



Using Potato Plant Diseases Data to Build CNN Modeling Comparison

[Date : 2025.05]

1. 서론

- 감자는 세계 주요 식량 작물 중 하나로, 특히 **역병(Late Blight, Phytophthora infestans)** 은 감자 생산량에 치명적인 영향을 미치는 대표적인 병해입니다. 특히 초기 역병과 후기 역병은 완전히 다른 질병이며 **초기 역병**은 잎의 작은 반점이나 모서리 변색 등으로 시작되어, 빠른 시간 내에 줄기 및 뿌리까지 전파된다. **후기 역병**은 이미 병이 확산된 상태에서 급격한 조직 괴사와 작물 고사 현상을 유발한다. 이 둘은 방제 시기 및 약제 선택이 다르기 때문에 **정확한 구분이 필수적이다**. 따라서 본 연구는 **감자의 초기 및 후기 역병 발생 시기의 정확한 분류 및 예측 모델**을 구축함으로써, 조기 방제 및 농가 생산성 향상에 기여하고자 한다.

2. 데이터셋 설명

- 이번 모델에서 사용한 데이터 셋은 Kaggle 사이트에 업로드 되어있는 **"Potato Plant Diseases Data"** 이다.
- ▼ 구글 드라이브에 데이터 셋을 업로드 후, 불러오는 식으로 진행하였다.

```
import os

# 구글 드라이브 데이터셋 경로
dataset_path = "/content/drive/MyDrive/Colab Notebooks/[카카오 부트캠프]

# 클래스별 이미지 개수 출력 함수
def count_images_per_class(dataset_path):
```

```

print(f"기준 경로: {dataset_path}\n")
for class_name in os.listdir(dataset_path):
    class_path = os.path.join(dataset_path, class_name)
    if os.path.isdir(class_path):
        img_count = len([
            f for f in os.listdir(class_path)
            if f.lower().endswith(('.jpg', '.jpeg', '.png'))
        ])
        print(f"클래스 '{class_name}': {img_count}장")

# 실행
count_images_per_class(dataset_path)

# 클래스별 폴더 확인 (파일이 아닌 폴더만 리스트에 추가)
categories = [category for category in os.listdir(dataset_path) if os.path.isd

# 클래스 목록 출력
print(f"클래스 개수: {len(categories)}")
print(f"클래스 목록: {categories}")

# 출력 결과
클래스 개수: 3
클래스 목록: ['Potato__Early_blight', 'Potato__Late_blight', 'Potato__health

클래스 'Potato__Early_blight': 1000장
클래스 'Potato__Late_blight': 1000장
클래스 'Potato__healthy': 152장

```

- 해당 데이터 셋은
 - 'Potato__Early_blight'
 - 'Potato__Late_blight'
 - 'Potato__healthy'

으로 총 3개 클래스이며, 각각 1000장 그리고 152장으로 구성 되어있다.

- 하지만 클래스 불균형(Class Imbalance)이 심하여 작은 클래스는 무시하는 편향된 학습이 발생할 수 있다.

- 따라서 데이터 증강과 더불어 `WeightedRandomSampler` & `CrossEntropyLoss`의 `weight` 기법을 사용하였다.

WeightedRandomSampler

- 목적:
 - 훈련 시 적은 클래스가 자주 등장하도록 조정
- 방식:
 - Pytorch의 `WeightedRandomSampler`는 각 샘플에 대해 등장 확률을 부여
 - 데이터가 적은 클래스는 더 자주 샘플링 되도록 높은 확률 부여
 - 데이터가 많은 클래스는 덜 자주 등장하도록 낮은 확률 부여
- 기대효과:
 - 학습 데이터의 클래스 분포를 균등하게 만듦

CrossEntropyLoss의 weight

- 목적:
 - 모델의 손실(loss)을 계산할 때 적은 클래스의 오차를 더 크게 반영
- 방식:
 - `CrossEntropyLoss`에 `weight` 인자를 넣으면, 각 클래스별로 손실에 주는 **중요도 (weight)** 를 조절함
 - 예시: 데이터가 적은 클래스 → 가중치를 더 부여하여 손실을 크게 반영
- 기대효과:
 - 모델이 적은 클래스도 중요하게 학습됨

-
- 클래스 불균형이 심한 경우 이러한 기법들을 같이 적용시키면 "훈련 데이터 등장 비율과 손실 계산시 중요도 모두 균형 있게 반영된다."
 - 그래서 데이터 증강 과 위 기법들을 적용시킨 방식을 비교 분석 하여 데이터 불균형을 어떻게 처리 할 것인지 실험을 진행 하였다.
 - 해당 실험에 대해서 "ResNet50"을 선택 하였는데, 이유는 다음과 같다.
 - **풍부한 사전 학습 기반 가중치:** 깊은 네트워크 구조에도 불구하고 학습 안정성이 확보되어 있음

- **잔차 연결(Residual Block)** 구조를 통해 깊은 모델에서도 gradient vanishing 문제가 완화됨
- **다양한 실험에서 기준점(Baseline)으로 널리 사용됨**
- 특히, 본 실험 초기에는 다른 모델들과의 성능 비교가 아직 이루어지지 않은 상태이므로, 구조적으로 안정적이며 파인튜닝이 잘 되어있는 ResNet50을 기준 모델로 선정하여 실험을 진행 하였다.
 - Baseline : 기본 증강
 - Refactored : Sampler + Weighted Loss
- 그 전에 앞서 데이터양이 매우 적은 편이므로 **데이터 증강** 및 **데이터 분할**을 train : val : test = 8:1:1로 구성하여 train 학습에 좀더 집중하였다.

▼ Baseline

```
import os
import torch
from torchvision import transforms, datasets
from torch.utils.data import DataLoader, WeightedRandomSampler
import numpy as np
from torch import nn

# 데이터셋 경로
DATASET_PATH = "/content/drive/MyDrive/Colab Notebooks/[카카오 부

# 데이터 증강 (훈련용)
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.RandomApply([transforms.ColorJitter(brightness=0.2, co
    transforms.RandomAffine(degrees=15, translate=(0.1, 0.1), scale=(0.
    transforms.GaussianBlur(kernel_size=3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                          std=[0.229, 0.224, 0.225])
])

# 검증/테스트용
```

```

val_test_transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                          std=[0.229, 0.224, 0.225])
])

# 데이터셋 로드
train_dataset_weighted = datasets.ImageFolder(os.path.join(DATASET_PATH, "train"))
val_dataset = datasets.ImageFolder(os.path.join(DATASET_PATH, "val"))
test_dataset = datasets.ImageFolder(os.path.join(DATASET_PATH, "test"))

# 클래스별 개수 수동 입력
class_counts = [1000, 1000, 152] # Early, Late, Healthy
class_weights = 1. / torch.tensor(class_counts, dtype=torch.float)
sample_weights = [class_weights[label] for label in train_dataset_weighted.classes]

# WeightedRandomSampler 생성
sampler = WeightedRandomSampler(sample_weights, num_samples=len(train_dataset_weighted))

# 데이터로더 생성
BATCH_SIZE = 32
train_loader_weighted = DataLoader(train_dataset_weighted, batch_size=BATCH_SIZE, sampler=sampler)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=True)

# 손실 함수 정의 (class weight 적용)
loss_weights = torch.tensor([1.0, 1.0, 1000 / 152], dtype=torch.float)
criterion = nn.CrossEntropyLoss(weight=loss_weights)

# 데이터셋 개수 확인
print(f"훈련 데이터 개수: {len(train_dataset_weighted)}")
print(f"검증 데이터 개수: {len(val_dataset)}")
print(f"테스트 데이터 개수: {len(test_dataset)}")

# 클래스 정보 출력
print(f"클래스 개수: {len(train_dataset_weighted.classes)}")

```

```

print(f"클래스 목록: {train_dataset_weighted.classes}")

# 출력 결과
훈련 데이터 개수: 1721
검증 데이터 개수: 215
테스트 데이터 개수: 216
클래스 개수: 3
클래스 목록: ['Potato__Early_blight', 'Potato__Late_blight', 'Potato__he

```

▼ Refactored

```

import os
import torch
from torchvision import transforms, datasets
from torch.utils.data import DataLoader

# 데이터셋 경로 설정
DATASET_PATH = "/content/drive/MyDrive/Colab Notebooks/[카카오 부

# 훈련 데이터에 데이터 증강 추가
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224), # 중심 크롭 후 224x224 변환
    transforms.RandomHorizontalFlip(), # 좌우 반전
    transforms.RandomRotation(15), # 15도 이내 랜덤 회전
    transforms.RandomApply([transforms.ColorJitter(brightness=0.2, co
    transforms.RandomAffine(degrees=15, translate=(0.1, 0.1), scale=(0.
    transforms.GaussianBlur(kernel_size=3), # 블러 효과 추가
    transforms.ToTensor(), # Tensor로 변환
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.2
]))

# 검증 & 테스트 데이터는 원본 유지
val_test_transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.2
]))

```

```

# 데이터셋 로드 (훈련 데이터에는 증강 적용)
train_dataset_baseline = datasets.ImageFolder(root=os.path.join(DATA
val_dataset = datasets.ImageFolder(root=os.path.join(DATASET_PATH,
test_dataset = datasets.ImageFolder(root=os.path.join(DATASET_PATH

# 데이터 로더 생성
BATCH_SIZE = 32
train_loader_baseline = DataLoader(train_dataset_baseline, batch_size
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuff

# 데이터셋 개수 확인
print(f"훈련 데이터 개수: {len(train_dataset_baseline)}")
print(f"검증 데이터 개수: {len(val_dataset)}")
print(f"테스트 데이터 개수: {len(test_dataset)}")

# 클래스 정보 출력
print(f"클래스 개수: {len(train_dataset_baseline.classes)}")
print(f"클래스 목록: {train_dataset_baseline.classes}")

# 출력 결과
훈련 데이터 개수: 1721
검증 데이터 개수: 215
테스트 데이터 개수: 216
클래스 개수: 3
클래스 목록: ['Potato__Early_blight', 'Potato__Late_blight', 'Potato__he

```

- 각 기법 마다 가중치 & 전체 모델(.pt & .pth)을 저장한 후 Epoch별 log 정보를 보기위해 .csv 파일 또한 저장하였다.

▼ 모델 저장 코드(weighted 기법 예시)

```

# 구글 드라이브 내 모델 저장 경로 설정
SAVE_DIR = "/content/drive/MyDrive/Colab Notebooks/[카카오 부트캠프]/[카
os.makedirs(SAVE_DIR, exist_ok=True) # 폴더가 없으면 생성
CSV_SAVE_PATH = "/content/drive/MyDrive/Colab Notebooks/[카카오 부트

# best_model_path: 리팩토링 모델 저장 경로 (가중치만 저장되어 있음)

```

```

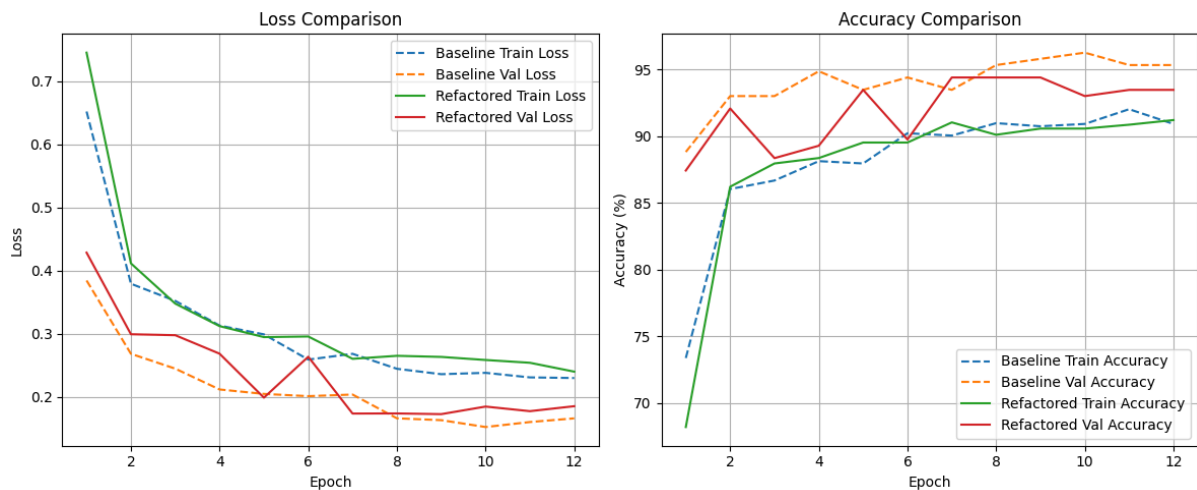
torch.save(torch.load(best_model_path), os.path.join(SAVE_DIR, "resnet_v
print(f"최종 모델 가중치 저장 완료: {os.path.join(SAVE_DIR, 'resnet_weighted.

# 전체 모델 저장 (구조 + 가중치 포함)
torch.save(model, os.path.join(SAVE_DIR, "resnet_weighted_model_full(8:
print(f"최종 모델 전체 저장 완료: {os.path.join(SAVE_DIR, 'resnet_weighted_m

df = pd.DataFrame({
    "epoch": list(range(1, len(train_losses_weighted) + 1)),
    "train_loss": train_losses_weighted,
    "val_loss": val_losses_weighted,
    "train_accuracy": train_accuracies_weighted,
    "val_accuracy": val_accuracies_weighted
})
df.to_csv(CSV_SAVE_PATH, index=False)
print(f"학습 로그 CSV 저장 완료: {CSV_SAVE_PATH}")

```

결론:



```

=== 최종 테스트 성능 비교 ===
Baseline   - Loss: 0.1762 | Accuracy: 96.30%
Refactored - Loss: 0.1901 | Accuracy: 95.83%

```

- 단순히 클래스 간 불균형을 해결하기 위해 Weighted Sampler 및 Loss Weight를 도입하였으나, 이번 실험에서는 데이터 증강만 적용한 기법(Baseline)이 더 안정적인 성능을 보였다. 이는 클래스 간 불균형보다, 데이터 증강 자체가 모델의 일반화에 더 큰 기여를 한 결과로 판단된다. 따라서 향후 모델별 비교 실험 시에는 **데이터 증강 기반**

Baseline을 기준으로 삼고, 불균형 해소 전략은 필요할 때만 추가 적용하는 방향이 바람직해 보인다.

3. 모델 설명

- 이번 프로젝트에서 감자 식물 역병 데이터셋을 학습시키기위해 훈련한 모델은 총 5가지 모델로 ResNet50, ResNet18, VGG16, MobileNet, GoogLeNet 등을 사용했다.
 - ResNet50
 - ▼ 조기종단(early stopping)을 적용한 모델학습 함수

```
import os
import numpy as np
import torch

best_model_path = "/content/drive/MyDrive/Colab Notebooks/[카카오

# 얼리 스탑 설정
patience = 5 # 5 에포크 동안 개선이 없으면 중단
min_delta = 0.001 # 개선이 min_delta 이하이면 의미 없는 개선으로 간주
best_val_loss = np.inf # 처음에는 무한대로 설정
counter = 0 # 개선되지 않은 횟수 카운트

EPOCHS = 50 # 최대 50 에포크까지 학습

train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

# 모델 평가 함수
def evaluate(model, dataloader):
    model.eval() # 평가 모드로 변경
    correct = 0
    total = 0
    running_loss = 0.0
```

```

with torch.no_grad(): # 그래디언트 계산 비활성화 (속도 최적화)
    for images, labels in dataloader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        running_loss += loss.item()

        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    avg_loss = running_loss / len(dataloader)
    accuracy = correct / total * 100
    return avg_loss, accuracy

for epoch in range(EPOCHS):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        running_loss += loss.item()

    # 학습 손실 및 정확도 계산
    train_loss = running_loss / len(train_loader)
    train_acc = correct / total * 100

```

```

train_losses.append(train_loss)
train_accuracies.append(train_acc)

# 검증 손실 및 정확도 계산
val_loss, val_acc = evaluate(model, val_loader)
val_losses.append(val_loss)
val_accuracies.append(val_acc)

# 현재 학습률 확인
current_lr = optimizer.param_groups[0]['lr']

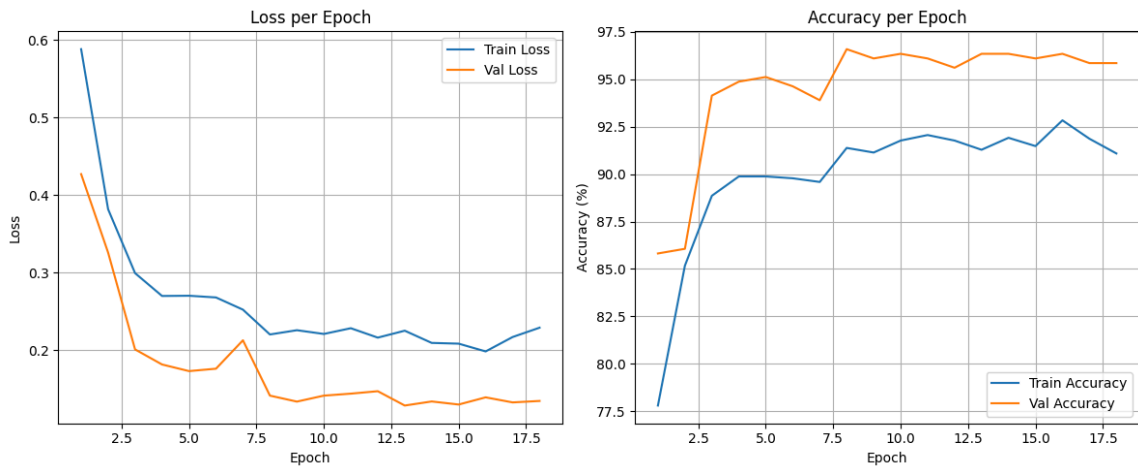
# 로그 출력
print(f"\n Epoch [{epoch+1}/{EPOCHS}]")
print(f"Train Loss: {train_loss:.4f} | Train Acc: {train_acc:.2f}%")
print(f"Val Loss: {val_loss:.4f} | Val Acc: {val_acc:.2f}%")
print(f"Learning Rate: {current_lr:.6f}")

# 얼리 스탑 확인
if val_loss < best_val_loss - min_delta:
    best_val_loss = val_loss
    counter = 0
    torch.save(model.state_dict(), best_model_path)
    print(f"성능 향상! 모델 저장됨: {best_model_path}")
else:
    counter += 1
    print(f"개선 없음 (Counter: {counter}/{patience})")

# 얼리 스탑 조건 충족 시 학습 중단
if counter >= patience:
    print(f"\n얼리 스탑 발생! {epoch+1} 에포크에서 학습 종료")
    break

# 학습률 스케줄러 업데이트
scheduler.step()

```



ResNet50 모델 학습 시각화

```
import matplotlib.pyplot as plt

# 모델 평가 함수
def evaluate(model, dataloader):
    model.eval() # 평가 모드
    correct = 0
    total = 0
    running_loss = 0.0

    with torch.no_grad(): # 그래디언트 계산 X (속도 최적화)
        for images, labels in dataloader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            running_loss += loss.item()

            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    avg_loss = running_loss / len(dataloader)
    accuracy = correct / total * 100
    return avg_loss, accuracy

# 최종 테스트 데이터 평가
test_loss, test_acc = evaluate(model, test_loader)
```

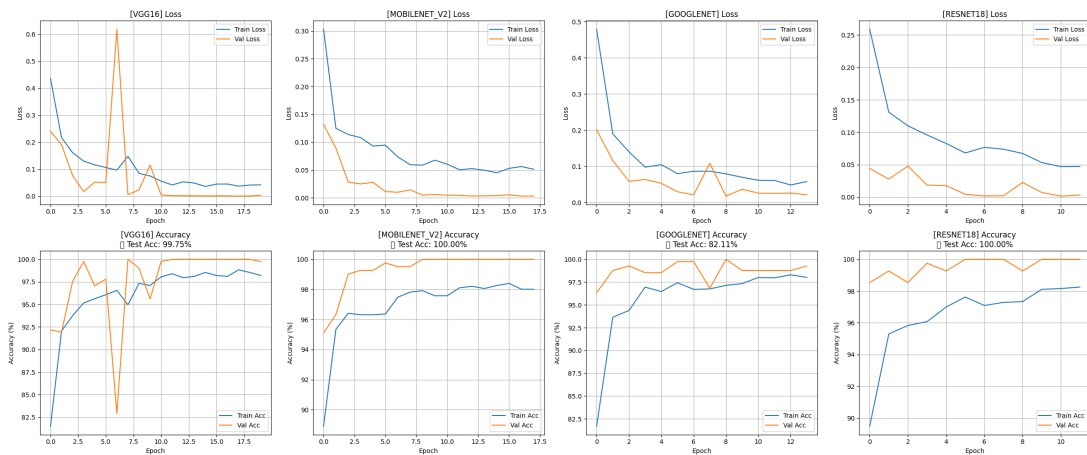
```
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_acc:.2f}%")
```

결과

Test Loss: 0.1191

Test Accuracy: 97.79%

- 역시나 **ResNet50** 모델은 매우 높은 성능을 보여주었다.
→ ImageNet 대규모 데이터셋에서 사전학습된 가중치를 사용이 큰 이유이다.
- 추가로 **경량화**를 염두하여 **ResNet18** 모델과 여러 모델들을 함께 모델학습을 진행하였다.



왼쪽부터 차례대로 VGG16, MobileNet, GoogLeNet, ResNet18 모델

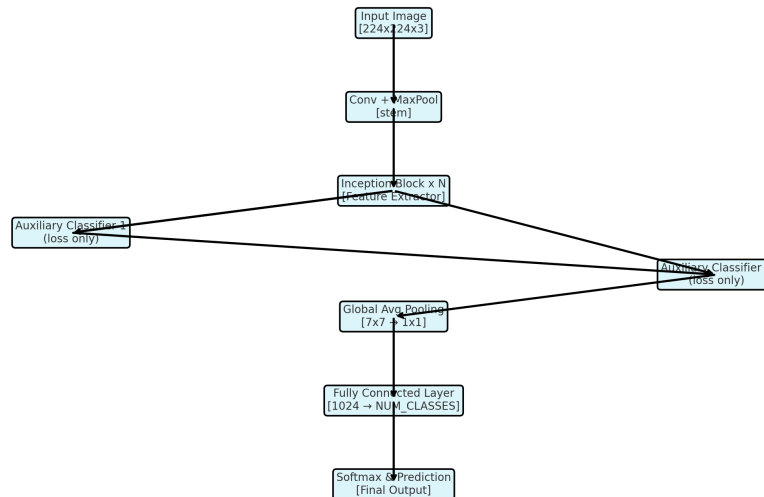
Model	Train Acc	Val Acc	Test Acc
vgg_result	98.21%	99.76%	99.75%
mobilenet_result	98.01%	100.00%	100.00%
googlenet_result	98.01%	99.27%	82.35%
resnet18_result	98.26%	100.00%	100.00%

VGG16, MobileNet, GoogLeNet, ResNet18 모델의 각 Train, Val, Test 정확도

- ResNet50&18, VGG16 그리고 MobileNet 모델을 사용결과 사전학습된 모델이다 보니 성능이 매우 좋은 반면, 상대적으로 학습성능이 떨어진 **GoogLeNet** 모델을 점차 **강화학습**을 진행하는 방향으로 진행하였다.
- GoogLeNet 모델의 특징
 1. **Auxiliary Classifiers (보조 분류기)** 가 중간 레이어에 존재
 - 학습 시 보조 손실로만 사용되며 보조 분류기로 인한 기울기 소실 완화

- 학습 속도 및 안전성 향상 기대
- 2. 작은 모델 크기에도 성능이 우수해서 리소스가 제한된 환경에 적합
- 3. 학습 안정성을 위해 보조 분류기를 쓰는 점이 소규모 데이터셋 학습에도 유리

□ GoogLeNet Architecture Flow



GoogLeNet 모델 구조 요약 그림

4. 실험 방법

GoogLeNet 모델 기반 점진적 성능 개선 전략

- 기존 일반 GoogLeNet 모델에서의 성능이 낮은 이유는 다음과 같다.

1. 보조 분류기(aux) 비활성화 : `aux_logits=False`

2. `CrossEntropyLoss` 사용

```

from torchvision import models
import torch.nn as nn

googlenet = models.googlenet(pretrained=True, aux_logits=False)
googlenet.fc = nn.Linear(googlenet.fc.in_features, NUM_CLASSES)

```

- 따라서 보조분류기(aux) 활성화 및 파인튜닝을 진행하였다.
- ▼ 조기중단(early stopping)을 적용한 해당 트레이닝 함수를 설정하여 진행하였다.

```

# 얼리 스탑 학습 함수
def train_with_early_stopping(model, model_name, train_loader, val_loader, device):
    model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-4)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.5)

    best_val_loss = np.inf
    counter = 0
    train_losses, val_losses = [], []
    train_accs, val_accs = [], []

    save_path = os.path.join(SAVE_DIR, f"{model_name}_best.pth")

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)

            # GoogLeNet 처리
            if isinstance(outputs, tuple) or hasattr(outputs, 'logits'):
                outputs = outputs.logits

            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

```

```

train_loss = running_loss / len(train_loader)
train_acc = correct / total * 100
val_loss, val_acc = evaluate(model, val_loader, criterion)

train_losses.append(train_loss)
val_losses.append(val_loss)
train_accs.append(train_acc)
val_accs.append(val_acc)

print(f"{model_name.upper()} Epoch [{epoch+1}/{num_epochs}]")
print(f"Train Loss: {train_loss:.4f} | Train Acc: {train_acc:.2f}%")
print(f"Val Loss: {val_loss:.4f} | Val Acc: {val_acc:.2f}%")
print(f"LR: {optimizer.param_groups[0]['lr']:.6f}")

# 얼리 스탑 체크
if val_loss < best_val_loss - min_delta:
    best_val_loss = val_loss
    counter = 0
    torch.save(model.state_dict(), save_path)
    print(f"성능 향상! 모델 저장됨: {save_path}")
else:
    counter += 1
    print(f"개선 없음 (Counter: {counter}/{patience})")

if counter >= patience:
    print(f"얼리 스탑 발생! {epoch+1} 에포크에서 학습 종료 ")
    break

scheduler.step()

return {
    "name": model_name,
    "train_loss": train_losses,
    "val_loss": val_losses,
    "train_acc": train_accs,
    "val_acc": val_accs
}

```


GoogLeNet 성능 개선

1. 1차 파인튜닝 (`use_aux=True, freeze_base=False`)

```
from torchvision import models
import torch.nn as nn
import torch

def get_finetuned_googlenet(num_classes, use_aux=True, freeze_base=False):
    model = models.googlenet(pretrained=True, aux_logits=use_aux)

    # 메인 분류기 교체
    model.fc = nn.Linear(model.fc.in_features, num_classes)

    # 보조 분류기도 교체 (aux_logits=True일 때만)
    if use_aux:
        model.aux1.fc2 = nn.Linear(model.aux1.fc2.in_features, num_classes)
        model.aux2.fc2 = nn.Linear(model.aux2.fc2.in_features, num_classes)

    # 백본 freeze (선택 시)
    if freeze_base:
        for name, param in model.named_parameters():
            if "fc" not in name and "aux" not in name:
                param.requires_grad = False

    return model
```

- `aux1,aux2` 를 출력 레이어 클래스 수 `num_classes` 에 맞춰 재정의 하였음
- `freeze_base=False` 덕분에 실제로는 이미 전체 파라미터가 `requires_grad=True` 인 상태임
→ 아무 것도 얼리지(freeze) 않았기 때문에, 전체 파라미터가 기본적으로 학습 대상이 된다는 뜻
- 따라서 위 코드 함수는 유연하게 **전체 파인튜닝**도, **Gradual Unfreeze** 초기도 모두 커버할 수 있는 구조

```

# Fine-tuning 1차 (전체 fine-tuning + 보조 분류기 포함)
finetuned_googlenet = get_finetuned_googlenet(NUM_CLASSES, use_aux:

# 학습 실행
googlenet_result = train_with_early_stopping(
    model=finetuned_googlenet,
    model_name="googlenet_finetuned",
    train_loader=train_loader,
    val_loader=val_loader,
    num_epochs=50,
    patience=5
)

```

2. 2차 파인튜닝 (use_aux=True, freeze_base=True)

```

# Fine-tuning 2차 (Feature Extractor & Gradual Unfreeze 방식 한번에)

# Feature Extractor 방식
model_fe = get_finetuned_googlenet(NUM_CLASSES, use_aux=True, freeze_b
result_fe = train_with_early_stopping(
    model_fe,
    model_name="googlenet_feature_extractor",
    train_loader=train_loader,
    val_loader=val_loader,
    gradual_unfreeze=False
)

# Gradual Unfreeze 방식
model_gu = get_finetuned_googlenet(NUM_CLASSES, use_aux=True, freeze_k
result_gu = train_with_early_stopping(
    model_gu,
    model_name="googlenet_gradual_unfreeze",
    train_loader=train_loader,
    val_loader=val_loader,
    gradual_unfreeze=True,

```

```
unfreeze_at=5
)
```

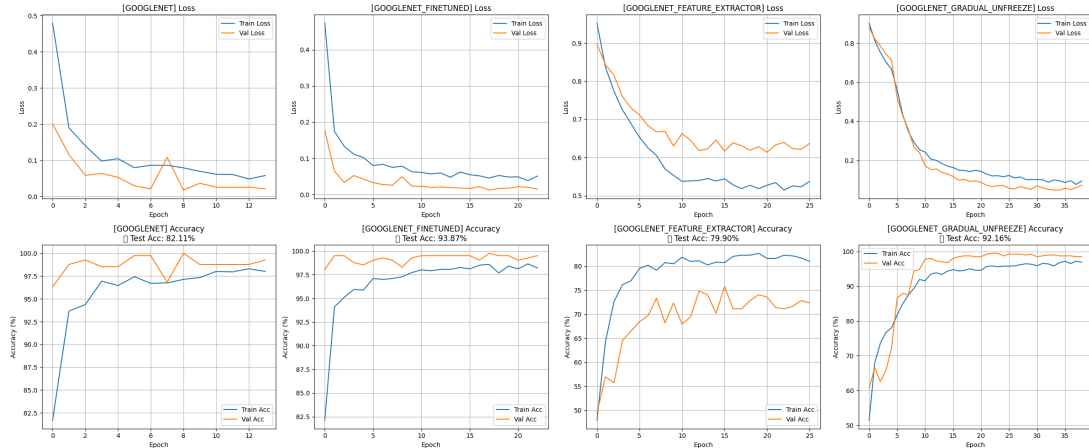
- 한번에 두 모델 생성하여 두 전략을 동시에 진행하였다.
 - **Feature Extractor 방식**
 - 백본(Conv Layer)은 고정(freeze) → `requires_grad = False`
 - 분류기(FC + aux1, aux2)만 학습
 - 즉, 기존 pretrained weight는 그대로 유지하면서 새로운 task에 맞는 출력 레이어만 학습
 - **Gradual Unfreeze**(점진적 파라미터 해제) 방식
 - 처음에는 Feature Extractor와 동일 → `freeze_base=True`
 - 지정된 epoch 이후, 백본도 `requires_grad=True`로 변경 → 전체 파인튜닝 전환
 - 학습이 안정화된 후 백본을 조금씩 푸는 전략
- ▼ 각 모델 학습 후 csv파일로 저장하였다.

```
# 드라이브 내 저장 경로
CSV_SAVE_DIR = "/content/drive/MyDrive/Colab Notebooks/[카카오 부
os.makedirs(CSV_SAVE_DIR, exist_ok=True)

def save_result_to_csv_if_not_exists(result_dict, filename, save_dir):
    path = os.path.join(save_dir, filename)
    if os.path.exists(path):
        print(f"이미 존재함: {path} → 저장 생략")
        return
    df = pd.DataFrame({
        "Epoch": list(range(1, len(result_dict["train_loss"]) + 1)),
        "Train Loss": result_dict["train_loss"],
        "Val Loss": result_dict["val_loss"],
        "Train Acc": result_dict["train_acc"],
        "Val Acc": result_dict["val_acc"],
    })
    df.to_csv(path, index=False)
    print(f"저장 완료: {path}")
```

```
save_result_to_csv_if_not_exists(googlenet_result, "googlenet_finetune
```

- 전체 모델 시각화
 - 각 모델을 학습 시킨 후 시각화를 진행하였음



GoogLeNet 계열 성능 요약 (최종 Epoch 기준)

Model	Train Acc	Val Acc	Test Acc
GoogLeNet (Pretrained)	98.01%	99.27%	82.11%
GoogLeNet (Fine-tuned)	98.21%	99.51%	93.87%
GoogLeNet (Feature Extractor)	81.01%	72.37%	79.90%
GoogLeNet (Gradual Unfreeze)	96.95%	98.53%	92.16%

- 파인튜닝을 적용한 모델이 기본 모델(GoogLeNet_Pretrained) 보다 훨씬 개선된 것이 있었으며, 반대로 개선이 안되고 오히려 테스트 정확도가 떨어진 모델이 발견되었다(**Feature Extractor**).
- Feature Extractor 방식에서 성능이 하락한 이유

Feature Extractor 방식은 GoogLeNet 모델의 백본(Backbone)을 고정(freeze)하고, 출력층(fc, aux1, aux2)만을 학습하는 방식이다. 이 전략은 일반적으로 데이터 양이 적고, 기존 사전학습된 특징이 새로운 도메인과 유사할 때 유효하다. 그러나 본 실험에서는 해당 방식이 오히려 Pretrained GoogLeNet보다 낮은 정확도를 보였다(Test Acc: 79.90% vs. 82.11%).

→ 이러한 결과는 다음과 같은 요인으로 설명할 수 있다

1. 학습 가능한 파라미터의 제한

Feature Extractor 방식은 **출력층**만 학습하기 때문에, 병해와 같은 특수한 이미지 도메인에 대해 충분한 표현 학습이 어려움 특히 패턴이 미묘하게 변

화하는 감자 병해 이미지에 대해 일반적인 백본 특징만으로는 성능이 제한됨 → 과적합 발생

2. Pretrained 모델의 일반화 능력

사전학습된 GoogLeNet은 ImageNet 데이터에 기반하여 다양한 시각 특징을 학습했기 때문에, 간단한 분류 문제에서는 Feature Extractor보다 오히려 더 일반적인 표현 능력을 제공할 수 있음

3. 데이터셋 도메인 간 차이 및 규모 이슈

본 프로젝트의 데이터셋은 클래스 수(3개)가 적고, 전체 이미지 수 또한 대규모가 아님

이러한 조건에서는 전체 파라미터를 조정할 수 있는 Fine-tuning 또는 Gradual Unfreeze 전략이 더 효과적으로 동작함

따라서 Feature Extractor 전략은 학습 범위가 제한적이기 때문에, 데이터의 특성과 도메인 간 차이를 충분히 반영하기 어려웠으며, 이는 전체적인 분류 성능의 하락으로 보임

3. Loss/Scheduler 변경

• Label Smoothing Loss 적용

◦ Loss Function은 정답에 얼마나 가까운지 수치화 하는 것이 핵심이다.

- `CrossEntropyLoss` : 정답 레이블만을 기준으로 확실히 맞춰야 한다고 학습
- `LabelSmoothingLoss` : 정답 이외 클래스도 약간의 확률을 부여해 과적합을 줄이는 방향으로 학습

◦ 핵심 특징

- `nn.CrossEntropyLoss(label_smoothing=0.1)` 적용
- 모델의 overconfidence 완화
- 작은 데이터셋에 유리함

◦ 코드

▼ `LabelSmoothingLoss` 통합 학습 함수 정의(기본모델 + 1차 파인튜닝)

```
def train_and_log(model, model_name, train_loader, val_loader, test_loader,
                  num_epochs=50, patience=5, min_delta=0.001,
                  loss_fn=None, collection_name=None):
```

```

model = model.to(device)
criterion = loss_fn if loss_fn else nn.CrossEntropyLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()))
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

# MongoDB 컬렉션 지정
collection = db[collection_name or model_name]

# 저장 경로
pth_path = os.path.join(BASE_DIR, f"{model_name}_best.pth")
csv_path = os.path.join(BASE_DIR, f"{model_name}_result.csv")
pt_path = os.path.join(BASE_DIR, f"{model_name}_probs.pt")

best_val_loss = float("inf")
counter = 0
logs = []

for epoch in range(num_epochs):
    model.train()
    start_time = time.strftime('%Y-%m-%d %H:%M:%S')
    correct, total, running_loss = 0, 0, 0.0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        if isinstance(outputs, tuple): outputs = outputs.logits
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)

    train_loss = running_loss / len(train_loader)
    train_acc = correct / total * 100

```

```

# 검증
model.eval()
val_loss, val_correct, val_total, val_running_loss = 0, 0, 0, 0.0
with torch.no_grad():
    for images, labels in val_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        if isinstance(outputs, tuple): outputs = outputs.logits
        loss = criterion(outputs, labels)
        val_running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        val_correct += (preds == labels).sum().item()
        val_total += labels.size(0)
val_loss = val_running_loss / len(val_loader)
val_acc = val_correct / val_total * 100
end_time = time.strftime('%Y-%m-%d %H:%M:%S')

is_best = val_loss < best_val_loss - min_delta
if is_best:
    best_val_loss = val_loss
    counter = 0
    torch.save(model.state_dict(), pth_path)
else:
    counter += 1
scheduler.step()

# MongoDB 저장
log = {
    "model": model_name,
    "epoch": epoch + 1,
    "start_time": start_time,
    "end_time": end_time,
    "train_loss": round(train_loss, 4),
    "val_loss": round(val_loss, 4),
    "train_acc": round(train_acc, 2),
    "val_acc": round(val_acc, 2),
    "is_best": is_best
}

```

```

collection.insert_one(log)
logs.append(log)

print(f"[{epoch+1:02d}] TrainAcc: {train_acc:.2f}% | ValAcc: {val_acc:.2f}%")
if counter >= patience:
    print("Early Stopping!")
    break

# CSV 저장
df = pd.DataFrame(logs)
df.to_csv(csv_path, index=False)

# Softmax 저장 (테스트셋)
model.load_state_dict(torch.load(pth_path))
model.eval()
probs_list, labels_list = [], []
with torch.no_grad():
    for images, labels in test_loader:
        images = images.to(device)
        outputs = model(images)
        if isinstance(outputs, tuple): outputs = outputs.logits
        probs = F.softmax(outputs, dim=1)
        probs_list.append(probs.cpu())
        labels_list.append(labels)
torch.save({"probs": torch.cat(probs_list), "labels": torch.cat(labels_list)}, pth_path)
print(f"저장 완료: {model_name} → .pth / .csv / .pt")

```

```

from torch.nn import functional as F

# Label Smoothing Loss 적용
loss_fn = nn.CrossEntropyLoss(label_smoothing=0.1)

# GoogLeNet base (Label Smoothing)
googlenet_smooth = models.googlenet(pretrained=True, aux_logits=True)
googlenet_smooth.fc = nn.Linear(googlenet_smooth.fc.in_features, googlenet_smooth.fc.out_features)
googlenet_smooth.aux1.fc2 = nn.Linear(googlenet_smooth.aux1.fc2.in_features, googlenet_smooth.aux1.fc2.out_features)
googlenet_smooth.aux2.fc2 = nn.Linear(googlenet_smooth.aux2.fc2.in_features, googlenet_smooth.aux2.fc2.out_features)
"""

```


보조 분류기 무시할 거면 왜 aux1,aux2에 대해 fc2를 설정했는가?

> 구조적으로 NUM_CLASSES에 맞게 보조 분류기의 출력층만 맞추는 것임
학습함수 `train_and_log()` 내부에서 `outputs = outputs.logits` 처리를
실제로 aux출력은 사용을 안함!!

-> aux1,aux2는 구조상 존재만 하고 학습&출력에는 영향을 안끼침

"""

🏃 학습 시작

```
train_and_log(  
    model=googlenet_smooth,  
    model_name="googlenet_smooth",  
    train_loader=train_loader,  
    val_loader=val_loader,  
    test_loader=test_loader,  
    loss_fn=loss_fn,  
    collection_name="googlenet_smooth" # 별도 컬렉션  
)
```

1차 파인튜닝: 전체 학습 + 보조 분류기(aux) 포함

```
googlenet_finetuned_smooth = models.googlenet(pretrained=True)  
googlenet_finetuned_smooth.fc = nn.Linear(googlenet_finetuned_s  
googlenet_finetuned_smooth.aux1.fc2 = nn.Linear(googlenet_finet  
googlenet_finetuned_smooth.aux2.fc2 = nn.Linear(googlenet_finet
```

전체 파라미터 학습 (requires_grad=True)

```
for param in googlenet_finetuned_smooth.parameters():  
    param.requires_grad = True
```

학습 실행

```
train_and_log(  
    model=googlenet_finetuned_smooth,  
    model_name="googlenet_finetuned_smooth",  
    train_loader=train_loader,  
    val_loader=val_loader,  
    test_loader=test_loader,  
    loss_fn=loss_fn, # Label Smoothing Loss
```

```
collection_name="googlenet_finetuned_smooth" # 별도 컬렉션
)
```

▼ LabelSmoothingLoss통합 학습 함수 정의 (Gradual Unfreeze _ 2차 파인 튜닝)

```
def train_and_log_with_unfreeze(model, model_name, train_loader,
                                num_epochs=50, patience=5, min_delta=0.001,
                                loss_fn=None, collection_name=None,
                                gradual_unfreeze=False, unfreeze_at=5):
    model = model.to(device)
    criterion = loss_fn if loss_fn else nn.CrossEntropyLoss()
    """
    외부에서 loss_fn을 넘겨주면 그것을 사용하고,
    넘겨주지 않으면 기본값인 nn.CrossEntropyLoss()를 사용한다.
    """

    optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()))
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)
    collection = db[collection_name or model_name]

   .pth_path = os.path.join(BASE_DIR, f"{model_name}_best.pth")
    csv_path = os.path.join(BASE_DIR, f"{model_name}_result.csv")
    pt_path = os.path.join(BASE_DIR, f"{model_name}_probs.pt")

    best_val_loss = float("inf")
    counter = 0
    logs = []

    for epoch in range(num_epochs):
        model.train()
        start_time = time.strftime('%Y-%m-%d %H:%M:%S')

        # 🗝 Gradual Unfreeze
        if gradual_unfreeze and epoch == unfreeze_at:
            print(f"🗝 Gradual Unfreeze 시작 (Epoch {epoch})")
            for param in model.parameters():
                param.requires_grad = True
```

```

optimizer = optim.Adam(model.parameters(), lr=1e-5)

running_loss, correct, total = 0.0, 0, 0
for images, labels in train_loader:
    images, labels = images.to(device), labels.to(device)
    optimizer.zero_grad()
    outputs = model(images)
    if isinstance(outputs, tuple): outputs = outputs.logits
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    running_loss += loss.item()
    _, preds = torch.max(outputs, 1)
    correct += (preds == labels).sum().item()
    total += labels.size(0)

train_loss = running_loss / len(train_loader)
train_acc = correct / total * 100

# 검증
val_loss, val_correct, val_total, val_running_loss = 0, 0, 0, 0.0
model.eval()
with torch.no_grad():
    for images, labels in val_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        if isinstance(outputs, tuple): outputs = outputs.logits
        loss = criterion(outputs, labels)
        val_running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        val_correct += (preds == labels).sum().item()
        val_total += labels.size(0)
val_loss = val_running_loss / len(val_loader)
val_acc = val_correct / val_total * 100
end_time = time.strftime('%Y-%m-%d %H:%M:%S')

is_best = val_loss < best_val_loss - min_delta
if is_best:

```

```

        best_val_loss = val_loss
        counter = 0
        torch.save(model.state_dict(), pth_path)
    else:
        counter += 1
    scheduler.step()

# MongoDB 저장
log = {
    "model": model_name, "epoch": epoch+1,
    "start_time": start_time, "end_time": end_time,
    "train_loss": round(train_loss, 4), "val_loss": round(val_loss, 4),
    "train_acc": round(train_acc, 2), "val_acc": round(val_acc, 2),
    "is_best": is_best
}
collection.insert_one(log)
logs.append(log)

print(f"[{epoch+1:02d}] TrainAcc: {train_acc:.2f}% | ValAcc: {val_acc:.2f}%")
if counter >= patience:
    print("Early Stopping!")
    break

# CSV 저장
pd.DataFrame(logs).to_csv(csv_path, index=False)

# Softmax 저장
model.load_state_dict(torch.load(pth_path))
model.eval()
probs_list, labels_list = [], []
with torch.no_grad():
    for images, labels in test_loader:
        images = images.to(device)
        outputs = model(images)
        if isinstance(outputs, tuple): outputs = outputs.logits
        probs = F.softmax(outputs, dim=1)
        probs_list.append(probs.cpu())
        labels_list.append(labels)

```

```
torch.save({"probs": torch.cat(probs_list), "labels": torch.cat(labels_list)},
print(f"저장 완료: {model_name} → .pth / .csv / .pt")
```

```
from torchvision import models
```

```
# 2차 파인튜닝: Gradual Unfreeze
```

```
googlenet_gu_smooth = models.googlenet(pretrained=True, aux_logits=True)
googlenet_gu_smooth.fc = nn.Linear(googlenet_gu_smooth.fc.in_features, googlenet_gu_smooth.fc.out_features)
googlenet_gu_smooth.aux1.fc2 = nn.Linear(googlenet_gu_smooth.aux1.fc2.in_features, googlenet_gu_smooth.aux1.fc2.out_features)
googlenet_gu_smooth.aux2.fc2 = nn.Linear(googlenet_gu_smooth.aux2.fc2.in_features, googlenet_gu_smooth.aux2.fc2.out_features)
```

```
# Conv 레이어는 초기 freeze, fc/aux만 학습
```

```
for name, param in googlenet_gu_smooth.named_parameters():
    if "fc" in name or "aux" in name:
        param.requires_grad = True
    else:
        param.requires_grad = False
```

```
# 손실함수 정의 (함수 바깥)
```

```
loss_fn = nn.CrossEntropyLoss(label_smoothing=0.1)
```

```
# loss_fn정의를 함수 바깥에 존재해야함
```

```
# train_and_log_with_unfreeze() 함수 안에서 다시 정의하면 안됨
```

```
# 함수 내부에서 loss_fn을 다시 정의
```

```
# Gradual Unfreeze 학습 실행
```

```
train_and_log_with_unfreeze(
    model=googlenet_gu_smooth,
    model_name="googlenet_gu_smooth",
    train_loader=train_loader,
    val_loader=val_loader,
    test_loader=test_loader,
    loss_fn=loss_fn, # Label Smoothing 적용
    collection_name="googlenet_gu_smooth", # MongoDB 컬렉션명
    gradual_unfreeze=True, # 점진적 Unfreeze 사용
    unfreeze_at=5 # 5 epoch 후 전체 layer 활성화
)
```

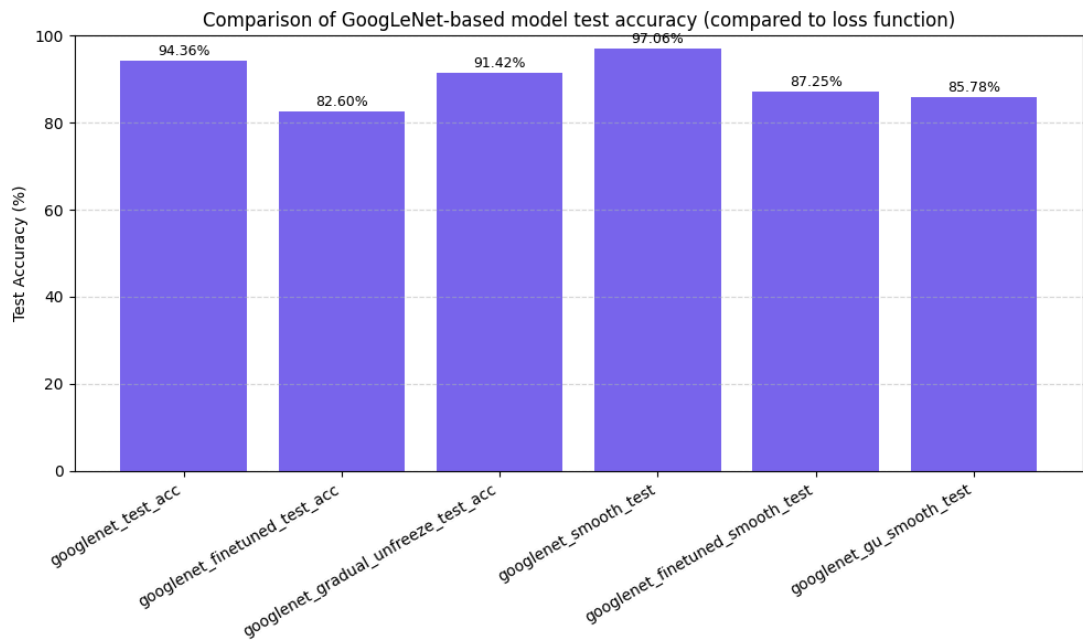
- 아래 결과는 **Label Smoothing Loss** 를 적용하고 난 후 테스트 정확도 이다.

```

# googlenet_smooth_test 테스트 정확도: 97.06% (MongoDB 저장 완료)
# googlenet_finnetuned_smooth_test 테스트 정확도: 87.25% (MongoDB 저장 완료)
# googlenet_gu_smooth_test 테스트 정확도: 85.78% (MongoDB 저장 완료)

```

- 아래 막대 그래프는 **Label Smoothing Loss** 유무에 따른 정확도 비교를 위해 시각화를 진행했다.(_smooth_가 **Label Smoothing Loss** 적용한 모델)



- 원래 의도는 Label Smoothing을 도입하여 모델이 **과적합을 방지하고 일반화 성능을 높이도록** 개선하려던 것이었는데, 오히려 2차 파인튜닝(gradual unfreeze)에서는 정확도가 **낮아지는** 결과가 나왔다.
- Gradual Unfreeze & Smoothing 조합이 안좋은 이유

구분	Gradual Unfreeze	Label Smoothing
목적	feature 보존 + 안정적 미세조정	과적합 방지 + regularization
특징	점진적으로 학습 강도 ↑	학습 강도를 ↓시키는 경향
충돌	학습의 민감한 초기 단계에서, 학습 대상이 너무 soft해져 → 유의미한 파라미터 업데이트가 어려움	

→ Gradual Unfreeze 는 점진적인 학습이 필요한데 **Label Smoothing은 학습 자체를 더 약하게 만들기 때문에** 파라미터가 풀린 후에도 모델이 제대로 학습하지 못한것으로 확인됨

- **Label Smoothing + ReduceLROnPlateau 스케줄러 적용**

- **Reduce Learning Rate On Plateau** : 학습이 평탄(plateau) 상태에 도달했을 때, 학습률(Learning Rate)을 자동으로 줄여주는 스케줄러

- **핵심 특징**

- 일정 epoch 동안 val_loss 개선 없을 시 learning rate 감소
- StepLR보다 더 부드럽고 정밀한 조절
- min_delta, factor, patience 조정 가능

```
scheduler = ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=10)
```

- 검증 손실이 2폭동안 개선 되지 않으면, 학습률을 현재의 50%로 감소시킨다는 뜻
- Label Smoothing은 일반적으로 학습을 부드럽게 유도하기에 고정적인 StepLR 보다 **동적으로 반응하는 ResuceLROnPlateu**로 진행

- 코드

- ▼ googlenet_smooth_rlrop (기존 구글넷 모델 + Smoothing + ReduceLROnPlateau)

```
def train_and_log(model, model_name, train_loader, val_loader, test_loader,
                  num_epochs=50, patience=5, min_delta=0.001,
                  loss_fn=None, collection_name=None):

    model = model.to(device)
    criterion = loss_fn if loss_fn else nn.CrossEntropyLoss()
    optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()))

    # ReduceLROnPlateau 스케줄러 적용
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, patience=10,
                                                    factor=0.5, patience=2,
                                                    verbose=True, min_lr=1e-6)

    collection = db[collection_name or model_name]
    pth_path = os.path.join(BASE_DIR, f"{model_name}_best.pth")
    csv_path = os.path.join(BASE_DIR, f"{model_name}_result.csv")
    pt_path = os.path.join(BASE_DIR, f"{model_name}_probs.pt")
```

```

best_val_loss = float("inf")
counter = 0
logs = []

for epoch in range(num_epochs):
    model.train()
    start_time = time.strftime('%Y-%m-%d %H:%M:%S')
    running_loss, correct, total = 0.0, 0, 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        if isinstance(outputs, tuple): outputs = outputs.logits
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)

    train_loss = running_loss / len(train_loader)
    train_acc = correct / total * 100

    # 검증
    model.eval()
    val_loss, val_correct, val_total, val_running_loss = 0, 0, 0, 0.0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            if isinstance(outputs, tuple): outputs = outputs.logits
            loss = criterion(outputs, labels)
            val_running_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            val_correct += (preds == labels).sum().item()
            val_total += labels.size(0)

```



```

val_loss = val_running_loss / len(val_loader)
val_acc = val_correct / val_total * 100
end_time = time.strftime('%Y-%m-%d %H:%M:%S')

# 스케줄러 업데이트
scheduler.step(val_loss)

is_best = val_loss < best_val_loss - min_delta
if is_best:
    best_val_loss = val_loss
    counter = 0
    torch.save(model.state_dict(), pth_path)
else:
    counter += 1

# MongoDB 저장
log = {
    "model": model_name,
    "epoch": epoch + 1,
    "start_time": start_time,
    "end_time": end_time,
    "train_loss": round(train_loss, 4),
    "val_loss": round(val_loss, 4),
    "train_acc": round(train_acc, 2),
    "val_acc": round(val_acc, 2),
    "is_best": is_best
}
collection.insert_one(log)
logs.append(log)

print(f"[{epoch+1:02d}] TrainAcc: {train_acc:.2f}% | ValAcc: {val_acc:.2f}%")
if counter >= patience:
    print("Early Stopping!")
    break

# CSV 저장
pd.DataFrame(logs).to_csv(csv_path, index=False)

```

```

# Softmax 저장
model.load_state_dict(torch.load(pth_path))
model.eval()
probs_list, labels_list = [], []
with torch.no_grad():
    for images, labels in test_loader:
        images = images.to(device)
        outputs = model(images)
        if isinstance(outputs, tuple): outputs = outputs.logits
        probs = F.softmax(outputs, dim=1)
        probs_list.append(probs.cpu())
        labels_list.append(labels)
torch.save({"probs": torch.cat(probs_list), "labels": torch.cat(labels_list)}, pth_path)
print(f"저장 완료: {model_name} → .pth / .csv / .pt")

```

```

from torchvision import models
import torch.nn as nn

# 손실 함수: Label Smoothing
loss_fn = nn.CrossEntropyLoss(label_smoothing=0.1)

# GoogLeNet base + Label Smoothing + ReduceLROnPlateau
googlenet_smooth_rlrop = models.googlenet(pretrained=True, aux=True)
googlenet_smooth_rlrop.fc = nn.Linear(googlenet_smooth_rlrop.fc.in_features, googlenet_smooth_rlrop.fc.out_features)
googlenet_smooth_rlrop.aux1.fc2 = nn.Linear(googlenet_smooth_rlrop.aux1.fc2.in_features, googlenet_smooth_rlrop.aux1.fc2.out_features)
googlenet_smooth_rlrop.aux2.fc2 = nn.Linear(googlenet_smooth_rlrop.aux2.fc2.in_features, googlenet_smooth_rlrop.aux2.fc2.out_features)

# 전체 파라미터 학습 가능하도록 설정
for param in googlenet_smooth_rlrop.parameters():
    param.requires_grad = True

# 학습 시작
train_and_log(
    model=googlenet_smooth_rlrop,
    model_name="googlenet_smooth_rlrop",
    train_loader=train_loader,
    val_loader=val_loader,
    test_loader=test_loader,

```

```

    loss_fn=loss_fn,
    collection_name="googlenet_smooth_rlrop"
)

```

```

from torchvision import models
import torch.nn as nn

# 손실 함수: Label Smoothing 유지
loss_fn = nn.CrossEntropyLoss(label_smoothing=0.1)

# GoogLeNet + 1차 파인튜닝 (ReduceLROnPlateau 포함)
googlenet_finetuned_smooth_rlrop = models.googlenet(pretrained=True)
googlenet_finetuned_smooth_rlrop.fc = nn.Linear(googlenet_finetuned_smooth_rlrop.fc.in_features, googlenet_finetuned_smooth_rlrop.fc.out_features)
googlenet_finetuned_smooth_rlrop.aux1.fc2 = nn.Linear(googlenet_finetuned_smooth_rlrop.aux1.fc2.in_features, googlenet_finetuned_smooth_rlrop.aux1.fc2.out_features)
googlenet_finetuned_smooth_rlrop.aux2.fc2 = nn.Linear(googlenet_finetuned_smooth_rlrop.aux2.fc2.in_features, googlenet_finetuned_smooth_rlrop.aux2.fc2.out_features)

# 전체 파라미터 학습 가능하도록 설정
for param in googlenet_finetuned_smooth_rlrop.parameters():
    param.requires_grad = True

# 학습 실행
train_and_log(
    model=googlenet_finetuned_smooth_rlrop,
    model_name="googlenet_finetuned_smooth_rlrop",
    train_loader=train_loader,
    val_loader=val_loader,
    test_loader=test_loader,
    loss_fn=loss_fn,
    collection_name="googlenet_finetuned_smooth_rlrop"
)

```

▼ GoogLeNet + Gradual Unfreeze + Label Smoothing + ReduceLROnPlateau 통합 셀

```

def train_and_log_with_unfreeze_rlrop(model, model_name, train_loader, val_loader, test_loader, num_epochs=50, patience=5, min_delta=0.01, loss_fn=None, collection_name=None, gradual_unfreeze=False, unfreeze_at=5):

```

```

model = model.to(device)
criterion = loss_fn if loss_fn else nn.CrossEntropyLoss()

optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.

# ReduceLROnPlateau 스케줄러 적용
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, n
                                                    factor=0.5, patience=2,
                                                    verbose=True, min_lr=1e-6)

collection = db[collection_name or model_name]
pth_path = os.path.join(BASE_DIR, f"{model_name}_best.pth")
csv_path = os.path.join(BASE_DIR, f"{model_name}_result.csv")
pt_path = os.path.join(BASE_DIR, f"{model_name}_probs.pt")

best_val_loss = float("inf")
counter = 0
logs = []

for epoch in range(num_epochs):
    model.train()
    start_time = time.strftime('%Y-%m-%d %H:%M:%S')

    # Gradual Unfreeze
    if gradual_unfreeze and epoch == unfreeze_at:
        print(f"Gradual Unfreeze 시작 (Epoch {epoch})")
        for param in model.parameters():
            param.requires_grad = True
        optimizer = optim.Adam(model.parameters(), lr=1e-5)

    running_loss, correct, total = 0.0, 0, 0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        if isinstance(outputs, tuple): outputs = outputs.logits
        loss = criterion(outputs, labels)
        loss.backward()

```

```

optimizer.step()
running_loss += loss.item()
_, preds = torch.max(outputs, 1)
correct += (preds == labels).sum().item()
total += labels.size(0)

train_loss = running_loss / len(train_loader)
train_acc = correct / total * 100

# 검증
model.eval()
val_loss, val_correct, val_total, val_running_loss = 0, 0, 0, 0.0
with torch.no_grad():
    for images, labels in val_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        if isinstance(outputs, tuple): outputs = outputs.logits
        loss = criterion(outputs, labels)
        val_running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        val_correct += (preds == labels).sum().item()
        val_total += labels.size(0)
val_loss = val_running_loss / len(val_loader)
val_acc = val_correct / val_total * 100
end_time = time.strftime('%Y-%m-%d %H:%M:%S')

# 스케줄러 적용
scheduler.step(val_loss)

is_best = val_loss < best_val_loss - min_delta
if is_best:
    best_val_loss = val_loss
    counter = 0
    torch.save(model.state_dict(), pth_path)
else:
    counter += 1

# MongoDB 저장

```

```

log = {
    "model": model_name, "epoch": epoch + 1,
    "start_time": start_time, "end_time": end_time,
    "train_loss": round(train_loss, 4), "val_loss": round(val_loss, 4),
    "train_acc": round(train_acc, 2), "val_acc": round(val_acc, 2),
    "is_best": is_best
}
collection.insert_one(log)
logs.append(log)

print(f"[{epoch+1:02d}] TrainAcc: {train_acc:.2f}% | ValAcc: {val_acc:.2f}%")
if counter >= patience:
    print("Early Stopping!")
    break

# CSV 저장
pd.DataFrame(logs).to_csv(csv_path, index=False)

# Softmax 저장
model.load_state_dict(torch.load(pth_path))
model.eval()
probs_list, labels_list = [], []
with torch.no_grad():
    for images, labels in test_loader:
        images = images.to(device)
        outputs = model(images)
        if isinstance(outputs, tuple): outputs = outputs.logits
        probs = F.softmax(outputs, dim=1)
        probs_list.append(probs.cpu())
        labels_list.append(labels)
torch.save({"probs": torch.cat(probs_list), "labels": torch.cat(labels_list)}, pth_path)
print(f"저장 완료: {model_name} → .pth / .csv / .pt")

```

```

from torchvision import models
import torch.nn as nn

```

```

# Label Smoothing 손실 함수
loss_fn = nn.CrossEntropyLoss(label_smoothing=0.1)

```

```

# GoogLeNet + Gradual Unfreeze + LabelSmoothing + ReduceLROnPlateau
googlenet_gu_smooth_rlrop = models.googlenet(pretrained=True,
googlenet_gu_smooth_rlrop.fc = nn.Linear(googlenet_gu_smooth_rlrop.fc.in_features,
googlenet_gu_smooth_rlrop.aux1.fc2 = nn.Linear(googlenet_gu_smooth_rlrop.aux1.fc2.in_features,
googlenet_gu_smooth_rlrop.aux2.fc2 = nn.Linear(googlenet_gu_smooth_rlrop.aux2.fc2.in_features,

# Conv 레이어 초기 freeze
for name, param in googlenet_gu_smooth_rlrop.named_parameters():
    if "fc" in name or "aux" in name:
        param.requires_grad = True
    else:
        param.requires_grad = False

train_and_log_with_unfreeze_rlrop(
    model=googlenet_gu_smooth_rlrop,
    model_name="googlenet_gu_smooth_rlrop",
    train_loader=train_loader,
    val_loader=val_loader,
    test_loader=test_loader,
    loss_fn=loss_fn,
    collection_name="googlenet_gu_smooth_rlrop", # MongoDB 별명
    gradual_unfreeze=True, # 점진적 파라미터 해제
    unfreeze_at=5          # 5 epoch 후 전체 unfrozen
)

```

▼ 테스트 정확도 평가 및 MongoDB에 저장

```

from pymongo import MongoClient
import certifi

# MongoDB 연결 (이미 연결된 경우 생략 가능)
MONGO_URI = "mongodb+srv://dnjsgh1820:dkf1ckrp@cluster-ktb.mongodb.net/"
client = MongoClient(MONGO_URI, tlsCAFile=certifi.where())
db = client["model_logs"]

# 테스트 정확도 계산 함수
def evaluate_test_accuracy(model, dataloader):

```

```

model.eval()
correct, total = 0, 0
with torch.no_grad():
    for images, labels in dataloader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        if isinstance(outputs, tuple): outputs = outputs.logits
        _, preds = torch.max(outputs, 1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)
return round(correct / total * 100, 2)

```

MongoDB에 저장 함수

```

def log_test_accuracy_to_mongodb(model_name, model, test_loader):
    # 먼저 테스트 정확도 계산
    test_acc = evaluate_test_accuracy(model, test_loader)

    # 이미 같은 type='final_test' 로그가 있는지 확인
    collection = db[model_name]
    existing = collection.find_one({"type": "final_test"})

    if existing:
        print(f"{model_name} 이미 테스트 정확도가 저장되어 있음 → 덮어쓰기")
        print(f"{model_name} 테스트 정확도: {test_acc}% (MongoDB 저장)")
        return # 중복 방지용 종료

    # 새로 저장
    result = {
        "model": model_name,
        "test_accuracy": test_acc,
        "type": "final_test"
    }
    collection.insert_one(result)
    print(f"{model_name} 테스트 정확도: {test_acc}% (MongoDB 저장)")

```

```

model_names = [
    "googlenet_smooth_rlrop",
    "googlenet_finetuned_smooth_rlrop",

```



```

    "googlenet_gu_smooth_rlrop" # 새로 추가된 모델
]

# 공통 함수: 모델 로드
def load_trained_model(model_name, weights_path, num_classes):
    model = models.googlenet(pretrained=False, aux_logits=True)
    model.fc = nn.Linear(model.fc.in_features, num_classes)
    model.aux1.fc2 = nn.Linear(model.aux1.fc2.in_features, num_classes)
    model.aux2.fc2 = nn.Linear(model.aux2.fc2.in_features, num_classes)
    model.load_state_dict(torch.load(weights_path, map_location=device))
    model.to(device)
    model.eval()
    return model

# 정확도 측정 및 MongoDB 저장
for model_name in model_names:
    weights_path = os.path.join(BASE_DIR, f"{model_name}_best.pt")
    model = load_trained_model(model_name, weights_path, NUM_CLASSES)
    log_test_accuracy_to_mongodb(model_name, model, test_loader)

```

```

# 3그룹 분류 시각화 코드
import matplotlib.pyplot as plt

def visualize_test_accuracy_by_group(db):
    # 3그룹 분류
    groups = {
        "Fine tuning": [
            "googlenet_test_acc",
            "googlenet_finetuned_test_acc",
            "googlenet_gradual_unfreeze_test_acc"
        ],
        "LabelSmoothing Only": [
            "googlenet_smooth_test",
            "googlenet_finetuned_smooth_test",
            "googlenet_gu_smooth_test"
        ],
        "LabelSmoothing + ReduceLROnPlateau": [

```

```

        "googlenet_smooth_rlrop",
        "googlenet_finetuned_smooth_rlrop",
        "googlenet_gu_smooth_rlrop"
    ]
}

# 3개의 서브플롯 생성
fig, axs = plt.subplots(1, 3, figsize=(18, 6), sharey=True)

for i, (group_name, model_names) in enumerate(groups.items()):
    accs = []
    labels = []

    for name in model_names:
        acc = None
        if name in db.list_collection_names():
            result = db[name].find_one({"type": "final_test"})
            if result and "test_accuracy" in result:
                acc = result["test_accuracy"]
            accs.append(acc if acc is not None else 0.0)
            labels.append(name.replace("_test", ""))

    # 막대그래프
    bars = axs[i].bar(labels, accs, color='mediumseagreen')
    axs[i].set_title(group_name, fontsize=13)
    axs[i].set_ylim(0, 100)
    axs[i].tick_params(axis='x', rotation=20)
    axs[i].grid(axis='y', linestyle='--', alpha=0.3)
    axs[i].set_ylabel("Test Accuracy (%)\" if i == 0 else "")

    for bar, acc in zip(bars, accs):
        label = f"{acc:.2f}%" if acc > 0 else "N/A"
        axs[i].text(bar.get_x() + bar.get_width() / 2, bar.get_height(),
                    label, ha='center', va='bottom', fontsize=9)

plt.suptitle("GoogLeNet Model Accuracy Comparison by Strateg
plt.tight_layout()

```

```
plt.show()
visualize_test_accuracy_by_group(db)
```

▼ 테스트 확인 & 시각화 진행

```
import matplotlib.pyplot as plt

def visualize_all_test accuracies_flexible(model_names, db):
    test_accs = []
    labels = []

    for name in model_names:
        acc = None

        # 우선 name+'_test' 컬렉션에서 찾아보기
        test_collection_name = name + "_test"
        if test_collection_name in db.list_collection_names():
            result = db[test_collection_name].find_one({"type": "final_test"})
            if result and "test_accuracy" in result:
                acc = result["test_accuracy"]

        # 그게 안되면 name 컬렉션에서 찾아보기
        if acc is None and name in db.list_collection_names():
            result = db[name].find_one({"type": "final_test"})
            if result and "test_accuracy" in result:
                acc = result["test_accuracy"]

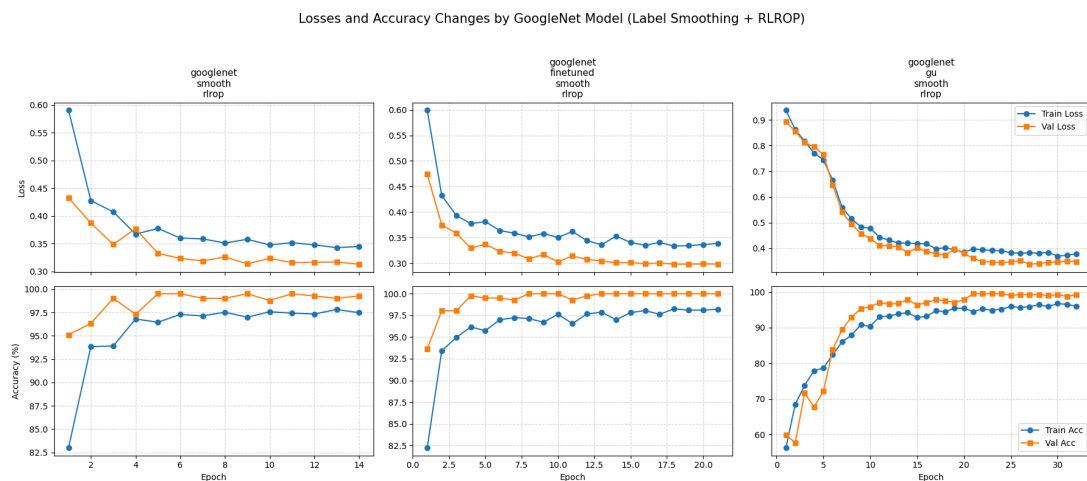
        test_accs.append(acc if acc is not None else 0.0)
        labels.append(name)

    # 시각화
    plt.figure(figsize=(12, 6))
    bars = plt.bar(labels, test_accs, color='cornflowerblue')
    for bar, acc in zip(bars, test_accs):
        label = f"{acc:.2f}%" if acc > 0 else "N/A"
        plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 1,
                 label, ha='center', va='bottom', fontsize=9)
```

```
plt.ylim(0, 100)
plt.ylabel("Test Accuracy (%)")
plt.title("GoogLeNet Overall Model 9 Testing Accuracy Comparison")
plt.xticks(rotation=10, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.2)
plt.tight_layout()
plt.show()
```

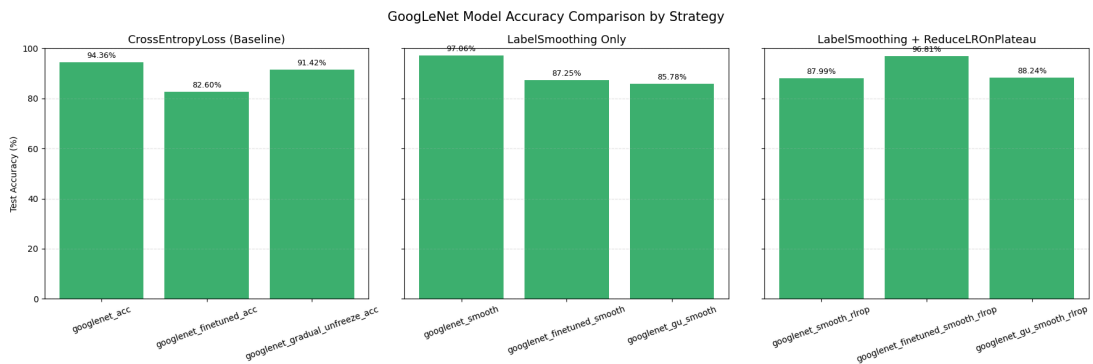
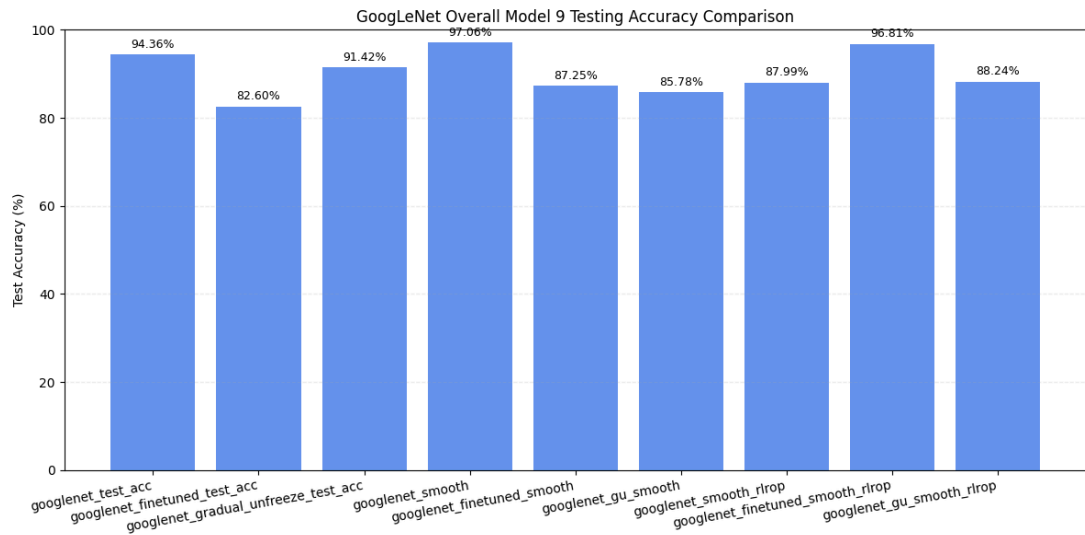
```
all_9_models = [
    "googlenet_test_acc",
    "googlenet_finetuned_test_acc",
    "googlenet_gradual_unfreeze_test_acc",
    "googlenet_smooth",
    "googlenet_finetuned_smooth",
    "googlenet_gu_smooth",
    "googlenet_smooth_rlrop",
    "googlenet_finetuned_smooth_rlrop",
    "googlenet_gu_smooth_rlrop"
]
```

```
visualize_all_test accuracies_flexible(all_9_models, db)
```



```
⚠ googlenet_smooth_rlrop 이미 테스트 정확도가 저장되어 있음 → 덮어쓰기 생략
✖ googlenet_smooth_rlrop 테스트 정확도: 87.99% (MongoDB 저장 완료)
⚠ googlenet_finetuned_smooth_rlrop 이미 테스트 정확도가 저장되어 있음 → 덮어쓰기 생략
✖ googlenet_finetuned_smooth_rlrop 테스트 정확도: 96.81% (MongoDB 저장 완료)
⚠ googlenet_gu_smooth_rlrop 이미 테스트 정확도가 저장되어 있음 → 덮어쓰기 생략
✖ googlenet_gu_smooth_rlrop 테스트 정확도: 88.24% (MongoDB 저장 완료)
```

- 총 9가지 모델에 대하여 총 테스트 정확도를 시각화 하였다(아래).



그룹별 시각화 진행

5. 실험 결과

- 모델별 테스트 정확도 결과는 아래와 같다

모델명	Test Accuracy (%)
googlenet	94.36
googlenet_finetuned	82.60
googlenet_gradual_unfreeze	91.42
googlenet_smooth	97.06
googlenet_finetuned_smooth	87.25
googlenet_gu_smooth	85.78
googlenet_smooth_rlrp	87.99
googlenet_finetuned_smooth_rlrp	96.81
googlenet_gu_smooth_rlrp	88.24

- 인사이트
 - **Label Smoothing** 은 일반적인 GoogLeNet 학습보다 **분명한 성능 향상을** 보임 (Base 기준 94.36 → 97.06%)
 - **Gradual Unfreeze** 는 정규화 계열 기법(Loss Smoothing, ReduceLROnPlateau 등)과의 조합 시 **과도한 일반화로 성능 하락** 가능성
 - **ReduceLROnPlateau** 는 overfitting 방지, val_loss 안정화에 유리함 → 일부 모델에서 큰 폭의 정확도 상승
- 고려사항
 - **googlenet_finetuned** 모델은 오히려 정확도 저하 → **초기 학습률 / 과적합** 가능성 추정
 - Gradual Unfreeze 전략은 **label smoothing 없이 적용** 시 더 유리할 수도 있음
 - 정규화/스케줄링 기법이 항상 이점이 되지 않음을 보여주는 예시 포함

6. 결론

- 본 프로젝트는 여러 모델기법중, 낮은 정확도를 보인 GoogLeNet의 기본 구조에서 출발하여, 보조 분류기 활성화, Gradual Unfreeze, Label Smoothing, 학습률 스케줄링 (ReduceLROnPlateau) 기법을 점진적으로 적용하면서 모델 성능을 단계적으로 향상시키는 것을 목표로 하였음

특히 **Finetuning+Label Smoothing + ReduceLROnPlateau** 조합은 작은 데이터 셋 환경에서 효과적이었으며, GoogLeNet 1차 파인튜닝 모델에서 높은 테스트 정확도인 96.81%를 달성하였음

이는 전체적인 파라미터 학습을 통해 사전학습된 백본(Backbone) + Fully Connected + 보조 분류기(aux) 가 모두 업데이트 되면서 모델의 표현력과 적응력이 최대로 발휘된 것으로 보임

추가로 **ReduceLROnPlateau** 가 일정 Epoch 이후 val_loss 개선이 멈추면 학습률을 절반으로 낮추어 더 정밀한 학습으로 전환된 점을 고려해 과적합 타이밍을 잡아준 것으로 보임
하지만 일반 GoogLeNet 모델에서는 오히려 정확도가 떨어지는 것을 확인할 수 있는데 (97.06%→87.99%), 이는 데이터의 클래스 수가 3개로 적기에 오히려 **Label Smoothing** 의 정규화 효과 과다로 보임


또한 일반 GoogLeNet 모델은 보조분류기(aux)가 비활성화 된 상태인데, **Label Smoothing**이 **과도하게 confidence**를 낮추고 ReduceLROnPlateau가 이를 바탕으로 조기에 learning rate를 떨어뜨리면서 결과적으로 **충분한 학습을 방해한 것으로 보임**

따라서 최적 전략은 기본 **GoogLeNet** 모델에 **Label Smoothing 적용** (googlenet_smooth) 하거나 **1차 파인튜닝 + Label Smoothing + ReduceLROnPlateau 조합**(googlenet_finetuned_smooth_rlrop)으로 하는 것이 적절하다고 판단된다.

코드 링크:

- ResNet50 모델 학습 & 데이터 증강_비교 테스트


Google Colab

 <https://colab.research.google.com/drive/1KaoYjCGbDyiXGhz4ub2yvm5UIFvT-TE9?usp=sharing>



- 이외 여러 모델 학습 & 비교 코드


Google Colab

 https://colab.research.google.com/drive/1TswtCKCh0EAiYc3kv_udFya8xf6MnGIN?usp=sharing



- 최종 모델링 & 학습 코드

Google Colab

 https://colab.research.google.com/drive/1-8qLkn-sNGnwFZhwF_BoKulnyJPNzWt8?usp=sharing

