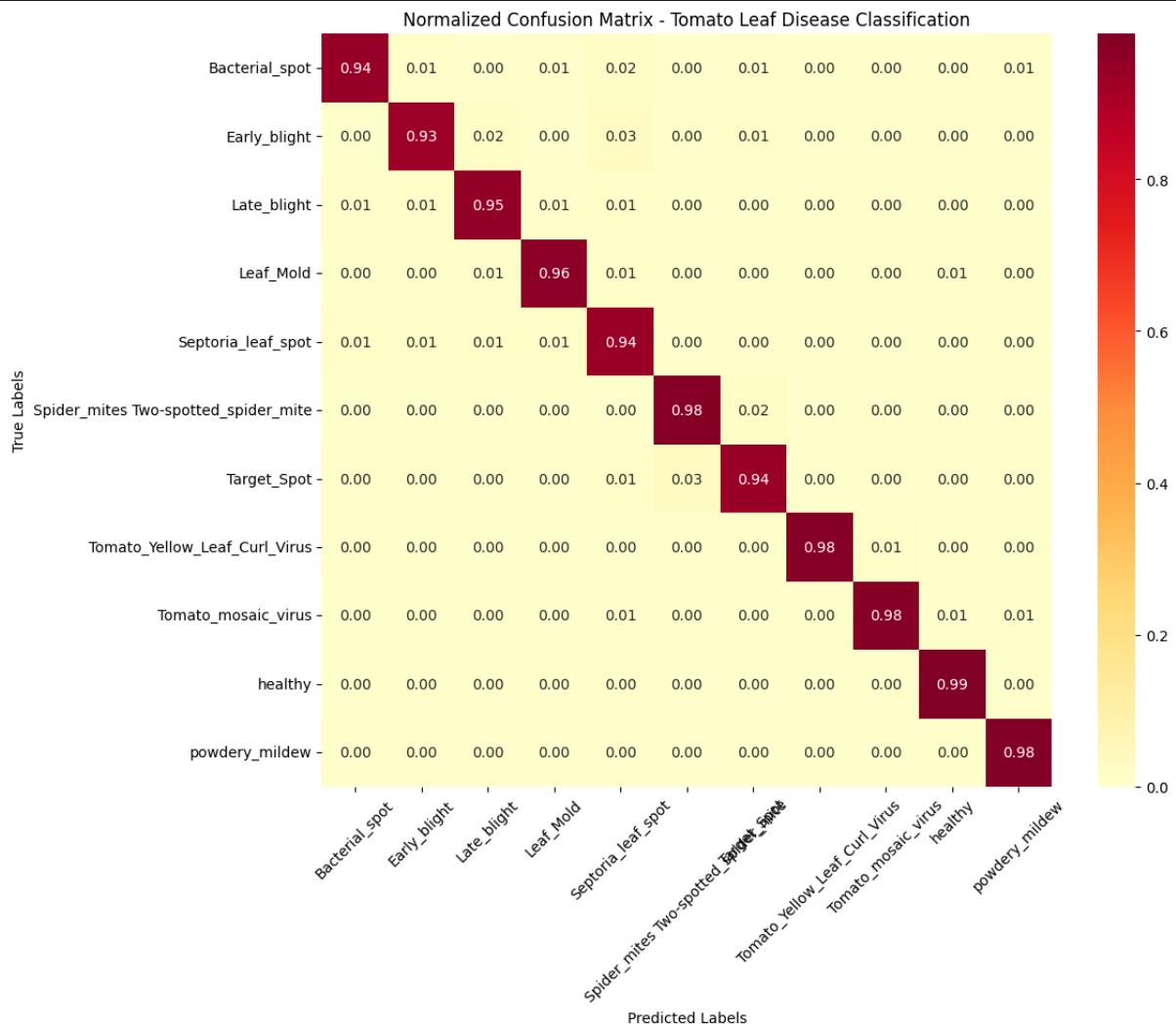
# 정규화된 흔동 행렬
cm_norm = confusion_matrix(y_true, y_pred, normalize='true')

# 시각화
plt.figure(figsize=(12, 10))
sns.heatmap(cm_norm, annot=True, fmt='.2f', cmap='YlOrRd',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
```

```

plt.title('Normalized Confusion Matrix - Tomato Leaf Disease Classification')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

```



정규화된 혼동 행렬에서도 클래스 간 예측 비율이 0.9 이상으로 나타나 전체적으로 모델이 클래스 간 구분을 잘 수행하고 있음을 보여준다.

e. Precision, Recall, F1-score 지표

```

# 예측값 및 실제값
y_pred_probs = model.predict(validation_generator)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = validation_generator.classes
class_names = list(validation_generator.class_indices.keys())

# 분류 성능 리포트 출력
report = classification_report(y_true, y_pred,
target_names=class_names, digits=4)
print("Classification Report (Precision, Recall, F1-score):\n")

```

```

print(report)
105/105 6s 61ms/step
Classification Report (Precision, Recall, F1-score):

          precision    recall   f1-score   support
Bacterial_spot      0.9678    0.9440    0.9557     732
Early_blight        0.9493    0.9316    0.9403     643
Late_blight         0.9581    0.9520    0.9550     792
Leaf_Mold           0.9673    0.9621    0.9647     739
Septoria_leaf_spot   0.9175    0.9397    0.9285     746
Spider_mites Two-spotted_spider_mite  0.9572    0.9770    0.9670     435
Target_Spot          0.9449    0.9387    0.9418     457
Tomato_Yellow_Leaf_Curl_Virus  0.9919    0.9779    0.9848     498
Tomato_mosaic_virus   0.9744    0.9760    0.9752     584
healthy              0.9779    0.9913    0.9846     805
powdery_mildew       0.9466    0.9841    0.9650     252

accuracy             -         -         0.9596     6683
macro avg            0.9594    0.9613    0.9602     6683
weighted avg         0.9597    0.9596    0.9596     6683

```

마지막으로 분류 리포트에서는 전체 정확도 95.96%, macro 평균 F1-score 는 96.02%로 나타났다.

e. Precision, Recall, F1-score 시각화

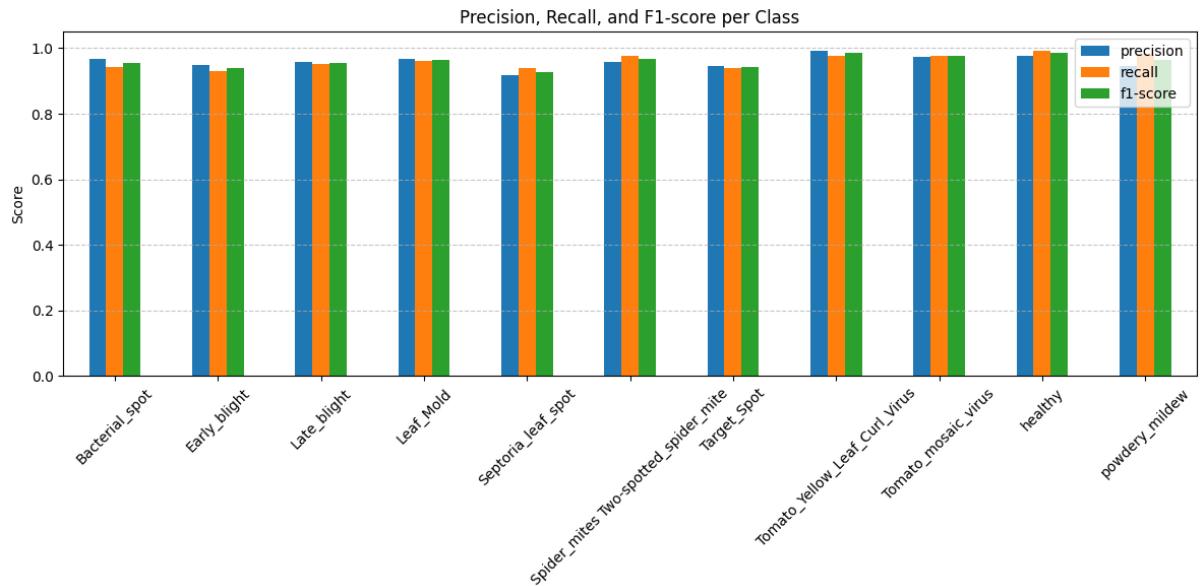
```

# 분류 보고서 생성 (dict 형태)
report = classification_report(y_true, y_pred,
target_names=class_names, output_dict=True)
report_df = pd.DataFrame(report).transpose()

# Precision, Recall, F1-score만 추출
metrics_to_plot = report_df.loc[class_names, ['precision', 'recall',
'f1-score']]

# 바 차트 시각화
metrics_to_plot.plot(kind='bar', figsize=(12, 6))
plt.title('Precision, Recall, and F1-score per Class')
plt.ylabel('Score')
plt.ylim(0, 1.05)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

```



[2] DB 저장하기

위 코드는 모델 학습 과정에서 발생한 에폭별 정확도(accuracy), 손실(loss), 검증 정확도(val_accuracy), 검증 손실(val_loss)을 Google Cloud SQL의 training_logs 테이블에 저장하는 기능을 수행한다.

```
import mysql.connector

# 1. MySQL DB 연결
conn = mysql.connector.connect(
    host='',
    user='',
    password='',
    database=''
)
cursor = conn.cursor()

# 2. 에폭별 학습 결과 저장
for epoch in range(len(history.history['accuracy'])):
    acc = history.history['accuracy'][epoch]
    val_acc = history.history['val_accuracy'][epoch]
    loss = history.history['loss'][epoch]
    val_loss = history.history['val_loss'][epoch]

    cursor.execute("""
        INSERT INTO training_logs (model_name, epoch, accuracy,
        val_accuracy, loss, val_loss)
        VALUES (%s, %s, %s, %s, %s, %s)
    """, ("DenseNet121", epoch + 1, acc, val_acc, loss, val_loss))

conn.commit()
conn.close()
```

```

CREATE TABLE IF NOT EXISTS training_logs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    model_name VARCHAR(100),
    epoch INT,
    accuracy FLOAT,
    val_accuracy FLOAT,
    loss FLOAT,
    val_loss FLOAT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)

```

```

SELECT * FROM training_logs
WHERE model_name = 'DenseNet121'
ORDER BY epoch ASC;

```

결과

id	model_name	epoch	accuracy	val_accuracy	loss	val_loss
1	DenseNet121	1	0.235005	0.463864	2.34503	1.73059
93	DenseNet121	1	0.236692	0.492444	2.2481	1.6501
136	DenseNet121	1	0.236692	0.492444	2.2481	1.6501
2	DenseNet121	2	0.480635	0.649858	1.62601	1.23362
94	DenseNet121	2	0.4993	0.656741	1.54076	1.20197
137	DenseNet121	2	0.4993	0.656741	1.54076	1.20197
3	DenseNet121	3	0.611616	0.722729	1.2509	0.963931
95	DenseNet121	3	0.624753	0.729762	1.20308	0.940681
138	DenseNet121	3	0.624753	0.729762	1.20308	0.940681
4	DenseNet121	4	0.686241	0.770163	1.02581	0.792011
96	DenseNet121	4	0.696795	0.771211	0.985259	0.77374
139	DenseNet121	4	0.696795	0.771211	0.985259	0.77374

[3] 모델 테스트

위 코드는 학습된 모델을 활용하여 무작위로 선택된 토마토 잎 이미지에 대한 질병을 예측하는 과정이다. 예측 결과는 가장 높은 확률을 가진 클래스 이름과 해당 확률을 출력하며, 이미지와 함께 시각화된다

```

import os
import random
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import cv2

# 클래스 이름 정의 (폴더 이름 순서와 동일하게!)
dataset_path = "/content/tomato_disease_data/train"

```

```

class_names = sorted(os.listdir(dataset_path)) # 폴더 이름 기준으로 자동 정의됨

# 예측에 사용할 샘플 이미지 경로 가져오기
sample_class = random.choice(class_names)
sample_folder = os.path.join(dataset_path, sample_class)
sample_image_name = random.choice(os.listdir(sample_folder))
sample_image_path = os.path.join(sample_folder, sample_image_name)

print(f"선택된 샘플 이미지: {sample_image_path} (클래스: {sample_class})")

# 모델 불러오기
model = load_model('/content/model.h5')

# 이미지 불러오기 및 전처리
img = image.load_img(sample_image_path, target_size=(160, 160))
img_array = image.img_to_array(img) / 255.0
img_array = np.expand_dims(img_array, axis=0)

# 예측
pred = model.predict(img_array)
pred_class = np.argmax(pred, axis=1)[0]
confidence = np.max(pred) * 100

# 결과 출력
print(f"예측 클래스: {class_names[pred_class]}")
print(f"모델 확신도: {confidence:.2f}%")

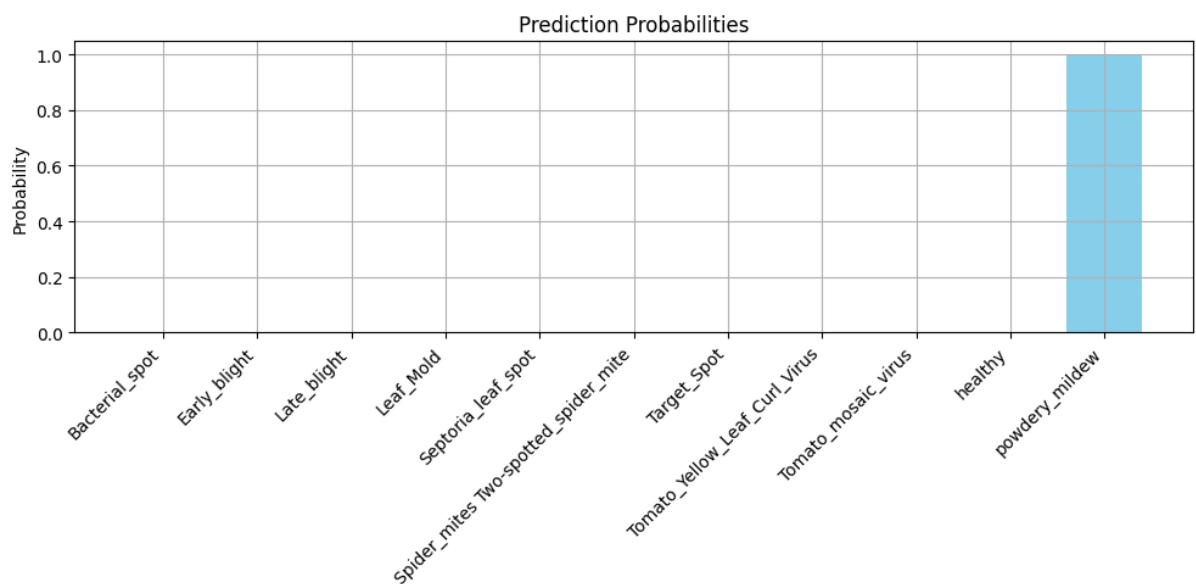
# 이미지와 함께 시각화
plt.imshow(img)
plt.axis('off')
plt.title(f"Prediction: {class_names[pred_class]} ({confidence:.2f}%)")
plt.show()

# Softmax 확률 시각화
plt.figure(figsize=(10, 5))
plt.bar(class_names, pred[0], color='skyblue')
plt.xticks(rotation=45, ha='right')
plt.ylabel('Probability')
plt.title('Prediction Probabilities')
plt.grid(True)
plt.tight_layout()
plt.show()

```

1/1 7s 7s/step
 예측 클래스: powdery_mildew
 모델 확신도: 99.73%

Prediction: powdery_mildew (99.73%)



결과 이미지에서는 powdery_mildew 로 분류되었고, 예측 확률은 약 99.73%로 매우 높았다. 또한 막대그래프를 통해 모든 클래스에 대한 예측 확률 분포를 확인할 수 있다.

6. 결론

본 연구에서는 토마토 잎에 발생하는 10 가지 질병과 정상 잎을 포함한 11 개 클래스를 대상으로, CNN 기반 DenseNet121 모델을 활용하여 자동 질병 분류 시스템을 개발하였다. 데이터 불균형을 완화하기 위해 'powdery_mildew' 클래스에만 데이터 증강을 적용하고, 클래스 가중치를 반영하여 학습 성능을 높였다. 사전 학습된 DenseNet121 모델의 특성을 활용한 전이 학습과 파인튜닝을 통해 약 96%의 정확도를 달성하였고, 혼동 행렬 및 정량 평가 지표(Precision, Recall, F1-score)를 통해 모델의 안정성과 클래스 간 구분 능력을 입증하였다.

최종적으로, 학습 이력을 데이터베이스에 저장하고 임의 이미지를 통한 예측 기능을 구현하여 실시간 응용 가능성까지 확보하였으며, 이는 스마트팜 환경에서의 질병 조기 진단과 친환경 농업 실현에 기여할 수 있는 가능성을 보여준다.

7. 참고 문헌 및 부록

- <https://www.kaggle.com/datasets/kaustubhb999/tomatoleaf/code?datasetId=619181&sortBy=voteCount>
- <https://www.kaggle.com/code/samanfatima7/tomato-leaf-disease-94-accuracy>
- <https://chatgpt.com/>