CNN 및 Transformer 기반 전이 학습을 활용한 꽃 이미지 분류 성능 비교 연구

작성자 : ethan.kim(김용범)/인공지능

목차

1. 서론

1.1 연구 배경 및 이미지 분류의 개요

이미지 분류(Image Classification)는 컴퓨터 비전(Computer Vision)의 대표적인 과제로, 주어진 이미지를 특정한 클래스로 정확히 구별하는 기술이다. 최근 몇 년간 딥러닝의 발전과 함께 CNN(Convolutional Neural Network)이 등장하면서 이미지 분류 분야는 비약적인 발전을 이루었다. CNN은 이미지를 공간적으로 처리하며, 레이어가 깊어질수록 이미지의 다양한 특성과 세부 정보를 효과적으로 학습하여 높은 성능을 달성할 수 있다. 이러한 CNN의 대표적인 예시로 VGGNet, ResNet, EfficientNet 등이 있으며, 특히 ResNet은 기존 CNN이 갖던 심층 구조에서의 성능 저하 문제를 Skip Connection으로 해결하면서 큰 주목을 받았다.

최근 이미지 분류의 새로운 흐름으로는 Transformer 구조를 활용한 Vision Transformer(ViT), Swin Transformer 등의 모델이 등장하였다. Transformer는 본래 자연어 처리(NLP) 분야에서 뛰어난 성과를 보였으며, 이를 이미지 처리에 도입하여 CNN 대비 더욱 뛰어난 성능과 일반화 능력을 보이고 있다. Transformer 계열의 모델들은 Self-Attention 메커니즘을 활용하여 이미지의 전역적(global) 정보를 효과적으로 학습할 수 있어, 데이터셋이 크고 다양한 경우에 특히 강력한 성능을 보이고 있다.

1.2 전이 학습(Transfer Learning)의 중요성 및 활용

이미지 분류 문제를 다룰 때 데이터셋의 규모가 작거나 라벨링된 데이터가 부족한 경우가 많다. 이러한 환경에서는 모델이 충분히 학습되지 못하여 과적합(overfitting)이 발생하거나 성능 저하가 나타날 수 있다. 이때 효과적으로 사용될 수 있는 전략이 전이 학습(Transfer Learning)이다. 전이 학습은 기존의 대규모 이미지 데이터셋(예: ImageNet)을 통해 미리 학습된 모델의 가중치를 사용하여, 상대적으로 작은 데이터셋에서도 높은 분류 성능을 얻을 수 있도록 도와준다.

최근 연구에서는 이러한 전이 학습을 CNN뿐만 아니라 Transformer 계열 모델에서도 적극적으로 활용하고 있다. 특히 ViT(Vision Transformer), Swin Transformer, ConvNeXt와 같은 최신 모델들도 대규모 데이터셋을 통해 미리 훈련된 가중치를 활용해 작은 데이터셋에서도 높은 성능을 달성할 수 있음을 증명하고 있다.

1.3 본 연구의 목적 및 연구 문제 정의

본 연구의 목적은 Kaggle에서 제공하는 꽃 이미지 데이터셋을 활용하여 최신 CNN 기반 모델들과 Transformer 기반 모델들을 전이 학습을 통해 비교하고, 각 모델의 성능과 특징을 분석하는 데 있다. 구체적으로는 CNN 계열의 대표적인 모델인 VGG16, ResNet50, EfficientNetV2와 Transformer 계열의 최신 모델인 ViT, Swin Transformer, ConvNeXt를 선택하여 비교 실험을 진행한다. 본 연구에서는 전이 학습을 통한 하이퍼파라미터 튜닝(Fine-tuning)을 수행하고, 각 모델의 성능 차이와 과적합 방지를 위한 방법을 비교하여 실무적으로도 가장 적합한 모델을 선정하고자 한다.

1.4 연구의 중요성 및 기대효과

본 연구를 통해 다양한 이미지 분류 모델 간 성능을 명확히 비교하고 분석함으로써, 이미지 분류 분야에서 CNN과 Transformer 모델 간의 강점과 한계를 이해할 수 있을 것으로 기대한다. 또한, 작은 데이터셋 환경에서 전이 학습을 통해 모델 성능을 극대화하는 최적의 방법을 탐색하여, 실제 연구나 산업 현장에서 효율적인 모델 선택의 가이드라인을 제공할 수 있다.

이러한 연구 결과는 식물 분류, 생태계 관리, 농업 자동화, 생물 다양성 보존 등 실생활과 산업 분야에서 실질적인 도움을 줄 수 있는 지침이 될 것으로 기대한다.

2. 데이터셋 설명

2.1 데이터셋 출처

본 연구에 사용된 데이터셋은 Kaggle 플랫폼에서 공개된 Flowers Dataset이다. 해당 데이터셋은 이미지 분류 연구에서 흔히 사용되는 대표적인 벤치마크 중 하나이며, 꽃의 종류를 분류하는 과제를 다룰 때 널리 이용된다.

- 데이터 출처: Kaggle Flowers Dataset
- 데이터 공개자: imsparsh (Kaggle 사용자)

2.2 데이터셋 구조 및 특성

해당 데이터셋은 총 **3670**개의 이미지로 구성되어 있으며, 학습 데이터 **2746,** 테스트 데이터 **924**개로 구분되어 있다. 각 이미지는 다음과 같은 5가지 클래스에 속해 있다.

클래스 (꽃 종류)	학습용 이미지 수 (개)	특징
Daisy (데이지)	501	흰색 꽃잎, 중심부 노란색
Dandelion (민들레)	646	노란색, 꽃잎이 얇고 촘촘함
Rose (장미)	497	빨강, 핑크 등 다양한 색상과 복잡한 꽃잎 구조
Sunflower (해바라기)	495	노란색, 중심부가 크고 뚜렷함
Tulip (튤립)	607	여러 색상, 둥근 꽃잎 구조
총합	2746	

각 클래스는 약 500~600장 내외로 구성되어 클래스 간 이미지 수는 다소 균형적이다. 다만 이는 분류 모델의 학습에는 일반적으로는 적절할 것으로 보이나 일반화 성능 평가에는 다소 아쉬운 규모로 판단된다. 원본 이미지의 해상도는 일반적으로 320×240 픽셀이며, 모든 이미지 데이터는 JPEG 형식을 따른다.

2.3 데이터 전처리 및 데이터 증강 기법

본 연구에서는 모델별 특성에 따라 최적의 데이터 전처리(transform) 전략을 적용하여 학습 효율성과 일반화 성능을 극대화하였다. 특히, PyTorch 프레임워크 기반의 `transforms` API를 활용하여 각 모델에 적합한 입력 크기, 정규화 값, 증강 기법을 적용하였다.

(1) 모델별 이미지 전처리 전략

각 모델이 요구하는 입력 해상도 및 전처리 기준이 상이하므로, 다음과 같이 모델별 전처리 파이프라인을 구축하였다:

모델명	입력 크기	정규화(mean/std)	주요 증강 기법
VGG16, ResNet50	224x224	ImageNet 기준: [0.485, 0.456, 0.406] / [0.229, 0.224, 0.225]	`RandomResizedCrop`, `RandomHorizontalFlip`
EfficientNetV2	300x300	[0.5, 0.5, 0.5] / [0.5, 0.5, 0.5]	`AutoAugment`, `RandomResizedCrop`
ConvNeXt	224x224	[0.5, 0.5, 0.5] / [0.5, 0.5, 0.5]	`AutoAugment`, `RandomResizedCrop`
Swin Transformer	256x256	[0.5, 0.5, 0.5] / [0.5, 0.5, 0.5]	`RandomResizedCrop`, `RandomHorizontalFlip`

각 모델에 대해 train, validation, test에 대해 별도의 transform을 설정하였다. 일반적으로 학습용 데이터에는 데이터 증강을 포함하고, 검증 및 테스트 데이터에는 정적 전처리만 수행하였다.

(2) 데이터 정규화 및 Tensor 변환

모든 이미지 데이터는 다음과 같은 순서로 처리되었다.

- PIL 이미지 → Tensor (`transforms.ToTensor()`)
- 정규화(Normalization): 모델별로 설정한 평균/표준편차 기준으로 정규화
    - CNN 모델 (VGG, ResNet 등): ImageNet 기준 정규화 적용
      Transformer 모델 및 최신 CNN 모델: 평균 및 표준편차 = `[0.5, 0.5, 0.5]`

(3) 데이터셋 분할 전략

전체 이미지 데이터셋은 `StratifiedShuffleSplit`을 활용하여 클래스 비율을 유지한 채로 다음과 같이 분할하였다:

- 학습 데이터: 70%
- 검증 데이터: 15%
- 테스트 데이터: 15%

데이터셋 분할은 다음과 같은 절차로 이루어졌다:

1. 전체 데이터를 학습:비학습 = 7:3으로 분할 (1차 Stratified)
2. 비학습(30%) 데이터를 다시 검증:테스트 = 1:1로 분할 (2차 Stratified)
3. 최종적으로 `TransformedSubset` 클래스를 활용하여 각 subset에 알맞은 transform 적용

(4) 적용된 주요 증강 기법

- `RandomResizedCrop`: 다양한 스케일의 이미지를 학습하여 크기 불변성 확보
- `RandomHorizontalFlip`: 좌우 대칭 변형을 통한 데이터 다양성 확보

- `AutoAugment`: 자동 이미지 증강 정책을 통한 일반화 성능 강화
- `CenterCrop` / `Resize`: 검증 및 테스트 시 정적인 중앙 크롭으로 일관된 평가

2.5 데이터 분할

데이터셋은 전형적인 머신러닝 실험 방식에 따라 학습(Training), 검증(Validation), 테스트(Test) 집합으로 나누었다. 각각의 비율은 다음과 같다.

데이터 분할	이미지 수 (개)	비율 (%)
학습 (Training)	2196	80% (학습 데이터)
검증 (Validation)	550	20% (학습 데이터)
테스트 (Test)	924	100% (테스트 데이터)
총합	3650	200% (학습 데이터 + 테스트 데이터)

원본 데이터의 구성이 학습(train) 데이터와 테스트(test) 데이터로 나누어져 있다. 따라서, 학습 데이터셋을 학습 데이터(80%)와 검증 데이터(20%)로 나누었고, 테스트 데이터는 라벨이 없는 데이터이고, 주어진 데이터를 모두 사용했다.

3. 모델 설명

본 연구에서는 이미지 분류 성능을 비교하기 위해 CNN 계열 모델과 Transformer 계열의 최신 모델을 선정하였다. 모델은 이미지 분류에 사용되는 대표적인 모델들을 선정의 기준은 이미지 분류에서의 대표성, 전이 학습 가능성, 최근 성능 우수성 등이다. 각 모델은 PyTorch Hub, Hugging Face, timm에서 제공하는 사전 학습된(pretrained) 가중치를 기반으로 전이 학습을 진행하였다.

3.1 CNN 계열 모델

VGG16

VGG16은 2014년 Simonyan과 Zisserman이 개발한 CNN 모델로, 구조가 간결하며 깊이가 증가할수록 성능이 향상됨을 처음으로 명확히 제시한 모델이다. VGG16은 13개의 컨볼루션 층과 3개의 Fully Connected(FC) 층으로 구성되며, 모든 컨볼루션 커널 크기를 3×3으로 통일하여 구조가 단순한 반면 강력한 성능을 제공한다. 이미지 분류 연구의 기준 모델로 자주 활용되며, 본 연구에서도 기본 성능 측정과 비교를 위해 선택하였다.

ResNet50

ResNet(Residual Network)은 Microsoft Research에서 2015년 제안한 모델로, 이미지넷(ImageNet) 챌린지에서 우수한 성적을 기록하여 CNN 구조의 혁신을 일으켰다. ResNet은 깊은 신경망 구조에서 발생하는 Gradient 소실 문제를 해결하기 위해 "Skip Connection"이라는 잔차 연결(residual connection)을 도입하였다. 특히 ResNet50은 50개의 층으로 구성되어 있으며, Skip Connection 덕분에 깊이가 증가해도 효율적으로 학습할 수 있어 이미지 분류 연구에 가장 널리 활용되고 있다.

EfficientNetV2

EfficientNetV2는 Google에서 2021년에 제안한 최신 CNN 모델로, 기존의 EfficientNet을 개량하여 연산 효율성과 성능을 한층 향상시킨 구조이다. Compound Scaling 전략을 통해 모델의 깊이(depth), 너비(width), 해상도(resolution)를 균형 있게 최적화한 EfficientNet 시리즈의 장점을 계승하면서도, 학습 과정의 효율성 및 GPU 자원의 활용성을 극대화하였다. 특히, 작은 규모의 데이터셋에서도 뛰어난 성능과 빠른 수렴 속도를 보여 본 연구에서 유력한 성능 비교 모델로 선정하였다.

ConvNeXt

ConvNeXt는 Meta AI (구 Facebook AI)에서 2022년에 제안한 모델로, Transformer 기반 모델(ViT, Swin 등)의 성공적인 설계 원리를 CNN 구조에 도입하여 CNN의 단순성과 Transformer의 강력한 성능을 동시에 확보한 모델이다. ConvNeXt는 CNN의 주요 구성 요소들을 체계적으로 개선하여 CNN 모델이 Transformer와 경쟁할 수 있음을 입증하였다. 특히 ConvNeXt는 ResNet 대비 구조가 간결하면서도 더 높은 성능을 제공하여 최근 이미지 분류 분야에서 크게 주목받고 있다.

3.2 Transformer 계열 모델

Vision Transformer (ViT)

Vision Transformer(ViT)는 2021년 Dosovitskiy 등이 처음 제안한 Transformer 기반 이미지 분류 모델로, CNN과 달리 컨볼루션 연산 없이 Self-Attention 메커니즘을 사용하여 이미지를 처리한다. ViT는 이미지를 작은 패치(patch)로 나누고 이를 Flatten하여 Transformer의 입력으로 사용한다. CNN과 달리 이미지의 전역(global) 정보를 직접적으로 모델링할 수 있어 대규모 데이터셋에서 높은 성능을 기록하며 이미지 분류에서 CNN 모델의 성능을 뛰어넘는 새로운 접근 방식을 제시하였다.

Swin Transformer

Swin Transformer는 Microsoft에서 2021년 제안한 모델로, Vision Transformer의 한계를 보완하여 이미지 특성을 더 효과적으로 반영할 수 있도록 Hierarchical Transformer 구조를 도입하였다. 특히 Swin Transformer는 Shifted Window 방식을 통해 Self-Attention의 계산 비용을 낮추면서 이미지의 로컬(local) 특성과 글로벌(global) 특성을 동시에 효과적으로 학습한다. 그 결과, 작은 데이터셋에서도 CNN을 능가하는 뛰어난 일반화 성능을 보이며, 최근 다양한 컴퓨터 비전 분야의 벤치마크에서 최고의 성능을 기록하고 있다.

3.3 모델 선정 이유

본 연구에서 위의 CNN 및 Transformer 기반 모델들을 선정한 이유는 다음과 같다.

- 대표성 : VGG16, ResNet50 등은 이미지 분류 분야에서 가장 많이 활용되는 대표 모델로 성능 비교의 기준점 역할을 수행한다.
- 최신성 및 우수성 : EfficientNetV2, ViT, Swin Transformer, ConvNeXt 등 최신 모델들은 최근 연구에서 뛰어난 성능을 기록하며 이미지 분류 분야의 최신 흐름을 나타낸다.
- 다양한 접근 방식 비교 : CNN 모델과 Transformer 모델의 성능을 직접 비교함으로써 각 접근 방식의 장점과 한계를 명확히 분석할 수 있다.
- 전이 학습 가능성 : PyTorch Hub, Hugging Face, timm 등을 통해 사전 학습된 가중치를 활용할 수 있어 적은 데이터로도 충분한 성능 평가가 가능하다.

4. 실험 방법

본 연구는 CNN 및 Transformer 기반의 이미지 분류 모델을 동일한 실험 환경 하에서 전이 학습(Transfer Learning) 방식으로 학습시키고, 성능을 정량적으로 비교 분석하였다. 실험은 Google Colab Pro 환경에서 NVIDIA A100 GPU를 기반으로 진행되었다.

4.1 실험 환경 구성

항목	설명
플랫폼 (Platform)	Google Colab
하드웨어(GPU)	NVIDIA A100
프레임워크(Framework)	PyTorch 2.x
주요 라이브러리(Library)	torchvision, timm, Hugging Face Transformers, numpy, scikit-learn
프로그래밍 언어 (Programming Language)	Python 3.10

4.2 전이 학습 과정

모든 모델은 사전 학습된(pretrained) 가중치를 로딩하여 전이 학습을 수행하였다. 각 모델은 `PyTorch Hub`, `Hugging Face Transformers`, `timm` 라이브러리를 통해 로드하였으며, 다음과 같은 방식으로 학습을 진행하였다:

- 출력 계층(Classifier)은 현재 꽃 이미지 분류 데이터셋의 클래스 수(5개)에 맞게 수정
- 전체 파라미터 학습(Fine-tuning) 방식 적용
- 학습 에폭(epoch)은 모델별로 **10**회로 고정하여 성능 비교의 일관성을 유지

4.3 사용한 최적화 기법

모델 학습의 성능 향상과 최적의 하이퍼파라미터 탐색을 위해 **Optuna** 프레임워크를 활용한 자동화된 하이퍼파라미터 최적화 기법을 적용하였다.

(1) 최적화 대상 항목

Optuna의 `Trial` 객체를 활용하여 다음과 같은 하이퍼파라미터를 탐색하였다:

하이퍼파라미터	탐색 범위	설명
Learning Rate	1e-5 ~ 1e-1 (log scale)	학습 속도 조절
Optimizer	[“Adam”]	현재는 Adam 고정 (추후 확장 가능)
Batch Size	[16, 32, 64]	연산 효율성 및 일반화 영향

탐색 알고리즘은 Optuna의 기본 TPE(Tree-structured Parzen Estimator) sampler를 사용하였다.

(2) 튜닝 전략

- 목표 함수 정의: 학습된 모델의 검증 정확도**(Validation Accuracy)** 를 최대화하는 방향으로 탐색
- 학습 에폭 수: 초기 튜닝 시에는 탐색 속도를 고려하여 3 epoch만 학습 수행
- **Trial** 횟수: 총 10회의 탐색(trial)을 통해 최적의 조합을 선정

```
study = optuna.create_study(direction="maximize")
study.optimize(objective_function, n_trials=10)
```

(3) 최적화 결과 적용

튜닝이 완료되면, 최적의 하이퍼파라미터 조합을 모델에 반영하여 전체 에폭 기준의 재학습을 수행함:

```
model.update_hyperparameters(
    best_trial["learning_rate"],
    best_trial["optimizer"],
    best_trial["batch_size"]
)
```

Optuna를 활용함으로써 수작업 대비 효율적으로 성능을 개선할 수 있었으며, 실험 반복에 따른 일관된 최적화 결과를 얻을 수 있었다.

4.4 평가 방법 및 평가 지표

모델 성능 평가는 학습 과정 중 검증 데이터셋**(Validation Set)** 을 이용하여 에폭(epoch)마다 지속적으로 수행되었으며, 최종 성능은 테스트 데이터셋**(Test Set)** 을 통해 정량적으로 분석하였다.

(1) 학습 중 평가 (Training Phase Evaluation)

- 모델은 `train()` 모드로 설정된 후 학습 루프를 수행하며, 각 에폭마다 학습 손실(Loss)과 정확도(Accuracy)를 계산하였다.
  정확도는 `torch.max()`를 사용하여 예측 결과와 실제 레이블을 비교함으로써 계산되었다.
- 손실 함수는 다중 클래스 분류 문제에 적합한 `CrossEntropyLoss`를 사용하였다.
- 학습 중 발생할 수 있는 **NaN** 또는 **Inf** 값 검출을 위해, 입력 및 손실 텐서에 대해 `torch.isnan()` 및 `torch.isinf()`를 체크하는 방어 코드가 포함되었다.

```
if has_invalid(inputs):
    print("[경고] 입력에 NaN 또는 Inf 포함됨!")
if torch.isnan(loss):
    print(f"[에러] Epoch {epoch + 1}: loss가 NaN입니다!")
```

(2) 검증 단계 평가 (Validation Phase Evaluation)

- 각 에폭의 끝마다 `ImageClassificationModelValidator`를 통해 모델을 평가하였다.
  검증 정확도(`val_acc`)와 검증 손실(`val_loss`)은 모델의 일반화 성능을 평가하기 위한 주요 지표로 사용되었다.
- 학습 로그는 `train_logs` 리스트에 매 에폭마다 기록되며, 추후 시각화 및 분석에 활용되었다.
- 주요 기록 항목:
  - 학습/검증 손실 (`train_loss`, `val_loss`)
  - 학습/검증 정확도 (`train_acc`, `val_acc`)
  - 에폭 번호, 시작 및 종료 시간

```
train_logs.append({
    "epoch": epoch + 1,
    "start_time": start_time,
    "end_time": end_time,
    "train_loss": train_loss,
    "train_acc": train_accuracy,
    "val_loss": validation_loss,
    "val_acc": validation_accuracy
})
```

(3) 최종 성능 평가 (Test Phase Evaluation)

- 모델 학습이 완료된 후, `test_dataset`을 사용하여 정확도**(Accuracy)** 와 손실**(Loss)** 을 측정하였다.
- 테스트는 학습 및 검증에 사용되지 않은 데이터에 대해 수행되므로 모델의 최종 일반화 성능을 측정하는 기준이 된다.

(4) 추가 평가 및 과적합 분석

- 과적합 여부 분석: 학습 로그를 활용해 `Train vs. Validation` 정확도 및 손실 곡선을 시각화하고, 학습 도중 검증 성능이 하락하는 시점을 확인하였다.
- 추가 평가 지표 **(**선택 적용**)**:
  - `Confusion Matrix`: 클래스별 정밀도 확인
  - `Precision`, `Recall`, `F1-Score`: 다중 클래스 분류 성능의 세부 분석

(5) 시각화 및 로그 기록

- 학습 과정에서의 모든 로그는 모델 객체 내부의 `training_logs` 필드에 저장되었으며, `matplotlib` 또는 `seaborn`을 이용한 시각화에 활용되었다.
- 향후 로그 기반의 학습 곡선 분석, 성능 변화 추이 파악 등에 유용하게 활용할 수 있다.

[저장된 학습 성능 평가 이력]

```sql
SELECT M.model_name, TL.epoch, TL.start_datetime, TL.end_datetime, TL.training_loss, TL.training_accuracy, TL.validation_loss, TL.validation_accuracy
FROM (
    SELECT * FROM (
        SELECT * FROM TrainingLogs ORDER BY training_log_id DESC LIMIT 60
    ) as LTL ORDER BY LTL.training_log_id ASC
) as TL JOIN Training T JOIN Models M
WHERE TL.training_id = T.training_id and T.model_id = M.model_id
ORDER BY M.model_id, TL.epoch
```

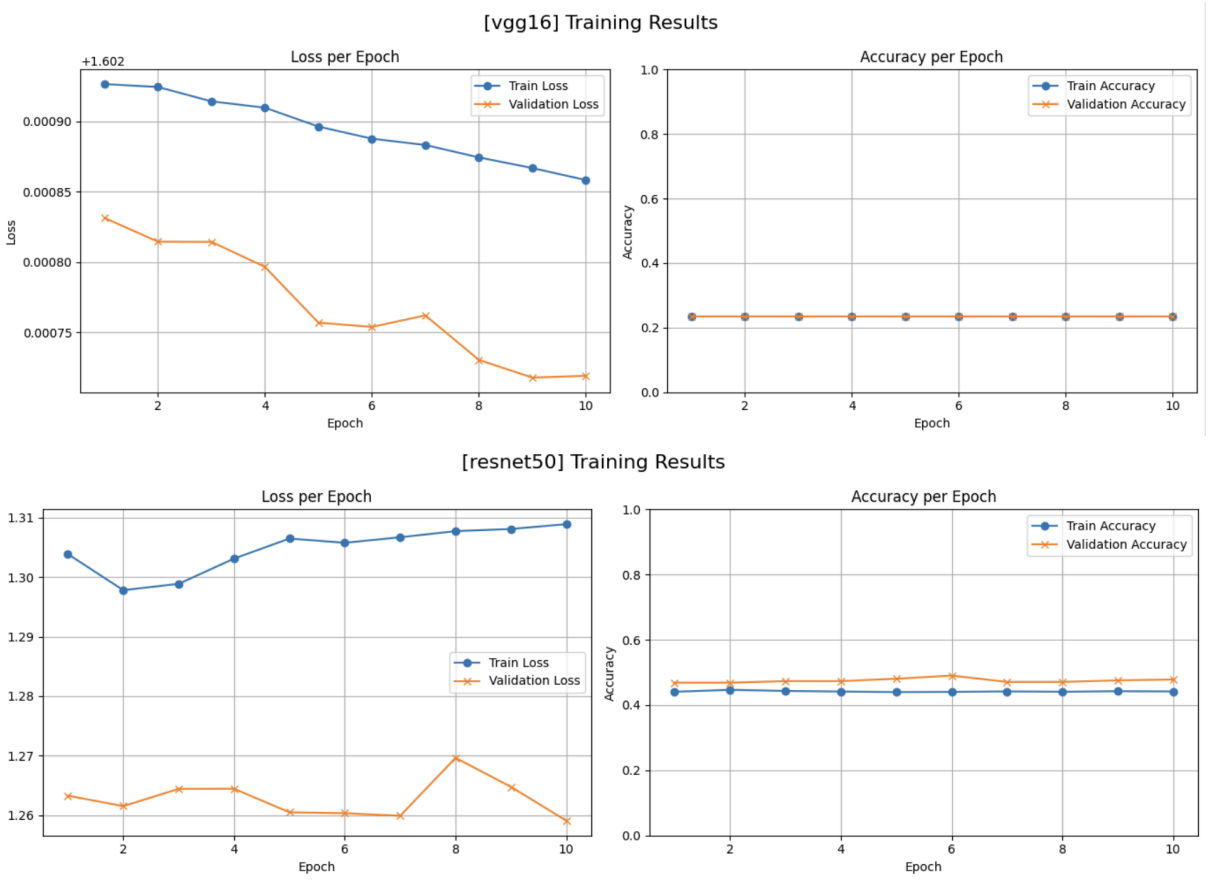
Models(+) 1

SELECT M.model_name, TL.epoch, TL.start_datetime, TL.end_da | Enter a SQL expression to filter results (use Ctrl+Space)

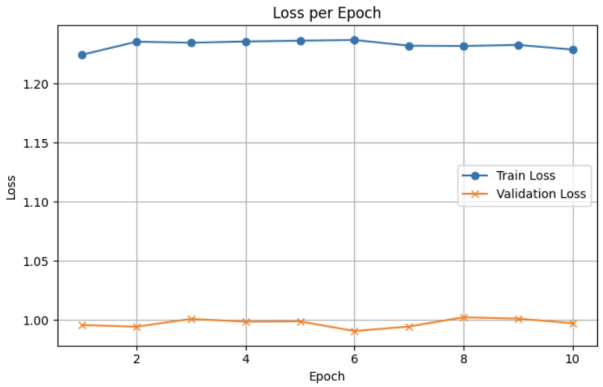
	model_name	epoch	start_datetime	end_datetime	training_loss	training_accuracy	validation_loss	validation_accuracy
1	vgg16	1	2025-03-30 07:20:00	2025-03-30 07:20:12	1.60293	0.235172	1.60283	0.235437
2	vgg16	2	2025-03-30 07:20:12	2025-03-30 07:20:24	1.60292	0.235172	1.60281	0.235437
3	vgg16	3	2025-03-30 07:20:24	2025-03-30 07:20:36	1.60291	0.235172	1.60281	0.235437
4	vgg16	4	2025-03-30 07:20:36	2025-03-30 07:20:49	1.60291	0.235172	1.6028	0.235437
5	vgg16	5	2025-03-30 07:20:49	2025-03-30 07:21:01	1.6029	0.235172	1.60276	0.235437
6	vgg16	6	2025-03-30 07:21:01	2025-03-30 07:21:13	1.60289	0.235172	1.60275	0.235437
7	vgg16	7	2025-03-30 07:21:13	2025-03-30 07:21:26	1.60288	0.235172	1.60276	0.235437
8	vgg16	8	2025-03-30 07:21:26	2025-03-30 07:21:38	1.60287	0.235172	1.60273	0.235437
9	vgg16	9	2025-03-30 07:21:38	2025-03-30 07:21:50	1.60287	0.235172	1.60272	0.235437
10	vgg16	10	2025-03-30 07:21:50	2025-03-30 07:22:03	1.60286	0.235172	1.60272	0.235437
11	resnet50	1	2025-03-30 07:28:22	2025-03-30 07:28:34	1.3039	0.440687	1.26328	0.468447
12	resnet50	2	2025-03-30 07:28:34	2025-03-30 07:28:46	1.29783	0.44667	1.26151	0.468447
13	resnet50	3	2025-03-30 07:28:46	2025-03-30 07:28:59	1.29889	0.443288	1.26442	0.473301
14	resnet50	4	2025-03-30 07:28:59	2025-03-30 07:29:11	1.30313	0.441337	1.26444	0.473301
15	resnet50	5	2025-03-30 07:29:11	2025-03-30 07:29:23	1.30651	0.439646	1.26048	0.480583

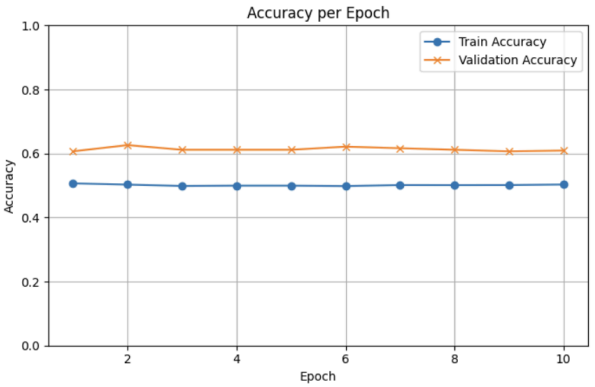
[모델 별 학습 성능]

- Training Loss & Validation Loss 그래프 (좌)
- Training Accuracy & Validation Accuracy 그래프 (우)

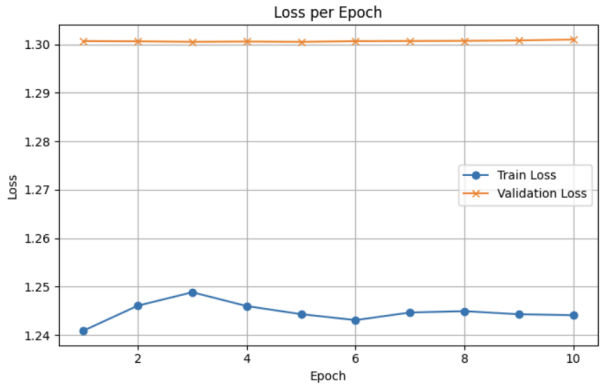


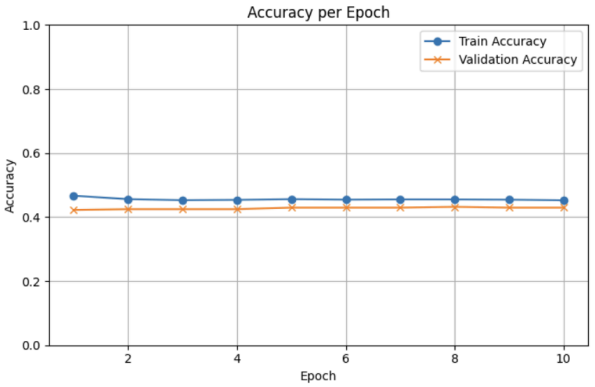
[tf_efficientnetv2_s] Training Results



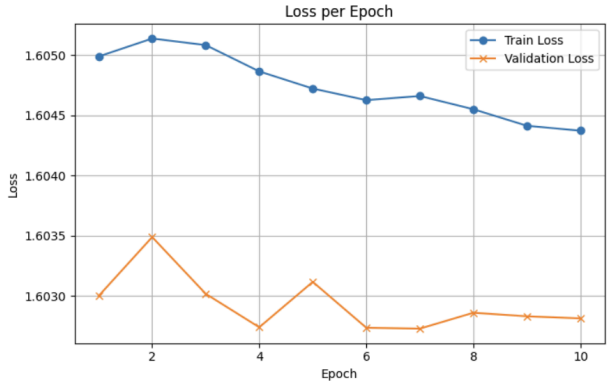


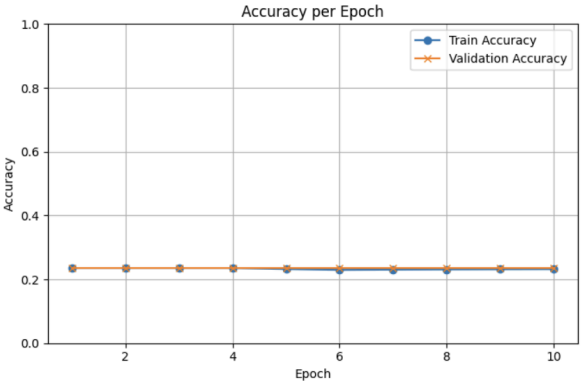
[vit_base_patch16_224] Training Results



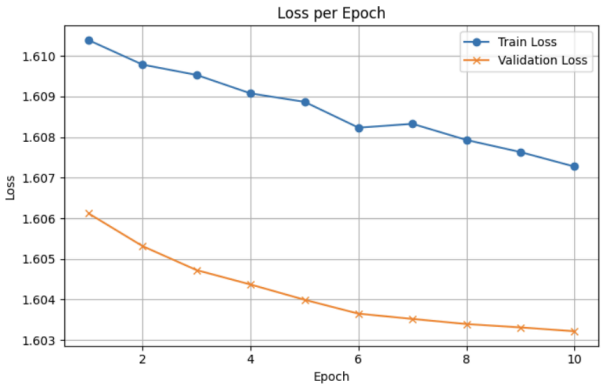


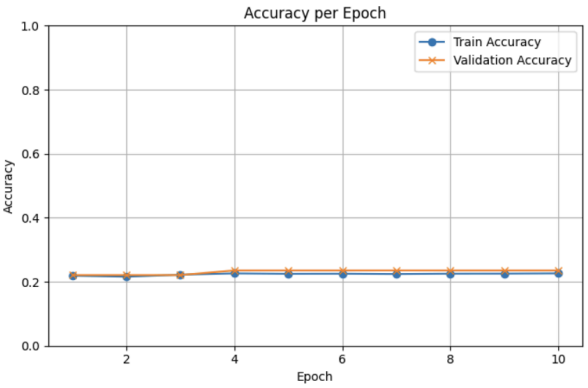
[convnext_tiny] Training Results





[swinv2_tiny_window8_256] Training Results





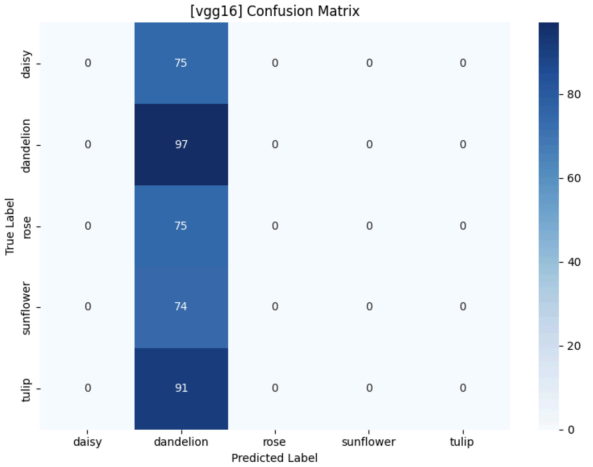
[모델 별 테스트 성능 평가]

- Performance Matrix & Classification Report (좌)
- Confusion Matrix (우)

📊 [vgg16] Performance Metrics
Accuracy : 0.2354
Precision: 0.0554
Recall : 0.2354
F1 Score : 0.0897

📋 Classification Report

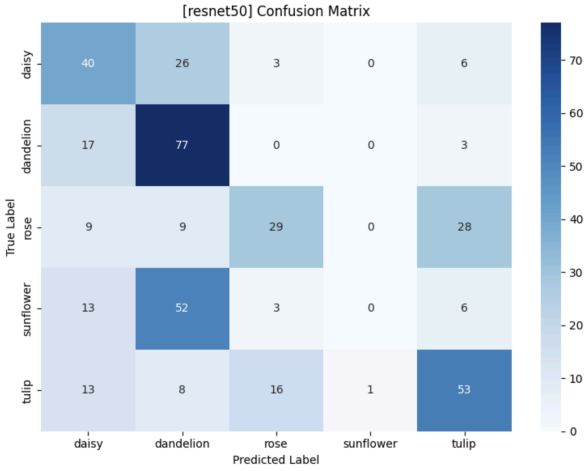
	precision	recall	f1-score	support
daisy	0.00	0.00	0.00	75
dandelion	0.24	1.00	0.38	97
rose	0.00	0.00	0.00	75
sunflower	0.00	0.00	0.00	74
tulip	0.00	0.00	0.00	91
accuracy			0.24	412
macro avg	0.05	0.20	0.08	412
weighted avg	0.06	0.24	0.09	412



📊 [resnet50] Performance Metrics
Accuracy : 0.4830
Precision: 0.4100
Recall : 0.4830
F1 Score : 0.4310

📋 Classification Report

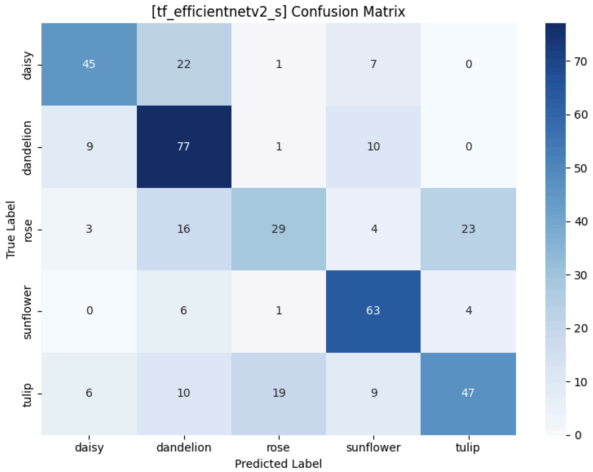
	precision	recall	f1-score	support
daisy	0.43	0.53	0.48	75
dandelion	0.45	0.79	0.57	97
rose	0.57	0.39	0.46	75
sunflower	0.00	0.00	0.00	74
tulip	0.55	0.58	0.57	91
accuracy			0.48	412
macro avg	0.40	0.46	0.42	412
weighted avg	0.41	0.48	0.43	412



📊 [tf_efficientnetv2_s] Performance Metrics
Accuracy : 0.6335
Precision: 0.6339
Recall : 0.6335
F1 Score : 0.6229

📋 Classification Report

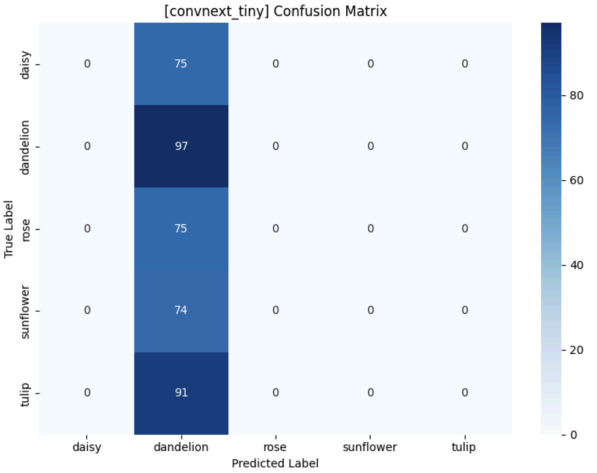
	precision	recall	f1-score	support
daisy	0.71	0.60	0.65	75
dandelion	0.59	0.79	0.68	97
rose	0.57	0.39	0.46	75
sunflower	0.68	0.85	0.75	74
tulip	0.64	0.52	0.57	91
accuracy			0.63	412
macro avg	0.64	0.63	0.62	412
weighted avg	0.63	0.63	0.62	412



📊 [convnext_tiny] Performance Metrics
Accuracy : 0.2354
Precision: 0.0554
Recall   : 0.2354
F1 Score : 0.0897

📋 Classification Report

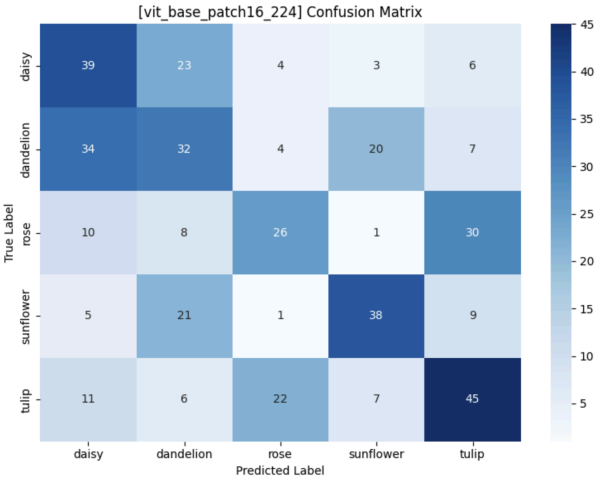
	precision	recall	f1-score	support
daisy	0.00	0.00	0.00	75
dandelion	0.24	1.00	0.38	97
rose	0.00	0.00	0.00	75
sunflower	0.00	0.00	0.00	74
tulip	0.00	0.00	0.00	91
accuracy			0.24	412
macro avg	0.05	0.20	0.08	412
weighted avg	0.06	0.24	0.09	412



📊 [vit_base_patch16_224] Performance Metrics
Accuracy : 0.4369
Precision: 0.4398
Recall   : 0.4369
F1 Score : 0.4351

📋 Classification Report

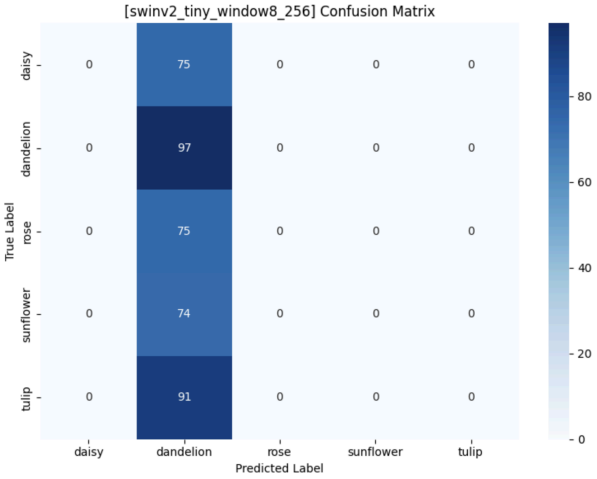
	precision	recall	f1-score	support
daisy	0.39	0.52	0.45	75
dandelion	0.36	0.33	0.34	97
rose	0.46	0.35	0.39	75
sunflower	0.55	0.51	0.53	74
tulip	0.46	0.49	0.48	91
accuracy			0.44	412
macro avg	0.44	0.44	0.44	412
weighted avg	0.44	0.44	0.44	412



📊 [swinv2_tiny_window8_256] Performance Metrics
Accuracy : 0.2354
Precision: 0.0554
Recall   : 0.2354
F1 Score : 0.0897

📋 Classification Report

	precision	recall	f1-score	support
daisy	0.00	0.00	0.00	75
dandelion	0.24	1.00	0.38	97
rose	0.00	0.00	0.00	75
sunflower	0.00	0.00	0.00	74
tulip	0.00	0.00	0.00	91
accuracy			0.24	412
macro avg	0.05	0.20	0.08	412
weighted avg	0.06	0.24	0.09	412



5. 결과 및 분석

실험 결과, 모든 모델(VGG16, ResNet50, EfficientNetV2, ConvNeXt, ViT, Swin Transformer)에서 전반적으로 성능이 기대 이하였으며, 일부 모델은 거의 학습되지 않은 수준의 정체된 성능을 보였다.

- **VGG16, ConvNeXt, SwinV2**의 경우, 학습 및 검증 정확도가 약 **23.5%** 수준에서 **10 epoch** 내내 동일하게 유지되며, 손실(Loss) 또한 거의 변화가 없었다. 이는 모델이 무작위 추측 수준에서 벗어나지 못했음을 의미한다.
- **ResNet50**은 다른 모델에 비해 상대적으로 정확도가 **47~48%** 수준까지 향상되었지만, 손실 값이 안정적으로 감소하지 않았고 성능이 수렴하지 않았다.
- **EfficientNetV2**, **ViT**는 일부 에폭에서 약간의 향상을 보였으나, 과적합 방지 및 일반화 성능 측면에서 유의미한 개선은 나타나지 않았다.

이러한 결과는 다음과 같은 문제 가능성을 시사한다:

- 데이터셋의 전처리/증강 방식이 모델 구조와 부적합하거나 너무 약할 수 있음
- 학습률, 배치 사이즈 등 하이퍼파라미터 설정이 전반적으로 부적절
- 모델이 Freeze되어 있는 경우 파라미터가 제대로 업데이트되지 않았을 가능성
- 전이 학습 시 Pretrained Weight가 제대로 적용되지 않았을 가능성

6. 결론

본 연구에서는 대표적인 CNN 및 Transformer 기반 모델을 활용하여 전이 학습을 통한 꽃 이미지 분류 성능을 비교하였다. 그러나 실험 결과, 전반적인 성능이 기대에 못 미쳤고, 특히 많은 모델이 학습 초기 수준에서 멈추는 학습 불능 또는 비수렴 현상을 보였다.

이러한 실패는 오히려 전이 학습 기반 이미지 분류 실험에서 고려해야 할 다양한 변수를 재확인하는 계기가 되었으며, 다음과 같은 점에서 추가적인 개선과 연구가 필요함을 확인하였다:

- 모델과 데이터 간의 적합성, 하이퍼파라미터, 전처리 전략이 성능에 미치는 영향을 정밀하게 분석해야 함
- 단순한 전이 학습 적용만으로는 고성능 분류 모델을 만들기 어렵고, 실험적이고 체계적인 튜닝이 병행되어야 함

7. 향후 계획

실패를 바탕으로, 다음과 같은 구체적인 개선 및 연구 계획을 수립하였다:

1. 데이터 품질 개선
    - 현재 전처리가 너무 단순하며, 증강도 모델에 최적화되지 않았을 수 있음
    - 향후 이상치 제거**,** 고급 데이터 증강 **(AutoAugment, CutMix** 등**)** 을 적용하고, 이미지 품질 분석을 통해 노이즈 제거 기법 도입

2. 모델 구조 및 동작 원리 학습
    - CNN과 Transformer 계열 모델에 대한 동작 메커니즘과 학습 방식을 보다 정밀하게 분석
    - 학습 가능 여부를 판단할 수 있는 **Feature Map** 시각화, **Gradient** 확인, 모델 **freeze** 여부 검사 등 도입

3. 하이퍼파라미터 튜닝 전략 고도화
    - 현재 Optuna를 이용한 일괄 튜닝은 한계가 있었음. 향후에는 모델별로 하이퍼파라미터 탐색 공간을 다르게 설정하고, **Multi-Fidelity** 방식, **Bayesian Optimization** 등 보다 정밀한 튜닝 전략 도입 예정

4. 전이 학습 방식 재점검
    - Pretrained weight가 실제로 적용되었는지 검증하고, 필요한 경우 **partial fine-tuning** 또는 **full fine-tuning** 방식을 구분하여 적용할 예정

5. 실험 로그 및 모델 관리 자동화
    - 모든 모델 학습, 검증, 테스트 이력을 효율적으로 기록하고 추적할 수 있는 실험 관리 시스템(Ex. SQLite, MLflow) 도입 예정
    - 모델 파라미터, 학습 설정, 결과를 통합적으로 관리하여, 추후 반복 실험 및 비교 분석의 기반을 마련할 것

참고문헌 **(References)**

[1] Kaggle Flowers Dataset, “Flowers Recognition Dataset,” Kaggle, 2023.
URL: https://www.kaggle.com/datasets/imsparsh/flowers-dataset/data. (Accessed: Mar. 24, 2025).

[2] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *International Conference on Learning Representations (ICLR)*, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[4] M. Tan and Q. V. Le, “EfficientNetV2: Smaller Models and Faster Training,” *Proceedings of the International Conference on Machine Learning (ICML)*, vol. 139, pp. 10096–10106, 2021.

[5] A. Dosovitskiy et al., “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” *International Conference on Learning Representations (ICLR)*, 2021.

[6] Z. Liu et al., “Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows,” *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 10012–10022, 2021.

[7] Z. Liu et al., “A ConvNet for the 2020s,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11976–11986, 2022.

[8] Ross Wightman, "PyTorch Image Models (timm)," Hugging Face, 2023.
URL: https://huggingface.co/docs/timm/index. (Accessed: Mar. 24, 2025).

[9] PyTorch Hub, "PyTorch Hub Documentation," PyTorch, 2023.
URL: https://pytorch.org/hub. (Accessed: Mar. 24, 2025).

[10] Hugging Face Transformers, "Vision Transformer Documentation," Hugging Face, 2023.
URL: https://huggingface.co/docs/transformers. (Accessed: Mar. 24, 2025).

부록 (Appendix)

Appendix 1. 전체 Source Code

- 전체 소스코드는 Google Colab 링크와 함께 Appendix로 이 문서에 첨부한다.
- Google Colab에 있는 Note를 제외한 순수 Source Code만 포함한다.
- Colab 코드 링크

[소스코드 전문]

```
!pip install optuna
!pip install pymysql sqlalchemy

# Google Colab
from google.colab import drive, files

# 시스템 명령 & 파일 처리
import os
import shutil

# PyTorch
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, random_split
import torch.optim as optim

# torchvision
import torchvision
from torchvision import datasets, transforms
from torchvision.models import (
    VGG16_Weights,
    ResNet50_Weights,
    ResNeXt50_32X4D_Weights
)

# hugging face timm
import timm

# Garbage Collector
import gc

# Python의 반복문에 진행상황을 시각적으로 보여주는 라이브러리
from tqdm import tqdm

# 데이터 분석 및 시각화
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# 이미지 처리
from PIL import Image

from datetime import datetime
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.metrics import classification_report, confusion_matrix

import optuna

class Logger:
    def __init__(self, name):
        self.name = name

    def log(self, message):
        now = datetime.now()
        date = now.strftime('%Y-%m-%d')
        time = now.strftime('%H:%M:%S.') + f'{int(now.microsecond /
1000):03d}'
        if self.name:
            print(f"[{date} {time}] [{self.name}] {message}")
        else:
            print(f"[{date} {time}] {message}")

    def __call__(self, message):
        self.log(message)

import pymysql
from sqlalchemy import create_engine, text
from datetime import datetime

class MysqlClient:
    def __init__(self, host, port, user, password, db):
        self.engine = create_engine(
            f"mysql+pymysql://{user}:{password}@{host}:{port}/{db}",
            echo=False,
            future=True
        )
        self._create_tables_if_not_exist()

    def _create_tables_if_not_exist(self):
        with self.engine.begin() as conn:
            conn.execute(text("""
                CREATE TABLE IF NOT EXISTS Models (
                    model_id INT AUTO_INCREMENT PRIMARY KEY,
                    model_name VARCHAR(255) UNIQUE NOT NULL
                );
            """))
```

```
            conn.execute(text("""
                CREATE TABLE IF NOT EXISTS Training (
                    training_id INT AUTO_INCREMENT PRIMARY KEY,
                    training_datetime DATETIME DEFAULT CURRENT_TIMESTAMP,
                    model_id INT,
                    FOREIGN KEY (model_id) REFERENCES Models(model_id)
                );
            """))
            conn.execute(text("""
                CREATE TABLE IF NOT EXISTS TrainingLogs (
                    training_log_id INT AUTO_INCREMENT PRIMARY KEY,
                    training_id INT,
                    epoch INT,
                    start_datetime DATETIME,
                    end_datetime DATETIME,
                    training_loss FLOAT,
                    training_accuracy FLOAT,
                    validation_loss FLOAT,
                    validation_accuracy FLOAT,
                    FOREIGN KEY (training_id) REFERENCES
Training(training_id)
                );
            """))
            conn.execute(text("""
                CREATE TABLE IF NOT EXISTS Test (
                    test_id INT AUTO_INCREMENT PRIMARY KEY,
                    test_name VARCHAR(255),
                    model_id INT,
                    FOREIGN KEY (model_id) REFERENCES Models(model_id)
                );
            """))
            conn.execute(text("""
                CREATE TABLE IF NOT EXISTS TestLogs (
                    test_log_id INT AUTO_INCREMENT PRIMARY KEY,
                    test_id INT,
                    image VARCHAR(512),
                    real_class VARCHAR(128),
                    pred_class VARCHAR(128),
                    FOREIGN KEY (test_id) REFERENCES Test(test_id)
                );
            """))

    # ========== 모델 관련 ==========
    def get_or_create_model(self, model_name):
        with self.engine.begin() as conn:
            result = conn.execute(text("SELECT model_id FROM Models WHERE
model_name = :model_name"),
                                  {"model_name": model_name}).fetchone()
```

```
            if result:
                return result[0]
            conn.execute(text("INSERT INTO Models (model_name) VALUES
(:model_name)"),
                         {"model_name": model_name})
            return conn.execute(text("SELECT LAST_INSERT_ID()")).scalar()

    # ========== 학습 기록 ==========
    def create_training(self, model_name):
        model_id = self.get_or_create_model(model_name)
        with self.engine.begin() as conn:
            result = conn.execute(
                text("INSERT INTO Training (model_id) VALUES (:model_id)"),
                {"model_id": model_id}
            )
            training_id = conn.execute(text("SELECT
LAST_INSERT_ID()")).scalar()
            return training_id

    def insert_training_log(self, training_id, epoch, start_dt, end_dt,
                            train_loss, train_acc, val_loss, val_acc):
        with self.engine.begin() as conn:
            conn.execute(text("""
                INSERT INTO TrainingLogs (training_id, epoch,
start_datetime, end_datetime,
                                          training_loss, training_accuracy,
                                          validation_loss,
validation_accuracy)
                VALUES (:training_id, :epoch, :start_dt, :end_dt,
                        :train_loss, :train_acc, :val_loss, :val_acc)
            """), {
                "training_id": training_id,
                "epoch": epoch,
                "start_dt": start_dt,
                "end_dt": end_dt,
                "train_loss": train_loss,
                "train_acc": train_acc,
                "val_loss": val_loss,
                "val_acc": val_acc
            })

    def get_training_logs(self, training_id):
        with self.engine.begin() as conn:
            result = conn.execute(text("""
                SELECT * FROM TrainingLogs WHERE training_id = :training_id
            """), {"training_id": training_id}).fetchall()
            return result
```

```
    # ========== 테스트 기록 ==========
    def create_test(self, model_name, test_name):
        model_id = self.get_or_create_model(model_name)
        with self.engine.begin() as conn:
            result = conn.execute(
                text("INSERT INTO Test (test_name, model_id) VALUES
(:test_name, :model_id)"),
                {"test_name": test_name, "model_id": model_id}
            )
            test_id = conn.execute(text("SELECT LAST_INSERT_ID()")).scalar()
            return test_id

    def insert_test_log(self, test_id, image, real_class, pred_class):
        with self.engine.begin() as conn:
            conn.execute(text("""
                INSERT INTO TestLogs (test_id, image, real_class,
pred_class)
                VALUES (:test_id, :image, :real_class, :pred_class)
            """), {
                "test_id": test_id,
                "image": image,
                "real_class": real_class,
                "pred_class": pred_class
            })

    def get_test_logs(self, test_id):
        with self.engine.begin() as conn:
            result = conn.execute(text("SELECT * FROM TestLogs WHERE test_id
= :test_id"),
                                  {"test_id": test_id}).fetchall()
            return result

    # 모델 없으면 삽입, 있으면 무시
    def add_model_if_not_exists(self, model_name):
        with self.engine.begin() as conn:
            result = conn.execute(text("SELECT model_id FROM Models WHERE
model_name = :model_name"),
                                  {"model_name": model_name}).fetchone()
            if not result:
                conn.execute(text("INSERT INTO Models (model_name) VALUES
(:model_name)"),
                             {"model_name": model_name})
                print(f"[INFO] 모델 '{model_name}' 새로 추가됨.")
            else:
                print(f"[INFO] 모델 '{model_name}' 이미 존재함. 추가하지
않음.")

    # 모델 목록 조회 및 출력
```

```
    def list_models(self):
        with self.engine.begin() as conn:
            result = conn.execute(text("SELECT model_id, model_name FROM
Models")).fetchall()
            if not result:
                print("[INFO] 등록된 모델이 없습니다.")
            else:
                print("[모델 목록]")
                for row in result:
                    print(f" - ID: {row.model_id}, Name: {row.model_name}")
            return result

from google.colab import userdata
mysql_pw = userdata.get('MYSQL_PW')

client = MysqlClient(
    host = 'ybkim-mysql-ybkim-test.h.aivencloud.com',
    port = 19485,
    user = 'avnadmin',
    password = mysql_pw,
    db = 'defaultdb'
)

client.add_model_if_not_exists("vgg16")
client.list_models()

# Kaggle 인증 및 데이터 수집
# 드라이브 마운트
drive.mount('/content/drive')
base_dir = "/content/drive/MyDrive/Colab Notebooks"

# Kaggle 설정 및 데이터 로딩
kaggle_json_dir = os.path.join(base_dir, "Kaggle")
kaggle_json_file_name = "kaggle.json"
kaggle_json_file_path = os.path.join(kaggle_json_dir, kaggle_json_file_name)
kaggle_cli_root_path = "/root/.kaggle"
kaggle_cli_root_path_file = os.path.join(kaggle_cli_root_path,
kaggle_json_file_name)
model_save_dir = os.path.join(base_dir, "flower_models")

# 필요한 폴더 생성
## Google Drive
os.makedirs(kaggle_json_dir, exist_ok = True) # kaggle.json 파일 저장 폴더
생성
os.makedirs(model_save_dir, exist_ok = True) # 학습된 모델 저장용 폴더 생성
## Google Colab
### ~/.kaggle 디렉토리 생성 및 이동
### Kaggle CLI는 이 위치에 있는 kaggle.json 파일을 사용하도록 설계되어 있음.
```

```
(기본 인증 경로)
os.makedirs(kaggle_cli_root_path, exist_ok = True) # kaggle CLI용
kaggle.json 파일 폴더 생성

# kaggle.json을 Google Drive에서 찾고, 있으면 Colab으로 복사, 없으면 업로드.
if os.path.exists(kaggle_json_file_path):
    shutil.copy(kaggle_json_file_path, kaggle_cli_root_path_file)
else:
    files.upload()
    shutil.copy(kaggle_json_file_name, kaggle_json_file_path)

# kaggle.json 파일 권한 설정
# 600 = 해당 파일을 소유한 사용자만 읽고 쓸 수 있도록 설정
os.chmod(kaggle_cli_root_path_file, 600)

# Kaggle 데이터 다운로드
## -d 옵션은 --dataset 이고, 어떤 데이터셋을 받을지 데이터셋의 ID를 지정하는 옵션
!kaggle datasets download -d imsparsh/flowers-dataset

# Kaggle 데이터 압축 해제
## -q : quiet mode (출력 메시지 생략)
## -d : destination directory 지정
!unzip -q flowers-dataset.zip -d flowers

## 이미지 분류 클래스 설계
class ImageClassificationModel:
    """
    모델 이름, 모델, 하이퍼파라미터 등을 저장하는 이미지 분류 모델 클래스
    """
    DEFAULT_LEARNING_RATE = 0.001
    DEFAULT_OPTIMIZER = "Adam"
    DEFAULT_BATCH_SIZE = 32

    def __init__(self, device, model_name:  str, num_classes: int = 5):
        self.device = device

        self.model_name = model_name
        self.num_classes = num_classes
        self.model = None

        self._is_need_to_tuning = True
        self._is_need_to_training = True

        self.learning_rate = ImageClassificationModel.DEFAULT_LEARNING_RATE
        self.optimizer = ImageClassificationModel.DEFAULT_OPTIMIZER
        self.batch_size = ImageClassificationModel.DEFAULT_BATCH_SIZE

        self._logger = Logger(model_name)
```

```
        self.train_logs = []
        self.test_logs = []
        self.test_all_preds = []
        self.test_all_targets = []

    def load(self):
        """
        모델 정보를 기반으로 모델을 로딩하고 Classification Layer를 모델별로
수정하여 반환한다.
        """
        self._logger("로딩 시작")

        weights_map = {
            "vgg16": VGG16_Weights.DEFAULT,
            "resnet50": ResNet50_Weights.DEFAULT,
        }

        if self.model_name in ["vgg16", "resnet50"]:
            weights = weights_map.get(self.model_name, None)
            self.model = getattr(torchvision.models,
self.model_name)(weights = weights)
            in_features = self.model.classifier[6].in_features if "vgg" in
self.model_name else self.model.fc.in_features

            if "vgg" in self.model_name:
                self.model.classifier[6] = nn.Linear(in_features,
self.num_classes)
            else:
                self.model.fc = nn.Linear(in_features, self.num_classes)
        else:
            self.model = timm.create_model(self.model_name, pretrained =
True, num_classes = self.num_classes)

        self._logger("로딩 완료")

    def update_dataset(self, train_dataset, validation_dataset,
test_dataset):
        self.train_dataset = train_dataset
        self.validation_dataset = validation_dataset
        self.test_dataset = test_dataset

    def update_hyperparameters(self, learning_rate: float, optimizer: str,
batch_size: int):
        """
        모델의 하이퍼 파라미터를 업데이트한다.
        learning_rate, optimizer, batch_size
        """
```

```
        self._logger("Hyperparameter 튜닝 시작")
        self._logger(f"learning_rate = {self.learning_rate} ->
{learning_rate}")
        self._logger(f"optimizer = {self.optimizer} -> {optimizer}")
        self._logger(f"_batch_size = {self.batch_size} -> {batch_size}")

        self.learning_rate = learning_rate
        self.optimizer = optimizer
        self.batch_size = batch_size

        self._logger("Hyperparameter 튜닝 완료")

    def save(self):
        """
        학습된 모델을 저장한다.
        """
        self._logger("저장 시작")

        model_path = os.path.join(model_save_dir, f"{self.model_name}.pth")
        torch.save(self.model.state_dict(), model_path)
        print(f"[INFO] 모델이 저장됨: {model_path}")

        training_id = client.create_training(self.model_name)
        for log in self.train_logs:
            client.insert_training_log(
                training_id = training_id,
                epoch = log["epoch"],
                start_dt = log["start_time"],
                end_dt = log["end_time"],
                train_loss = log["train_loss"],
                train_acc = log["train_acc"],
                val_loss = log["val_loss"],
                val_acc = log["val_acc"]
            )

        test_id = client.create_test(self.model_name, "test")
        for log in self.test_logs:
            client.insert_test_log(
                test_id = test_id,
                image = log["image"],
                real_class = log["real_class"],
                pred_class = log["pred_class"]
            )

        self._logger("저장 완료")
        pass

    def unload(self):
```

```
        """
        모델을 언로딩한다. (메모리 최적화)
        """
        self._logger("언로딩 시작")

        del self.model
        torch.cuda.empty_cache()  # GPU 캐시 정리
        self._logger("Model 제거")

        self.train_dataset = None
        self.validation_dataset = None
        self.test_dataset = None

        self._logger("데이터셋 제거")

        gc.collect()
        self._logger("Garbage  Collection 실행")

        self._logger("언로딩 완료")
        pass

    def __repr__(self):
        return f"ImageClassificationModel(model_name={self.model_name},
learning_rate={self.learning_rate}, optimizer={self.optimizer},
batch_size={self.batch_size})"

def create_transform_for_model(imageClassificationModel:
ImageClassificationModel):
    train_transform = None
    val_transform = None
    test_transform = None

    mean = [0.5, 0.5, 0.5]
    std = [0.5, 0.5, 0.5]

    # train, val, Test transform 만들기
    if imageClassificationModel.model_name in ["vgg16", "resnet50"]:
        mean = [0.485, 0.456, 0.406]
        std = [0.229, 0.224, 0.225]

        train_transform = transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(mean = mean, std = std),
        ])

        val_transform = transforms.Compose([
```

```
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean = mean, std = std),
        ])
    elif imageClassificationModel.model_name in ["tf_efficientnetv2_s"]:
        train_transform = transforms.Compose([
            transforms.RandomResizedCrop(300),
            transforms.RandomHorizontalFlip(),
            transforms.AutoAugment(),
            transforms.ToTensor(),
            transforms.Normalize(mean = mean, std = std),
        ])

        val_transform = transforms.Compose([
            transforms.Resize(320),
            transforms.CenterCrop(300),
            transforms.ToTensor(),
            transforms.Normalize(mean = mean, std = std),
        ])
    elif imageClassificationModel.model_name in ["convnext_tiny"]:
        train_transform = transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.AutoAugment(),
            transforms.ToTensor(),
            transforms.Normalize(mean = mean, std = std),
        ])

        val_transform = transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean = mean, std = std),
        ])
    elif imageClassificationModel.model_name in ["swinv2_tiny_window8_256"]:
        train_transform = transforms.Compose([
            transforms.RandomResizedCrop(256),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(mean = mean, std = std),
        ])
        val_transform = transforms.Compose([
            transforms.Resize(288),
            transforms.CenterCrop(256),
            transforms.ToTensor(),
            transforms.Normalize(mean = mean, std = std),
        ])
```

```
    elif imageClassificationModel.model_name == "vit_base_patch16_224":
        train_transform = transforms.Compose([
            transforms.Resize(256),
            transforms.RandomCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.5, 0.5, 0.5],
                                 std=[0.5, 0.5, 0.5])
        ])
        val_transform = transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.5, 0.5, 0.5],
                                 std=[0.5, 0.5, 0.5])
        ])
    else:
        raise ValueError(f"지원하지 않는 모델입니다.
{imageClassificationModel.model_name}")

    test_transform = val_transform

    return train_transform, val_transform, test_transform

class TransformedSubset(Dataset):
    def __init__(self, base_dataset, indices, transform):
        self.base_dataset = base_dataset
        self.indices = indices
        self.transform = transform

    def __len__(self):
        return len(self.indices)

    def __getitem__(self, idx):
        image, label = self.base_dataset[self.indices[idx]]
        if self.transform:
            image = self.transform(image)
        return image, label

class DataPreprocessor:
    def __init__(self, data_path):
        self.data_path = data_path
        pass

    def split(self, imageClassificationModel: ImageClassificationModel,
train_rate: float = 0.7, validation_rate: float = 0.15, test_rate: float =
0.15):
        """
```

```
        모델 별 transform을 정의하고 데이터를 training, validation, testing의
비율에 맞게 반환한다.
        """
        if train_rate + validation_rate + test_rate != 1.0:
            raise ValueError("train_rate + validation_rate + test_rate
should be 1.0")

        # 모델 별 transform 생성하기
        train_transform, val_transform, test_transform =
create_transform_for_model(imageClassificationModel)

        # 전체 데이터셋 로딩
        full_dataset = datasets.ImageFolder(root = self.data_path, transform
= None)
        targets = np.array(full_dataset.targets)

        # 전체 데이터셋을 70%, 30%로 나누기
        sss1 = StratifiedShuffleSplit(n_splits=1, test_size=0.3,
random_state=42)
        train_idx, temp_idx = next(sss1.split(np.zeros(len(targets)),
targets))

        # 위에서 30%로 나눈 데이터를 전체의 15% 15%가 되도록 나누기 (30%의
1/2씩. 반반 나누기)
        temp_targets = targets[temp_idx]
        sss2 = StratifiedShuffleSplit(n_splits=1, test_size=0.5,
random_state=42)
        val_rel_idx, test_rel_idx =
next(sss2.split(np.zeros(len(temp_targets)), temp_targets))
        val_idx = temp_idx[val_rel_idx]
        test_idx = temp_idx[test_rel_idx]

        # 최종 데이터셋 설정
        train_dataset = TransformedSubset(full_dataset, train_idx,
train_transform)
        val_dataset   = TransformedSubset(full_dataset, val_idx,
val_transform)
        test_dataset  = TransformedSubset(full_dataset, test_idx,
test_transform)

        return train_dataset, val_dataset, test_dataset

class ImageClassificationModelValidator:
    """
    이미지 분류 모델의 훈련 과정 검증 담당
    """
    def __init__(self):
        pass
```

```
    def validate_model(self, imageClassificationModel:
ImageClassificationModel, epoch, epochs):
        optimizer_name = imageClassificationModel.optimizer
        batch_size = imageClassificationModel.batch_size

        loss_function = nn.CrossEntropyLoss()

        validation_dataloader =
DataLoader(imageClassificationModel.validation_dataset, batch_size =
batch_size, shuffle = False)

        imageClassificationModel.model.eval()
        running_loss = 0.0
        correct, total = 0, 0

        with torch.no_grad():
            for inputs, targets in tqdm(validation_dataloader, desc =
f"Epoch ({epoch + 1}/{epochs}) - Validation"):
                inputs = inputs.to(imageClassificationModel.device)
                targets = targets.to(imageClassificationModel.device)

                outputs = imageClassificationModel.model(inputs)
                loss = loss_function(outputs, targets)

                running_loss += loss.item() * inputs.size(0)
                _, preds = torch.max(outputs, 1)
                correct += (preds == targets).sum().item()
                total += targets.size(0)

        val_loss = running_loss / total
        val_acc = correct / total
        return val_loss, val_acc

def has_invalid(tensor):
    return torch.isnan(tensor).any() or torch.isinf(tensor).any()

class ImageClassificationModelTrainer:
    """
    이미지 분류 모델 훈련 담당
    """
    def __init__(self, validator: ImageClassificationModelValidator):
        self._validator = validator
        pass

    def train_model(self, imageClassificationModel:
ImageClassificationModel, epochs):
        # 모델 파라미터 및 하이퍼파라미터 로딩
```

```
        lr = imageClassificationModel.learning_rate
        optimizer_name = imageClassificationModel.optimizer
        batch_size = imageClassificationModel.batch_size

        # 손실 함수, Optimizer 설정
        loss_function = nn.CrossEntropyLoss()
        optimizer = {
            # "SGD": optim.SGD(imageClassificationModel.model.parameters(),
lr = lr, momentum = 0.9),
            "Adam": optim.Adam(imageClassificationModel.model.parameters(),
lr = lr)
        }[optimizer_name]

        # 데이터 로더 생성
        train_dataloader =
DataLoader(imageClassificationModel.train_dataset, batch_size = batch_size,
shuffle = True)

        # 학습
        imageClassificationModel.model.to(imageClassificationModel.device)

        running_loss = 0.0
        correct, total = 0, 0

        train_logs = []
        for epoch in range(epochs):  # 에폭 수는 필요에 따라 조정
            start_time = datetime.now()

            # 학습 모드 설정
            imageClassificationModel.model.train()

            # 학습 과정
            for inputs, targets in tqdm(train_dataloader, desc=f"Epoch
({epoch + 1}/{epochs}) - Training"):
                if has_invalid(inputs):
                    print("[경고] 입력에 NaN 또는 Inf 포함됨!")
                if has_invalid(targets):
                    print("[경고] 타겟에 NaN 또는 Inf 포함됨!")

                inputs = inputs.to(imageClassificationModel.device)
                targets = targets.to(imageClassificationModel.device)

                # 옵티마이저 초기화
                optimizer.zero_grad()

                outputs = imageClassificationModel.model(inputs)   # 순전파
                loss = loss_function(outputs, targets)      # 손실 계산
                loss.backward()                             # 역전파
```

```
                optimizer.step()                                # 최적화

                running_loss += loss.item() * inputs.size(0)
                _, preds = torch.max(outputs, 1)
                correct += (preds == targets).sum().item()
                total += targets.size(0)

            train_loss = running_loss / total
            train_accuracy = correct / total

            validation_loss, validation_accuracy =
self._validator.validate_model(imageClassificationModel, epoch, epochs)

            end_time = datetime.now()

            if torch.isnan(loss):
                print(f"[에러] Epoch {epoch + 1}: loss가 NaN입니다!")
                print("입력 샘플:", inputs[0])
                print("타겟:", targets[0])
                break

            print(f"Epoch ({epoch + 1}/{epochs}), Train Loss: {train_loss},
Train Accuracy: {train_accuracy}")
            print(f"Epoch ({epoch + 1}/{epochs}), Validation Loss:
{validation_loss}, Validation Accuracy: {validation_accuracy}")
            train_logs.append({
                "epoch": epoch + 1,
                "start_time": start_time,
                "end_time": end_time,
                "train_loss": train_loss,
                "train_acc": train_accuracy,
                "val_loss": validation_loss,
                "val_acc": validation_accuracy
            })
        print(train_logs)
        imageClassificationModel.train_logs = train_logs

class ImageClassificationModelTester:
    """
    이미지 분류 모델 평가 담당
    """
    def __init__(self):
        pass

    def test_model(self, imageClassificationModel: ImageClassificationModel,
is_validator = False):
        batch_size = imageClassificationModel.batch_size
```

```
        loss_function = nn.CrossEntropyLoss()

        dataloader = None
        if is_validator:
            dataloader =
DataLoader(imageClassificationModel.validation_dataset, batch_size =
batch_size, shuffle = False)
        else:
            dataloader = DataLoader(imageClassificationModel.test_dataset,
batch_size = batch_size, shuffle = False)

        imageClassificationModel.model.eval()
        all_preds = []
        all_targets = []
        correct, total = 0, 0

        with torch.no_grad():
            for inputs, targets in tqdm(dataloader, desc = "Testing"):
                inputs = inputs.to(imageClassificationModel.device)
                targets = targets.to(imageClassificationModel.device)

                outputs = imageClassificationModel.model(inputs)
                _, preds = torch.max(outputs, 1)
                total += targets.size(0)
                correct += (preds == targets).sum().item()

                if is_validator == False:
                    all_preds.extend(preds.cpu().numpy())
                    all_targets.extend(targets.cpu().numpy())

        imageClassificationModel.test_all_preds = all_preds
        imageClassificationModel.test_all_targets = all_targets

        accuracy = correct / total
        return accuracy

class ImageClassificationModelTuner:
    """
    이미지 분류 모델 하이퍼 파라미터 튜닝 담당
    Optuna 사용
    """
    def __init__(self, trainer, tester):
        self._trainer = trainer
        self._tester = tester
        pass

    def tune_model(self, model: ImageClassificationModel):
        def tune_by_accuracy(trial):
```

```
            # 튜닝하고 싶은 하이퍼 파라미터 나열
            lr = trial.suggest_float("learning_rate", 1e-5, 1e-1, log =
True)
            optimizer_name = trial.suggest_categorical("optimizer",
["Adam"])
            batch_size = trial.suggest_categorical("batch_size", [16, 32,
64])

            # 학습
            model.update_hyperparameters(lr, optimizer_name, batch_size)
            self._trainer.train_model(model, epochs = 3)

            # 검증
            accuracy = self._tester.test_model(model, is_validator = True)

            return accuracy

        # Optuna 스터디 생성 및 최적화 수행
        study = optuna.create_study(direction = "maximize")
        study.optimize(tune_by_accuracy, n_trials = 10)

        print("Best Trial:")
        print(f"  Accuracy: {study.best_trial.value}")
        print(f"  Params: {study.best_trial.params}")

        model.update_hyperparameters(
            study.best_trial.params["learning_rate"],
            study.best_trial.params["optimizer"],
            study.best_trial.params["batch_size"]
        )

def get_models():
    """
    테스트에 사용할 모델들을 반환하는 Generator
    """
    model_names = [
        # CNN 기반
        "vgg16",                        # [VGG16] 단순한 구조, 고전 CNN
        "resnet50",                     # [ResNet50] Skip Connection 도입
        ## "resnext50_32x4d",           # [ResNeXt] ResNet 향상형,
Cardinality 추가
        ## "efficientnet_b0",            # [EfficientNet] 파라미터/연산
최적화
        "tf_efficientnetv2_s",          # [EfficientNetV2] 최신 버전
(tensorflow 기반 구현 및 학습)
        "convnext_tiny",                # [ConvNeXt] CNN에 Transformer 요소
도입. 최신 SOTA CNN
```

```
        ## Transformer 기반
        "vit_base_patch16_224",           # [ViT] 순수 Transformer 기반 시초
모델
        ## "deit_base_patch16_224",       # [DeiT] ViT 경량화, 데이터 효율
개선
        ## "swin_tiny_patch4_window7_224", # [Swin] 계층적 구조
        "swinv2_tiny_window8_256"         # [SwinV2] 최신 버전, 더 나은 성능
    ]

    models = []

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    for model_name in model_names:
        model = ImageClassificationModel(device, model_name)
        models.append(model)

    return models

model_validator = ImageClassificationModelValidator()
model_trainer = ImageClassificationModelTrainer(model_validator)
model_tester = ImageClassificationModelTester()
model_tuner = ImageClassificationModelTuner(model_trainer, model_tester)

imageClassificationModels = get_models()

data_path = 'flowers/train'
dataPreprocessor = DataPreprocessor(data_path)

for imageClassificationModel in imageClassificationModels:
    # 1. 데이터 전처리
    train_dataset, val_dataset, test_dataset =
dataPreprocessor.split(imageClassificationModel)
    imageClassificationModel.update_dataset(train_dataset, val_dataset,
test_dataset)

    # 2. 모델 로딩
    imageClassificationModel.load()

    # 3. 하이퍼 파라미터 튜닝
    model_tuner.tune_model(imageClassificationModel)

    # 4. 모델 학습
    model_trainer.train_model(imageClassificationModel, epochs = 10)

    # 5. 모델 평가 (테스트)
    model_tester.test_model(imageClassificationModel)
```

```
    # 6. 모델 저장
    imageClassificationModel.save()

    # 7. 모델 언로드 (메모리 확보)
    imageClassificationModel.unload()

import matplotlib.pyplot as plt

def display_training_results(model):
    """
    train_logs가 list[dict] 형태일 때, Loss/Accuracy를 시각화
    - Loss는 자동 Y축
    - Accuracy는 Y축을 0.0~1.0으로 고정
    """

    logs = model.train_logs

    # 리스트에서 값 추출
    epochs = [log['epoch'] for log in logs]
    train_loss = [log['train_loss'] for log in logs]
    val_loss = [log['val_loss'] for log in logs]
    train_acc = [log['train_acc'] for log in logs]
    val_acc = [log['val_acc'] for log in logs]

    # 시각화
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Loss (자동 스케일)
    axes[0].plot(epochs, train_loss, label='Train Loss', marker='o')
    axes[0].plot(epochs, val_loss, label='Validation Loss', marker='x')
    axes[0].set_title('Loss per Epoch')
    axes[0].set_xlabel('Epoch')
    axes[0].set_ylabel('Loss')
    axes[0].legend()
    axes[0].grid(True)

    # Accuracy (0.0 ~ 1.0 고정)
    axes[1].plot(epochs, train_acc, label='Train Accuracy', marker='o')
    axes[1].plot(epochs, val_acc, label='Validation Accuracy', marker='x')
    axes[1].set_title('Accuracy per Epoch')
    axes[1].set_xlabel('Epoch')
    axes[1].set_ylabel('Accuracy')
    axes[1].set_ylim([0.0, 1.0])
    axes[1].legend()
    axes[1].grid(True)

    plt.suptitle(f"[{model.model_name}] Training Results", fontsize=16)
    plt.tight_layout()
```

```
    plt.show()

for model in imageClassificationModels:
    display_training_results(model)

from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score, precision_score, recall_score, f1_score
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

def visualize_model_performance(model: ImageClassificationModel,
class_names=None):
    """
    Confusion Matrix + Classification Report 시각화
    - model.test_all_preds, model.test_all_targets 기반
    """
    preds = model.test_all_preds
    targets = model.test_all_targets

    if not preds or not targets:
        print(f"[{model.model_name}] 평가 지표 시각화를 위한 데이터가
없습니다.")
        return

    # Confusion Matrix
    cm = confusion_matrix(targets, preds)
    num_classes = model.num_classes

    if class_names is None:
        class_names = [f"Class {i}" for i in range(num_classes)]

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=class_names, yticklabels=class_names)
    plt.title(f"[{model.model_name}] Confusion Matrix")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.tight_layout()
    plt.show()

    # Accuracy, Precision, Recall, F1
    acc = accuracy_score(targets, preds)
    precision = precision_score(targets, preds, average='weighted',
zero_division=0)
    recall = recall_score(targets, preds, average='weighted',
zero_division=0)
    f1 = f1_score(targets, preds, average='weighted', zero_division=0)
```

```
    print(f"\n📊 [{model.model_name}] Performance Metrics")
    print(f"Accuracy : {acc:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall   : {recall:.4f}")
    print(f"F1 Score : {f1:.4f}")

    # Classification Report
    print("\n📋 Classification Report")
    print(classification_report(targets, preds, target_names=class_names,
zero_division=0))

class_names = ["daisy", "dandelion", "rose", "sunflower", "tulip"]
for model in imageClassificationModels:
    visualize_model_performance(model, class_names=class_names)

# 소스코드 끝.
```