「데이터 증강 기법의 적용 유무에 따른 이미지 분류 성능 분석」

요약

일반적으로 데이터 증강(Data Augmentation) 기법은 보유한 데이터를 변형하여 데이터의 양을 늘리고, 모델의 일반화 성능을 높이는 것을 목적으로 한다. 그러나 반드시 데이터 증강 기법이 데이터 품질 향상으로 이어지지는 않는다. 예컨대 밝기 변형의 경우, 너무 어둡거나 밝게 조정하면 오히려 원본 데이터의 특징이 왜곡될 수 있다. 본 연구에서는 RESNET50 모델을 파인튜닝하여 쌀의 질병 데이터를 분류하는 실험을 진행하였다. 실험 결과 데이터 증강을 적용하지 않았을 때 오히려 더 높은 성능이 나타났다. 이를 확인하기 위해 추가로 쌀 품종 데이터를 사용하여 실험한 결과, 같은 현상이 관찰되었다. 따라서 본 연구에서는 데이터 증강 기법의 효과를 구체적으로 분석하기 위해 각 회전 변형 방법을 조사하고, 변형의 유형과 강도에 따라 데이터 품질 향상에 어느 정도의 영향을 미치는지를 분석하였다.

Ⅰ. 서론

1.1 연구 배경 및 필요성

이미지 데이터 증강은 딥러닝 모델의 일반화 성능을 향상시키기 위한 대표적인 기법 중 하나로, 회전, 좌우 반전, 밝기 조절 등 다양한 방법이 존재한다. 일반적으로 데이터 증강을 수행하면 데이터셋의 다양성이 증가하여 모델의 성능이 향상된다고 알려져 있다. 그러나 본 프로젝트를 진행하는 과정에서 데이터 증강을 적용하지 않았을 때 오히려 더 높은 성능이 나오는 현상이 발견되었다. 이는 기존의 일반적인 가설과 상반되는 결과로, 데이터 증강이 반드시 모델의 성능을 향상시키는 것이 아니라 특정한 조건에서 오히려 성능 저하를 초래할 가능성이 있다는 점을 시사한다. 이에 따라, 데이터 증강이 모델 성능에 미치는 영향을 체계적으로 분석하고, 최적의 증강 방법 및 강도를 찾을 필요가 있다고 판단하여 본 연구를 진행하게 되었다.

1.2 연구 목표

본 연구의 목표는 다양한 데이터 증강 방법이 모델 성능에 미치는 영향을 분석하여, 어떤 증강 기법이 성능 향상에 가장 효과적인지와 어느 정도의 변형이 적절한지를 규명하는 것이다. 이를 통해, 데이터 증강이 항상 긍정적인 영향을 미친다는 통념을 검증하고, 모델 학습에 최적화된 증강 전략을 도출하고자 한다.

1.3 연구 범위 및 한계

본 연구에서는 이미지 데이터에 대한 증강 기법만을 다루며, 자연어 처리 및 기타 데이터 유형(예: 시계열 데이터)은 연구 범위에서 제외한다. 또한, 실험은 특정 모델(예: CNN 기반 모델)과 특정 데이터셋을 대상으로 수행되며, 연구 결과가 다른 모델이나 데이터셋에서도 동일하게 적용될지는 추가적인 검증이 필요하다. 본 연구의 결과는 해당 조건에서의 데이터 증강 최적화에 대한 실증적 근거를 제공하는 데 중점을 둔다.

Ⅱ. 관련 연구 및 기술 분석

2.1 기존 연구 및 관련 기술 분석

데이터 증강은 딥러닝 모델의 성능을 향상시키기 위해 널리 사용되는 기법이며, 여러 연구에서 그 효과가 입증되었다. 기존 연구에서는 이미지 증강 기법이 과적합을 방지하고 모델의 일반화 성능을 향상시키는 데 도움이 된다고 보고하고 있다. 대표적인 연구 사례로는 AutoAugment, RandAugment 등의 자동 증강 기법이 있으며, 이들은 다양한 증강 조합을 자동으로 탐색하여 최적의 증강 방법을 찾는 방식으로 동작한다.

그러나 일부 연구에서는 특정 증강 기법이 모델의 성능을 저하시킬 수도 있다는 점을 지적하였다. 예를 들어, 과도한 변형(예: 심한 회전, 강한 노이즈 추가)이 오히려 데이터의 원래 특성을 손상시키고 학습을 방해할 수 있다는 연구 결과도 존재한다. 이에 따라, 증강 기법의 효과를 정량적으로 분석하고 최적의 변형 강도를 설정하는 것이 중요한 연구 과제이다.

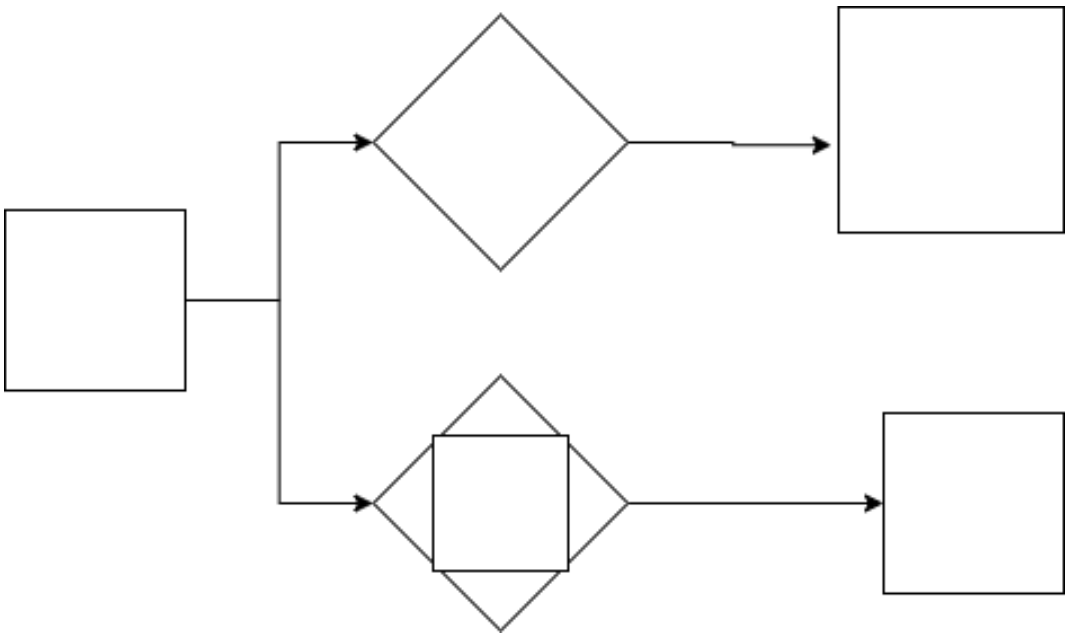
2.2 주요 기술 개요

본 연구에서 분석할 주요 데이터 증강 기법은 다음과 같다:

- 기본 증강 기법: 좌우 반전, 회전, 크기 조절, 밝기 및 대비 조절
- 고급 증강 기법: CutMix, MixUp, Random Erasing 등
- 자동 증강 기법: AutoAugment, RandAugment 등의 자동화된 증강 기법

각 기법이 모델 성능에 미치는 영향을 비교 분석하여, 최적의 증강 방법을 도출하는 것이 본 연구의 핵심이다.

- 이미지 회전의 기법



위에서 보이는 네모는 이미지이다 먼저 위에는 이미지를 회전 시킨 후 정사각형의 크기를 맞추기 위해서 패딩 방식 혹은 보간법을 사용한다. 두 번째는 이미지를 회전 시킨 후 확대를 하는 방법이다. 일반적으로 파이토치나 텐서플로우에서는 회전의 각도만 입력한다면 검정색으로 패딩 방식을 사용한다 아래 사진은 파이토치의 rotation을 적용한 것이다.

아래의 사진 찍는 남자의 사진은 보간 방법을 이용한 것으로 이미지를 회전시킨 후 정 사각형의 크기만큼 주위의 밝기로 채우고 이미지를 축소시킨다다

아래 표는 파이토치에서 사용할 수 있는 rotation의 매개변수이다

파라미터	설명	예시	기본값
🔧 `degrees`	회전 각도의 범위 설정	`45` → `-45도 ~ +45도` `(-30, 30)` → `-30도 ~ +30도`	**필수**
🎛️ `interpolation`	회전 시 픽셀 보간 방식	`InterpolationMode.BILINEAR` (부드러움) `InterpolationMode.NEAREST` (빠르지만 거칠음)	`InterpolationMode.NEAREST`
⬆️ `expand`	회전 후 이미지 크기를 확장할지 여부 (= 잘림 방지)	`True` → 이미지 크기 확장, 패딩처럼 보임 `False` → 원본 크기 유지, 회전 후 잘림 발생	`False`
🎯 `center`	회전 중심 좌표 지정 (왼쪽 위 기준 (x, y))	`(100, 100)` → 특정 좌표 중심으로 회전 기본값은 이미지 중앙	이미지 중앙
🎨 `fill`	회전 후 생긴 빈 공간 채우기 색상	`0` → 검정색 `255` → 흰색 `(128,128,128)` → 회색 RGB 튜플도 가능	`0` (검정색)

2.3 차별성 및 기여점

본 연구는 기존 연구들이 데이터 증강의 효과를 일반적으로 긍정적으로 평가한 것과 달리, 특정 상황에서는 증강이 오히려 성능 저하를 초래할 수 있음을 실험적으로 검증한다. 이를 통해, 무조건적인 증강이 아닌 최적의 증강 전략을 탐색하고자 한다. 또한, 다양한 증강 기법을 비교 분석하여 실용적인 가이드를 제시하는 것이 연구의 주요 기여점이다.

Ⅲ. 설계 및 구현

3.1 시스템 아키텍처

연구를 위한 실험 환경은 다음과 같이 구성된다:

- 데이터셋: rice-disaster 등
- 모델: ResNet50 CNN 기반 모델
- 프레임워크: PyTorch
- 실험 환경: GPU 기반 서버

3.2 주요 기능 및 설계

- 데이터 증강 기법별 실험 진행
- 모델 학습 및 성능 평가
- 증강 기법별 비교 분석 및 시각화

3.3 기술 스택 및 개발 환경

- 프로그래밍 언어: Python
- 라이브러리: PyTorch

3.4 데이터 처리 및 **AI** 모델 적용 방식

- 원본 데이터셋을 다양한 증강 방법으로 변형
- 변형된 데이터를 학습에 사용하고 성능 비교

Ⅳ. 실험 및 결과

4.1 실험 구성

- 실험 모델: ResNet50 (사전 학습된 가중치, FC 레이어만 fine-tuning)
- 실험 데이터: 쌀 질병 이미지(3클래스), 쌀 품종 이미지(5클래스)
- 회전 증강 각도: 15도, 30도, 45도
- 비교 기준: Test Accuracy, Test Loss

회전 각도	보간법	Test Loss	Test Accuracy
x	x	0.0119	1.0000
15	x	0.0353	0.9883
15	o	0.0338	0.987
30	x	0.046	0.9800
45	x	0.06	0.9700

4.2 훈련 과정

우선 데이터를 캐글에서 불러온

```
import kagglehub

path =
kagglehub.dataset_download("muratkokludataset/rice-image-dataset")
path+='/Rice_Image_Dataset'
print("Path to dataset files:", path)
```

```
import torch

from torch.utils.data import DataLoader

from torch import nn

from torchvision import transforms, datasets

from torchvision.models import resnet50, ResNet50_Weights

from torch.optim import Adam

from torch.optim import AdamW

from tqdm.auto import tqdm

from timeit import default_timer as timer

import os

import copy

device = (
```

```
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")


!pip install split-folders
```

그리고 import가 필요한 것들은 불러온다

```
import splitfolders  # splitfolders 라이브러리 불러오기


splitfolders.ratio(
    path,  # 원본 데이터 폴더 경로
    output="splitted",  # 분할된 데이터가 저장될 폴더명 (기본값: "splitted")
    seed=1337,  # 랜덤 시드 값 (재현 가능성 보장)
    ratio=(0.8, 0, 0.2)  # 훈련 80%, 검증 0%, 테스트 20% 비율로 데이터 분할
)
```

훈련셋과 테스트셋을 나눠준다

```
train_transform_rn50 = transforms.Compose([
```

```
    transforms.Resize(size=232,
interpolation=transforms.InterpolationMode.BILINEAR),

    transforms.CenterCrop(size=224),

    transforms.RandomRotation(45), # augmentation: random rotation 45
degree

    transforms.ToTensor(),

    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])

])



test_transform_rn50 = transforms.Compose([

    transforms.Resize(size=232,
interpolation=transforms.InterpolationMode.BILINEAR),

    transforms.CenterCrop(size=224),

    transforms.ToTensor(),

    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])

])
```

본 연구에서 중점은 위 훈련셋의 변환에 관해서이다. 위의 코드를 회전 각도별 만들어준다

```
train_dataset_rn50 = datasets.ImageFolder(root=train_data_path,
transform=train_transform_rn50)

test_dataset_rn50 = datasets.ImageFolder(root=test_data_path,
transform=test_transform_rn50)

est_data_loader = DataLoader(test_dataset_rn50, batch_size=32,
shuffle=True)
```

```
train_data_loader = DataLoader(train_dataset_rn50, batch_size=32,
shuffle=True)

rn50_model = resnet50(weights=ResNet50_Weights.DEFAULT)

rn50_model.fc =
nn.Sequential(nn.Linear(in_features=rn50_model.fc.in_features,
out_features=5))

for param in rn50_model.parameters():

    param.requires_grad = False



for param in rn50_model.fc.parameters():

    param.requires_grad = True



rn50_model = rn50_model.to(device)

rn50_model2 = copy.deepcopy(rn50_model).to(device)

rn50_model3 = copy.deepcopy(rn50_model).to(device)

loss_fn = nn.CrossEntropyLoss()

optim_rn50 = Adam(params=rn50_model.parameters(), lr=0.001)
```

모델들을 따로 만들어주고 각각의 모델들을 fc 만 훈련할 수 있게 수정한다

그리고 loss와 optimizer를 설정한다

```
def training(model, dataloader, loss_fn, optimizer):

    model.train()

    train_loss = 0

    train_acc = 0



    for batch, (image, label) in enumerate(dataloader):
```

```
        image = image.to(device)

        label = label.to(device)



        label_pred = model(image)



        loss = loss_fn(label_pred,label)

        train_loss += loss.item()



        optimizer.zero_grad()



        loss.backward()



        optimizer.step()



        label_pred_class = torch.argmax(torch.softmax(label_pred,
dim=1), dim=1)

        train_acc += (label_pred_class ==
label).sum().item()/len(label_pred)



    train_loss = train_loss / len(dataloader)

    train_acc = train_acc / len(dataloader)

    return train_loss, train_acc

def train_process(model, train_dataloader, optimizer, loss_fn, epochs):

    results = {"train_loss": [], "train_acc": []}
```

```
    for epoch in tqdm(range(epochs)):

        train_loss, train_acc = training(model=model,
dataloader=train_dataloader, loss_fn=loss_fn, optimizer=optimizer)



        # TensorBoard에 Loss & Accuracy 기록

        writer.add_scalar("Loss/train", train_loss, epoch)

        writer.add_scalar("Accuracy/train", train_acc, epoch)



        print(f"Epoch: {epoch+1} | train_loss: {train_loss:.4f} |
train_acc: {train_acc:.4f}")




        results["train_loss"].append(train_loss)

        results["train_acc"].append(train_acc)



    writer.close()  # TensorBoard Writer 종료

    return results
```

위 코드들로 훈련을 하고 loss와 acc를 모은 results를 return 한다

```
import time

torch.manual_seed(42)

torch.cuda.manual_seed(42)



NUM_EPOCHS = 10



```

```
start_time = timer()



rn50_results =
train_process(model=rn50_model,train_dataloader=train_data_loader,optim
izer=optim_rn50,loss_fn=loss_fn,epochs=NUM_EPOCHS)



end_time = timer()

print(f"Total Training Time: {end_time-start_time:.3f} seconds")
```

위 코드로 걸린 시간과 결과를 나오게 한다

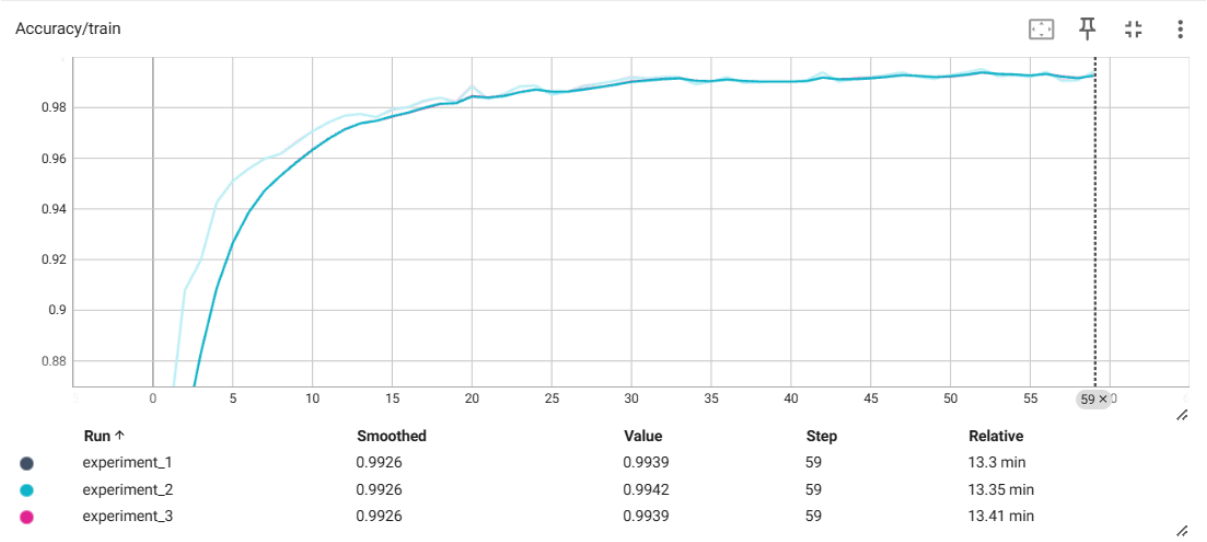
```
def test_step(model,dataloader,loss_fn):

    model.eval()

    test_loss, test_acc = 0, 0

    with torch.inference_mode():

        for batch, (X, y) in enumerate(dataloader):

            X, y = X.to(device), y.to(device)

            val_pred_logits = model(X)

            loss = loss_fn(val_pred_logits, y)

            test_loss += loss.item()

            val_pred_labels = val_pred_logits.argmax(dim=1)

            test_acc += ((val_pred_labels ==
y).sum().item()/len(val_pred_labels))

    test_loss = test_loss / len(dataloader)

    test_acc = test_acc / len(dataloader)

    return test_loss, test_acc
```

```
test_step(model=rn50_model, dataloader=test_data_loader,
loss_fn=loss_fn)
```

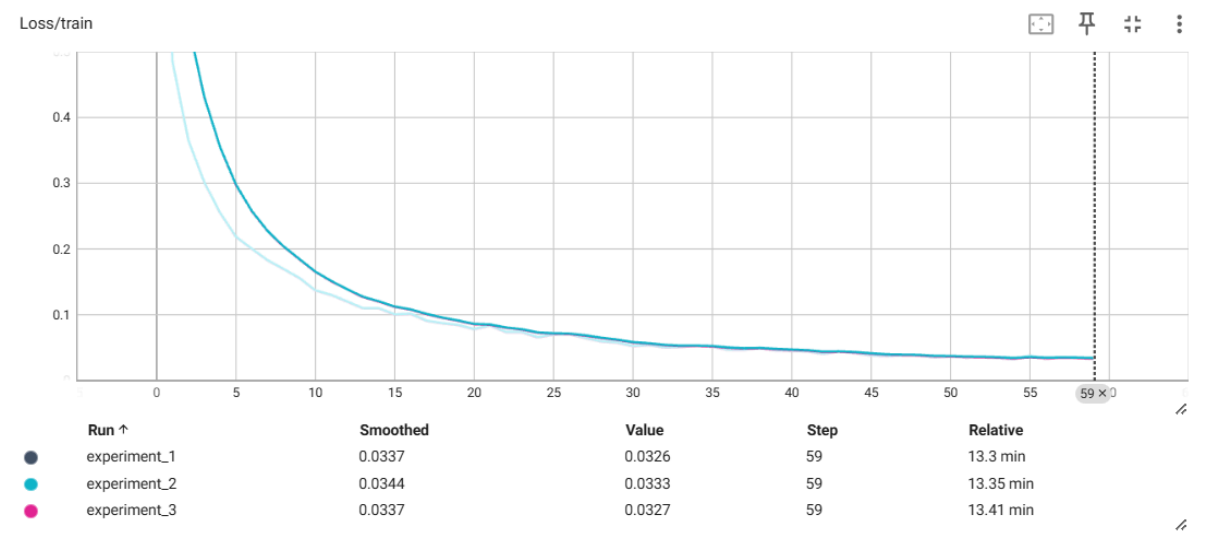
마지막은 테스트셋으로 모델의 성능을 확인한다.

4.3 추가 실험 결과

- 패딩 색상 변경(검정 → 흰색): 초기 학습 속도 향상, 최종 성능은 유사
- 보간법 변경(Bilinear): 유의미한 성능 차이 없음
- Optimizer 비교: Adam, AdamW, AdamP 간 큰 성능 차이 없음



Run	Smoothed	Value	Step	Relative
experiment_1	0.9926	0.9939	59	13.3 min
experiment_2	0.9926	0.9942	59	13.35 min
experiment_3	0.9926	0.9939	59	13.41 min



Run	Smoothed	Value	Step	Relative
experiment_1	0.0337	0.0326	59	13.3 min
experiment_2	0.0344	0.0333	59	13.35 min
experiment_3	0.0337	0.0327	59	13.41 min

```
# Example usage
predicted_class = predict(rn50_model, image)
predicted_class2 = predict(rn50_model2, image)
predicted_class3 = predict(rn50_model3, image)
print(f"Adam: {predicted_class}")
print(f"AdamW: {predicted_class2}")
print(f"AdamP: {predicted_class3}")
```

```
Adam: 0
AdamW: 0
AdamP: 0
```

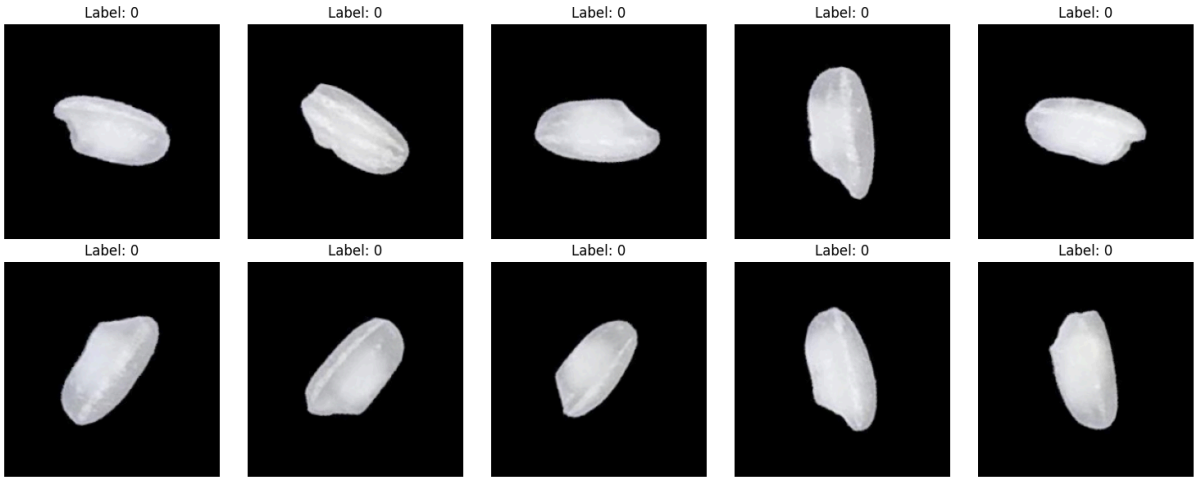
실제 데이터를 가지고 예측을 해 보았을 때 정상적으로 예측되는 것을 볼 수 있음 또한 모두 같은 결과를 나옴

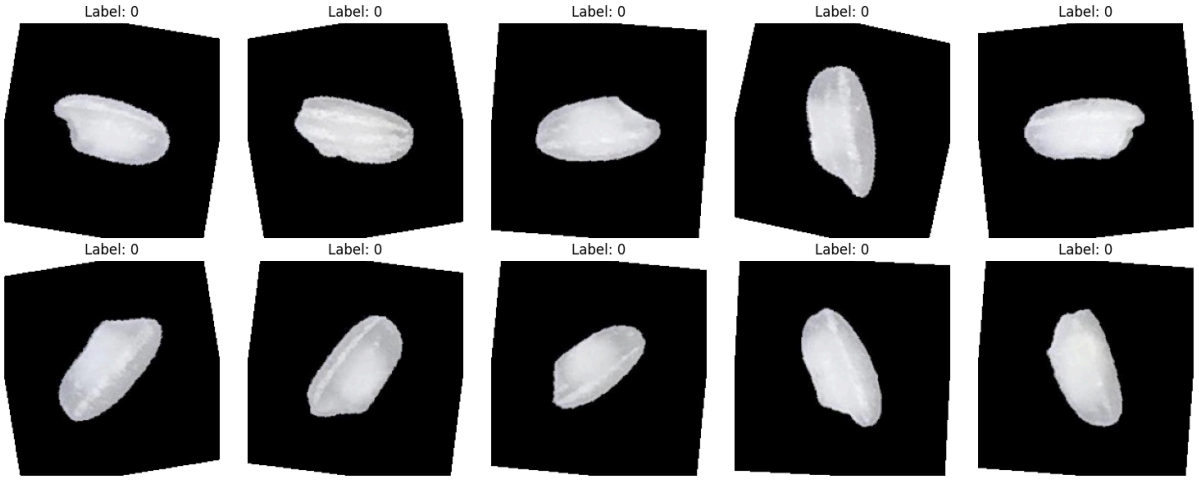
```
[ ]  # Example usage
predicted_class = predict(rn50_model, image)
predicted_class2 = predict(rn50_model2, image)
predicted_class3 = predict(rn50_model3, image)
print(f"Adam: {predicted_class}")
print(f"AdamW: {predicted_class2}")
print(f"AdamP: {predicted_class3}")
```

```
Adam: 1
AdamW: 1
AdamP: 1
```

V. 분석 및 논의

- 회전 각도가 커질수록 테스트 정확도가 하락함. 이는 회전으로 인해 생긴 이미지 외곽의 빈 공간(패딩) 또는 정보 손실이 주요 원인으로 분석됨.
- 패딩을 흰색으로 바꿔도 최종 성능에는 큰 차이가 없으나, 초기 학습 속도에는 영향을 미침. 이는 모델이 불필요한 정보(검정 영역)를 빠르게 무시할 수 있기 때문으로 판단됨.





- 이미 정제된 데이터셋에서는 무리한 증강 기법이 성능을 오히려 저하시킬 수 있으며, 오히려 원본 그대로 학습시키는 것이 더 효과적인 경우도 있음.

Ⅵ. 결론 및 향후 연구 제안

6.1 결론

본 연구에서는 쌀 이미지 데이터셋과 벼 질병 데이터셋을 활용하여 회전 중심의 데이터 증강 기법이 이미지 분류 모델에 미치는 영향을 분석하였다. 그 결과, 증강을 적용하지 않은 경우가 가장 높은 성능을 보였으며, 회전 각도가 커질수록 모델의 성능은 낮아졌다. 이는 데이터 증강이 항상 모델 성능을 향상시키는 것은 아니라는 점을 시사하며, 증강 기법의 선택과 강도는 데이터 특성에 맞게 조정되어야 한다. 또한 회전을 하고 패딩을 적용하는 방식 보다 보간법을 사용하여 빈 공간을 채우는 방식이 더 성능이 좋은 것을 볼 수 있다.

6.2 향후 연구 제안

- 밝기, 대비, 노이즈 추가, 수평 뒤집기 등 다양한 증강 기법과의 성능 비교 실험 진행
- AutoAugment, RandAugment와 같은 자동 증강 기법 적용
- 증강 기법에 따른 attention 변화 시각화(Grad-CAM 등 활용)
- 다른 백본 모델(EfficientNet, MobileNet 등)에서도 동일 실험 반복
- 데이터 편향, 클래스 불균형 상황에서의 증강 기법 효과 분석

이러한 추가 연구를 통해, 데이터 증강 기법의 실질적인 효과와 적용 조건에 대한 보다 명확한 가이드를 제시할 수 있을 것이다.

7. 데이터 베이스 사용

데이터 베이스는 RDB인 MYSQL 을 사용하였다. 어디서든 데이터베이스를 사용할 수 있게 구글 cloud에 데이터베이스를 만들어서 연결을 하였다

```
!pip install SQLAlchemy pymysql

from google.colab import userdata



from sqlalchemy import create_engine, text

engine = create_engine("mysql+pymysql://user:password@공개IP/goyong")







# 접속 정보 설정

user = 'root'                # Cloud SQL 사용자 이름

password = userdata.get('DB')        # 비밀번호

host = '34.47.81.137'            # 예: 34.64.xxx.xxx

database = 'goyong'            # 사용할 DB 이름



# SQLAlchemy 접속 URL 구성

url = f"mysql+pymysql://{user}:{password}@{host}/{database}"

engine = create_engine(url)

# 아래 코드에서는 db에 로스값과 정확도를 넣는다다

with engine.connect() as conn:

    for epoch, (loss, acc) in enumerate(zip(rn50_result['train_loss'],
rn50_result['train_acc']), start=1):

        values_to_insert = {

            'rotation': 15,
```

```
            'epoch': epoch,
            'train_loss': loss,
            'train_acc': acc
        }
        conn.execute(
            text("INSERT INTO training_metrics (rotation, epoch,
train_loss, train_acc) VALUES (:rotation, :epoch, :train_loss,
:train_acc)"),
            values_to_insert
        )
    conn.commit()
```

```
 with engine.connect() as conn:
#     # Use text() to create a SQLAlchemy TextClause object
     conn.execute(text("""
         CREATE TABLE IF NOT EXISTS training_metrics (
            id INT AUTO_INCREMENT PRIMARY KEY,
            rotation INT NOT NULL,
            interpolation INT NOT NULL,
            epoch INT NOT NULL,
            train_loss FLOAT,
            train_acc FLOAT
       );
    """))
```

테이블의 구조는 아래 표와 같다

id	rotation	interpolation	epoch	train_loss	train_acc
1	15	x	10	0.234	0.987
2	15	o	10	0.324	0.984

이렇게 테이블을 구성하였다.

아래 사진은 데이터베이스에 잘 들어갔는지 코랩에서 확인하는 코드와 결과입니다.