

kakao tech bootcamp

[personal project]

Transfer Learning 기반 쌀 잎 질병 분류 모델의
성능 비교 및 모니터링 시스템 구축



과정

팀

주강사님

이름

기간

인공지능

15 팀

Alex.lee(으)창신)

Anna.kim(김민선)

2025.03.17 - 2025.03.31

<목차>

- 1 서론
- 2 시스템 개요
 - 2.1 전체 구조 설명
 - 2.2 데이터셋 설명
- 3 모델 학습
 - 3.1 학습 모델 구조
 - 3.2 실험방법
 - 3.2.1 Custom CNN
 - 3.2.2 Resnet50
 - 3.2.3 VGG16
 - 3.2.4 Mobilenetv2
 - 3.2.5 Ensemble
- 4 MongoDB 및 시각화 결과분석
 - 4.1 데이터베이스 설계
 - 4.2 시각화 및 결과분석
 - 4.2.1 모델 상세 보기
 - 4.2.2 모델 비교 보기
 - 4.2.3 시간 필터
 - 4.2.4 학습 상태 실시간 모니터링
 - 4.2.4.1 실시간 에폭 반영 문제 해결
- 5 결론 및 향후 계획
- 6 참고문헌

1. 서론

최근 인공지능 기술의 발전과 함께 다양한 분야에서 딥러닝 기반 모델이 활용되고 있으며, 특히 영상 및 이미지 데이터를 활용한 분류, 탐지, 인식 분야에서 성과를 내고 있다. 그러나 단일 모델의 한계로 인해 데이터의 복잡성이나 다양성을 충분히 반영하지 못하는 문제가 존재한다. 이러한 한계를 극복하기 위해 최근에는 여러 모델을 융합하거나 다양한 학습 기법을 적용하는 방식의 연구가 활발히 진행되고 있다.

본 연구의 목적은 모델 학습 과정을 정밀하게 추적하고, 하이퍼파라미터 조정에 따른 성능 변화를 시각적으로 분석할 수 있는 시스템을 구축하는 데 있다. 학습의 전 과정에서 수집된 정보는 MongoDB를 기반으로 저장되며, Streamlit을 활용한 대시보드를 통해 사용자가 실시간으로 학습 상태를 확인할 수 있도록 한다. 특히, 에폭별 정확도와 손실값의 변화를 실시간으로 시각화함으로써, 학습의 흐름을 직관적으로 파악할 수 있는 환경을 제공한다.

이와 같은 시스템은 모델 학습의 투명성과 신뢰성을 높이는 데 기여할 수 있으며, 반복 실험이나 하이퍼파라미터 튜닝 과정에서 시간과 자원을 효율적으로 사용할 수 있는 기반을 마련한다. 특히, 복수의 실험을 동일한 조건 하에 반복 수행하여 성능의 일관성을 확인하고, 모델 개선 여부를 명확하게 판단할 수 있는 객관적 기준을 제공한다는 점에서 연구의 의의가 있다.

2. 시스템 개요

본 장에서는 본 연구에서 구현한 시스템의 전체 구조와 데이터셋, 사용된 기술 스택, 그리고 데이터 흐름 및 처리 파이프라인에 대해 설명한다.

2.1 전체 구조 설명

본 시스템은 다양한 CNN 모델(VGG16, ResNet50, MobileNetV2)을 학습시킨 뒤, 각 모델의 예측 결과를 양상화하여 최종 분류 성능을 향상시키는 구조로 구성되어 있다. 모델 학습 과정에서는 PyTorch 프레임워크를 활용하였으며, 학습 중 발생하는 에폭별 로그 및 상태 정보를 MongoDB에 저장하고, 이를 Streamlit 기반의 웹 대시보드에서 실시간으로 시각화할 수 있도록 하였다.

2.2 데이터셋 설명

본 연구에서는 Kaggle에 공개된 Rice Leaf Diseases Dataset을 사용하였다. EDA를 해보았을 때, 이 데이터셋은 벼 잎에 발생하는 주요 질병에 대한 이미지로 구성되어 있으며, [Figure1]에서처럼 총 3 개의 클래스로 구분된다:

- Bacterialblight (세균성 벼잎마름병)
- Brownspot (갈반병)
- Leafsmut (잎무늬병)



[Figure1. 클래스별 대표 이미지 10 장]

전체 데이터셋은 총 4,684 장의 RGB 이미지로 구성되어 있으며, 모든 이미지는 모델 입력에 적합하도록 224x224 크기로 resize 하여 사용하였다. 각 이미지는 .jpg 포맷으로 저장되어 있으며, 라벨은 폴더 구조를 통해 자동으로 지정된다.

데이터는 학습, 검증, 테스트 세트로 분할되었으며, 총 3,431 장의 이미지로 구성되어 있다. 각 분할의 이미지 수는 다음과 같다:

- 학습(Train): 2,058 장 (약 60%)
- 검증(Validation): 686 장 (약 20%)
- 테스트(Test): 687 장 (약 20%)

이미지 클래스 간의 분포는 비교적 균등하게 유지되어 있어, 데이터 불균형 문제 없이 학습이 가능하였다.

모델의 성능 향상 및 과적합 방지를 위해 학습 데이터에는 다양한 데이터 증강 기법을 적용하였다. 검증 및 테스트 데이터는 원본 특성을 유지하도록 최소한의 전처리만 수행하였다.

- 학습 데이터 변환
 - Resize(224x224)
 - RandomHorizontalFlip() (좌우 반전)
 - RandomVerticalFlip() (상하 반전)
 - RandomRotation(20 도)
 - ColorJitter() (밝기, 대비, 채도, 색조 조정)
 - RandomAffine() (위치 및 회전 이동)
 - ToTensor(), Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
- 검증 / 테스트 데이터 변환

- o Resize(224x224)
- o ToTensor(), Normalize(...)

```

7 # 데이터 경로
8 data_dir = "rice-plant-diseases/rice leaf diseases dataset"
9
10 # Train 데이터 변환 (Data Augmentation 적용)
11 train_transform = transforms.Compose([
12     transforms.Resize((224, 224)),
13     transforms.RandomHorizontalFlip(), # 좌우 반전
14     transforms.RandomVerticalFlip(), # 상하 반전 추가
15     transforms.RandomRotation(20), # 랜덤 회전
16     transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1), # 색상 변형 추가
17     transforms.RandomAffine(degrees=30, translate=(0.1, 0.1)), # 이미지 이동 추가
18     transforms.ToTensor(),
19     transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
20 ])
21
22 # Validation & Test 데이터 변환 (원본 유지)
23 val_test_transform = transforms.Compose([
24     transforms.Resize((224, 224)),
25     transforms.ToTensor(),
26     transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
27 ])

```

[Figure 2. Google Colab에서 수행된 이미지 전처리 코드 일부]

이 데이터셋은 농업 분야에서 벼의 질병을 자동으로 분류하는 인공지능 시스템 개발에 활용될 수 있으며, 고해상도 이미지와 균형 잡힌 클래스 분포 덕분에 다양한 딥러닝 모델 실험에도 적합하다. 특히 CNN 기반의 분류 모델 실험, 전이 학습, 앙상블 모델링 등에서 유용하게 사용될 수 있다.

3. 모델 학습

3.1 학습 모델 구조

본 프로젝트에서는 이미지 분류 정확도 향상과 다양한 모델 성능 비교를 위해 총 네 가지 모델 구조를 실험하였다: Custom CNN, ResNet50, VGG16, MobileNetV2. 이들 중 ResNet50, VGG16, MobileNetV2는 PyTorch 의 torchvision.models에 내장된 사전 훈련(pretrained) 모델을 기반으로 하며, 각 모델에는 Feature Extraction 을 적용하고 Custom Fully Connected (FC) Layer 를 재구성하여 도메인 특화 분류기로 수정하였다.

사전 훈련 모델을 선택한 이유는 제한된 데이터셋 상황에서 학습 속도와 정확도 모두를 확보할 수 있기 때문이다. 특히 ResNet50은 깊은 네트워크에서도 성능 저하 없이 정보를 유지할 수 있는 skip connection 구조를 가지며, MobileNetV2는 경량 모델로서 빠른 추론 속도와 효율성을 제공한다. VGG16은 비교적 단순한 구조로 실험 초기 안정적인 기준선 모델로 활용되었다.

모든 모델은 Global Average Pooling 레이어 이후, Linear → BatchNorm → ReLU → Dropout → Linear 형태의 FC 구조를 공통적으로 사용하였으며, Dropout(0.5)은 과적합 방지를 위해, BatchNorm은 학습 안정화와 수렴 속도 향상을 위해 적용되었다.

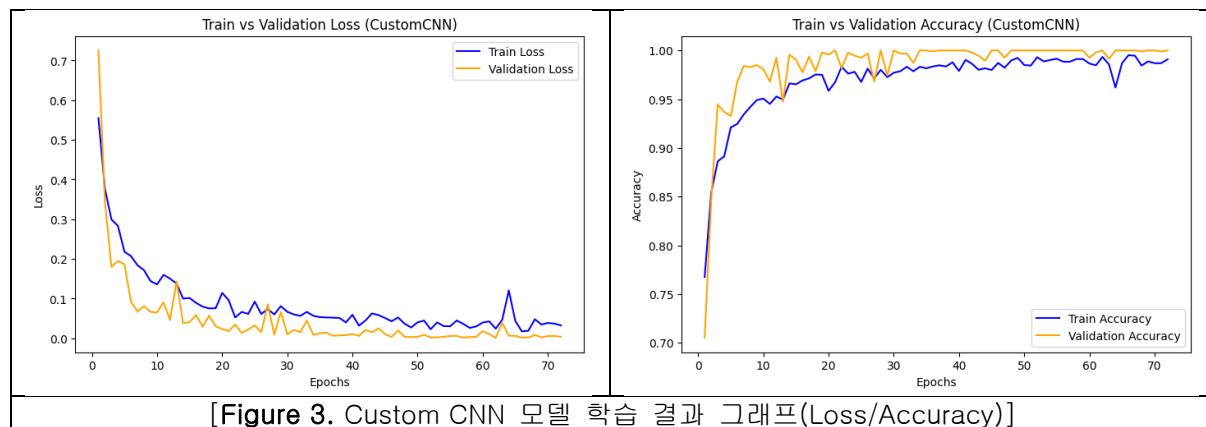
3.2 실험방법

학습 시 Early Stopping 을 도입하여 과적합을 방지하였으며, 최대 100 Epoch, Patience 는 10 으로 설정하였다. Loss 함수는 CrossEntropyLoss, Optimizer 는 Adam(learning rate=0.001, weight_decay=1e-4)을 공통적으로 사용하였다.

모델별 특성에 따라 Feature Extraction 을 적용하거나 Dropout, BatchNorm 등을 구성에 포함하였으며, 특히 VGG16 과 ResNet50 은 처음에는 정확도가 낮게 나왔으나 Fine-tuning 을 통해 성능을 크게 개선하였다. 학습 중 loss 나 accuracy 가 불안정하게 튕는 현상이 있는 경우에는 네트워크 구조나 learning rate 를 조정하고, 일부 layer 의 requires_grad 설정을 조절하는 방식으로 안정성을 확보하였다.

3.2.1 Custom CNN

Custom CNN 은 가장 단순한 구조의 모델로, 3 개의 Conv-BN-ReLU 블록과 Global Average Pooling, 그리고 두 개의 FC Layer 로 구성되어 있다. 각 계층 뒤에는 BatchNorm 과 Dropout(0.5)을 적용하여 과적합 방지를 유도하였다. 일반적인 CNN 모델의 흐름을 따르되, ResNet 등 고도화된 구조에 비해 파라미터 수가 적어 빠른 학습이 가능하며, baseline 모델로서 역할을 하였다.



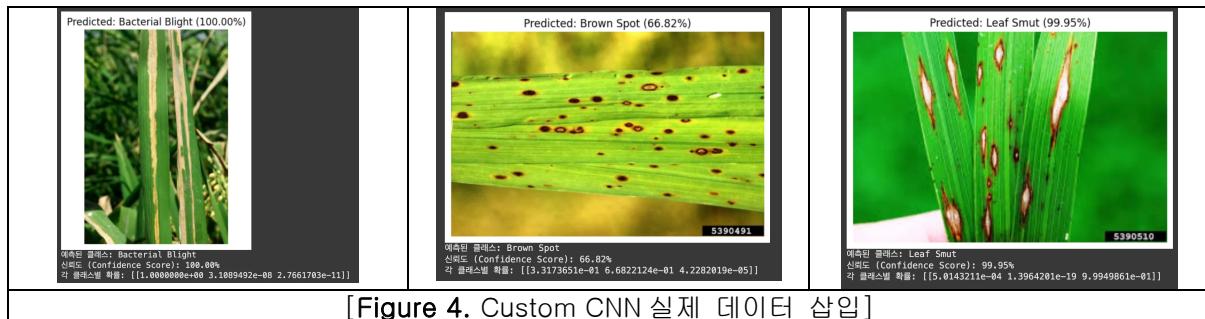
Test Accuracy: 1.0000

Test Loss: 0.0011

Loss 그래프는 전반적으로 train 과 val 의 차이가 크지 않고 동반 하락하므로 과적합 징후는 거의 없으며 모델이 안정적으로 학습되었다. 60epoch 부근에서 train loss 가 일시적으로 상승하는 구근이 보이나, 이후 빠르게 회복되어 전체적인 안정성에는 큰 영향을 주지 않는다.

Accuracy 그래프는 train 과 validation 이 거의 유사한 수준으로 유지되고 있으며, 이 또한 일반화 성능이 우수한 학습 결과임을 나타낸다. 20epoch 이후에 val accuracy 는 거의 1.0 수준에 도달하여 거의 완벽한 예측 성능을 보여준다.

종합적으로 Custom CNN 은 비교적 단순한 구조임에도 불구하고, 적절한 적절한 BatchNorm, Dropout, 그리고 적은 파라미터 수 덕분에 높은 성능과 빠른 수렴을 보여주었다.



[Figure 4. Custom CNN 실제 데이터 삽입]

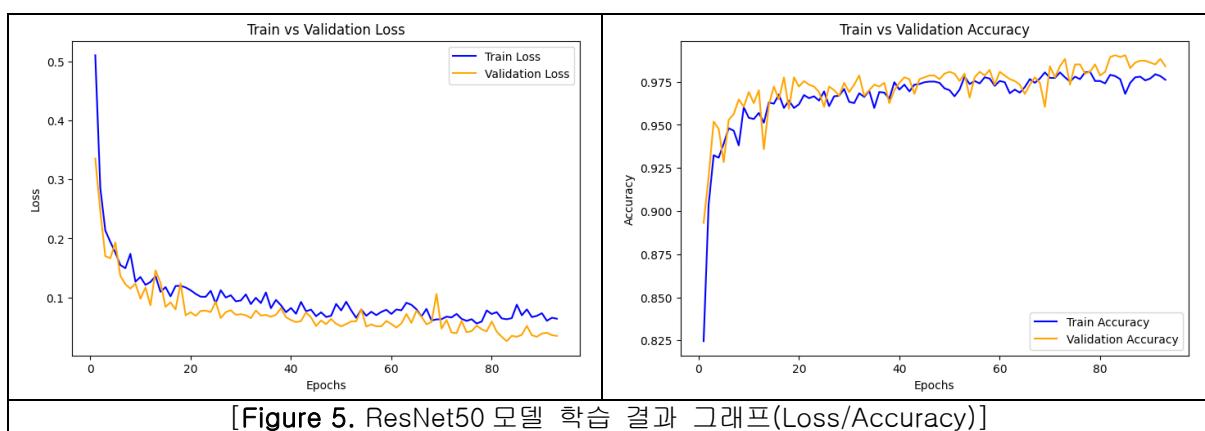
테스트 데이터 외에, 실제 이미지를 넣어서 모델이 이미지에 대해 어떻게 예측을 수행하는지 확인해보았다. 세 클래스 모두 정확히 예측되었으며, 특히 Bacterial Blight 와 Leaf Smut 의 경우 신뢰도가 매우 높았다. Brown Spot 은 유일하게 신뢰도가 낮았지만, 실제 클래스와 예측 클래스는 일치하기에 모델의 판단 근거가 명확하다는 점에서 긍정적으로 평가한다. 모델이 단순히 외형이 비슷한 이미지를 구분하는 것이 아니라, 질병별 특징을 학습하여 분류하고 있다는 것을 보여주는 사례이다.

3.2.2 ResNet50

사전 학습된 ResNet50 모델을 기반으로 Feature Extraction 을 적용하고, Fully Connected Layer 를 재구성한 Custom 모델(CustomResNet50_FE)을 학습에 사용하였다.

ResNet50 의 기본 구조는 깊은 계층과 skip connection 을 활용하여, 학습이 깊어질수록 발생할 수 있는 gradient vanishing 문제를 해결하는데 강점을 가진다. 본 실험에서는 ResNet50 의 기존 feature extractor 를 고정(requires_grad=False)한 뒤, 최종 FC 레이어를 $2048 \rightarrow 64 \rightarrow 3$ 으로 재설계하고 BatchNorm, ReLU, Dropout(0.5) 을 적용해 일반화 성능을 확보하고자 했다.

학습에는 CrossEntropyLoss 와 Adam Optimizer 를 사용하였으며, learning rate 는 0.001, weight decay 는 $1e-4$ 로 설정하였다. 학습은 최대 100 Epoch 까지 수행하되, 10 번 연속으로 validation loss 개선이 없을 경우 조기 종료되는 Early Stopping 기법을 도입하였다. 학습 중 검증 정확도 및 loss 가 불안정하게 흔들리는 현상을 방지하기 위해 일부 layer 의 파라미터 고정과 learning rate 조정을 병행하였다.



[Figure 5. ResNet50 모델 학습 결과 그래프(Loss/Accuracy)]

Test Accuracy: 0.9851

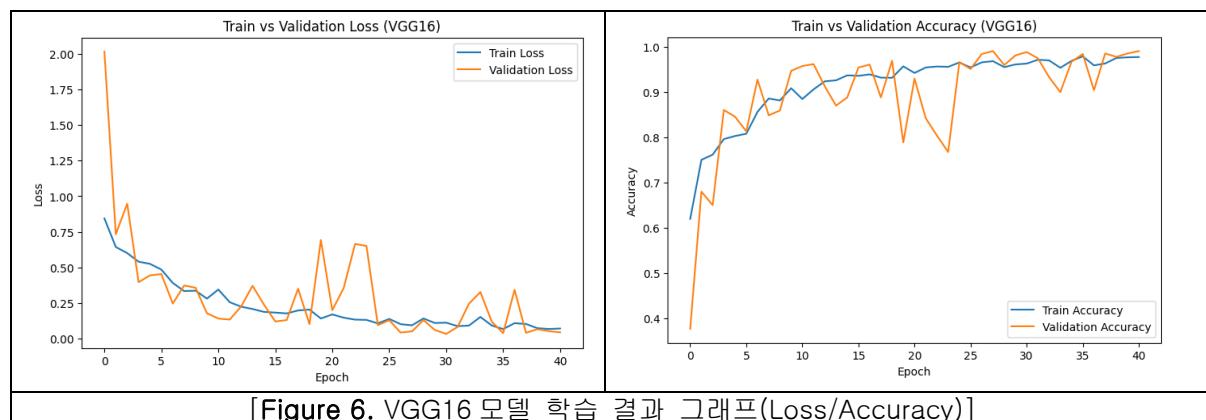
Test Loss: 0.0396

Loss 그래프를 보면 학습 초반에는 빠르게 감소하며 이후 비교적 완만하게 안정화된다. 전반적으로 학습 손실과 검증 손실 간의 간격이 크지 않아 과적합 현상은 적다.

Accuracy 그래프는 약 20 Epoch 이후부터는 Train과 Validation Accuracy 가 모두 95~98% 수준에서 유지되며, 학습이 지나치게 train 데이터에 치우치지 않도록 조절됨을 알 수 있다.

3.2.3 VGG16

VGG16 또한 Feature Extractor 를 중심으로 구성되었으며, 기본 feature layer 이후 AdaptiveAvgPool 과 FC Layer($512 \rightarrow 64 \rightarrow 3$)를 도입하였다. 실험 초반에는 성능이 95%가 나왔으나 이후 실험을 진행하며, Feature Layer 일부에 Fine-tuning 을 적용하고 Dropout, BatchNorm 등의 정규화 기법을 활용하여 정확도를 99%까지 올릴 수 있었다. 다른 모델들과 비교했을 때, 구조적으로 단순하지만 fine-tuning 설정에 따라 성능이 달라질 수 있다는 점에서 실험적으로 의미 있는 결과를 보였다.



Test Accuracy: 0.9893

Test Loss: 0.0356

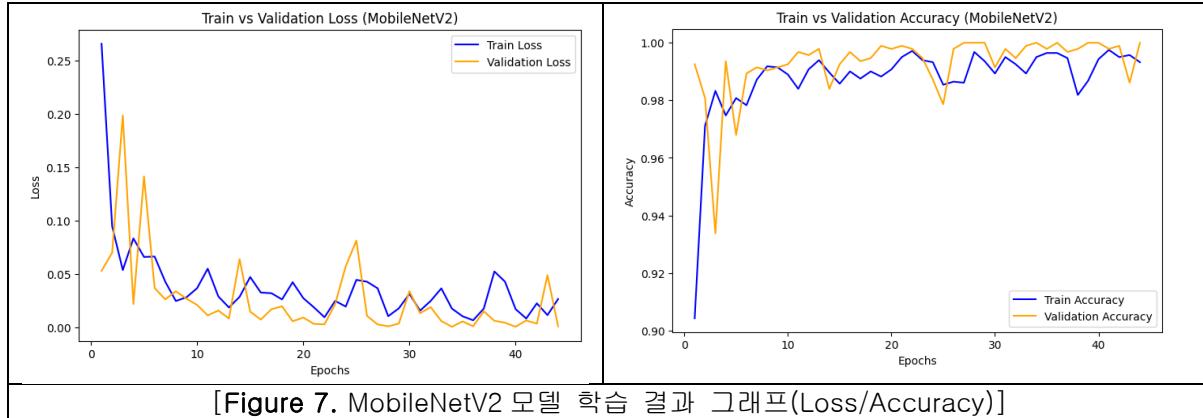
학습 초기에는 loss 와 accuracy 가 다소 불안정하게 변동하였으나, epoch 0이 진행되면서 점차 안정화되었으며 fine-tuning 을 통해 성능이 큰 폭으로 개선되었다. 특히, validation accuracy 는 후반부에 95% 이상을 지속적으로 유지하며 모델이 과적합 없이 일반화된 학습을 수행하였다.

[Figure 6]의 학습 그래프를 통해 확인할 수 있듯, train/validation loss 는 전체적으로 안정적인 하강 곡선을 그리며, accuracy 또한 train/validation 간 유사한 추이를 보여주어 과적합 없이 훈련이 진행되었음을 확인할 수 있다.

3.2.4 MobileNetV2

MobileNetV2 모델은 경량화된 구조와 효율적인 성능으로 잘 알려진 사전 학습(pretrained) 모델로, 본 프로젝트에서는 feature extractor 를 고정한 후 classification head 만 새롭게 정의해 fine-tuning 을 수행하였다. Feature Extraction 단계에서 기존 MobileNetV2 의 features 부분은 그대로 유지하면서, 최종 분류를 위한 FC 구조는 $1280 \rightarrow 256 \rightarrow 3$ 으로 구성하였고, 중간에 BatchNorm 과 ReLU 활성화 함수를 삽입하였다. 일부 실험에서는 Dropout 을 제거하여 더욱 경량화된 모델로도 실험하였다.

학습에는 CrossEntropyLoss 와 Adam Optimizer($\text{lr}=0.001$, $\text{weight_decay}=1e-4$)를 사용하였으며, 최대 100 epoch 중 Early Stopping 기준으로 patience는 10 으로 설정하였다. 학습 초기에 95% 수준의 정확도를 보였으나, fine-tuning 을 통해 99% 이상까지 정확도를 끌어올릴 수 있었다. 이는 사전 학습된 feature map 위에서 Custom FC 레이어를 효과적으로 학습시킨 결과로 볼 수 있다.

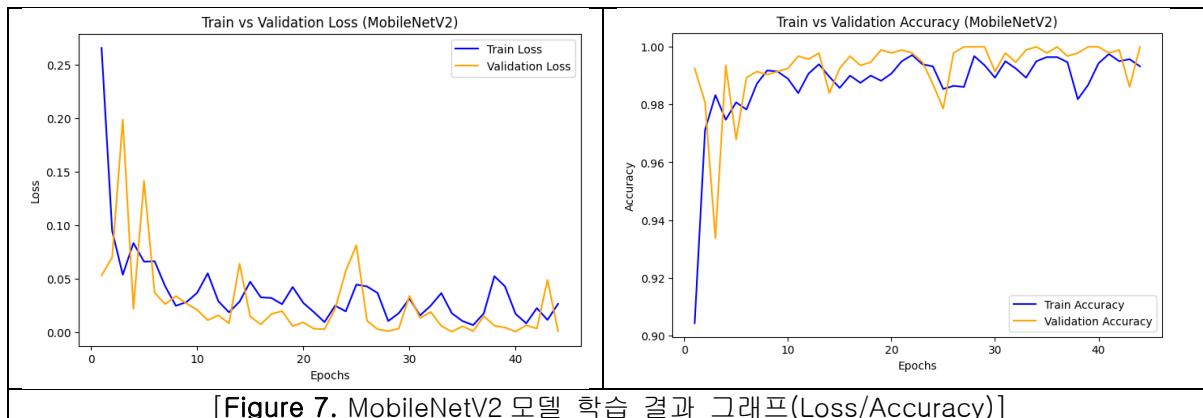


Test Accuracy: 0.9989

Test Loss: 0.0018

[Figure 7]을 통해 확인할 수 있듯이, loss 와 accuracy 의 추이는 전반적으로 안정적인 형태를 보였으며, 특히 validation accuracy 는 빠르게 0.99 수준까지 수렴하였다. 학습 및 검증 간 큰 성능 차이가 없고, loss 그래프의 진동 폭도 크지 않아 일반화 성능이 우수함을 확인할 수 있었다.

전체 모델에 대한 비교는 4.2.2 에서 대시보드를 통한 시각화된 표와 함께 진행한다.

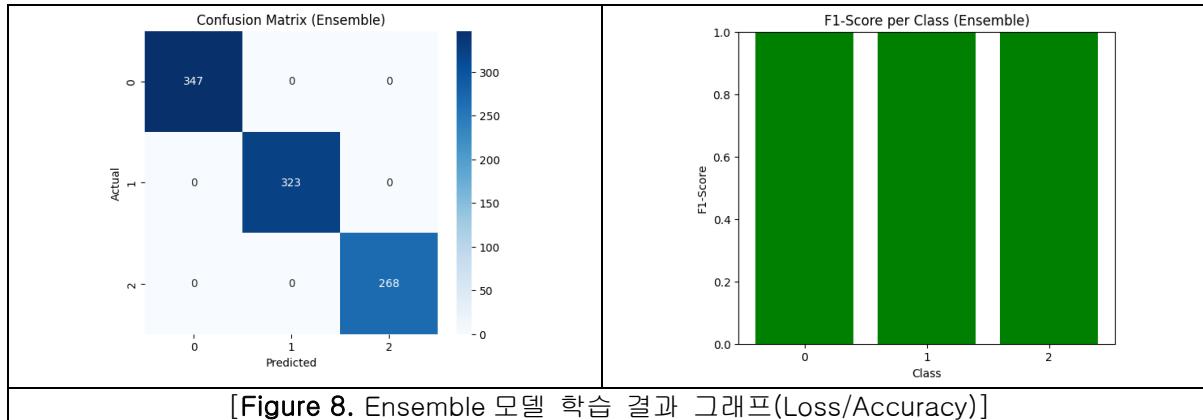


3.2.5 Ensemble

모델을 실행할 때마다 정확도나 결과가 다소 달라지는 현상을 경험했다. 특히 MobileNetV2 와 VGG16 모델은 특정 학습에서 100%에 가까운 높은 정확도를 보였지만, 이는 과적합일 가능성도

배제할 수 없었다. 이러한 불안정성을 완화하고 일반화 성능을 향상시키기 위해 양상을 기법을 도입하였다.

본 프로젝트에서는 Custom ResNet50, VGG16, MobileNetV2 세 가지 모델의 예측 결과를 평균화하는 방식으로 소프트 보팅(soft voting) 양상을 수행하였다. 각 모델의 출력 logits 을 평균낸 후 argmax 를 취해 최종 클래스를 결정하였으며, test 데이터셋에 대해 다음과 같은 결과를 얻었다



양상을 적용함으로써 모든 클래스에 대해 완벽한 분류 결과를 달성할 수 있었으며, 개별 모델에서는 나타나던 미세한 오차가 효과적으로 보완되었다. 특히 단일 모델의 과적합 가능성이나 데이터 편향 문제를 최소화할 수 있었고, 예측 안정성이 대폭 향상되었다는 점에서 실험적으로 큰 의미를 가진다.

4. MongoDB 및 시각화 결과분석

4.1 데이터베이스 설계

본 프로젝트에서는 각 모델의 학습 결과를 체계적으로 기록하고 실시간 시각화하기 위해 MongoDB 기반의 데이터베이스 설계를 적용하였다. 학습 도중 발생하는 로그를 수집하고, Streamlit 대시보드에서 이를 실시간으로 시각화할 수 있도록 하기 위해 세 개의 주요 컬렉션(trainings, epochs, status)을 구성하였다.

- trainings: 각 학습(run_id)에 대한 전체 요약 정보를 저장 (모델 이름, 정확도, 손실, 소요 시간 등)
- epochs: 각 epoch 별 loss, accuracy, 학습 시간 기록
- status: 학습 진행 상태(status), 시작/종료 시간, 완료 여부 등 실시간 추적용

MongoDB 는 NoSQL 방식의 유연한 데이터 스키마를 제공하므로, 모델마다 구조가 다르거나 로그 항목이 조금씩 다른 경우에도 제약 없이 기록이 가능하다는 장점이 있다. 예를 들어, VGG16 모델에는 dropout, learning_rate 정보가 명시되었지만, CustomCNN이나 MobileNetV2 에서는 일부 항목이 생략되어도 문제가 발생하지 않는다.

MongoDB 를 선택한 이유는 다음과 같다:

- RDS 계열(MySQL, PostgreSQL 등)은 기존 프로젝트에서 자주 사용해보았기 때문에, 이번에는 새로운 방식의 데이터베이스를 경험해보고자 NoSQL 기반 MongoDB를 선택하였다.
- SQLite 또한 가볍게 테스트한 적은 있었지만, 이번 프로젝트처럼 동적 구조의 데이터 저장 및 빠른 시각화 연동이 필요한 상황에는 다소 제한적이었다.

따라서 본 프로젝트에서는 다양한 모델 실험 데이터를 효율적으로 저장하고, 이를 실시간으로 조회·분석할 수 있는 유연한 환경 구성을 위해 MongoDB를 도입하게 되었다. 이 구조는 추후 모델 비교, 하이퍼파라미터 튜닝 분석, 실시간 모니터링 등에서도 확장성이 높다.

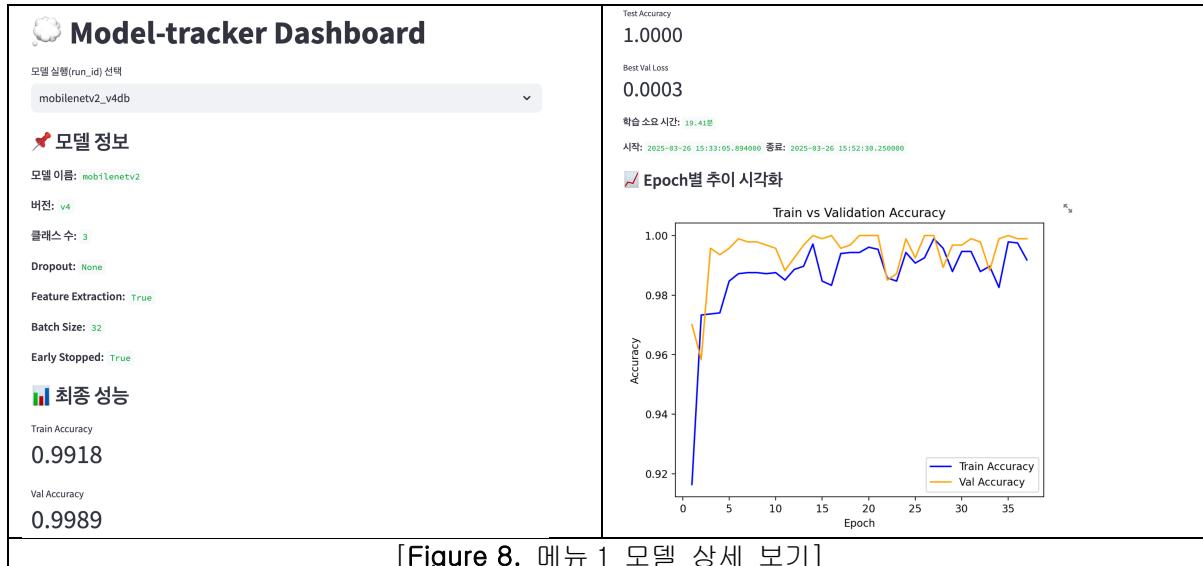
4.2 시각화 및 결과분석

배포 url: [model tracker](#)

추가 과제인 mongoDB를 활용하기 위해 본 프로젝트에서는 모델 학습 과정을 실시간으로 추적하고 다양한 지표를 시각화하기 위해 Streamlit 기반의 웹 대시보드를 설계하였다. 대시보드는 모델별 성능 비교 및 하이퍼파라미터 분석을 한눈에 확인할 수 있도록 구성되어 있으며, 실시간 모니터링 기능까지 지원한다.

아래에서 각각의 메뉴 4 개에 대해 설명한다.

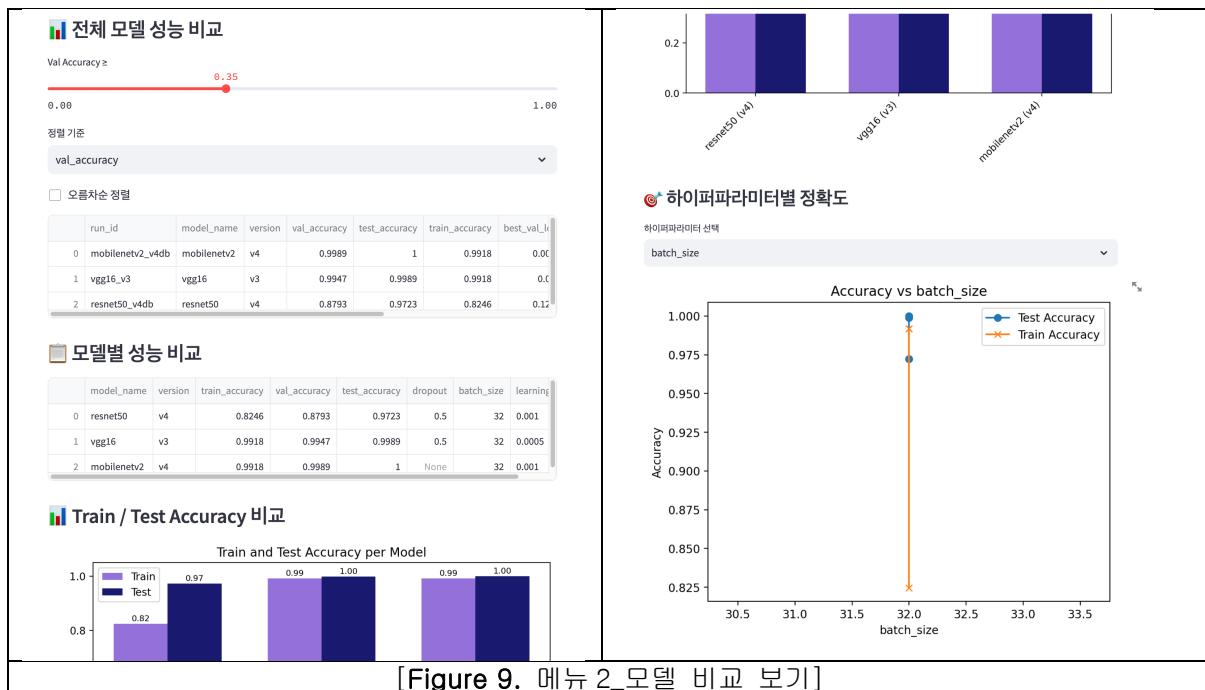
4.2.1 모델 상세 보기



해당 메뉴는 선택한 학습(run_id)에 대한 상세 정보를 확인할 수 있도록 구성된 화면이다. 사용자 인터페이스 상단에서는 드롭다운 메뉴를 통해 특정 모델 실험(run_id)을 선택할 수 있으며, 선택 즉시 해당 모델의 모든 정보가 실시간으로 조회된다.

모델 이름, 버전, 클래스 수, 적용된 Feature Extraction 여부, Dropout 유무, Batch Size, Early Stopping 여부 등을 확인할 수 있다. 이 정보는 학습 당시 설정된 하이퍼파라미터와 네트워크 구조를 이해하는 데 도움을 준다. 또한 학습 완료 시점의 주요 성능 지표가 표시된다. Train/Validation/Test Accuracy 및 Best Validation Loss가 함께 출력되며, 학습 소요 시간, 시작~종료 시간도 함께 제공되어 실험 관리에 용이하도록 하였다.

4.2.2 모델 비교 보기



모델 비교 보기 메뉴는 학습된 다양한 모델들의 성능을 직관적으로 비교·분석할 수 있도록 구성되었다. 사용자 선택에 따라 정렬 기준 및 필터 조건을 다르게 설정할 수 있으며, 핵심 성능 지표를 한눈에 확인할 수 있다.

	model_name	version	train_accuracy	val_accuracy	test_accuracy	dropout	batch_size	learning_rate
0	resnet50	v4	0.8246	0.8793	0.9723	0.5	32	0.001
1	vgg16	v3	0.9918	0.9947	0.9989	0.5	32	0.0005
2	mobilenetv2	v4	0.9918	0.9989	1	None	32	0.001

표는 ResNet50(v4), VGG16(v3), MobileNetV2(v4) 세 모델의 성능 지표와 학습에 사용된 주요 하이퍼파라미터 설정을 정리한 것이다. 이를 통해 모델 간의 일반화 성능, 안정성, 과적합 여부 등을 비교해볼 수 있다.

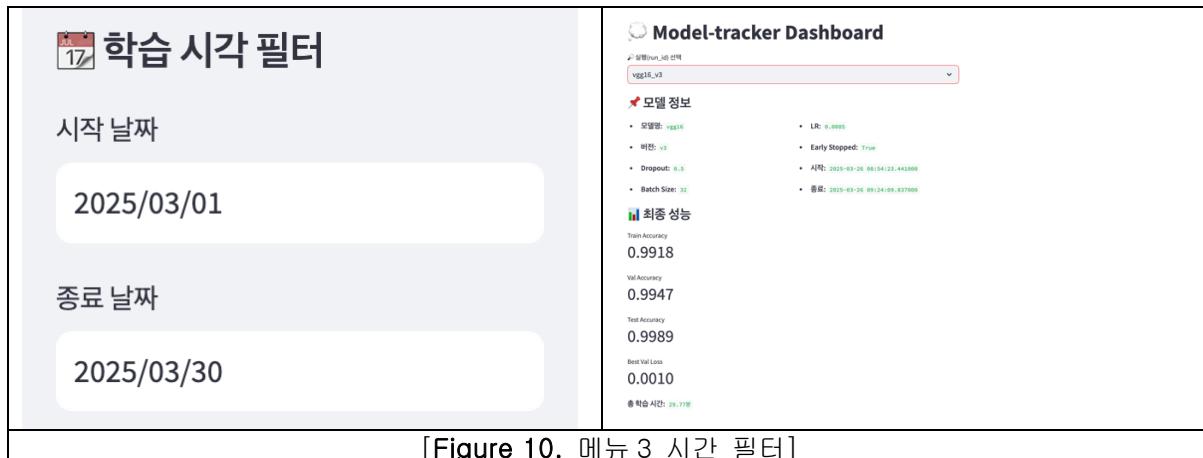
ResNet50은 Train Accuracy 가 다른 모델에 비해 낮고, Val/Test Accuracy에서도 상대적으로 성능이 떨어져 보이지만 일반적으로 보면 매우 높은 정확도이다. 이는 fine-tuning 이 덜 적용되었거나, 깊은 네트워크 구조에 비해 데이터셋 규모가 작았던 점 등이 원인일 수 있다.

VGG16과 MobileNetV2는 모두 학습 정확도와 검증 정확도가 99% 이상이며, MobileNetV2는 100%의 테스트 정확도를 기록해 가장 뛰어난 일반화 성능을 보여준다.

세 모델 모두 Early Stopping 을 적용했으며, validation loss 기준으로 조기 종료되어 과적합을 방지하였다. Dropout 은 ResNet50과 VGG16에 적용되어 과적합 방지에 기여했으나, MobileNetV2는 dropout 없이도 안정적인 학습 성능을 기록했다. Batch Size 는 세 모델 모두 동일하게 32로 설정되었으며, 비교를 위한 통일된 실험 조건을 반영한 것이다. Learning Rate 는 VGG16에서만 더 낮은 값인 0.0005가 적용되었는데, 이는 상대적으로 민감한 fine-tuning

구조에 맞춰 안정적인 수렴을 유도하기 위함이다. 실제로 lr을 다른 모델들과 똑같이 실험했을 때 loss 그래프에서 튀는 현상이 있었다.

4.2.3 시간 필터



[Figure 10. 메뉴 3_시간 필터]

시간 필터는 사용자가 원하는 학습 시작일과 종료일을 기준으로 모델 학습 결과를 필터링하여 확인할 수 있도록 구성한 기능이다. 주로 다수의 모델이 비동기적으로 학습되거나, 특정 기간 내에 진행된 실험 결과만 따로 분석할 필요가 있을 때 유용하다.

시간 기반으로 실험 결과를 관리하면 **장기간 프로젝트에서의 실험 관리 및 버전 추적**이 수월해진다. 예를 들어 "3월에 진행한 모델만 보고 싶다" 또는 "최근 일주일 간 학습한 모델 성능만 비교해보자" 같은 요구를 쉽게 해결할 수 있다.

4.2.4 학습 상태 실시간 모니터링



[Figure 11. 메뉴 4_학습 상태 실시간 모니터링]

이 메뉴는 모델 학습이 진행되는 **도중의 상태를 실시간으로 확인**할 수 있도록 설계된 기능이다. 학습이 오래 걸리는 환경이나 여러 모델을 동시에 실험하는 경우, 모니터링 기능은 효율적인 운영에 큰 도움이 된다. 실제로 colab pro+를 사용하면서도 모델이 돌아가는 것이 멈추는 경우가 몇몇 있었는데 휴대폰으로 간편하게 확인 가능해서 좋은 것 같다.

실시간 상태 확인은 실험 효율성을 높이고, 중단 여부 판단이나 리소스 낭비를 줄이는 데 핵심 역할을 한다. 특히 클라우드 환경이나 리소스 공유 환경에서 유용하며, 향후 Slack/Discord 알림 등과 연동하면 확장성 있는 실험 관리 체계도 가능하다.

4.2.4.1 실시간 에폭 반영 문제 해결

초기에는 학습 중 MongoDB의 epochs 컬렉션에 insert_one()으로 에폭 정보를 저장하고 있었으나, Streamlit 화면에서는 이 변경이 즉각 반영되지 않는 문제가 발생했다.

원인: Streamlit은 기본적으로 페이지 전체를 새로 고치지 않는 구조이기 때문에, DB가 갱신되어도 화면에는 반영되지 않는 자연 현상이 발생.

해결 방식:

- Streamlit 내 st.experimental_rerun() 또는 st.session_state 재할당을 통해 주기적으로 새로고침 시도
- 일정 간격으로 데이터를 polling하여 변경 여부를 감지 후 업데이트 반영
- 한 에폭당 DB에 insert_one() → update_one() 방식으로 이원화하여, 학습 초기에 입력 후 각 지표(val_loss, val_acc, epoch_time_sec 등)는 update_one()으로 갱신하는 구조로 변경

5. 결론 및 향후 계획

본 프로젝트에서는 쌀 잎 질병 이미지 분류 문제를 해결하기 위해 다양한 사전 학습 기반 모델(Custom CNN, ResNet50, VGG16, MobileNetV2)을 적용하고, 각 모델의 학습 과정을 체계적으로 기록·시각화할 수 있는 Streamlit 기반 실시간 대시보드 시스템을 구축하였다. 학습 로그는 MongoDB를 통해 저장되며, 실시간으로 모델의 정확도, 손실, 시간 등을 추적할 수 있게 하였다.

모델별로 Dropout, BatchNorm, Feature Extraction 등의 설정을 달리하여 실험을 진행하였고, Fine-tuning 기법을 통해 초기 정확도가 낮았던 모델들도 성능을 성공적으로 개선하였다. 특히 MobileNetV2는 가볍고 빠른 구조임에도 불구하고 높은 정확도를 보여주었고, ResNet50과 VGG16 또한 fine-tuning 이후 테스트 정확도 97% 이상을 기록하였다.

또한, 실시간 에폭별 결과 반영을 위해 insert_one과 update_one을 조합한 방식으로 MongoDB 갱신 속도를 최적화하였으며, streamlit.experimental_rerun 사용 시 발생하는 오류도 디버깅하여 안정적인 대시보드 갱신 구조를 완성하였다.

이외에도 모델 탐색 효율을 높이기 위해 Grid Search 기반의 하이퍼파라미터 조정 도구를 도입하고자 한다. 또한 다양한 실험 결과를 비교 분석할 수 있는 Model Comparison Platform (MCP) 기능을 추가해, 각 모델의 성능, 학습 환경, 파라미터 조건 등을 직관적으로 비교할 수 있도록 구현하고자 한다.

이번 프로젝트는 단순한 모델 개발을 넘어서, 데이터 실험 관리 자동화, 성능 시각화, 그리고 모델 간 비교와 분석의 편의성을 통합적으로 고려한 결과물이다. 향후에는 이 실험 관리 체계를 다양한 도메인에 적용하고, 자동화 및 추천 시스템을 접목해 연구와 실무 모두에 활용 가능한 AI 실험 통합 플랫폼으로 고도화해 나갈 계획이다.

6. 참고문헌

- Jay7080dev. *Rice Plant Diseases Dataset*, Kaggle.
<https://www.kaggle.com/datasets/jay7080dev/rice-plant-diseases-dataset>
- 문상현. (머신러닝) 양상을 학습이란?, DAWN@CAU (중앙대학교 AI 연구 커뮤니티), Medium, 2023.
<https://medium.com/dawn-cau/머신러닝-양상을-학습-이란-cf1fcb97f9d0>
- PyTorch Tutorials. *Saving and Loading Models* (official documentation).
https://pytorch.org/tutorials/beginner/saving_loading_models.html
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
(딥러닝 기본 개념 및 모델 학습 방식 참고)
- Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media.
(하이퍼파라미터 튜닝, 양상을 학습 관련 기법 참고)