Orange Disease Classification

오랜지 다중 클래스 데이터 세트를 통해 오랜지 상태 분류

Study Period **10 ~ 25, March ,2025**

Researcher **Shim, Gyuwon**

---

1. 개요

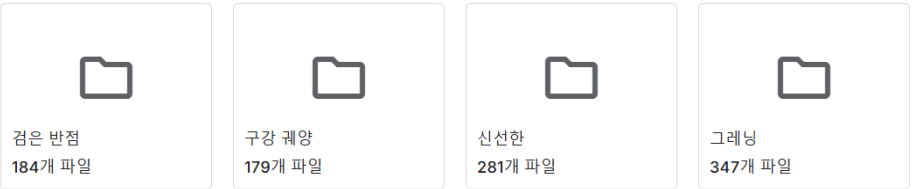
1. **연구의 목적**
    a. 신선한 오랜지, 질병이 있는 오랜지를 분류하는 최적 AI 모델, 알고리즘 구현
    b. 오랜지의 다양한 상태 분류에 적용 할 수 있는 AI 모델 구현
    c. 연구를 통해 이미지 분류 AI 모델과 튜닝에 대한 특성 파악
2. **문제의 정의**
    a. 오랜지의 정상적인 상태와 질병인 상태를 분별할 수 있는지
    b. 오랜지가 비정상일 경우 어떤 질병인지 구분이 정확한지
    c. 오랜지 이미지 데이터셋의 적은 데이터 보완 가능 여부 (약 1000K, 1%의 일반적인 양)
    d. 오랜지 상태를 분석하기 위한 적합한 AI 모델
    e. 오랜지 상태의 AI 모델의 적절한 파인튜닝 방식과 적용되는 최적 값 연구

c. **연구의 활용**
    a. 신선한 오랜지와 질병 있는 오랜지를 정확하게 분류하는 프로세스에 적용
    b. 부적합한 오랜지를 파악 및 제거를 통해 오랜지 생산성 개선
    c. 오랜지 뿐 아니라 과일의 상태를 정확하게 파악하는 AI 모델 구현
    d. 이미지 상태를 효과적으로 파악하는데 적합한 AI 모델 튜닝 방법, 값 개발

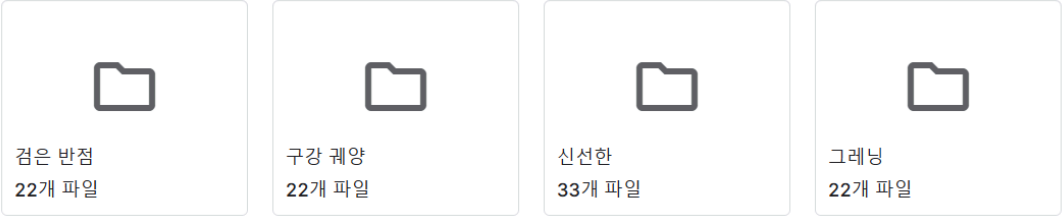
2. 오랜지 데이터 셋 분석

https://www.kaggle.com/datasets/jonathansilva2020/orange-diseases-dataset

- 오렌지의 질병 분류를 수행하기 위한 머신 러닝 및 딥 러닝 알고리즘을 구축하기 위해 큐레이션 되어있음
- 데이터 셋에는 신선한 오렌지가 있는 클래스와 감귤 암종, 검은 반점 및 감귤 녹화의 세 가지 다른 질병 클래스가 포함됨
- 훈련 (학습) 이미지 991개 4개 클래스

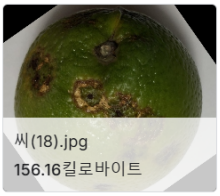


- 검증 (테스트) 이미지 99개 4개 클래스



- 클래스별 예시

검은 반점 예시 이미지



구강 궤양 예시 이미지

신선한 이미지

그레인 이미지

3. 적용 AI 모델

3-1. EfficientNetV2S Info

- EfficientNetV2는 Google Brain에서 개발한 효율적인 이미지 분류(이미지넷 등) 모델로, 이전 버전인 EfficientNet을 개선한 버전
- EfficientNetV2는 더 빠르고, 더 작은 모델이면서도 더 높은 정확도를 제공
- Fused MBConv 적용: 초반 레이어에서 MBConv 대신 표준 합성곱을 사용하여 연산 속도 향상
- Gradual Learning 적용: 작은 해상도의 이미지를 먼저 학습하고 점진적으로 해상도를 증가시켜 학습 효율 증가
- 더 작은 모델 크기: 동일한 성능을 유지하면서도 모델 크기를 줄임
- 연산 속도 최적화: 메모리 사용량을 줄이고 GPU 연산 속도를 증가
- EfficientNetV2S 구조

> 참고자료 1 https://arxiv.org/abs/2104.00298
> 참고자료 2. https://da2so.tistory.com/45
> 참고자료 3. https://audrb1999.tistory.com/7

3-2. MBConv vs. Fused MBConv 비교

항목	MBConv	Fused MBConv
채널 확장 방식	1×1 Conv	**3×3 또는 5x5 Conv**
구조	1×1 Expand → DWConv → 1×1 Project	**3×3 Conv (fused)** → 1×1 Project
공간 정보 추출	Depthwise Conv	일반 Conv에서 함께 수행
병렬화 성능	떨어짐 (Depthwise가 병렬화 어려움)	**하드웨어 친화적 (GPU/TPU에 적합)**
연산량	작음	약간 많음
속도	실제로 느릴 수 있음	**실제로 빠름**
적용 모델	EfficientNet, MobileNetV2 등	**EfficientNetV2**

- Depthwise Conv는 이론적으로 연산량이 적지만, 실제 하드웨어에서는 메모리 접근이 많아서 느림
- Fused MBConv는 연산량은 조금 늘어나지만, GPU와 TPU에서 병렬처리가 잘 돼서 실제로는 훨씬 빠름.

- EfficientNetV2의 shallow layer(초기 층)에서는 Fused MBConv를 사용하고, 깊은 층에서는 MBConv를 사용해 속도 + 정확도를 모두 잡는 구조

3-3. efficientNetV2 모델 종류

- EfficientNetV2는 모델 크기에 따라 S, M, L 3가지 버전이 존재
- EfficientNetV2-S: 작은 모델, 모바일이나 Edge 기기용
- EfficientNetV2-M: 중간 크기, 일반적인 GPU 환경에서 사용
- EfficientNetV2-L: 가장 큰 모델, 고성능 서버에서 최적화됨

3-4. EfficientNetV2 활용 분야

- 이미지 분류(Classification): ImageNet과 같은 데이터셋에서 높은 정확도 제공
- 객체 탐지(Object Detection): Faster R-CNN, YOLO 등에 백본(Backbone)으로 사용 가능
- 세분화(Segmentation): 의료 영상 분석, 자율 주행, 위성 영상 처리 등
- 모바일 및 Edge 디바이스: 경량화된 모델을 활용하여 저사양 환경에서도 빠른 추론 가능

3-5. Vision Transformer

- ViT는 트랜스포머(Transformer) 구조를 이미지 처리에 적용한 모델로, 2020년에 Google Research에서 발표
- NLP(자연어 처리)에서 쓰던 구조를 CNN 없이 이미지 분류에 적용

참고 자료

https://www.ibm.com/kr-ko/think/topics/transformer-model

https://www.hpe.com/kr/ko/what-is/transformer-model.html

https://ko.wikipedia.org/wiki/트랜스포머_(기계_학습)

3-6. Vision Transformer 구조

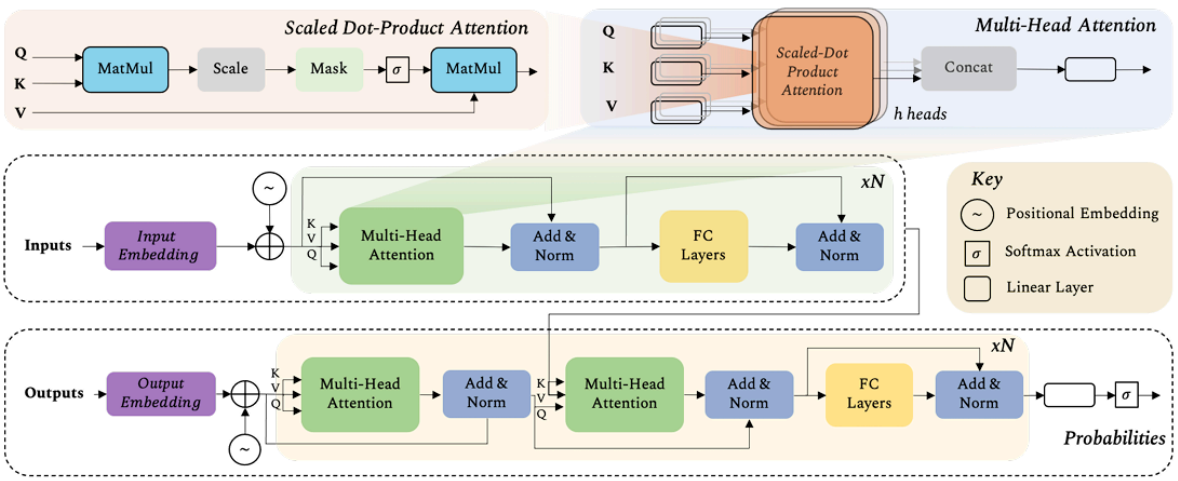


Fig. 2. Architecture of the Transformer Model [1].

- 비전 트랜스포머(Vision Transformer, ViT)는 자연어 처리용 트랜스포머 구조를 컴퓨터 비전에 적용한 모델로, 이미지를 패치 단위로 분할해 순차적 입력으로 처리
- 합성곱 신경망(CNN)과 달리 셀프 어텐션을 통해 이미지의 **글로벌 컨텍스트**를 포착

1. **이미지 패치 분할(Image Patching)**
   - 입력 이미지(예: 32x32×3)를 고정된 크기의 패치(예: 16x16)로 분할
   - 32×32 이미지를 16x16 패치로 나누면 4개(2x2)의 패치가 생성
   - 각 패치를 1차원 벡터로 변환 (예: 16x16×3 → 768차원 벡터)
2. **패치 임베딩(Patch Embedding)**
   - 선형 투영(Linear Projection): 패치 벡터를 저차원 임베딩 공간으로 매핑
   - 예: 4x768 → 4×3 (hidden dimension=3으로 가정)
   - 레이어 정규화(LayerNorm) 적용 후 최종 임베딩 벡터 생성
3. **위치 임베딩(Positional Embedding)**
   - 패치의 공간적 위치 정보를 추가하기 위해 학습 가능한 1D 벡터를 사용
   - CLS 토큰 : 분류任务를 위해 패치 시퀀스 시작에 추가되는 학습 가능한 벡터1
   - 최종 임베딩: `[CLS] + 패치 임베딩 + 위치 임베딩`
4. **트랜스포머 인코더(Transformer Encoder)**
   - 인코더는 **멀티헤드 셀프 어텐션(MSA)**과 **MLP 블록**으로 구성
5. **멀티헤드 셀프 어텐션(MSA)**
   - 입력을 여러 헤드로 분할해 병렬 어텐션 계산:

     ```python
     # 예: 3개 헤드, 헤드 차원=4
     Query, Key, Value = split_heads(input)
     # 3×5×4
     attention_output = concat(attention_heads)
     # 5×12
     ```
   - 어텐션 결과를 결합하고 선형 변환해 원래 차원 복원
6. **MLP 블록**
   - GELU 활성화 함수와 드롭아웃을 포함한 2층 신경망
   - 예: 5x3 → 5×4 (확장) → 5×3 (축소)
   - 잔차 연결(Residual Connection) 적용

3-7. 오랜지 분류 이미지를 사용하여 AI 모델 추론 정확도 예상

모델	정확도(예상)	속도 (추론)	모델 크기	모바일 사용 가능 여부
EfficientNetV2	**최고 (85~88%)**	빠름	작음	가능
ResNet50	좋음 (80~85%)	중간	중간	가능
ViT (Vision Transformer)	매우 높음 (85~90%)	느림	큼	어려움

4. 모델 적용 실험

- GPU의 한정 자원을 사용하며, 가능한 여러번, 다수의 파라미터를 적용해 보기 위해 경량한 모델인 EfficientNetV2S 모델 중심으로 테스트
- 데이터 증강, 사전 학습, Fine Tunning 단계 적용
- EfficienceNetV2S 목표 : 83% 검증 정확도
- ViTransformer 목표 : 95% 검증 정확도

4-1. 모델 적용 결과 (중간 리포팅 1 ~ 10th)

실험 순번	데이터 증강 1	데이터 증강 2	적용 모델	사전 학습 epoch	튜닝 epoch	정확도	손실
1th	리스케일, 회전, 넓이높이 이동, 좌우 반전, 줌 조정		**EfficientNetV2S**	8	0	0.9588	0.1736
2th	리스케일, 회전, 넓이높이 이동, 좌우 반전, 줌 조정	밝기 조절, 기울이기 적용, 채널 색상 변경	**EfficientNetV2S**	15	10	0.7706	0.6252
3, 4th	리스케일, 회전, 넓이높이 이동, 좌우 반전, 줌 조정	밝기 조절, 기울이기 적용, 채널 색상 변경, 노이즈 추가, 믹스업	**EfficientNetV2S**	10	10	0.6478	1.0158
5th	리스케일, 회전, 넓이높이 이동, 좌우 반전, 줌 조정	밝기 조절, 기울이기 적용, 채널 색상 변경, 배치 사이즈 증대	**EfficientNetV2S**	15	10	0.7706	0.6252
6th	리스케일, 회전, 넓이높이 이동, 좌우 반전, 줌 조정	밝기 조절, 기울이기 적용, 채널 색상 변경	**EfficientNetV2S**	30	10	0.6989	0.7173
7th	리스케일, 회전, 넓이높이 이동, 좌우 반전, 줌 조정	밝기 조절, 기울이기 적용, 채널 색상 변경, 노이즈 추가, 믹스업	**EfficientNetV2S**	32	10	0.6	0.78
8th	리스케일, 회전, 넓이높이 이동, 좌우 반전, 줌 조정	밝기 조절, 기울이기 적용, 채널 색상 변경	**EfficientNetV2S**	35	15	0.8365	0.3724
9th	리사이즈	-	**ViT (Vision Transformer)**	3	0		0.168718
10th	리사이즈, 수평회전, 랜덤회전, 컬러지터,	-	**ViT (Vision Transformer)**	5	0		0.19080

4-2. 추가 실험 결과 (13 ~ 14th)

- 1th의 데이터 증강 수준으로 줄이고 epoch 수, drop out 적용으로 안정화 적용
- 11 ~ 12th NoSql 연결 실험으로 결과 정보에서는 사용하지 않음

실험 순번	데이터 증강 1	데이터 증강 2	적용 모델	사전 학습 epoch	튜닝 epoch	정확도	손실
13th	리스케일, 회전, 넓이높이 이동, 좌우 반전, 줌 조정 (1th와 동일)		**EfficientNetV2S**	20	8	0.9475	0.1367
14th	리스케일, 회전, 넓이높이 이동, 좌우 반전, 줌 조정 (1th와 동일)		**EfficientNetV2S**	20	8	0.8479	0.2600

4-2. 실험 상세 노트

1th EXP

https://colab.research.google.com/drive/1gqW6-qQX1MHBh6pCnkwsQFAkjMzXCbQo?usp=sharing

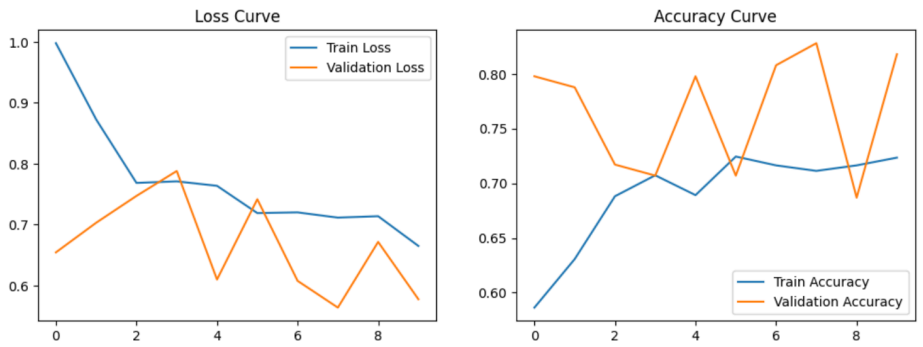
- 데이터 증강

```python
# 데이터 증강 적용
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.2
)
```

- EfficientNetV2S 적용값

```python
# EfficientNetV2 모델 불러오기 (Pretrained weights 사용)
base_model = EfficientNetV2S(weights="imagenet", include_top=False, input_shape=(256, 256, 3))

# 기존 가중치 동결 (전이 학습)
base_model.trainable = False

# 분류기 추가
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(256, activation="relu")(x)
x = Dropout(0.3)(x)
x = Dense(len(classes), activation="softmax")(x)  # 클래스 개수만큼 출력 노드 설정
```

- 결과



- 9ms/step - accuracy: 0.9588 - loss: 0.1736 | Validation Accuracy: 0.9596
- Fine-Tunning 없이 사전 학습 진행
- 현재 모델은 학습이 진행되면서 훈련 데이터에는 적응하고 있지만, 검증 데이터에서는 변동성이 크며 일반화가 어렵다는 문제가 있음.
- Validation 데이터의 변동이 큰 이유를 확인하는 것도 중요 → 데이터셋의 크기, 불균형
- 과적합 가능성이 있으므로 모델 일반화를 위해 추가 조치가 필요

2nd EXP

https://colab.research.google.com/drive/139fg9h_92hhIH1MGkjUVX0m1ATUCtB49?usp=sharing

- 데이터 증강 추가

```python
# 강력한 데이터 증강 적용
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    rotation_range=45,  # 회전 범위 증가
    width_shift_range=0.2,  # 가로 이동
    height_shift_range=0.2,  # 세로 이동
    horizontal_flip=True,  # 좌우 반전
    brightness_range=[0.5, 1.5],  # 밝기 조절
    zoom_range=0.3,  # 줌 조정
    shear_range=0.2,  # 기울이기 적용
    channel_shift_range=50.0  # 채널 색상 변형
)
```

- 모델튜닝 :전이 학습 후 일부 레이어를 Fine-Tuning Dropout 적용하여 과적합 방지

```python
# EfficientNetV2 모델 불러오기 (Pretrained weights 사용)
base_model = EfficientNetV2S(weights="imagenet", include_top=False, input_shape=(256, 256, 3))

# 기존 가중치 동결
base_model.trainable = False

# 분류기 추가
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(512, activation="relu")(x)  # 더 큰 히든 레이어 추가
x = Dropout(0.4)(x)  # Dropout 증가
x = Dense(256, activation="relu")(x)
x = Dropout(0.3)(x)
x = Dense(len(classes), activation="softmax")(x)  # 클래스 개수만큼 출력 노드 설정

# 모델 생성
model = Model(inputs=base_model.input, outputs=x)

# 모델 컴파일 (적응형 학습률 사용)
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

- 추가 학습 : 학습률 1e-5

```python
# EfficientNetV2의 상위 30% 레이어를 학습 가능하게 변경
for layer in base_model.layers[-int(len(base_model.layers) * 0.3):]:
    layer.trainable = True

# 모델 재컴파일 (학습률 감소)
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

# 추가 학습 (Fine-Tuning)
EPOCHS_FINE_TUNE = 10  # 추가 학습 10 Epoch
```

결과

- accuracy: 0.7706 - loss: 0.6252 / 최종 Validation Accuracy: 0.7677
- F-ine-Tuning이 모델의 성능을 향상
  - Validation Accuracy가 더 높고 안정적
  - Loss가 더 낮아지고 일반화가 잘됨
  - 정확도가 80% 밑이라 더 개선 필요

3, 4th EXP

https://colab.research.google.com/drive/1rxFzeatvEAwvSqf0s-zCe2OHDXK5a0S9?usp=sharing

- 더 강력한 데이터 증강
  - 기본 변형: 회전, 이동, 반전, 확대
  - 추가 변형: 밝기 조절, 대비 조정,
  - 노이즈 추가 고급 기법 : CutMix & MixUp 적용 (데이터 혼합 기법)

```
# 강력한 데이터 증강 적용
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    rotation_range=45,  # 회전 범위 증가
    width_shift_range=0.2,  # 가로 이동
    height_shift_range=0.2,  # 세로 이동
    horizontal_flip=True,  # 좌우 반전
    brightness_range=[0.5, 1.5],  # 밝기 조절
    zoom_range=0.3,  # 줌 조정
    shear_range=0.2,  # 기울이기 적용
    channel_shift_range=50.0,  # 채널 색상 변형
    preprocessing_function=lambda img: cv2.GaussianBlur(img, (5, 5), 0)  # 노이즈 추가
)
```

```
import tensorflow.keras.backend as K

def mixup(batch_x, batch_y, alpha=0.2):
    """MixUp 적용"""
    batch_size = batch_x.shape[0]
    l = np.random.beta(alpha, alpha, batch_size)
    X_l = l.reshape(batch_size, 1, 1, 1)
    Y_l = l.reshape(batch_size, 1)

    index = np.random.permutation(batch_size)
    mixed_x = batch_x * X_l + batch_x[index] * (1 - X_l)
    mixed_y = batch_y * Y_l + batch_y[index] * (1 - Y_l)

    return mixed_x, mixed_y
```

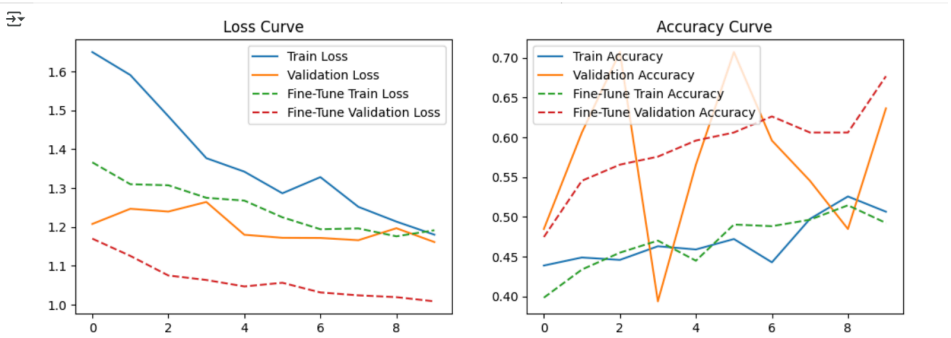
- 모델의 Drop Out 증대

```
from tensorflow.keras.applications import EfficientNetV2S
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, BatchNormalization

# EfficientNetV2 모델 불러오기 (Pretrained weights 사용)
base_model = EfficientNetV2S(weights="imagenet", include_top=False, input_shape=(256, 256, 3))

# 기존 가중치 동결
base_model.trainable = False

# 분류기 추가
x = GlobalAveragePooling2D()(base_model.output)
x = BatchNormalization()(x)  # Batch Normalization 적용
x = Dense(512, activation="relu")(x)
x = Dropout(0.5)(x)  # Dropout 증가 (과적합 방지)
x = Dense(256, activation="relu")(x)
x = Dropout(0.4)(x)
x = Dense(len(classes), activation="softmax")(x)  # 클래스 개수만큼 출력 노드 설정

# 모델 생성
model = Model(inputs=base_model.input, outputs=x)

# 모델 컴파일 (적응형 학습률 사용)
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

- 추가 학습 epoch 증대 5 → 10

```
# EfficientNetV2의 상위 40% 레이어를 학습 가능하게 변경
for layer in base_model.layers[-int(len(base_model.layers) * 0.4):]:
    layer.trainable = True

# 모델 재컴파일 (학습률 감소)
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

# 추가 학습 (Fine-Tuning)
EPOCHS_FINE_TUNE = 10  # 추가 학습 10 Epoch

history_finetune = model.fit(
    train_data,
    validation_data=val_data,
    epochs=EPOCHS_FINE_TUNE,
    verbose=1
)
```

- 결과



- accuracy: 0.6478 - loss: 1.0158 | 최종 Validation Accuracy: 0.6768
- cutmix/mixup 적용했으나 낮은 효과
- Fine-Tuning을 통해 모델의 성능을 향상
- Loss가 더 낮아지고 일반화 성능 향상
- 검증 정확도가 낮음

5th EXP

https://colab.research.google.com/drive/1xVuUjOeXTNzAOcb-ty5VaG_6654Gv3-v?usp=sharing

- 경량화를 위해 데이터 증강 감소, **배치 사이즈 증대**

```
# 이미지 크기 및 배치 설정
IMG_SIZE = (256, 256)
BATCH_SIZE = 64

# 강력한 데이터 증강 적용
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    rotation_range=45,  # 회전 범위 증가
    width_shift_range=0.2,  # 가로 이동
    height_shift_range=0.2,  # 세로 이동
    horizontal_flip=True,  # 좌우 반전
    brightness_range=[0.5, 1.5],  # 밝기 조절
    zoom_range=0.3,  # 줌 조정
    shear_range=0.2,  # 기울이기 적용
    channel_shift_range=50.0  # 채널 색상 변형
)

val_datagen = ImageDataGenerator(rescale=1.0 / 255.0)
```

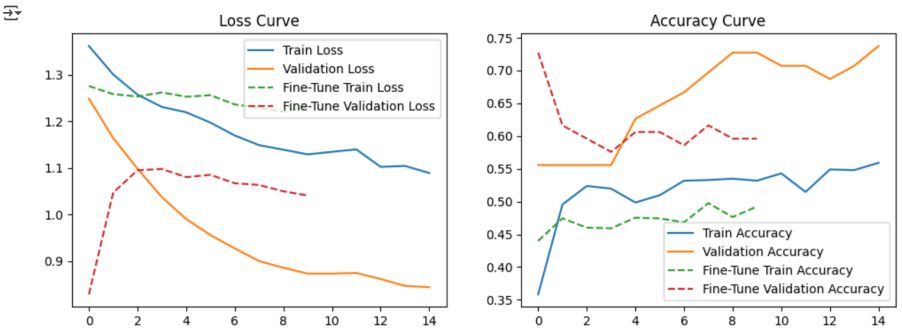
```python
from tensorflow.keras.applications import EfficientNetV2S
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout

# EfficientNetV2 모델 불러오기 (Pretrained weights 사용)
base_model = EfficientNetV2S(weights="imagenet", include_top=False, input_shape=(256, 256, 3))

# 기존 가중치 동결
base_model.trainable = False

# 분류기 추가
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(512, activation="relu")(x)  # 더 큰 히든 레이어 추가
x = Dropout(0.5)(x)  # Dropout 증가
x = Dense(256, activation="relu")(x)
x = Dropout(0.4)(x)
x = Dense(len(classes), activation="softmax")(x)  # 클래스 개수만큼 출력 노드 설정

# 모델 생성
model = Model(inputs=base_model.input, outputs=x)

# 모델 컴파일 (적응형 학습률 사용)
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.00005),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

- 기본 학습 epoch 증대

```python
EPOCHS = 15  # 기본 전이 학습

history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=EPOCHS,
    verbose=1
)
```

```python
# EfficientNetV2의 상위 30% 레이어를 학습 가능하게 변경
for layer in base_model.layers[-int(len(base_model.layers) * 0.4):]:
    layer.trainable = True

# 모델 재컴파일 (학습률 감소)
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-6),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

# 추가 학습 (Fine-Tuning)
EPOCHS_FINE_TUNE = 10  # 추가 학습 10 Epoch
```

- 결과



- accuracy: 0.7706 - loss: 0.6252 | 최종 Validation Accuracy: 0.7677
- Fine-Tuning이 Loss 감소 및 Accuracy 향상 효과적
  - Validation Loss가 더 낮음 → 일반화 성능 향상
  - Validation Accuracy가 더 높음
- Baseline 모델은 과소적합(Underfitting) 가능성
  - Train Accuracy가 낮고 Validation Accuracy와 차이가 큼 → 모델이 충분히 학습되지 않았을 가능성

- Fine-Tuning 초반 변동성이 크지만 이후 안정화됨

6th EXP.

https://colab.research.google.com/drive/1iVIX4TUJKBTQoRLEk5byi2AgzLSEWFqf?usp=sharing

- 데이터 증강은 이전값 유지

```
# 이미지 크기 및 배치 설정
IMG_SIZE = (256, 256)
BATCH_SIZE = 32

# 강력한 데이터 증강 적용
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    rotation_range=45,  # 회전 범위 증가
    width_shift_range=0.2,  # 가로 이동
    height_shift_range=0.2,  # 세로 이동
    horizontal_flip=True,  # 좌우 반전
    brightness_range=[0.5, 1.5],  # 밝기 조절
    zoom_range=0.3,  # 줌 조정
    shear_range=0.2,  # 기울이기 적용
    channel_shift_range=50.0  # 채널 색상 변형
)

val_datagen = ImageDataGenerator(rescale=1.0 / 255.0)
```

- Drop Out 줄임

```
from tensorflow.keras.applications import EfficientNetV2S
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout

# EfficientNetV2 모델 불러오기 (Pretrained weights 사용)
base_model = EfficientNetV2S(weights="imagenet", include_top=False, input_shape=(256, 256, 3))

# 기존 가중치 동결
base_model.trainable = False

# 분류기 추가
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(512, activation="relu")(x)  # 더 큰 히든 레이어 추가
x = Dropout(0.4)(x)  # Dropout 증가
x = Dense(256, activation="relu")(x)
x = Dropout(0.3)(x)
x = Dense(len(classes), activation="softmax")(x)  # 클래스 개수만큼 출력 노드 설정

# 모델 생성
model = Model(inputs=base_model.input, outputs=x)
```

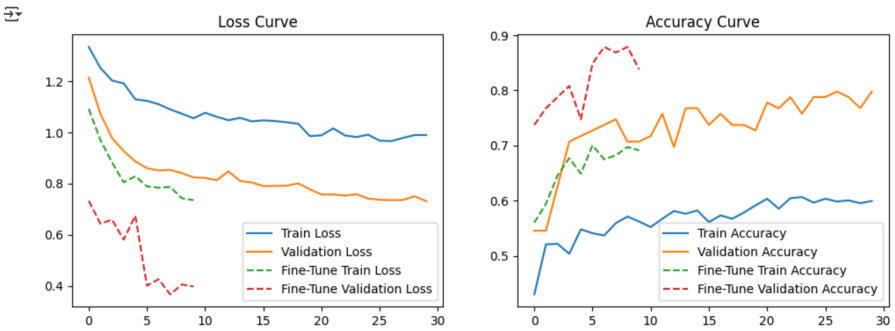
- 사전 학습 수 증대

```
[ ] EPOCHS = 30  # 기본 전이 학습

    history = model.fit(
        train_data,
        validation_data=val_data,
        epochs=EPOCHS,
        verbose=1
    )
```

- 최적화를 위해 학습률값 증대

```python
# EfficientNetV2의 상위 30% 레이어를 학습 가능하게 변경
for layer in base_model.layers[-int(len(base_model.layers) * 0.3):]:
    layer.trainable = True

# 모델 재컴파일 (학습률 감소)
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

# 추가 학습 (Fine-Tuning)
EPOCHS_FINE_TUNE = 10  # 추가 학습 10 Epoch
```

결과



- accuracy: 0.6989 - loss: 0.7173 - val_accuracy: 0.8384 - val_loss: 0.3978
- Fine-Tuning을 통해 모델 성능이 향상됨
- **Validation Accuracy가 83% 이상 도달 → 큰 성능 향상**
- Loss가 더 낮아지고 일반화 성능 향상

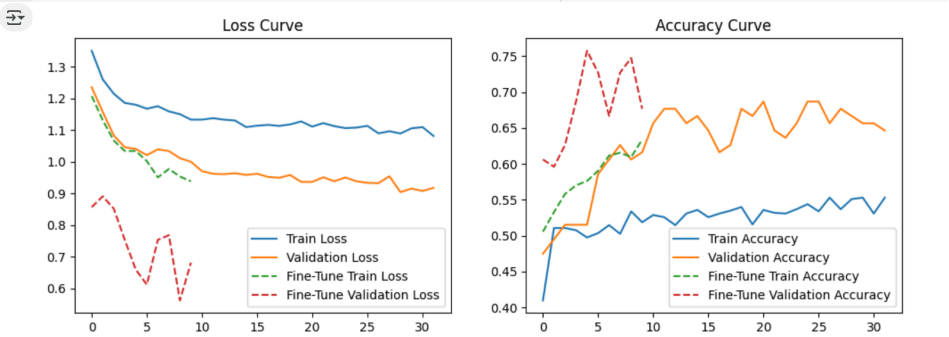
7th EXP

https://colab.research.google.com/drive/1dulbmcxS6Ydk4SAJihTwXlWwXBJY_XdP#scrollTo=kcY_tPcl3ZR7

- 노이즈 추가 고급 기법: CutMix & MixUp 적용 (데이터 혼합 기법)

```python
# 강력한 데이터 증강 적용
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    rotation_range=45,  # 회전 범위 증가
    width_shift_range=0.2,  # 가로 이동
    height_shift_range=0.2,  # 세로 이동
    horizontal_flip=True,  # 좌우 반전
    brightness_range=[0.5, 1.5],  # 밝기 조절
    zoom_range=0.3,  # 줌 조정
    shear_range=0.2,  # 기울이기 적용
    channel_shift_range=50.0,  # 채널 색상 변형
    preprocessing_function=lambda img: cv2.GaussianBlur(img, (5, 5), 0)  # 노이즈 추가
)
```

```python
import tensorflow.keras.backend as K

def mixup(batch_x, batch_y, alpha=0.2):
    """MixUp 적용"""
    batch_size = batch_x.shape[0]
    l = np.random.beta(alpha, alpha, batch_size)
    X_l = l.reshape(batch_size, 1, 1, 1)
    Y_l = l.reshape(batch_size, 1)

    index = np.random.permutation(batch_size)
    mixed_x = batch_x * X_l + batch_x[index] * (1 - X_l)
    mixed_y = batch_y * Y_l + batch_y[index] * (1 - Y_l)

    return mixed_x, mixed_y
```



- 6번 Exp에 3번 Exp에 적용했던 데이터 증강(CutMix & MixUp)을 적용 해 보았으나 성능은 더 낮아짐.
- Train Accuracy가 낮고 증가하지 않음 → 학습이 충분히 되지 않았거나, 모델이 너무 단순할 가능성.
- 정확도도 오히려 낮아짐 → 데이터 증강 및 변형이 학습이 되는 부분을 저해시킴

8th EXP.

https://colab.research.google.com/drive/1Nv4X75G4B8tiYa2MlNuAeMTVQ0BdBfm_?usp=sharing

- 6차 실험에 사전 학습 epoch 6번과 사후 튜닝 epoch 4번만 증가
- 결과

- accuracy: 0.8365 - loss: 0.3724 | 최종 Validation Accuracy: 0.8283
- Fine-Tuning을 통해 모델 성능이 확실히 향상됨
- Validation Accuracy가 83% 이상

9th. ViT (Vision transformer) 모델로 테스트 (not CNN)

https://colab.research.google.com/drive/1rW0JKqB8FT0eL7q_XJj0D6QaWNUfItB-?usp=sharing

- 이미지 전처리

```
[20] # 이미지 전처리 및 Feature Extractor 설정
     feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-base-patch16-224-in21k')

     transform = transforms.Compose([
         transforms.Resize((224, 224)),
         transforms.ToTensor(),
         transforms.Normalize(mean=feature_extractor.image_mean, std=feature_extractor.image_std)
     ])

     # 데이터 로딩
     train_dataset = ImageFolder(train_dir, transform=transform)
     test_dataset = ImageFolder(test_dir, transform=transform)

     # 데이터셋 클래스 재정의
     class ViTDataset(torch.utils.data.Dataset):
         def __init__(self, dataset):
             self.dataset = dataset

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

         def __getitem__(self, idx):
             image, label = self.dataset[idx]
             return {'pixel_values': image, 'labels': label}

     train_ds = ViTDataset(train_dataset)
     test_ds = ViTDataset(test_dataset)
```

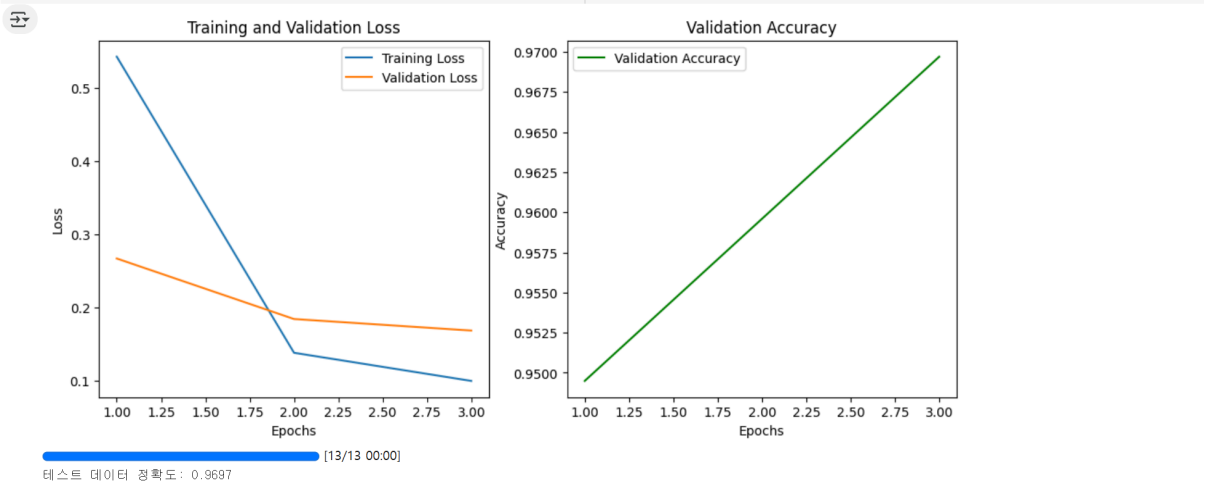
- **ViTransformer 모델 적용**

```
# 모델 설정
model = ViTForImageClassification.from_pretrained(
    'google/vit-base-patch16-224-in21k',
    num_labels=len(train_dataset.classes),
    id2label={str(i): label for i, label in enumerate(train_dataset.classes)},
    label2id={label: str(i) for i, label in enumerate(train_dataset.classes)}
)

# 학습 인자 설정
training_args = TrainingArguments(
    output_dir='./vit_orange_results',
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    evaluation_strategy="epoch",
    logging_strategy='epoch',
    save_strategy='epoch',
    load_best_model_at_end=True,
    metric_for_best_model='accuracy',
    logging_dir='./logs',
    learning_rate=2e-5,
)

# 평가 함수 정의
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    acc = accuracy_score(labels, predictions)
    return {'accuracy': acc}

# Trainer 설정
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_ds,
    eval_dataset=test_ds,
    compute_metrics=compute_metrics,
)
```

```
Some weights of ViTForImageClassification were not initialized from the model checkpoint at google/vit-base-patch16-224-in21k and are newly initialized: ['classifier.bias', 'c
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1594: FutureWarning: `evaluation_strategy` is deprecated and will be removed in version 4.46 of 🤗 Trans
  warnings.warn(
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a different run name by setting th
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter: · · · · · · · · · · ·
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: gwshim (gwshim-ktb) to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
Tracking run with wandb version 0.19.8
Run data is saved locally in /content/wandb/run-20250321_063559-xk2ezpfl
Syncing run ./vit_orange_results to Weights & Biases (docs)
View project at https://wandb.ai/gwshim-ktb/huggingface
View run at https://wandb.ai/gwshim-ktb/huggingface/runs/xk2ezpfl
[372/372 00:58, Epoch 3/3]
```

Epoch	Training Loss	Validation Loss	Accuracy
1	0.542400	0.267018	0.949495
2	0.138400	0.184452	0.959596
3	0.100000	0.168718	0.969697

- 결과



- e-poch 3 : **정확도 0.9697**

10th Vit tunning

https://colab.research.google.com/drive/1WKB5K6j5d9n42Wi6AhsC6a5LLaFaaBW1?usp=sharing

- 데이터 전처리 및 증강 추가

```python
# 데이터 전처리 및 증강
feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-base-patch16-224-in21k')

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=feature_extractor.image_mean, std=feature_extractor.image_std)
])
```

- 모델 하이퍼 파라미터 튜닝, epoch 증대, 배치 사이즈 벼경

```python
# 학습 인자 설정 >> epoch 추가, 배치사이즈 변경, 학습률 변경, Early stopping 추가
training_args = TrainingArguments(
    output_dir='./vit_orange_finetune',
    num_train_epochs=8,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    evaluation_strategy="epoch",
    save_strategy='epoch',
    load_best_model_at_end=True,
    metric_for_best_model='accuracy',
    learning_rate=1e-5,
    logging_dir='./logs'
)
```

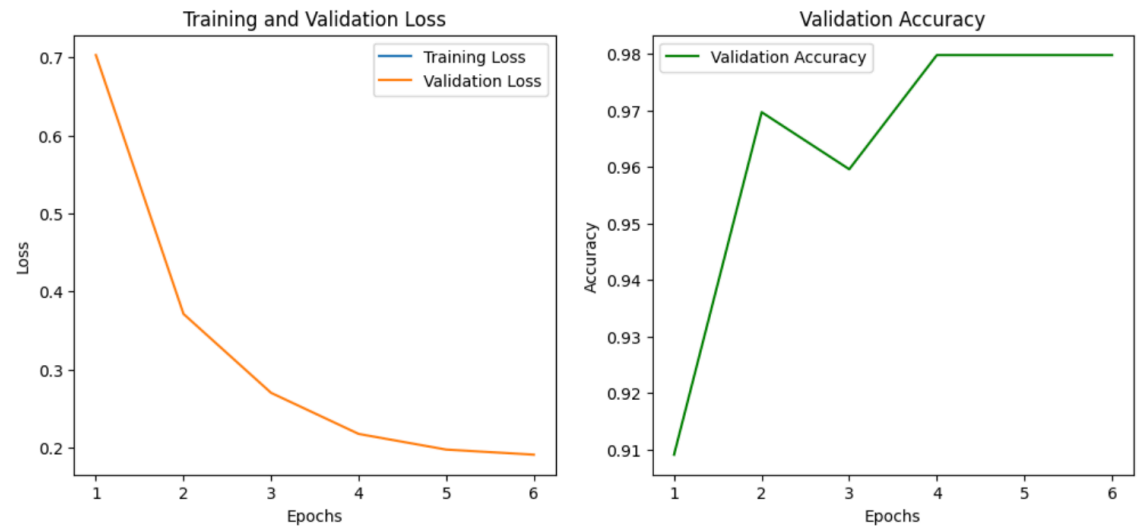
- 얼리 스탑 코드 추가

```python
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_ds,
    eval_dataset=test_ds,
    compute_metrics=compute_metrics,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=2)],
)
```

- 결과

```
wandb: Currently logged in as: gwshim (gwshim-ktb) to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
Tracking run with wandb version 0.19.8
Run data is saved locally in /content/wandb/run-20250321_083924-ts3vu75p
Syncing run ./vit_orange_finetune to Weights & Biases (docs)
View project at https://wandb.ai/gwshim-ktb/huggingface
View run at https://wandb.ai/gwshim-ktb/huggingface/runs/ts3vu75p
[372/496 01:56 < 00:39, 3.18 it/s, Epoch 6/8]
```

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.703095	0.909091
2	No log	0.371259	0.969697
3	No log	0.270063	0.959596
4	No log	0.217392	0.979798
5	No log	0.197314	0.979798
6	No log	0.190802	0.979798




```
[7/7 00:00]
테스트 데이터 정확도: 0.9798
```

- **ViT 모델 테스트 데이터 정확도: 0.9798 달성 98%**

---

* 중간 리포팅 이후 1차 데이터 증강을 줄인 Efficientnet V2S로 추가 실험 진행

13th Exp

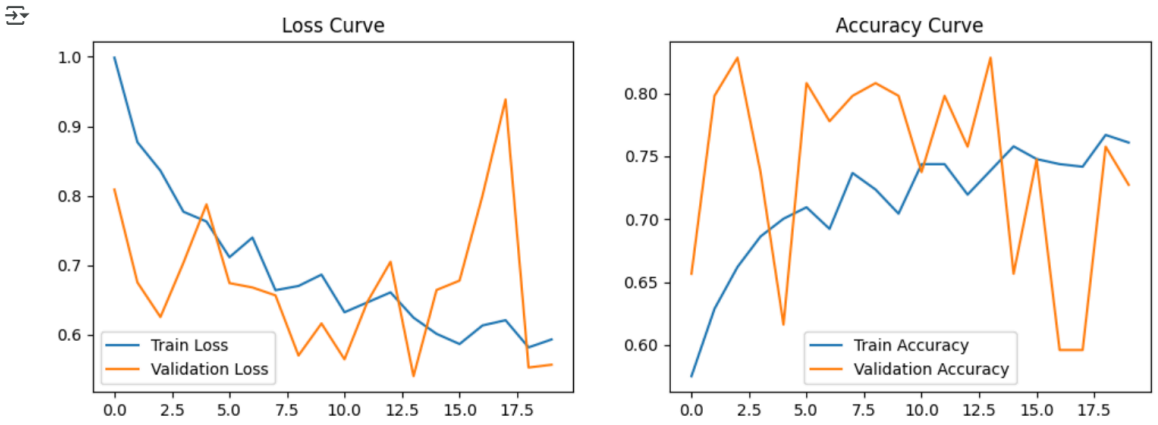
https://colab.research.google.com/drive/1qUnDMruOExtjSD9j1Nw7nwPl-F2kkyfs#scrollTo=albBpUX0hVQz

- 데이터 증강 : 1차 테스트 정도로 감소 시킴
- 사전 학습, 튜닝 epoch 개수만 증가 시킴

```
# 모델 학습 설정
mongo_logger = MongoLoggingCallback()

EPOCHS = 20

history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=EPOCHS,
    verbose=1,
    callbacks=[mongo_logger]
)
```

```
# 일부 레이어를 학습 가능하도록 설정
base_model.trainable = True

# 모델 재컴파일 (학습률 낮춤)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),

# 추가 학습
EPOCHS_FINE_TUNE = 8

history_finetune = model.fit(
    train_data,
    validation_data=val_data,
    epochs=EPOCHS_FINE_TUNE,
    verbose=1,
    callbacks=[mongo_logger]
)
```

- 결과



- accuracy: 0.9475 - loss: 0.1367 | Validation Accuracy: 0.9495
- Loss 감소, Accuracy 증가로 보아 **수렴은 잘 되고 있음**
- Validation Loss/Accuracy 진동 → **일반화 불안정**
- 특정 Epoch에서 **일시적 과적합 존재** 가능

14th Exp

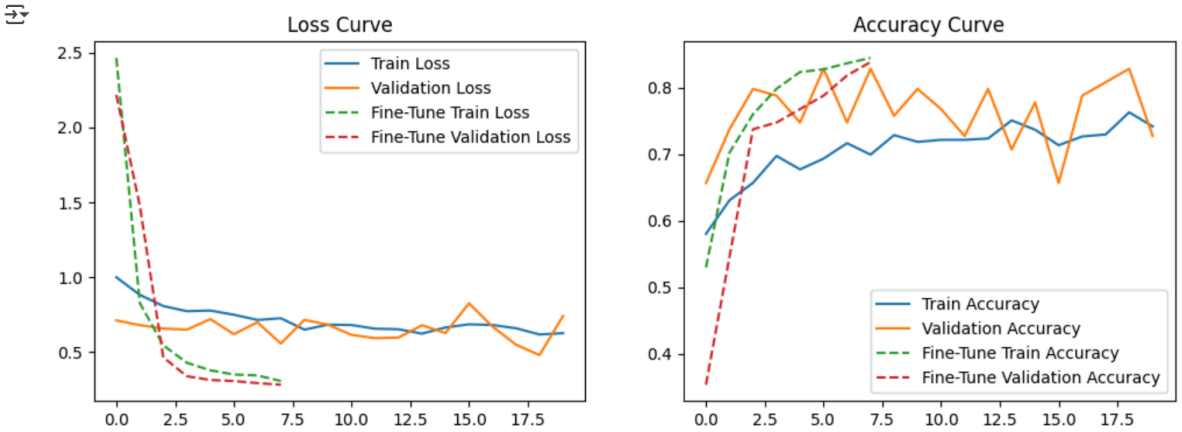
https://colab.research.google.com/drive/1qUnDMruOExtjSD9j1Nw7nwPl-F2kkyfs?usp=sharing

- 데이터 증강 : 1차 테스트 정도로 동일

- 1차 기준 사전 학습, 튜닝 epoch 개수만 증가
- 진동이 커서 Drop out 안정화 증가

```
# EfficientNetV2 모델 불러오기 (Pretrained weights 사용)
base_model = EfficientNetV2S(weights="imagenet", include_top=False, input_shape=(256, 256, 3))

# 기존 가중치 동결 (전이 학습)
base_model.trainable = False

# 분류기 추가
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(512, activation="relu")(x)  # 더 큰 히든 레이어 추가
x = Dropout(0.4)(x)  # Dropout 증가
x = Dense(256, activation="relu")(x)
x = Dropout(0.3)(x)
x = Dense(len(classes), activation="softmax")(x)  # 클래스 개수만큼 출력 노드 설정

# 모델 생성
model = Model(inputs=base_model.input, outputs=x)

# 모델 컴파일
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

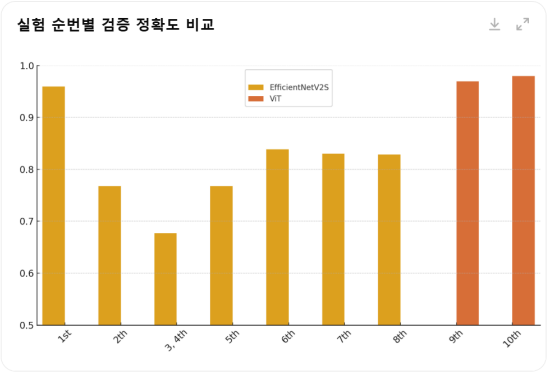
- 결과

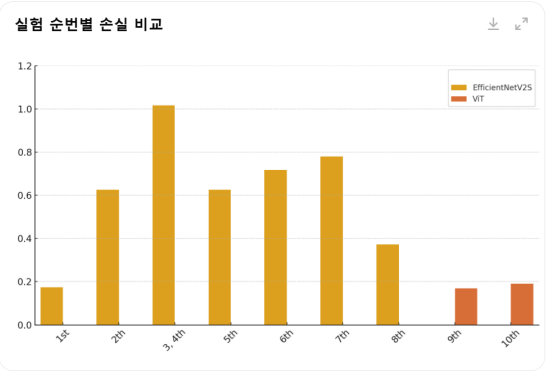


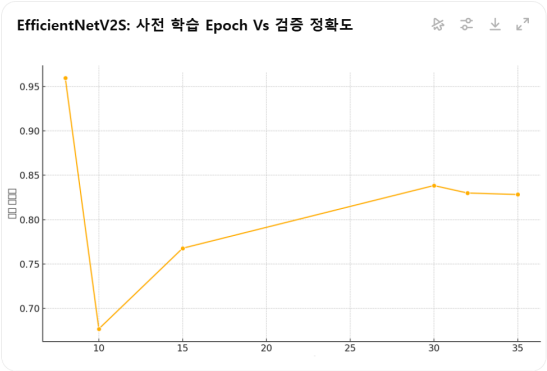
- accuracy: 0.8479 - loss: 0.2600 | Validation Accuracy: 0.8384
- Fine-tuning 구간에서 손실 감소 속도가 더 빠름
- 검증 정확도 또한 평균적으로 더 높은 수치를 유지
- 학습 효율성과 성능 안정성 높아짐

5. 결과 분석

5-1. 모델 테스트 분석 (중간 보고 1 ~ 10차 실험 기준)

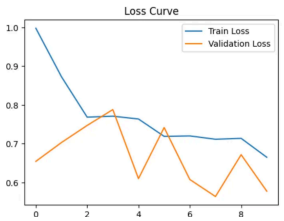


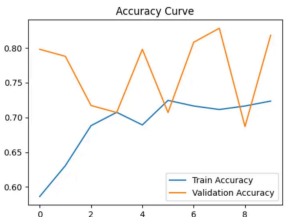




< 참고 사항 >

- 1th 실험의 정확도는 높게 나왔으나 진동이 크게 나와 안정도와 신뢰도가 낮아 최종 결과 분석에는 적용하지 않았다.
- 13, 14th 실험에 1th의 높은 정확도를 유지하기 위해 별도 테스트를 실행하였다.
- 1차 실험 결과 : 진동이 큰 것을 확인 할 수 있다.





5-2. 주요 결과 요약 (중간 보고 1 ~ 10차 실험 기준)

모델	실험 번호	검증 정확도	정확도	손실
EfficientNetV2S	8th	0.8365	0.8283	0.3724
ViT	10th	0.9797	0.9697	0.1908
ViT	9th	0.9697	0.9697	0.1687

- 성능
  - 최고 성능: ViT ：10th 실험
  - 최소 손실: ViT ：9th 실험
  - EfficientNet 최적 결과: 8st 실험
- 데이터 증강 분석
  - 단순한 증강(회전, 이동, 반전 등)은 모델 성능 향상에 효과적
  - 밝기/채도/기울이기/노이즈 추가 등 **과도한 증강은 오히려 성능 저하 발생**
  - 노이즈, mismatch 추가 실험(3, 4차)에서는 손실이 1.0158로 최악의 결과
- 학습 Epoch 영향 분석 (EfficientNetV2S)
  - 사전 학습 Epoch이 증가함에 따라 성능이 부분적으로 회복됨

- **EfficientNetV2S**
  - 경량 조건에서 뛰어난 성능 가능
  - 과도한 데이터 증강에는 민감

- **ViT**
  - 데이터 증강 없이도 높은 성능 달성
  - 적은 학습량으로도 손실이 낮고, 검증 정확도 높음.
  - 소량 데이터 또는 증강 어려운 상황에 유리.

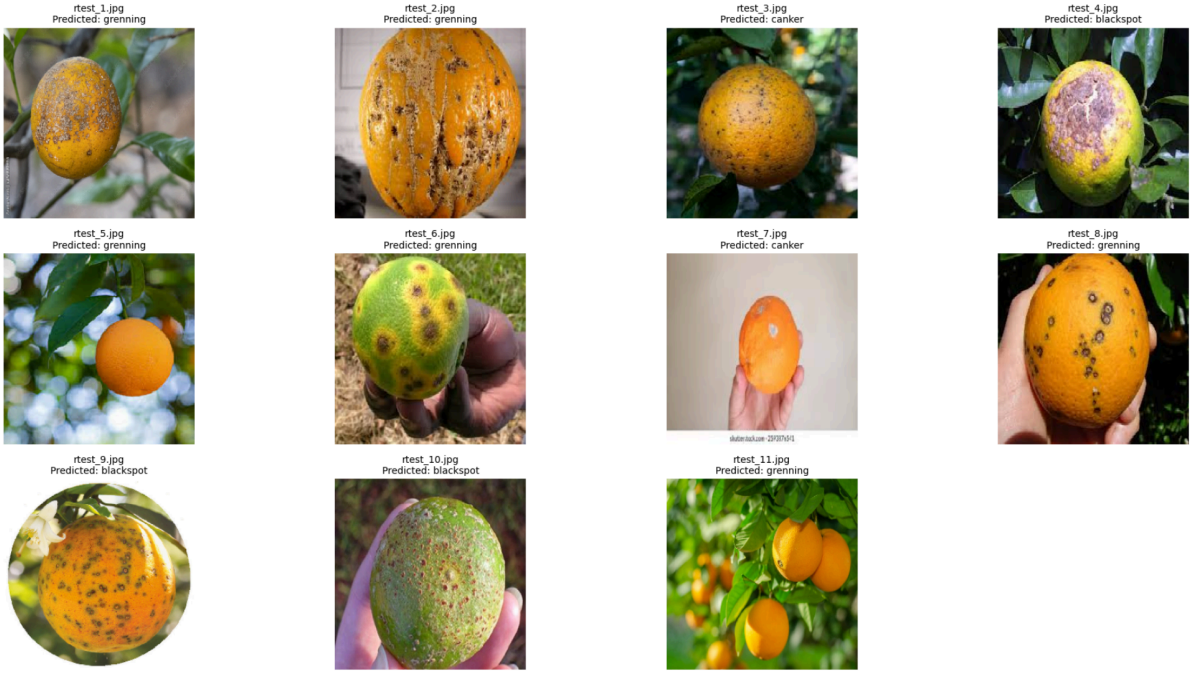
5-2. 주요 결과 결론 (중간 보고 1 ~ 10차 실험 기준)

- 효율적인 데이터 증강 전략이 중요하며, 단순한 변형 위주가 효과적이였음
- 자원 효율성을 고려할 경우, EfficientNet + 단순 증강 조합이 안정적 선택
- ViTransformer 모델의 성능이 확실히 더 빠르고 좋았으므로, 높은 성능을 원한다면 ViT 사용
- 데이터 증강시 회전, 반전, 위치 조정은 효과 있음.
- efficientNetV2S 초기 레이어의 Fused MPC 3×3 형식, 노이즈 증가, 교차 이미지는 효과가 적음.
- 사전 학습을 충분히 하는 부분도 좋음
- efficientNetV2S 과 잘 안맞았다고 생각되는 데이터 증강 기법 > CutMIX (이미지 패치), MixUp (필요한 이미지 수 자연스런 비율 조정)

5-3. 예측 실험 (중간 보고 1 ~ 10차 실험 기준)

https://colab.research.google.com/drive/1KE1bpCmurR5c9Eyxb2FaSySFwa9BqwEB

1. 성능 분류 검증 : 11개의 실제 이미지 분류 테스트 → 9정답 / 11개 : 약 85%의 성능 검증
2. 데이터셋과 다르게 배경이 있는 이미지의 분류에서는 Fresh 상태의 예측 성공이 낮음



5-4. EfficientnetV2S 모델 추가 분석 결과 (13, 14th)

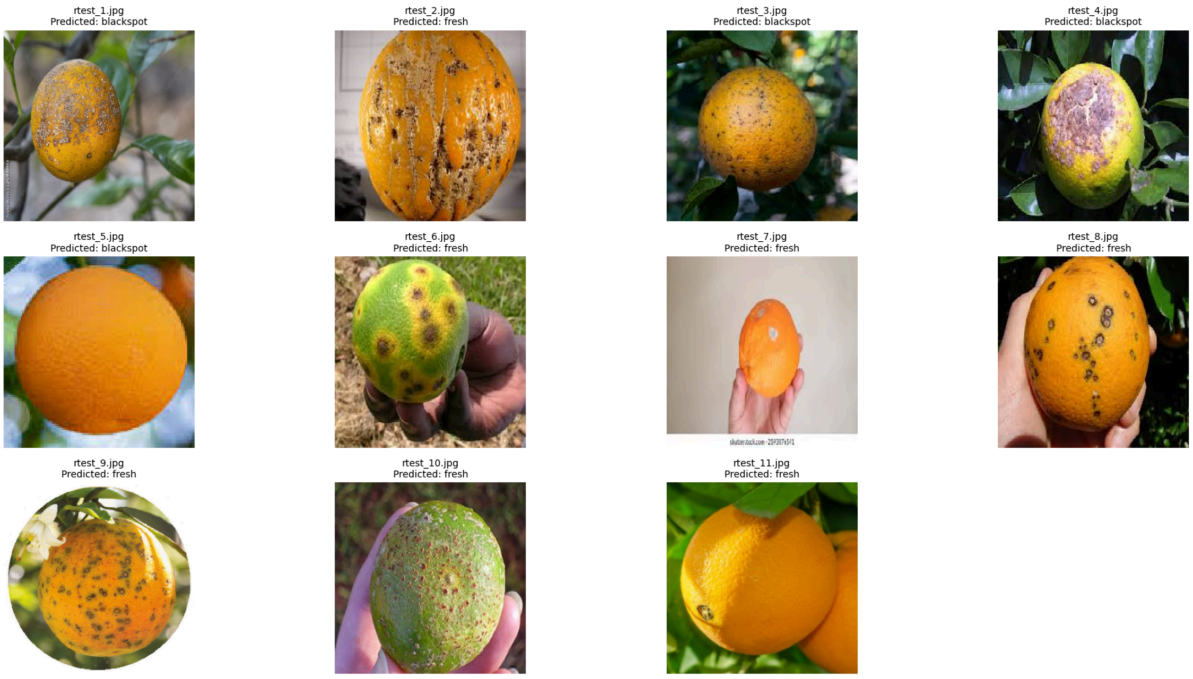
- 1th의 높은 성능을 안정적으로 구현해 보기 위해 데이터 증강은 최소화 하고 사전, 튜닝 학습수, drop out를 통한 안정성을 높이는 13, 14th 실험 추가 진행
- 13th 실험 분석
  - Train Loss는 꾸준히 감소.
  - Validation Loss는 기복이 매우 심하고 불안정, 특히 Epoch 17~18에서 급격히 증가.
  - 일시적 과적합 또는 데이터 불균형
  - Train Accuracy는 꾸준히 증가하여 약 0.76 도달.
  - Validation Accuracy는 진동이 심하고, 중간중간 0.6 이하로 급락
  - 검증 데이터와 학습 데이터 간 분포 차이 가능성
  - 데이터 증강 불균형 또는 부족한 정규화
- 14th 실험 분석
  - Train / Validation Loss 모두 훨씬 빠르게 감소 (Fine-tuning 시도된 부분)
  - Fine-tune 구간에서는 Loss가 0.3 이하로 안정적으로 유지됨
  - Validation Loss가 매우 안정적이며 낮음
  - Fine-tune Accuracy (점선)가 빠르게 상승하고 0.85 수준까지 도달.
  - Validation Accuracy 또한 진동이 적고 전반적으로 더 높음, 3~6 epoch 사이 최고 성능 기록
- 14th 실험이 수렴 속도, 손실 값 모두 높음

- 모델이 더 잘 학습되고, 일반화 성능이 더 뛰어남
- Fine-tuning을 적용한 14th 실험에서는 전체적으로 손실 값이 더 빠르게 감소하고 정확도도 더 높은 수준에서 안정적으로 수렴
- Validation Accuracy의 진동이 적고 최고 정확도가 높아, 모델이 학습 데이터 뿐 아니라 검증 데이터에 대해서도 일반화가 잘 되었음을 확인
- Fine-tuning을 통한 사전 학습 가중치 조정이 모델 성능 향상에 효과적을 증명

5-4. 예측 실험 (13, 14th)

https://colab.research.google.com/drive/1KE1bpCmurR5c9Eyxb2FaSySFwa9BqwEB

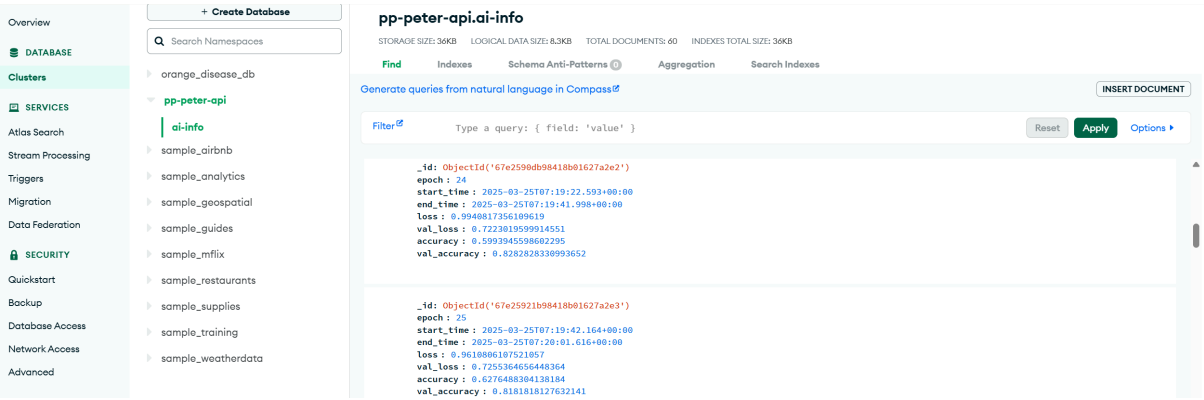
1. 성능 분류 검증 : 11개의 실제 이미지 분류 테스트 → 3정답 / 11개 : 약 30%의 낮은 예측
2. 데이터셋과 다른 형태의 이미지를 통한 클래스 분류에 매우 낮은 예측 성공률을 보임



5-5. DB 연동 테스트 (14th)

https://colab.research.google.com/drive/1KE1bpCmurR5c9Eyxb2FaSySFwa9BqwEB

- 모델 트레이닝 프로세싱 정보와 NoSQL(MongoDB)와 데이터 저장 및 조회 실습
- id, epoch, start time, end time, loss, acurracy, validation accuracy 값이 성공적으로 저장 및 조회되는 부분을 확인하였음



📊 MongoDB 학습 로그 조회 결과

	message	epoch	start_time	end_time	loss	val_loss	accuracy	val_accuracy	duration
0	Colab 연결 테스트 완료!	nan	NaT	NaT	nan	nan	nan	nan	NaT
1	nan	1.000000	2025-03-25 07:24:34.043000	2025-03-25 07:26:54.443000	1.065176	0.619954	0.564077	0.808081	0 days 00:02:20.400000
2	nan	1.000000	2025-03-25 07:10:03.717000	2025-03-25 07:12:10.398000	1.351745	1.171462	0.344097	0.545455	0 days 00:02:06.681000
3	nan	2.000000	2025-03-25 07:26:54.617000	2025-03-25 07:27:14.452000	0.930698	0.487641	0.609485	0.818182	0 days 00:00:19.835000
4	nan	2.000000	2025-03-25 07:12:10.571000	2025-03-25 07:12:30.009000	1.257730	1.058195	0.524723	0.555556	0 days 00:00:19.438000
5	nan	3.000000	2025-03-25 07:12:30.173000	2025-03-25 07:12:49.550000	1.219159	0.987155	0.503532	0.646465	0 days 00:00:19.377000
6	nan	3.000000	2025-03-25 07:27:14.622000	2025-03-25 07:27:34.203000	0.898672	0.458966	0.629667	0.797980	0 days 00:00:19.581000
7	nan	4.000000	2025-03-25 07:12:49.715000	2025-03-25 07:13:09.187000	1.182803	0.926229	0.526741	0.646465	0 days 00:00:19.472000
8	nan	4.000000	2025-03-25 07:27:34.370000	2025-03-25 07:27:53.855000	0.856699	0.530778	0.656912	0.727273	0 days 00:00:19.485000
9	nan	5.000000	2025-03-25 07:13:09.352000	2025-03-25 07:13:28.957000	1.150252	0.880628	0.528759	0.696970	0 days 00:00:19.605000
10	nan	5.000000	2025-03-25 07:27:54.021000	2025-03-25 07:28:13.569000	0.817378	0.392151	0.670030	0.808081	0 days 00:00:19.548000
11	nan	6.000000	2025-03-25 07:13:29.122000	2025-03-25 07:13:48.512000	1.130331	0.854313	0.554995	0.737374	0 days 00:00:19.390000
12	nan	6.000000	2025-03-25 07:28:13.734000	2025-03-25 07:28:33.291000	0.834283	0.371641	0.657921	0.848485	0 days 00:00:19.557000
13	nan	7.000000	2025-03-25 07:13:48.678000	2025-03-25 07:14:08.224000	1.115204	0.839566	0.547931	0.737374	0 days 00:00:19.546000
14	nan	7.000000	2025-03-25 07:28:33.455000	2025-03-25 07:28:52.647000	0.771301	0.428997	0.680121	0.797980	0 days 00:00:19.192000
15	nan	8.000000	2025-03-25 07:14:08.388000	2025-03-25 07:14:27.819000	1.098237	0.823544	0.551968	0.767677	0 days 00:00:19.431000
16	nan	8.000000	2025-03-25 07:28:52.812000	2025-03-25 07:29:12.281000	0.759865	0.513911	0.678103	0.797980	0 days 00:00:19.469000
17	nan	9.000000	2025-03-25 07:29:12.452000	2025-03-25 07:29:31.883000	0.755345	0.577335	0.686176	0.727273	0 days 00:00:19.431000
18	nan	9.000000	2025-03-25 07:14:27.994000	2025-03-25 07:14:47.273000	1.107908	0.820438	0.559031	0.757576	0 days 00:00:19.279000
19	nan	10.000000	2025-03-25 07:29:32.053000	2025-03-25 07:29:51.472000	0.761208	0.560005	0.678103	0.848485	0 days 00:00:19.419000

6. 결론

오렌지 질병 분류를 위해 EfficientNetV2S와 ViT(Vision Transformer) 모델을 활용하여 다양한 데이터 증강 및 학습 전략의 영향을 비교 분석하였습니다. EfficientNetV2S 모델은 상대적으로 경량화된 구조로, 제한된 연산 자원에서도 우수한 성능을 낼 수 있으며, 단순한 증강(회전, 이동, 반전 등)을 적용했을 때 가장 안정적이고 일관된 결과(검증 정확도 95.96%)를 보였습니다. 반면, 밝기 조정, 노이즈 추가, MixUp 등 과도한 증강 기법은 오히려 손실을 증가시키고 검증 정확도 저하를 가져왔습니다. 특히 Fine-tuning을 적용한 실험에서는 사전 학습된 가중치를 기반으로 손실이 빠르게 감소하고 검증 정확도가 크게 향상되는 등 일반화 성능의 개선이 뚜렷하게 나타났습니다. EfficientNetV2S는 사전 학습 epoch 수가 일정 수준 이상일 때 성능 회복력이 뛰어나며, Dropout과 같은 안정화 기법과 병행할 경우 더욱 효과적이었습니다. 반면, 과도한 데이터 증강은 모델의 수렴을 방해하거나 학습을 저해하는 결과를 보였습니다. 비교하기 위해 선택한 ViT 모델은 적은 학습량으로 가장 높은 정확도(97.97%)를 기록하며 이미지 분류에 있어 사용 비율이 증가 되고 있는 것처럼 실험에서도 높은 성능을 보였습니다. 결론적으로, EfficientNetV2S는 경량 조건에서 단순 증강과 적절한 학습 설정을 통해 안정적 성능을 확보할 수 있으며, Fine-tuning 전략을 병행할 경우 일반화와 수렴 속도 모두에서 큰 개선을 기대할 수 있습니다.

7. 참고 자료

1. **EfficientNetV2 관련 논문 및 자료**

EfficientNetV2: Smaller Models and Faster Training

https://da2so.tistory.com/45

https://audrb1999.tistory.com/7

2. **Vision Transformer (ViT) 관련 참고자료**

https://www.ibm.com/kr-ko/think/topics/transformer-model

https://www.hpe.com/kr/ko/what-is/transformer-model.html

https://ko.wikipedia.org/wiki/트랜스포머_(기계_학습)