knu University

Start

독러닝

(DOG LEARNING)

Subject : 강아지를 찾아내기

Submit by : 팀 5조(퍼피넷)

OVERVIEW

주제 선정 배경

치와와 VS 쿠키

푸들 VS 치킨

비숑 VS 기장떡

걸레 VS 삽살개

구별하기 힘든 클래스 분류

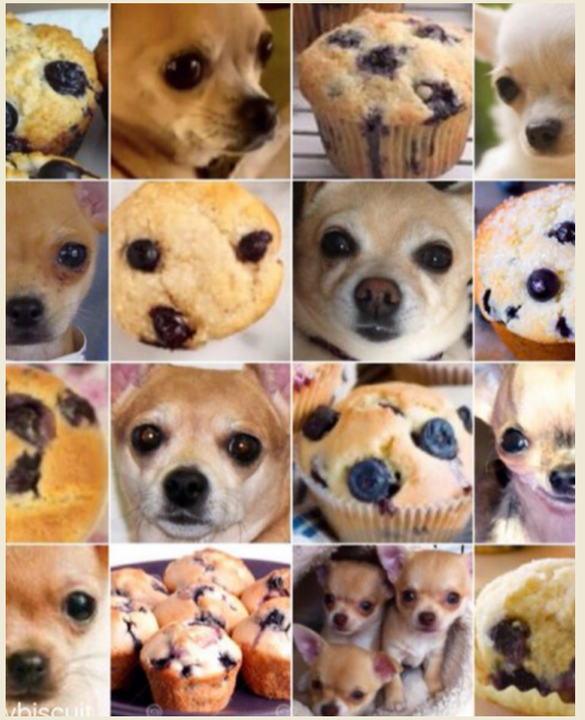
성능을 높이는 데 사용되는 중요한 지표는 무엇인가?

역할 분담

	손예림	임소영	이화은	명노아
주제 선정	✅		✅	
Github Readme		✅		✅
데이터 전처리	✅	✅	✅	✅
전이학습		✅		✅
CNN	✅		✅	
최종 결과물 산출	✅	✅	✅	✅

CHIHUAHUA MUFFIN

손예림



[1] 데이터 준비

KDT

chihuahua vs muffin

kaggle

Search

Create

Home

Competitions

Datasets

Models

Code

Discussions

Learn

More

Your Work

VIEWED

General Image Clas...

images_prediction_...

Muffin vs chihuahua...

Muffin versus ChiH...

Muffin vs Chihuahua

EDITED

Hotel Booking Anal...

SAMUEL CORTINHAS · UPDATED A YEAR AGO

68

New Notebook

Download (497 MB)

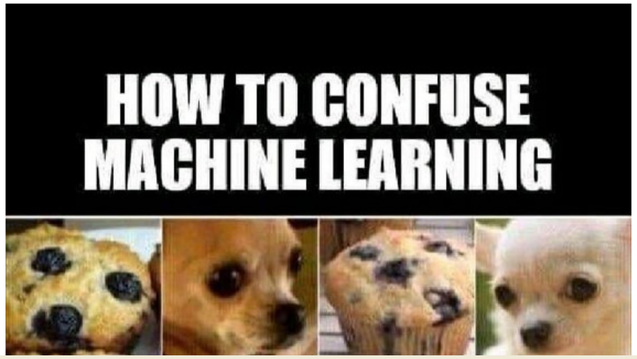
Muffin vs chihuahua

6,000 images of muffins and dogs

Data Card | Code (28) | Discussion (1) | Suggestions (0)

About Dataset

This dataset was inspired by the following meme:



Usability
8.75

License
CC0: Public Domain

Expected update frequency
Never

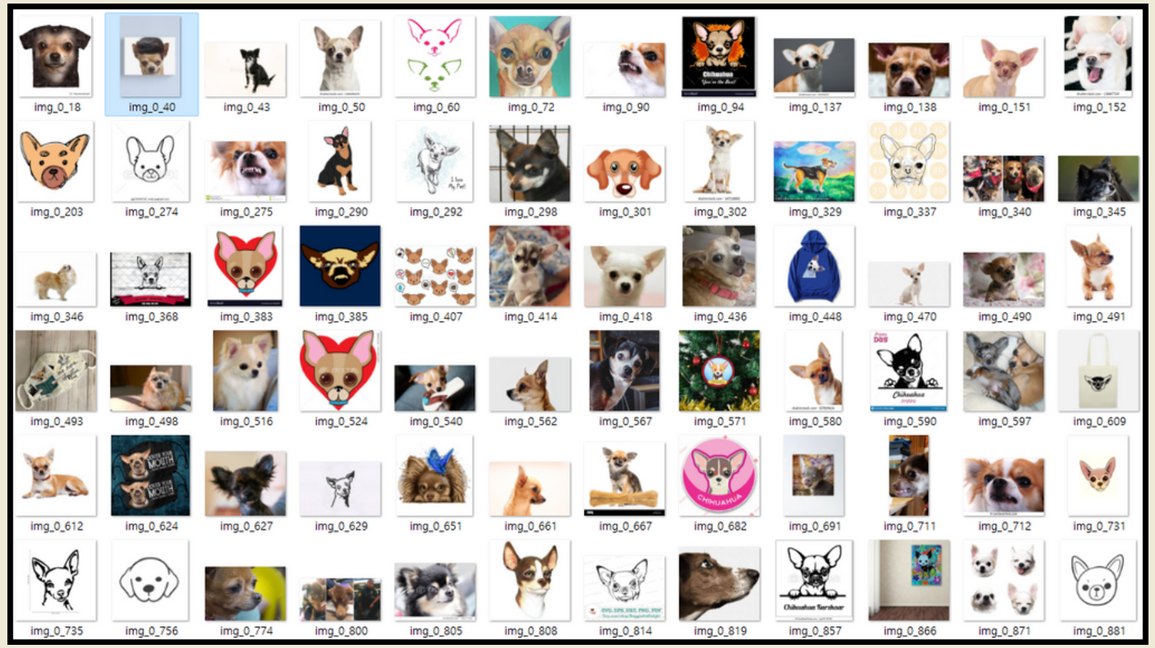
Tags

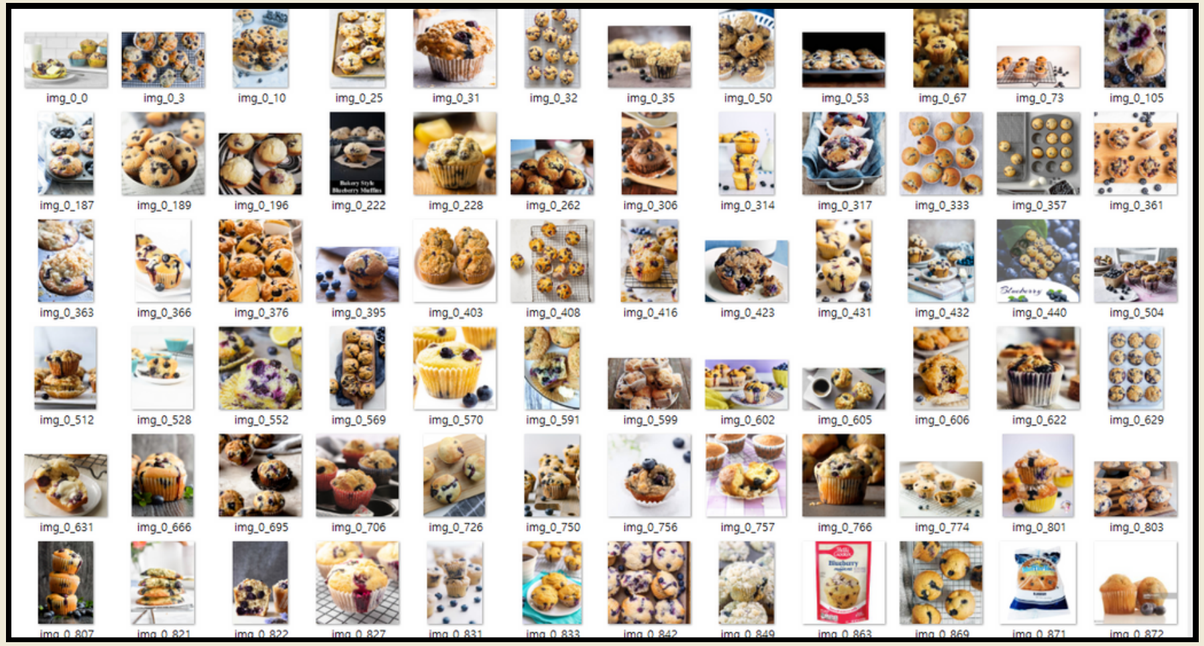
Food | Image | Beginner | Computer Vision | Animals

[1] 데이터 준비

KDT

chihuahua vs muffin





데이터 준비


```
train_transform = transforms.Compose([
    transforms.Resize((50, 50)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

test_transform = transforms.Compose([
    transforms.Resize((50, 50)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])
```

훈련 및 테스트 데이터에 대한 전처리 정의,
이미지 50x 50 크기로 조정,
텐서로 변환 ,
평균 및 표준편차를 사용하여 정규화

데이터셋 & 데이터로더


```
train_dataset = torchvision.datasets.ImageFolder(root='.
 ./data/img/train', transform=train_transform)
train_loader = torch.utils.data.DataLoader(train_dataset,
 batch_size=32, shuffle=True)

test_dataset = torchvision.datasets.ImageFolder(root='.
 ./data/img/test', transform=test_transform)
test_loader = torch.utils.data.DataLoader(test_dataset,
 batch_size=32, shuffle=False)
```

ImageFolder 데이터셋을 생성하고,
정의된 전처리를 적용

데이터셋을 미니배치로 나누고 셔플
링함

클래스 생성


```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels = 3,out_channels =8, kernel_size = 3, padding =1)
        self.conv2 = nn.Conv2d(in_channels =8, out_channels = 16, kernel_size = 3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(12*12*16,64)
        self.fc2 = nn.Linear(64,32)
        self.fc3 = nn.Linear(32,1)

    def forward(self,x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.pool(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.pool(x)

        x = x.view(-1, 12*12*16)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        x = self.Sig(x)
        return x
```

CNN 클래스 => 간단한 합성곱 신경망 모델 정의 두개의 합성곱 계층 & 두 개의 완전 연결 계층

forward 메서드 => 모델의 순전파 정의

훈련


```python
for epoch in range(10):  # 예시로 10 에폭으로 설정
    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data[0].to(device), data[1].float().to(device)  # 레이블을 float으로 변환
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels.unsqueeze(1))  # BCEWithLogitsLoss를 사용하므로 레이블의 shape을 [batch_size, 1]로 만들어줌
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if i % 10 == 9:  # 10 배치마다 손실 출력
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 10))
            running_loss = 0.0
```

criterion : BCELoss ()=> 이진 분류 엔트로피 손실

optimizer : SGD 옵티마이저 사용

평가


```python
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, labels = data[0].to(device), data[1].float().to(device)  # 레이블을 float으로 변환
        outputs = model(images)
        predicted = (outputs > 0).float()  # outputs가 0보다 크면 1로, 그렇지 않으면 0으로 예측
        total += labels.size(0)
        correct += (predicted == labels.unsqueeze(1)).sum().item()

print('Accuracy : %d %%' % (
    100 * correct / total))
```

```
Finished Training
Accuracy : 82 %
```

모델 저장 및 모델 정의


```python
from cnn import CNN

# 모델 저장
torch.save(model.state_dict(), 'best_cnn.pth')

# 모델 정의
model = CNN()
model.load_state_dict(torch.load('best_cnn.pth'))  # 저장된 모델을 불러옴
model.eval()
```

이미지 전처리


```python
# 이미지 로드 및 전처리
image_path = '../data/img/test/chihuahua/img_0_18.jpg'
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # OpenCV는 이미지를 BGR 형식으로 읽어옴 그 후 RGB로 변환
img_pil = to_pil_image(image)
input_image = transform(img_pil)
input_image = input_image.unsqueeze(0)
```

이미지를 BGR에서 RGB로 변환하는 이유

openCV의 기본 컬러 채널 순서 BGR

대부분의 이미지 처리 라이브러리 및 이미지 파일 형식은 RGB이므로 변경 !!!

예측결과


```
with torch.no_grad():
    output = model(input_image)
    _, predicted = torch.max(output, 1)
predicted_prob = torch.softmax(output, dim=1)[0][predicted.item()].item()
print(f"{'치와와' if predicted.item() == 1 else '머핀'}입니다!")
```

기울기 계산을 비활성화하여 메모리 사용 최적화
소프트맥스 함수 사용하여 확률로 변환

머핀입니다!

치와와를 넣었으나,,, 머핀이 나옴

POODLE

CHICKEN

임소영

QUESTIONS

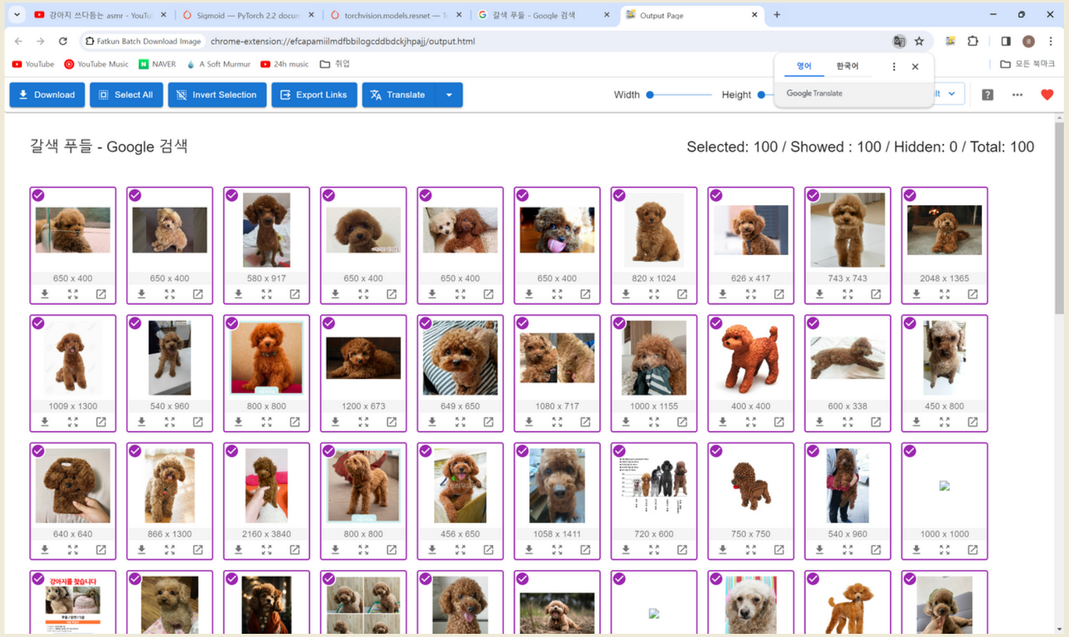
푸들?

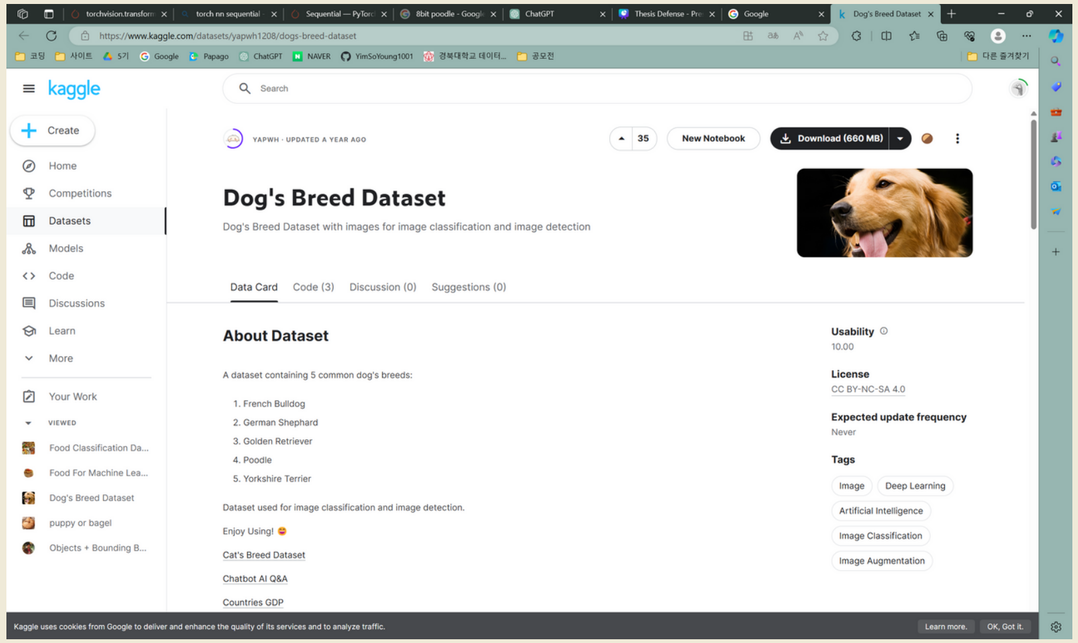
치킨?

[1] 데이터 준비

KDT

poodle vs chicken





크롬에서 검색어 입력 후 사진 다운로드

캐글에서 dataset 다운로드 후 일부 선정

검색어 : ‘갈색 푸들’ ‘간장 치킨’

chicken : 247개 poodle : 243개

[2] 전처리 및 DataSet 생성

KDT

poodle vs chicken

```
preprocessing = transforms.Compose([
    transforms.Resize(size = (150, 150)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

Executed at 2024.03.27 09:51:06 in 12ms

```
file_dir = './image'
imgDS = ImageFolder(root = file_dir, transform = preprocessing)
```

Executed at 2024.03.27 09:51:06 in 13ms

```
print(imgDS.classes, imgDS.class_to_idx)
```

Executed at 2024.03.27 09:51:06 in 13ms

```
['chicken', 'poodle'] {'chicken': 0, 'poodle': 1}
```

```
# dataset에서 train, valid, test를 나누어보자
seed_gen = torch.Generator().manual_seed(42)
tr, val, ts = 0.7,0.1,0.2
trainDS, validDS, testDS = random_split(imgDS, [tr, val, ts], generator=seed_gen)
print(len(trainDS), len(validDS), len(testDS))
```

Executed at 2024.03.27 09:51:06 in 14ms

```
343 49 98
```

전처리 순서

- resize : 사진 크기 (150,150)
- array => tensor로 변환
- 정규화 : 각 채널에서 mean 0.5 std 0.5이 되도록

Dataset 분리

- 비율 설정

  train : valid : test = 0.7 : 0.1 : 0.2

[3] Dataloader

KDT

poodle vs chicken

```
# 각 분류별로 dataloader를 생성해보자
batch_size = 10
train_dl = DataLoader(trainDS, batch_size=batch_size, shuffle=True)
valid_dl = DataLoader(validDS, batch_size=batch_size, shuffle=True)
test_dl = DataLoader(testDS, batch_size = batch_size, shuffle=True)
print(len(train_dl), len(valid_dl), len(test_dl))
```

Executed at 2024.03.27 09:51:06 in 15ms

```
35 5 10
```

[4] 모델 생성


```python
# 모델 인스턴스 생성 : 사전 학습된 모델 로딩
# => 가중치를 조절해보자!
res_model = models.resnet18(weights = "ResNet18_Weights.DEFAULT")

# 전결합층 변경
res_model.fc = nn.Linear(in_features = 512, out_features = 1)
```

Executed at 2024.03.27 09:51:07 in 216ms

```python
# 모델의 합성곱층 가중치 고정
for param in res_model.parameters():
    param.requires_grad = False
for param in res_model.fc.parameters():   # 완전 연결층은 학습
    param.requires_grad = True
```

전이 학습

- 사전 학습된 모델 : Resnet18
- 가중치 : "ResNet18_Weights.DEFAULT"
- 완전 결합층 변경
- 모델의 완전 연결층은 학습이 되도록 설정

[5] 함수 정의

KDT

poodle vs chicken

```
optimizer = torch.optim.Adam(res_model.fc.parameters(), lr=0.01)
cost = torch.nn.BCELoss() # 손실함수 정의
```

Executed at 2024.03.27 09:51:07 in 839ms

```
def training(dataloader):
    res_model.train()
    loss_list = []
    acc_list = []
    precision_list = []
    recall_list = []
    f1score_list = []
    for image, label in dataloader:
        # 학습
        pre_label = res_model(image)
        pre_label = F.sigmoid(pre_label)
        pre_label = pre_label.squeeze()
        # print(label.shape, pre_label.shape)
        # print(label, pre_label, sep = '\n\n')
```

```
        # 손실계산
        train_loss = cost(pre_label, label.float())
        # train_loss = cost(pre_label, label)

        # w,b 업데이트
        optimizer.zero_grad()
        train_loss.backward()
        optimizer.step()

        # 정확도
        acc = metrics.accuracy(pre_label, label, task = 'binary')
        precision = metrics.precision(pre_label, label, task = 'binary')
        recall = metrics.recall(pre_label, label, task = 'binary')
        f1score = metrics.f1_score(pre_label, label, task = 'binary')
```

optimizer : Adam

손실함수 : BCELoss

[6] 학습 및 검증

KDT

poodle vs chicken

training

valid check

checkpoint

scheduler

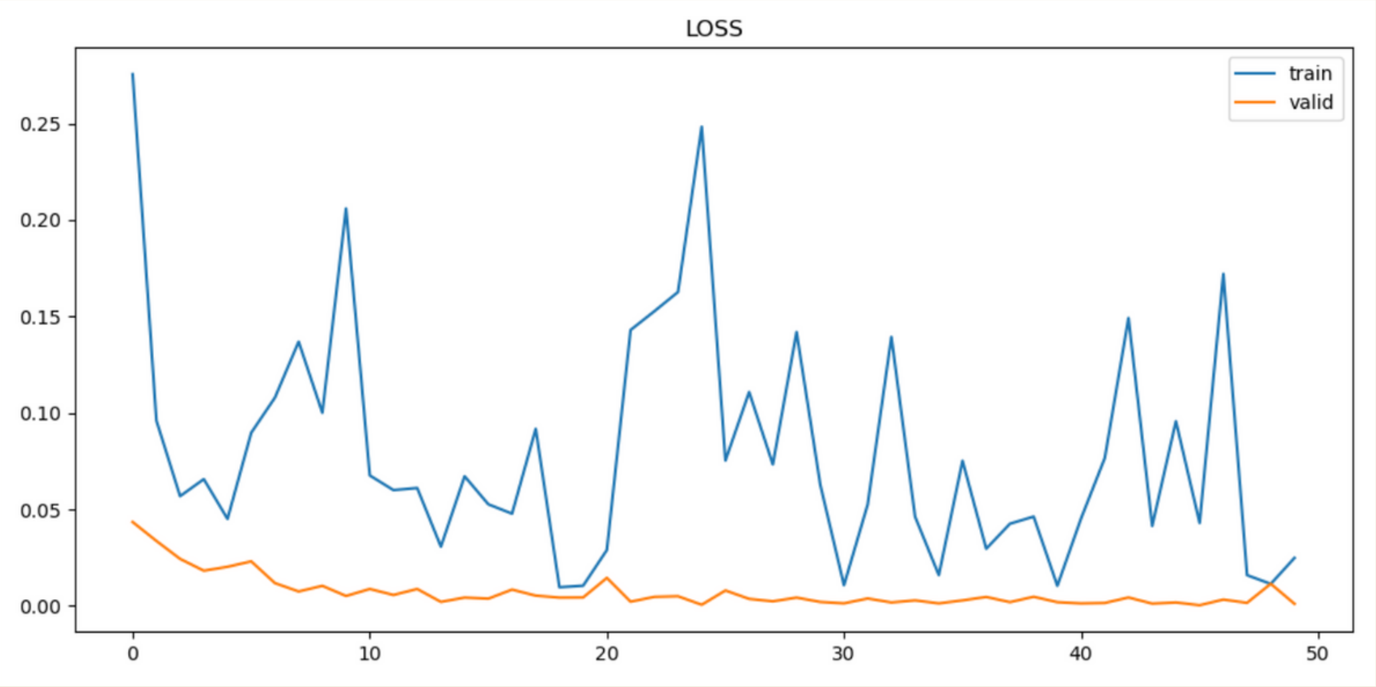
```python
scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones = [0, epoch_num], gamma =
 0.5)
# milestones => 어떤 에포크 구간에서 학습률을 조정할지 나타내는거

# 모델 저장 관련 변수
save_score_point = 5
```

```python
for epo in range(epoch_num):
    # 학습
    loss, acc, prec, rec, f1 = training(train_dl)
    print(loss, acc, prec, rec, f1)
    training_list[0].append(loss.item())
    training_list[1].append(acc.item())
    training_list[2].append(prec.item())
    training_list[3].append(rec.item())
    training_list[4].append(f1.item())
    print(f"epo => {epo}  학습중")
    # 검증
    loss, acc, prec, rec, f1 = valid_testing(valid_dl)
    validing_list[0].append(loss.item())
    validing_list[1].append(acc.item())
    validing_list[2].append(prec.item())
    validing_list[3].append(rec.item())
    validing_list[4].append(f1.item())
    print(f"epo => {epo}  검증중")

    # 검증 데이터 기준 학습된 모델 저장
    if loss < save_score_point:
        torch.save(res_model, filename)
        print('  모 델 저 장 완 료\n')

    # 스케줄러
    scheduler.step()
```

[7] 성능평가

KDT

poodle vs chicken

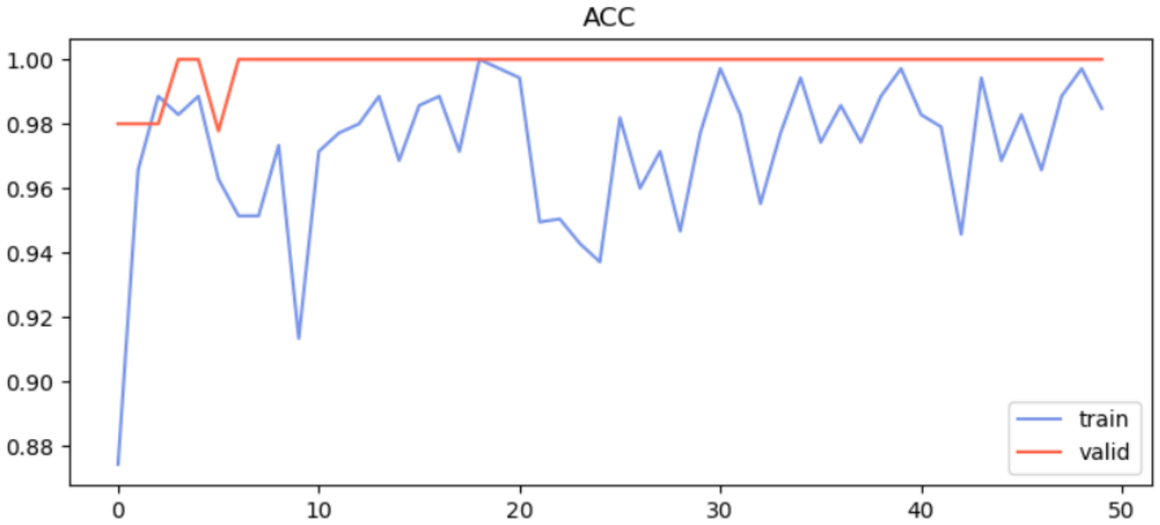


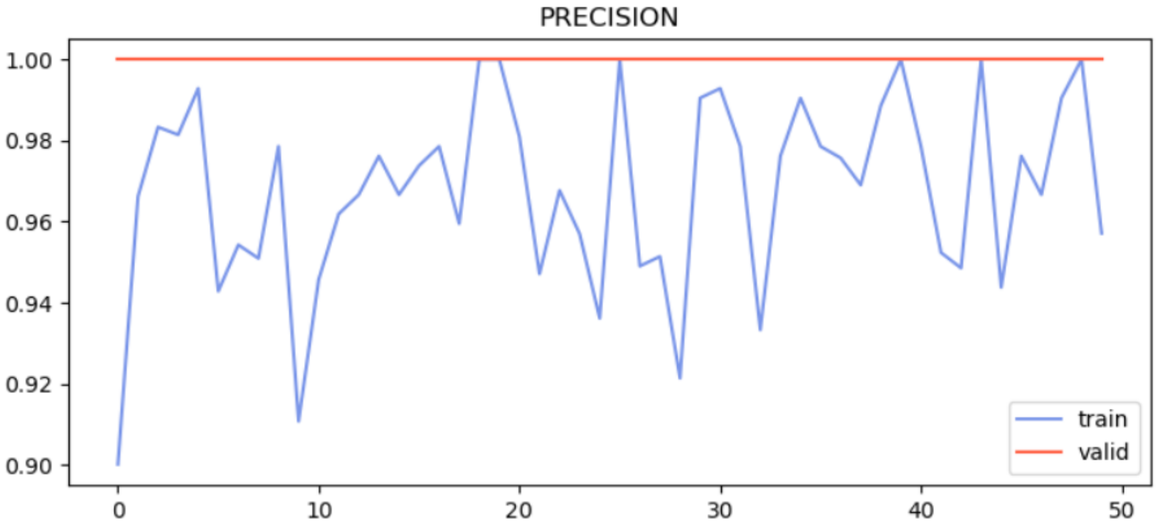
average	train	valid
loss	0.08	0.01
accuracy	0.97	0.09
precision	0.96	1.0
recall	0.97	0.98
f1_score	0.96	0.98

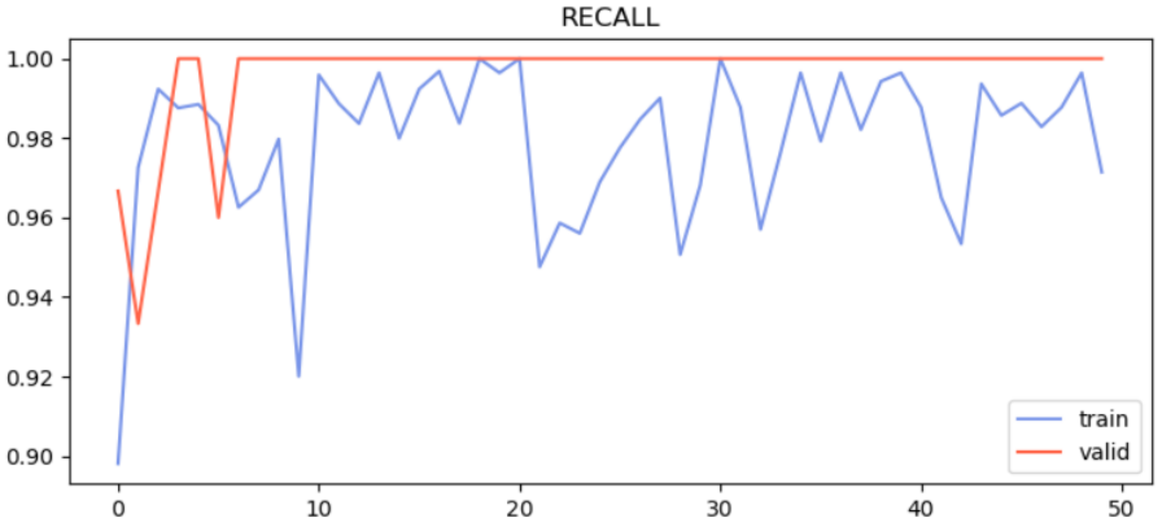
[7] 성능평가

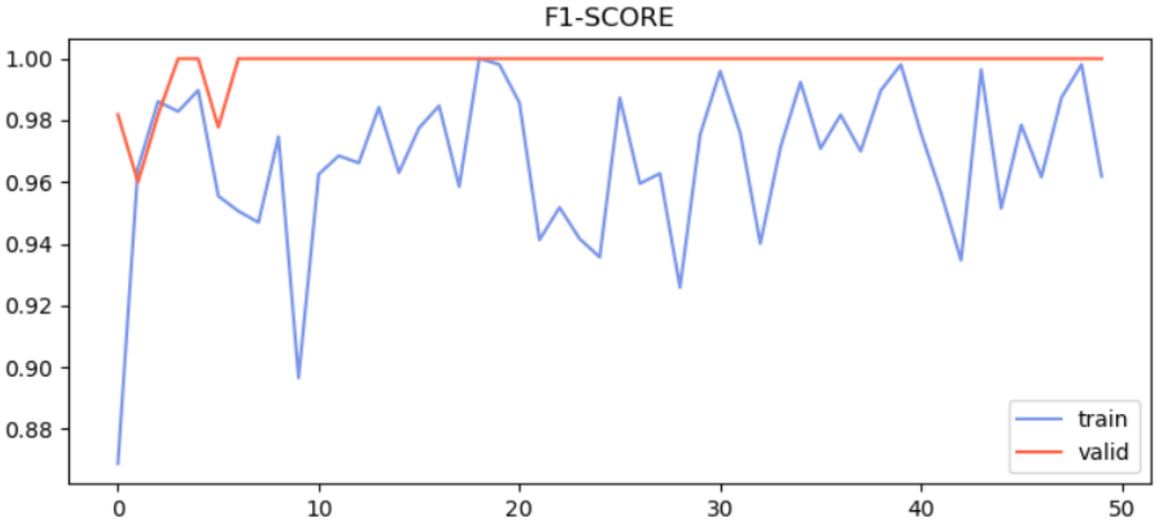
KDT

poodle vs chicken









[9] 예측

KDT

poodle vs chicken

test

```python
def predicting(dataloader):
    res_model.eval()
    loss_list = []
    acc_list = []
    precision_list = []
    recall_list = []
    f1score_list = []
    for image, label in dataloader:
        # 학습
        pre_label = res_model(image)
        pre_label = F.sigmoid(pre_label)
        pre_label = pre_label.squeeze()

        # 손실계산
        pred_loss = cost(pre_label, label.float())

        # 정확도
        acc = metrics.accuracy(pre_label, label, task = 'binary')
        precision = metrics.precision(pre_label, label, task = 'binary')
        recall = metrics.recall(pre_label, label, task = 'binary')
        f1score = metrics.f1_score(pre_label, label, task = 'binary')
```

Loss : 0.00436

ACC : 1.0

Precision : 1.0

Recall : 1.0

F1_score : 1.0

KDT

[+] resnet 비교

poodle vs chicken

※ 숫자가 높을수록 layer수가 더 많음

	resnet34	*resnet50*	*resnet101*	*resnet152*
loss	0.10	0.10	0.09	0.09
accuracy	0.96	0.95	0.96	0.96
precision	0.96	0. 95	0.95	0.96
recall	0.97	0.97	0.97	0.97
f1_score	0.95	0.95	0.95	0.96

[10] 모델 시연

KDT

poodle vs chicken

https://youtu.be/JmKinDs4KzI

푸들?

치킨?

BICHON

GI-JANG DDUCK

이화은

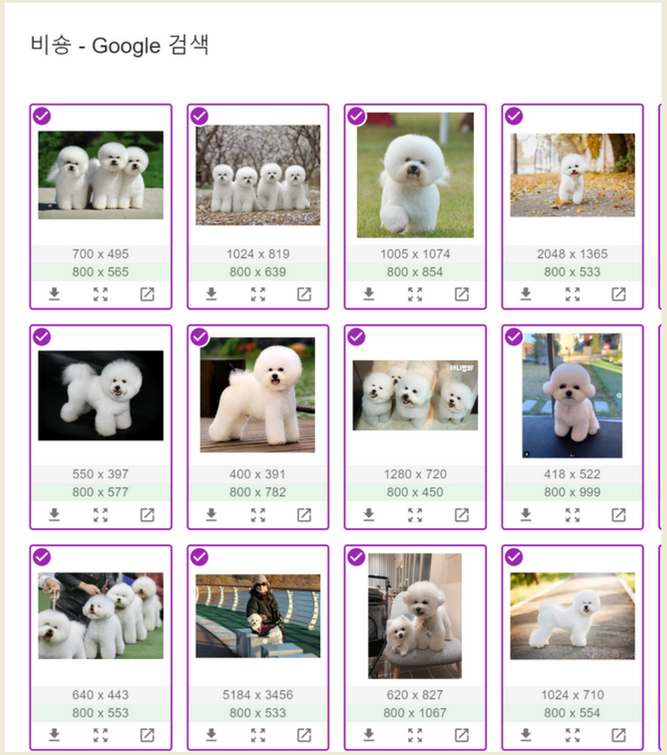
[1] 데이터 준비

KDT

bichon vs gi-jiang-dduck

<출처> : GOOGLE, NAVER, DAUM

-Fatkun Batch Download Image



Train (346) vs Test (80)

[2] 이미지 전처리

- resize : 64 X 64 조정
- ToTensor() : 텐서로 변환
- Normalize : 정규화

```
# 이미지 전처리 시 사용
preprocessing = transforms.Compose(
    [transforms.Resize(size=(64, 64)), # 사이즈 조정 64 * 64
    transforms.ToTensor(), # 텐서로 변환
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))] # 정규화
)
```

[2] 이미지 전처리

KDT

bichon vs gi-jiang-dduck

< 폴더 경로 설정 >

- **학습용 - 테스트용 데이터 경로 설정**
- **os.path.isdir(폴더) : 경로 맞는지 확인 ▶ ImageFolder : 이미지 데이터셋 가져오기**

```
1  # 폴더 경로 설정
2  train = '../train/' # 학습용 데이터 경로 설정
3  test = '../test/' # 테스트용 데이터 경로 설정
4
5  os.path.isdir(train), os.path.isdir(test) # 경로 맞는지 확인
```

```
5  from torchvision.datasets import ImageFolder
6
7  imgFolder = ImageFolder(root=train, transform=preprocessing)
8  testFolder = ImageFolder(root=test, transform=preprocessing)
9  imgFolder.classes, testFolder.classes
```

✓ 0.0s

```
(['Bichon', 'zangidduk'], ['bichon', 'zangidduk'])
```

[2] 이미지 전처리

< 데이터 로더 만들기 >

- 플래그 변수를 이용하여 첫 번째 이미지를 저장하고 이후에는 이미지와 라벨 출력.

```
imgDL = DataLoader(imgFolder, batch_size=8, shuffle=True)
testDL = DataLoader(testFolder, batch_size=8, shuffle=True)

#  플래그 변수를 이용하여 첫 번째 이미지와 라벨을 저장하고,
# 이후에는 이미지와 라벨을 출력하는 구문

flag=0 # 플래그 변수 생성, 첫번째 이미지와 라벨 저장.
img2=np.array([0]) # 첫 번째 이미지를 저장할 빈 배열 생성
label2=np.array([0]) # 첫 번째 라벨을 저장할 빈 배열을 생성
for (img, label) in imgDL : #이미지와 라벨을 반복해서 가져오는 문장.
    if not flag: # 플래그 변수가 0일 때 (즉, 아직 첫 번째 이미지와 라벨을 가져오지 않았을 때) 실행됩니다.
        img2, label2 = img, label # 첫 번째 이미지와 라벨을 img2와 label2에 저장합니다.
        flag=1 # 플래그 변수를 1로 설정하여 더 이상 첫 번째 이미지와 라벨을 가져오지 않도록 합니다.
    print(img.shape, label.shape, label) # 현재 가져온 이미지의 크기와 라벨을 출력합니다.
```

```
torch.Size([8, 3, 64, 64]) torch.Size([8]) tensor([1, 0, 1, 0, 0, 0, 0, 0])
torch.Size([8, 3, 64, 64]) torch.Size([8]) tensor([0, 1, 1, 0, 0, 1, 1, 1])
torch.Size([8, 3, 64, 64]) torch.Size([8]) tensor([0, 0, 0, 0, 0, 0, 0, 0])
torch.Size([8, 3, 64, 64]) torch.Size([8]) tensor([1, 0, 1, 1, 0, 0, 1, 1])
torch.Size([8, 3, 64, 64]) torch.Size([8]) tensor([1, 0, 1, 1, 0, 0, 0, 0])
```

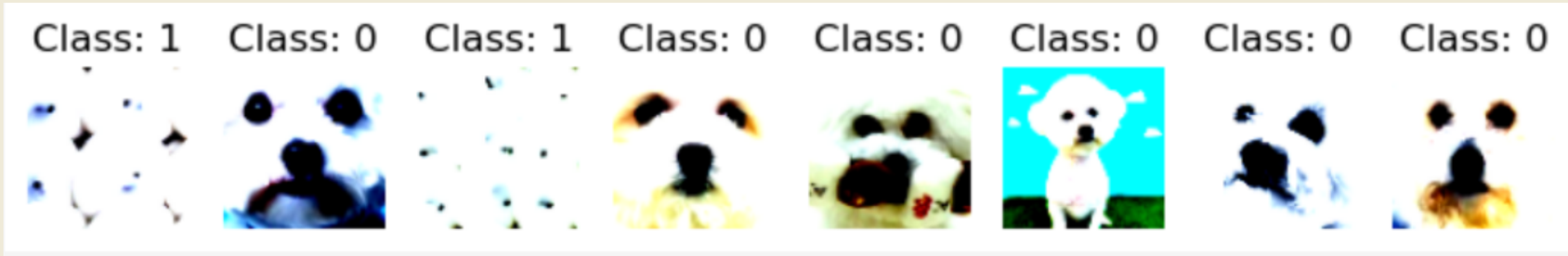
[2] 이미지 전처리

KDT

bichon vs gi-jiang-dduck

< 이미지 데이터 시각화 >

```
pltsize = 2 # pltsize = 2은 subplot을 표시할 때 사용할 figure의 크기를 조정하는 데 사용
plt.figure(figsize=(7, pltsize))

for i in range(8):
    plt.subplot(1, 8, i + 1)
    plt.axis("off")
    plt.imshow(np.transpose(img2[i], (1, 2, 0)))
    plt.title("Class: " + str(label2[i].item()))
plt.tight_layout()
plt.show()
```



 # 예상 : 정규화로 인해 사진이 밝게 나온 것 같음.

[3] CNN 모델 생성

KDT

bichon vs gi-jiang-dduck

< CNN >

```python
class CNN(nn.Module) :
    def __init__(self) : # 모델의 구조를 정의 후, 초기화
        super().__init__() # 부모 클래스인 nn.Module의 초기화 메서드를 호출
        self.conLayer = nn.Conv2d(in_channels=3, out_channels=25, kernel_size = 3, ) # 합성곱 레이어1 정의
        self.conLayer2 = nn.Conv2d(in_channels=25, out_channels=10, kernel_size = 3) # 합성곱 레이어2 정의
        self.pool1 = nn.MaxPool2d(2, 2) # 풀링 레이어를 정의, 최대 풀링(MaxPooling)을 사용하여 특징 맵의 크기 축소
        self.fc1 = nn.Linear(10*14*14, 1000) # fully connected 레이어를 정의
        self.fc2 = nn.Linear(1000, 500)
        self.fc3 = nn.Linear(500, 250)
        self.fc4 = nn.Linear(250, 125)
        self.fc5 = nn.Linear(125, 60)
        self.fc6 = nn.Linear(60, 2)

    def forward(self, x) :
        x = self.conLayer(x) # 입력 이미지에 첫 번째 합성곱 레이어를 적용
        x = F.relu(x) # ReLU 활성화 함수를 적용
        x = self.pool1(x) # 첫 번째 풀링 레이어를 적용
        x = self.conLayer2(x) #
        x = F.relu(x)
        x = self.pool1(x)
        x = x.view(-1, 10*14*14) # fully connected 레이어에 입력하기 위해 데이터를 평탄화(flatten)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        x = F.relu(x)
        x = self.fc4(x)
        x = F.relu(x)
        x = self.fc5(x)
        x = F.relu(x)
        x = self.fc6(x) # fully connected 레이어를 적용하여 최종 예측값을 출력

        return x
```

```python
# 실행 디바이스 실행

DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
# 함수를 사용하여 CUDA 지원 GPU가 사용 가능한지 확인.
# CUDA 지원 GPU가 있는 경우 True를 반환, 그렇지 않은 경우 False를 반환.

mdl = CNN().to(DEVICE)
# CNN 모델을 생성하고, 이를 선택한 실행 디바이스에 할당
```

✓ 0.0s

```python
# 최적화 인스턴스 생성 => 모델에서 사용하는 w와 b 변수들 전달
optimizer=optim.AdamW(mdl.parameters())
```

 # Adam을 썼는데 정확도가 안 나와서

팀원의 추천을 받아서 AdamW로 모델 변경

[4] 학습

KDT

bichon vs gi-jiang-dduck

< Training >

```python
import torchmetrics.functional as metrics
from torchmetrics.functional.classification import multiclass_accuracy
from torch.optim import lr_scheduler

scheduler = lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

def training() :
        mdl.train()
        for idx, (feature, target) in enumerate(imgDL) :
                feature, target = feature.to(DEVICE), target.to(DEVICE)

                # 학습
                pre_y = mdl(feature)
                pre_y2 = F.sigmoid(pre_y)

                # 손실 계산
                #print(f"손실함수 계산 전 shape 확인 => pre_y2 : {pre_y2.shape}  target : {target.shape}")

                target = target.unsqueeze(1).expand(-1, 2).float() # 돌아는 가는데 뭔가 잘못됨.
                loss = F.binary_cross_entropy(pre_y2, target)

                # w, b 업데이트
                optimizer.zero_grad()
                loss.backward()  # 손실함수 계산값으로 미분 진행하여 새로운 W, b 계산
                optimizer.step() # 새로운 값으로 W, b 업데이트

                # 스케줄러 업데이트
                #scheduler.step()
                #current_lr = scheduler.get_last_lr()[0]
                print()

                # 정확도
                accuracy = metrics.accuracy(pre_y2, target, task = 'binary')
                recall = metrics.precision(pre_y2, target, task = 'binary')
                multi_accuracy = multiclass_accuracy(pre_y2, target, num_classes=2)
                # multicalss_accuracy = 모델의 예측과 실제 레이블을 비교하여 올바르게 분류된 샘플의 비율을 계산

        return loss.item(), accuracy, recall, multi_accuracy, target
```

```python
EPOCHS = 100
loss_list = []
mul_acc_list = []
recall_list = []

for ep in range(EPOCHS):
    sample = []
    train_loss, accuracy, recall, train_multi_accuracy, current_lr = training()
    print(f"ep [{ep+1}] loss : {train_loss}")
    print(f"accuracy : {accuracy}")
    print(f'recall : {recall}')
    print(f'multiclass accuracy : {train_multi_accuracy}')
    loss_list.append(train_loss)
    recall_list.append(recall)
    mul_acc_list.append(train_multi_accuracy)
```

```
ep [100] loss : 0.0
accuracy : 1.0
recall : 1.0
multiclass accuracy : 1.0
```

[4] 학습

< Training >

```python
def get_accuracy(loader, model):
    total=0
    correct=0
    for data in loader:
        images, labels = data
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    return correct / total
```

```
1  get_accuracy(imgDL, mdl)
✓  1.2s

0.5057803468208093
```

[4] 테스트

KDT

bichon vs gi-jiang-dduck

```python
def test() :
    mdl.eval()
    with torch.no_grad():
        loss_list=[]
        for i, (feature, target) in enumerate(testDL) :
            feature, target = feature.to(DEVICE), target.to(DEVICE)

            pre_ytst = mdl(feature)
            # 분류값 변화
            pre_ytst2 = F.sigmoid(pre_ytst)


            # 오차 즉 손실 계산
            #print(f"손실함수 계산 전 shape 확인 => pre_y2 : {pre_ytst2.shape}  target : {target.shape}")
            target = target.unsqueeze(dim=1).expand(-1,2).float()
            loss = F.binary_cross_entropy(pre_ytst2, target) # (예측값, 정답)
            loss_list.append(loss.item())
            #print(f'[{idx+1}] Loss => {loss}')


            # 정확도 계산
            test_accuracy = metrics.accuracy(pre_ytst2, target, task = 'binary')
        return loss_list, test_accuracy
```

```
ep [1] loss : 2.9177271709812658e-09
test_accuracy : 0.875
ep [2] loss : 12.542451858520508
test_accuracy : 0.875
ep [3] loss : 3.027533335142607e-09
test_accuracy : 0.625
ep [4] loss : 25.0
test_accuracy : 0.875
ep [5] loss : 2.789740278785757e-07
test_accuracy : 0.875
ep [6] loss : 2.788029291878047e-07
test_accuracy : 0.375
ep [7] loss : 25.0
test_accuracy : 0.625
ep [8] loss : 25.042457580566406
test_accuracy : 1.0
ep [9] loss : 12.500005722045898
test_accuracy : 0.875
ep [10] loss : 50.0
test_accuracy : 0.5
```

```python
loss_list_tst = []
acc_list_tst = []
for ep in range(10):
    tst_loss, tst_accuracy= test()
    print(f"ep [{ep+1}] loss : {tst_loss[ep]}\ntest_accuracy : {tst_accuracy}")
    loss_list_tst.append(tst_loss)
    acc_list_tst.append(tst_accuracy)
```

```python
1  get_accuracy(testDL, mdl)
```

✓ 0.8s

```
0.225
```

[4] 예측

KDT

bichon vs gi-jiang-dduck

< 모델 불러오기 >

```
model_path = 'model.pth'
mdl = CNN()
```

✓ 0.0s

```
mdl = torch.load(model_path)
mdl.eval()
```

✓ 0.0s

< 모델 사진 - bichon>

```
import matplotlib.pyplot as plt
testPic = cv2.imread('bichon2.jpg')
test_pil = to_pil_image(testPic)

testPic2 = cv2.imread('zangidduk2.jpg')
test_pil2 = to_pil_image(testPic2)

plt.subplot(1,2,1)
plt.title('BICHON')
plt.imshow(testPic)
plt.axis('off')
```

< 출력 >

[4] 예측

KDT

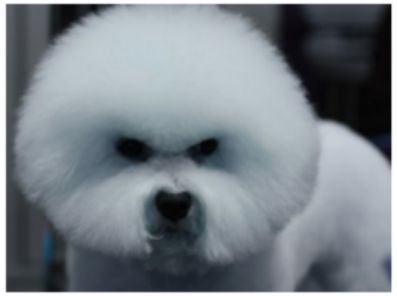
bichon vs gi-jiang-dduck

< 모델 시연 >

```
transformed_img = preprocessing(test_pil)
transformed_img = transformed_img.unsqueeze(0) # 배치 차원을 추가
```

```
def predict(trans_img) :
    with torch.no_grad() : # 그래디언트 업데이트 x
        output = mdl(trans_img) # 전처리된 이미지를 모델에 입력하여 출력을 계산
        _, predicted = torch.max(torch.softmax(output, dim=1).data, 1) # 모델의 출력에 소프트맥스 함수를 적용하여 확률값으로 변환
        print(f"{torch.softmax(output, dim=1).data[0][predicted].item():.2f}% 확률로 {'비숑' if predicted.item() else '기장떡'}입니다!")
```

BICHON



```
1  predict(transformed_img)
```

✓ 0.0s

0.78% 확률로 비숑입니다!

GIJIANG-DDUCK

```
1  predict(transformed_img2)
```

✓ 0.0s

0.72% 확률로 기장떡입니다!

MOP DOG

MOP

명노아

QUESTIONS

KDT

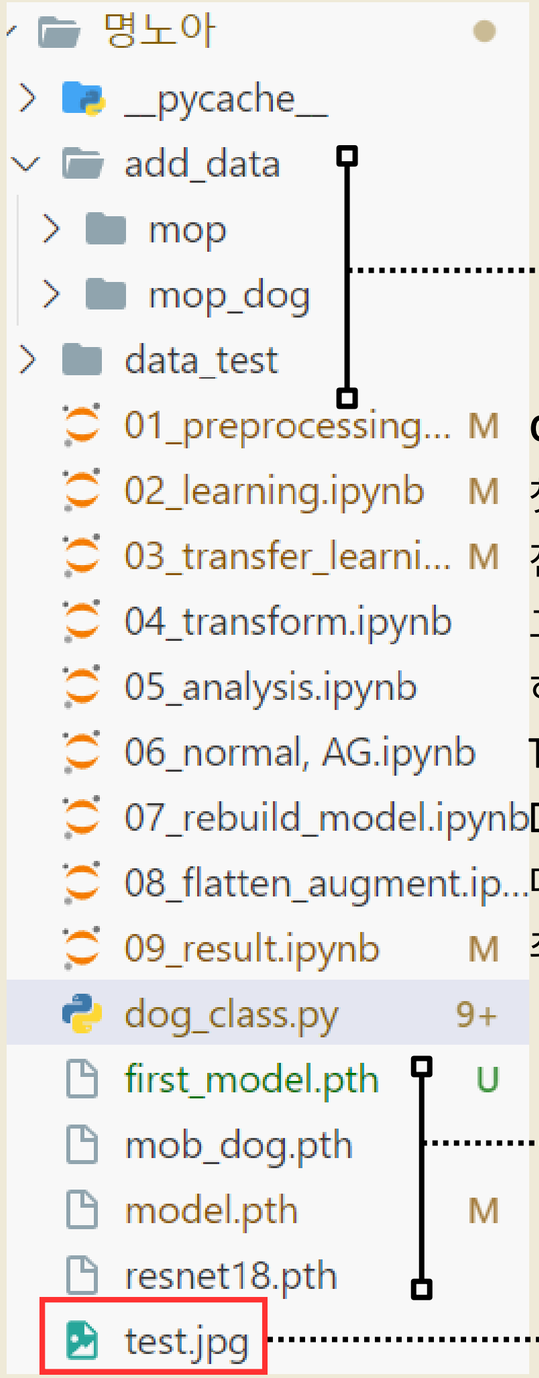
mop vs mop_dog

대걸레?

대걸레

강아지?

DIRECTORY



→Train/Test dataset

이미지 전처리

첫 학습

전이학습 (Resnet18)

그래프 출력

히스토그램 분석

Threshold적용

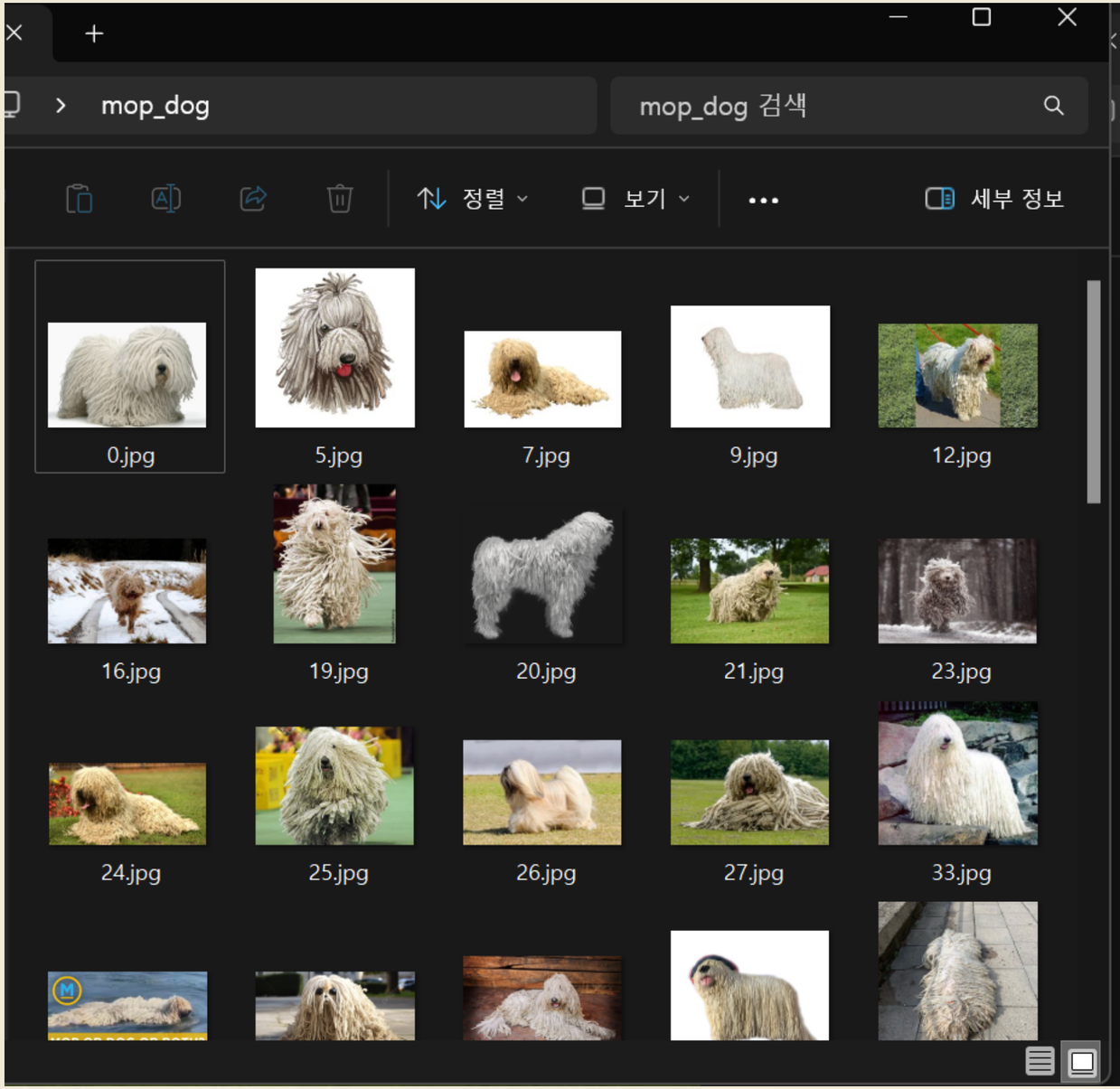
Dropout, Batchnorm 적용

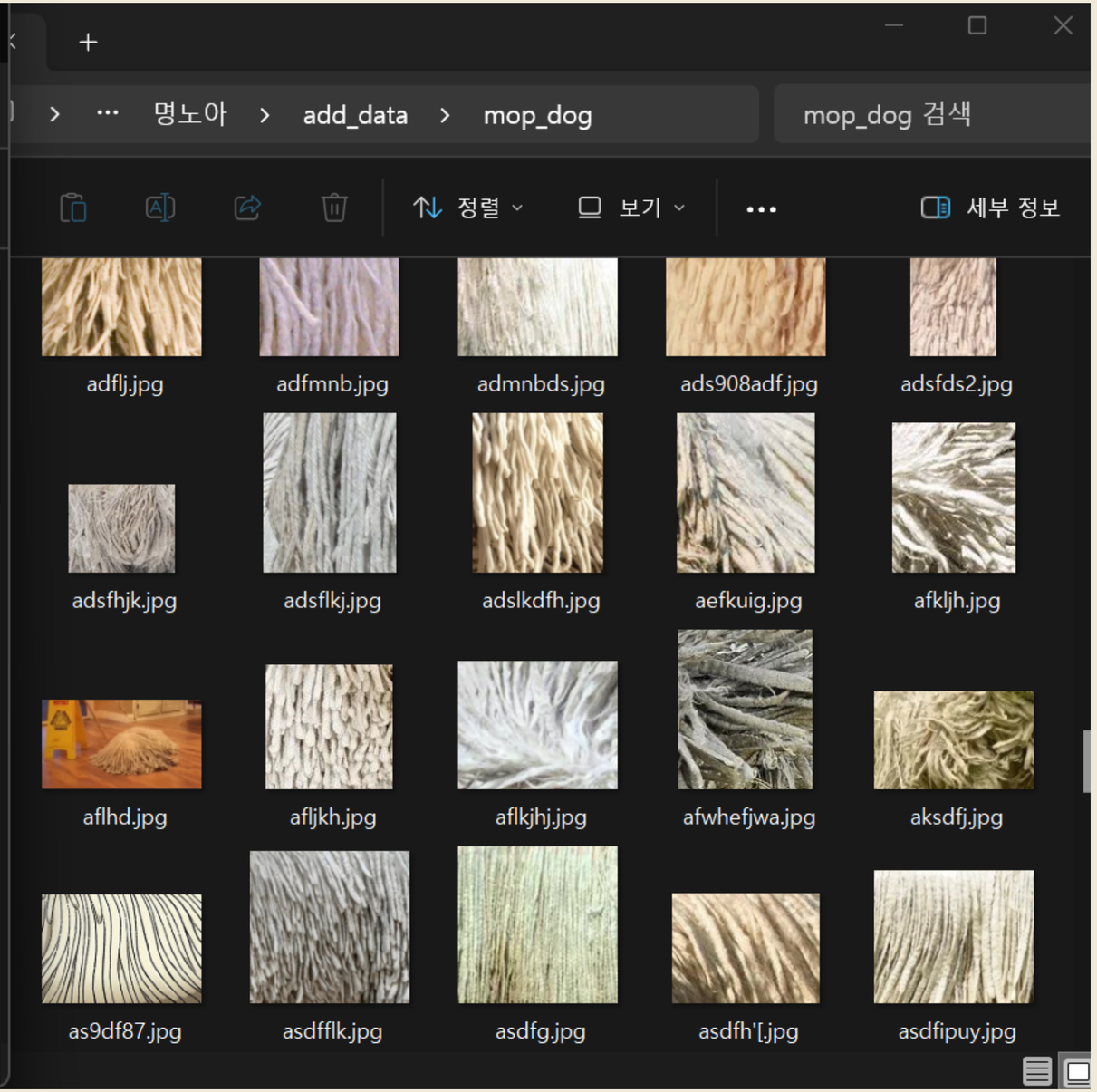
데이터 개수 늘린 후 진행

최종산출물

→각 훈련당 checkpoint 된 모델

→ 테스트 이미지





MOP : 254

MOP_DOG : 260

PREPROCESSING

```python
def onMouse(event,x,y,flags,param):
    global isDragging, x0, y0, img      # 전역변수 참조
    if event == cv2.EVENT_LBUTTONDOWN:  # 왼쪽 마우스 버튼 다운
        isDragging = True
        x0 = x
        y0 = y
    elif event == cv2.EVENT_MOUSEMOVE:  # 마우스 움직임
        if isDragging:                  # 드래그 진행 중
            img_draw = img.copy()
            cv2.rectangle(img_draw, (x0, y0), (x, y), blue, 2) # 드래그 진행 영역 표시
            cv2.imshow('img', img_draw) # 사각형 표시된 그림 화면 출력
    elif event == cv2.EVENT_LBUTTONUP:  # 왼쪽 마우스 버튼 업
        if isDragging:                  # 드래그 중지
            isDragging = False
            w = x - x0                  # 폭 계산
            h = y - y0                  # 높이 계산
            print("x:%d, y:%d, w:%d, h:%d" % (x0, y0, w, h))
            if w > 0 and h > 0:         # 왼쪽에서 오른쪽으로 긁었을 때,
                img_draw = img.copy()   # 선택 영역에 사각형 그림을 표시할 이미지 복제
                # 선택 영역에 빨간 사각형 표시
                cv2.rectangle(img_draw, (x0, y0), (x, y), red, 2)
                cv2.imshow('img', img_draw) # 빨간 사각형 그려진 이미지 화면 출력
                roi = img[y0:y0+h, x0:x0+w] # 원본 이미지에서 선택 영역만 ROI로 지정
                cv2.imshow('cropped', roi)  # ROI 지정 영역을 새창으로 표시
                cv2.moveWindow('cropped', 0, 0) # 새창을 화면 좌측 상단에 이동
                cv2.imwrite(param, roi)  # ROI 영역만 파일로 저장
                print("croped.")
            else:
                cv2.imshow('img', img)  # 드래그 방향이 잘못된 경우 사각형 그림이 없는 원본 이미지 출력
                print("좌측 상단에서 우측 하단으로 영역을 드래그 하세요.")

for i in file_names:
    try :
        img = cv2.imread(i)
        cv2.imshow('img', img)
        cv2.setMouseCallback('img', onMouse, param=i)
        cv2.waitKey()
        cv2.destroyAllWindows()
    except : # 그림이 열리지 않는 경우도 있어서, 이를 감안하여 try, catch구문 활용
        pass
```

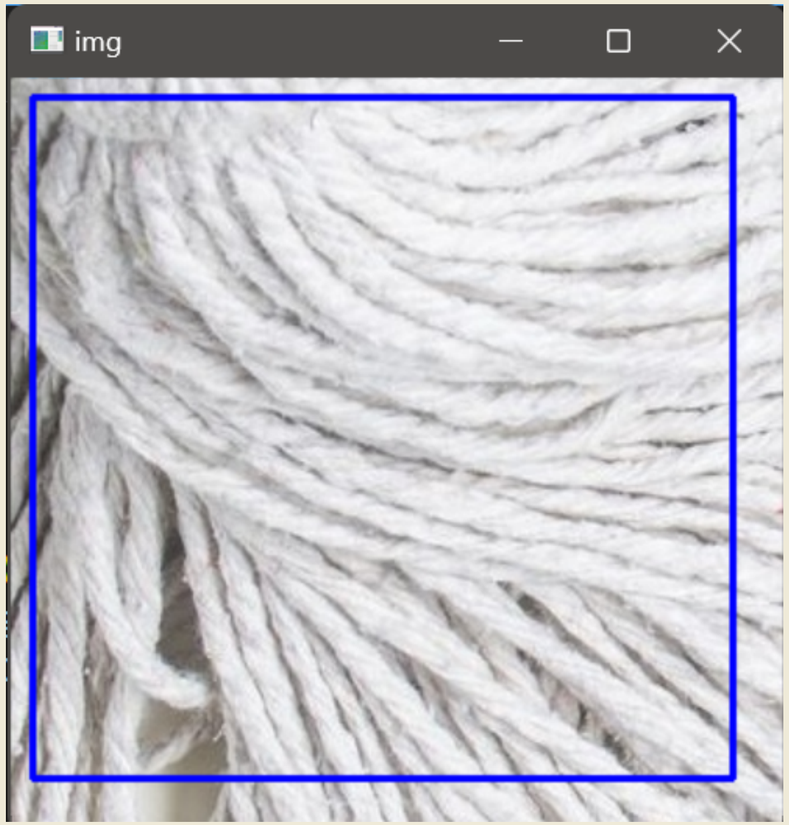
마우스 콜백 함수

=> 배경 제거, 확대

PREPROCESSING

KDT

mop vs mop_dog



ROI 선택 후 저장

=> 배경 제거, 확대

LEARNING

KDT

mop vs mop_dog

```
transform = transforms.Compose([
    transforms.Resize((50,50)),
    transforms.ToTensor()
])

data = ImageFolder(root="./data", transform=transform)
data2 = ImageFolder(root="./data_test", transform=transform)
```

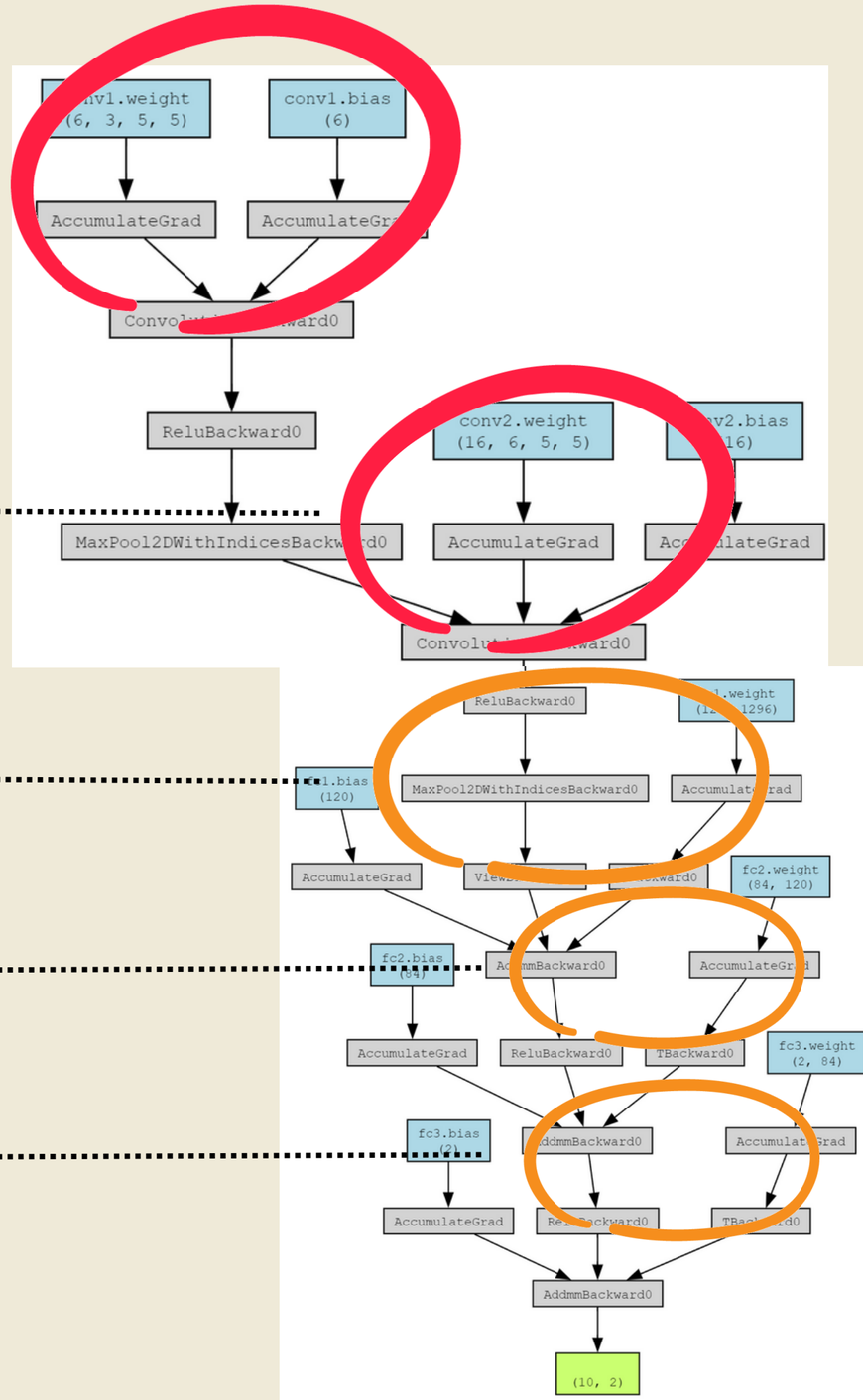
```
data_loader = DataLoader(data, batch_size=10, shuffle=True)
test_loader = DataLoader(data2, batch_size=8, shuffle=True)
```

=> IMAGEFOLDER

```python
class Mob_Dog(nn.Module): # 클래스 생성
    def __init__(self):
        super(Mob_Dog, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 9 * 9, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 2)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x))) # 두번 컨볼루션 연산 해주기



        x = x.view(-1, 16 * 9 * 9)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

=> MODEL

Shape (Batch, channel, height, width)				
10	3	50	50	
10	6	46	46	nn.Conv2d(3,6,5)
10	6	23	23	nn.MaxPool2d(2,2)
10	16	19	19	nn.Conv2d(6,16,5)
10	16	9	9	nn.MaxPool2d(2,2)
(Batch,			feature)	
10			16*9*9	x = x.view(-1, 16 * 9 * 9)
10			120	nn.Linear(16*9*9, 120)
10			84	nn.Linear(120, 84)
10			2	nn.Linear(84, 2)



LEARNING

KDT

mop vs mop_dog

```
# 모델 객체 생성, 옵티마이저 생성
model = Mob_Dog()
criterion = nn.CrossEntropyLoss()
optimizer = optim.AdamW(model.parameters(), lr=0.001)
```

```
def get_accuracy(loader, model):
    total=0
    correct=0
    for data in loader:
        images, labels = data
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1) # dim=1을 주어 최종 값 계산
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    return correct / total
```

=> ACCURACY

```
from tq                스 바 생성
from torch.optim.lr_scheduler import ReduceLROnPlateau # 스케줄러 생성
scheduler = ReduceLROnPlateau(optimizer, 'min', patience=15, verbose=True)

loss_list=[]
tas_list=[]
tes_list=[]
max=0
for epoch in range(100): # 10회 반복
    running_loss = 0.0 # 1 epoch당 누적 로스값
    #=============================================================================
    pbar = tqdm(enumerate(data_loader), total=len(data_loader))
    for i, data in pbar: # pbar가 데이터 로더 역할 수행
        inputs, labels = data
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item() # 로스 누적
        if i % 10 == 9:
            pbar.set_description(f'Epoch [{epoch + 1}/{10}], 횟수 [{i + 1}/{len(data_loader)}], Loss: {running_loss / 10:.4f}')
            loss_list.append(running_loss)
            running_loss = 0.0
    #=============================================================================
```

=> 학습진행

EVALUATE


```
with torch.no_grad():
    train_accuracy = get_accuracy(data_loader, model)
    test_accuracy = get_accuracy(test_loader, model)
print(f'{epoch} Epoch 종료 후 train_score : {(100 * train_accuracy)}')
print(f'{epoch} Epoch 종료 후 test_score : {(100 * test_accuracy)}')
tas_list.append(train_accuracy)
tes_list.append(test_accuracy)
if max < test_accuracy:
    max = test_accuracy
    torch.save(model.state_dict(), 'model.pth')

scheduler.step(loss)
if scheduler.num_bad_epochs >= scheduler.patience:
    print(f"Early Stopping at : {epoch} Epoch")
    break
```

=> 체크포인트

=> 스케줄러

LEARNING

KDT

mop vs mop_dog

```
1 Epoch 종료 후 test_score : 77.17391304347827
Epoch [3/10], 횟수 [30/43], Loss: 0.4491:  81%|████████   | 35/43 [00:00<00:00, 48.39it/s]
```

```
9 Epoch 종료 후 test_score : 86.95652173913044
Epoch [11/10], 횟수 [10/43], Loss: 0.3371:  23%|██         | 10/43 [00:00<00:00, 47.98it/s]
```

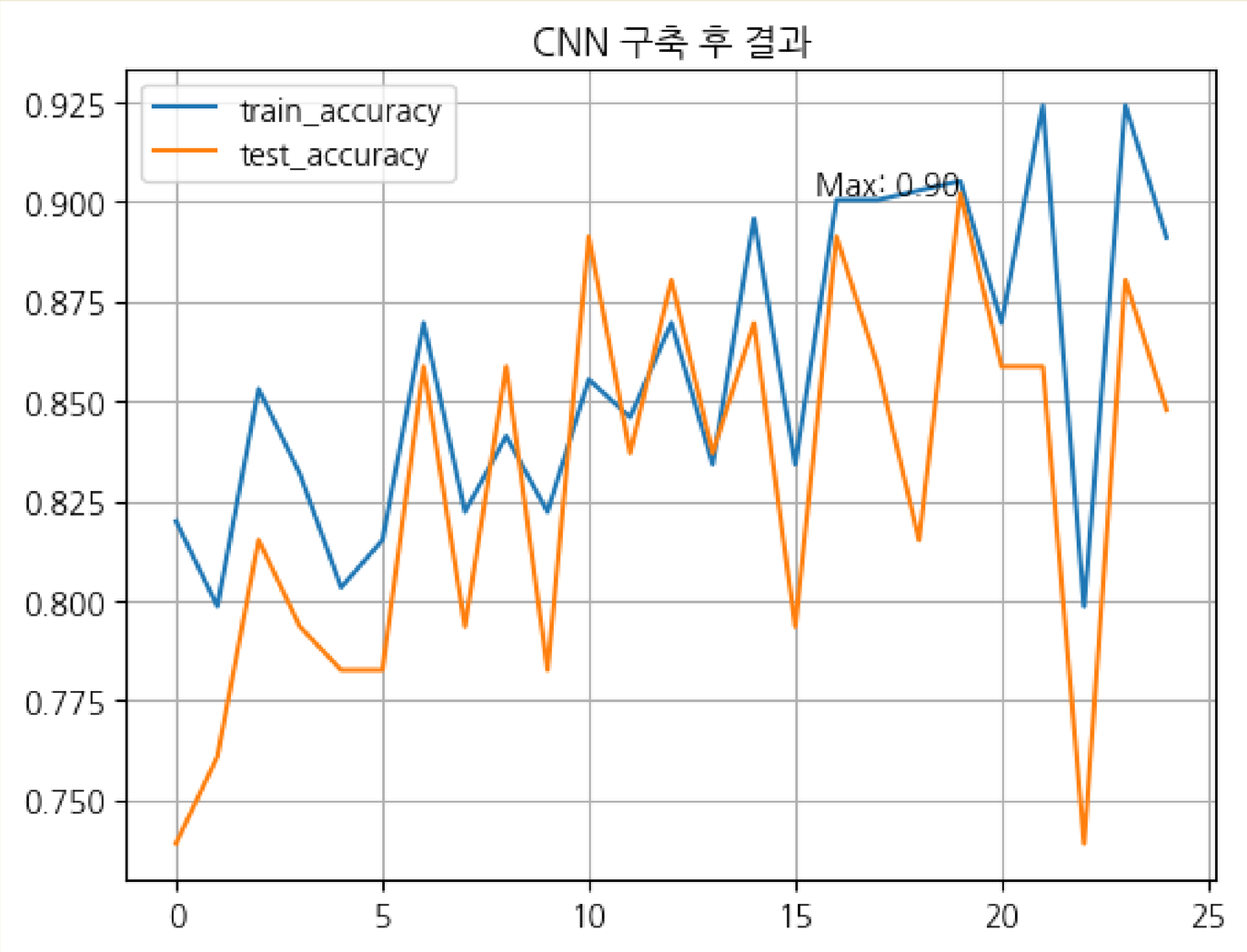
```
Epoch [13/10], 횟수 [40/43], Loss: 0.3227: 100%|██████████| 43/43 [00:00<00:00, 45.94it/s]
12 Epoch 종료 후 train_score : 85.07109004739335
12 Epoch 종료 후 test_score : 83.69565217391305
Epoch [14/10], 횟수 [40/43], Loss: 0.3476: 100%|██████████| 43/43 [00:00<00:00, 47.81it/s]
13 Epoch 종료 후 train_score : 87.91469194312796
13 Epoch 종료 후 test_score : 88.04347826086956
Epoch [15/10], 횟수 [40/43], Loss: 0.3234: 100%|██████████| 43/43 [00:00<00:00, 46.95it/s]
14 Epoch 종료 후 train_score : 87.44075829383885
14 Epoch 종료 후 test_score : 84.78260869565217
Epoch [16/10], 횟수 [40/43], Loss: 0.3440: 100%|██████████| 43/43 [00:00<00:00, 52.44it/s]
15 Epoch 종료 후 train_score : 88.62559241706161
15 Epoch 종료 후 test_score : 86.95652173913044
Epoch [17/10], 횟수 [40/43], Loss: 0.2467: 100%|██████████| 43/43 [00:00<00:00, 52.50it/s]
```

=> PROGRESS BAR

RESULT

KDT

mop vs mop_dog



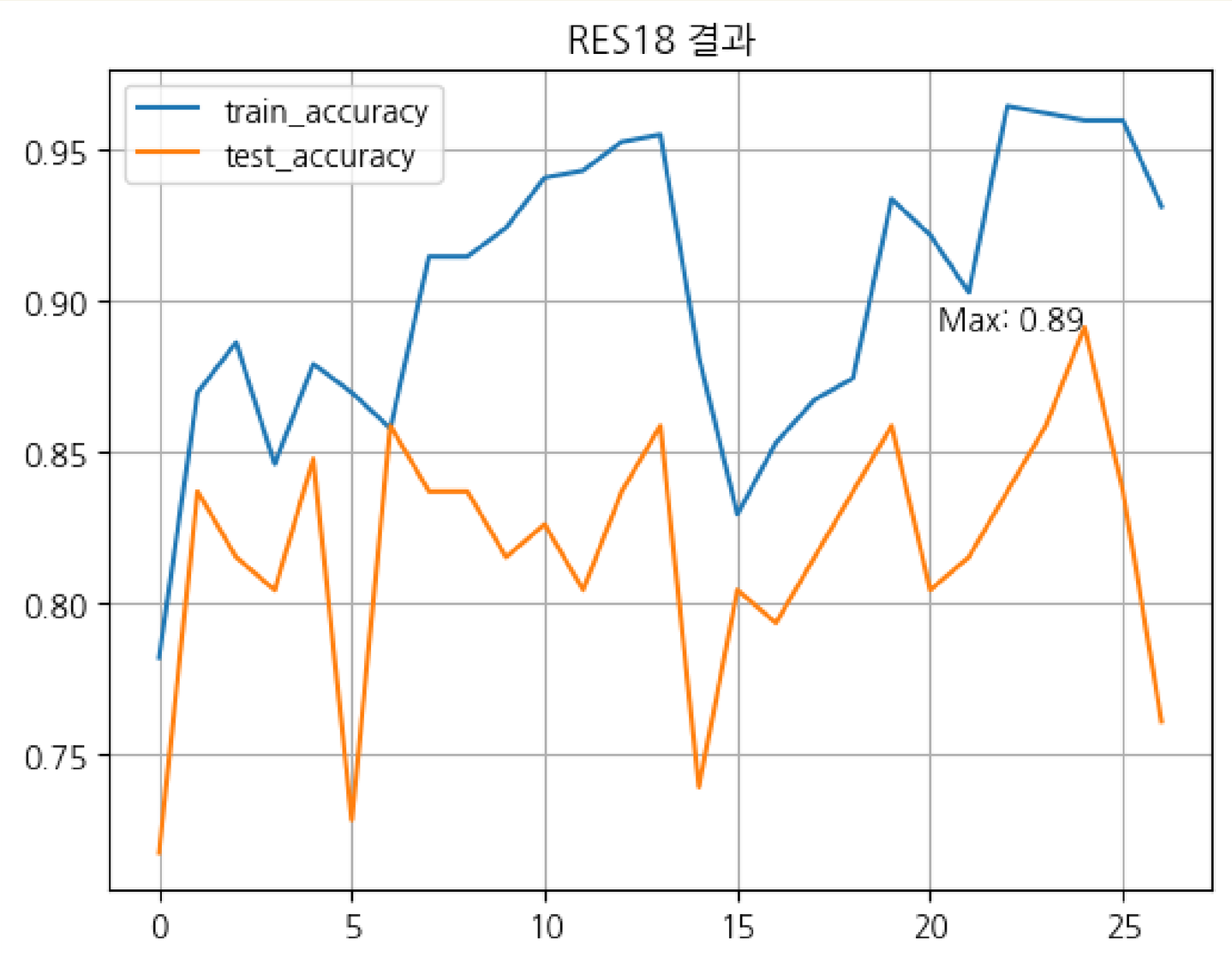
TRAIN SCORE : 90%

FIRST_MODEL.PTH

KDT

RESNOT18 사용

mop vs mop_dog



TEST SCPRE : 89%

과대적합....

분석-히스토그램

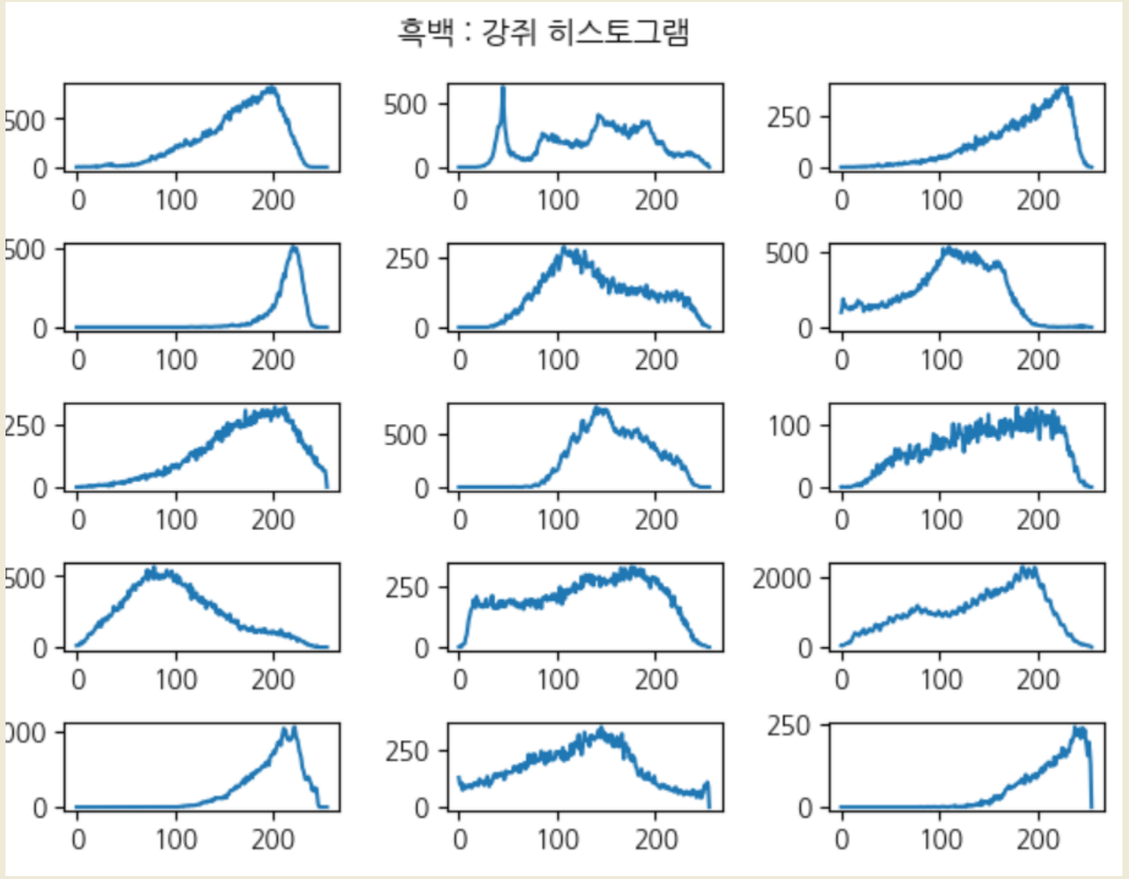

```
img_num=[0,5,7,9,12,16,19,20,21,23,24,25,26,27,33] # 15개
for i,v in enumerate(img_num):
    img = cv2.imread(f"./data/mop_dog/{v}.jpg", cv2.IMREAD_GRAYSCALE)
    hist = cv2.calcHist([img], [0], None, [256], [0,255])
    plt.subplot(5,3,i+1)
    plt.plot(hist)

plt.suptitle("흑백 : 강쥐 히스토그램")
plt.tight_layout()
plt.show()
```

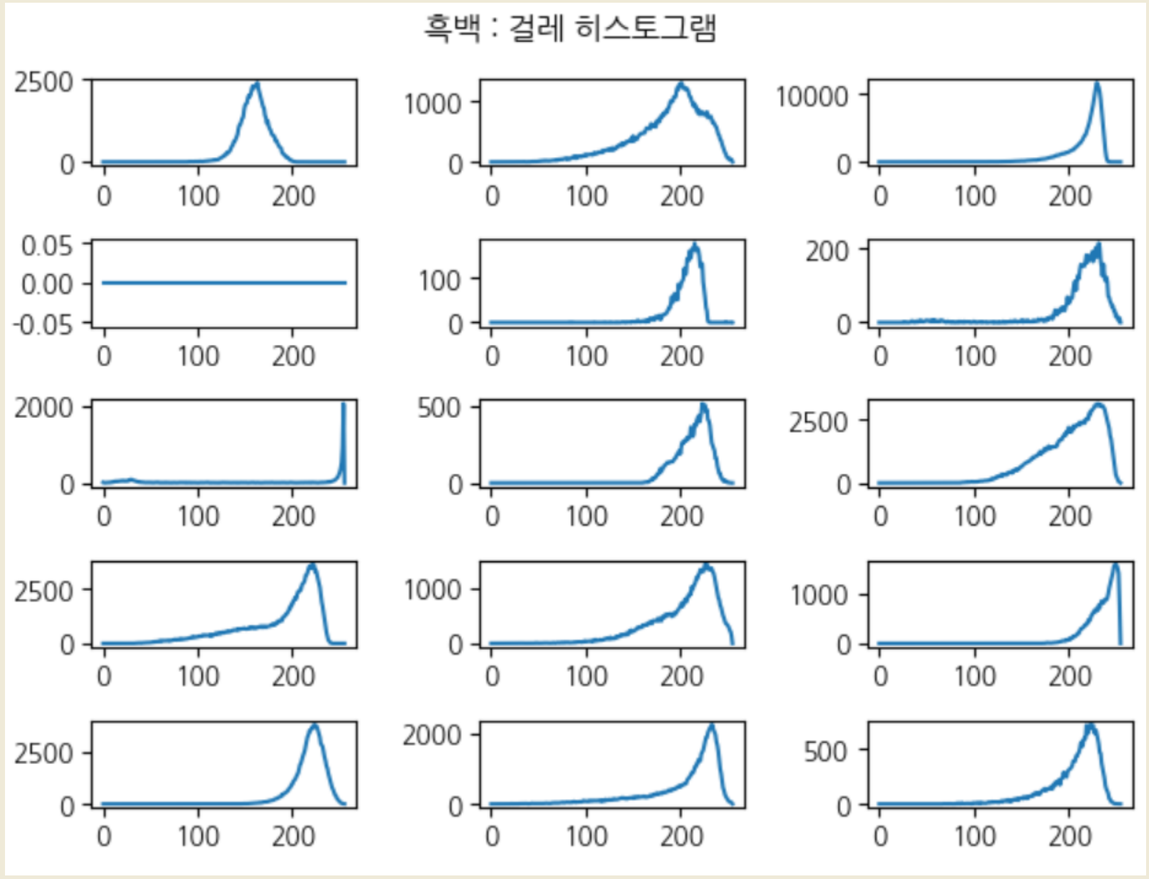
CV2.CALHIST 사용

KDT

흑백 히스토그램

mop vs mop_dog



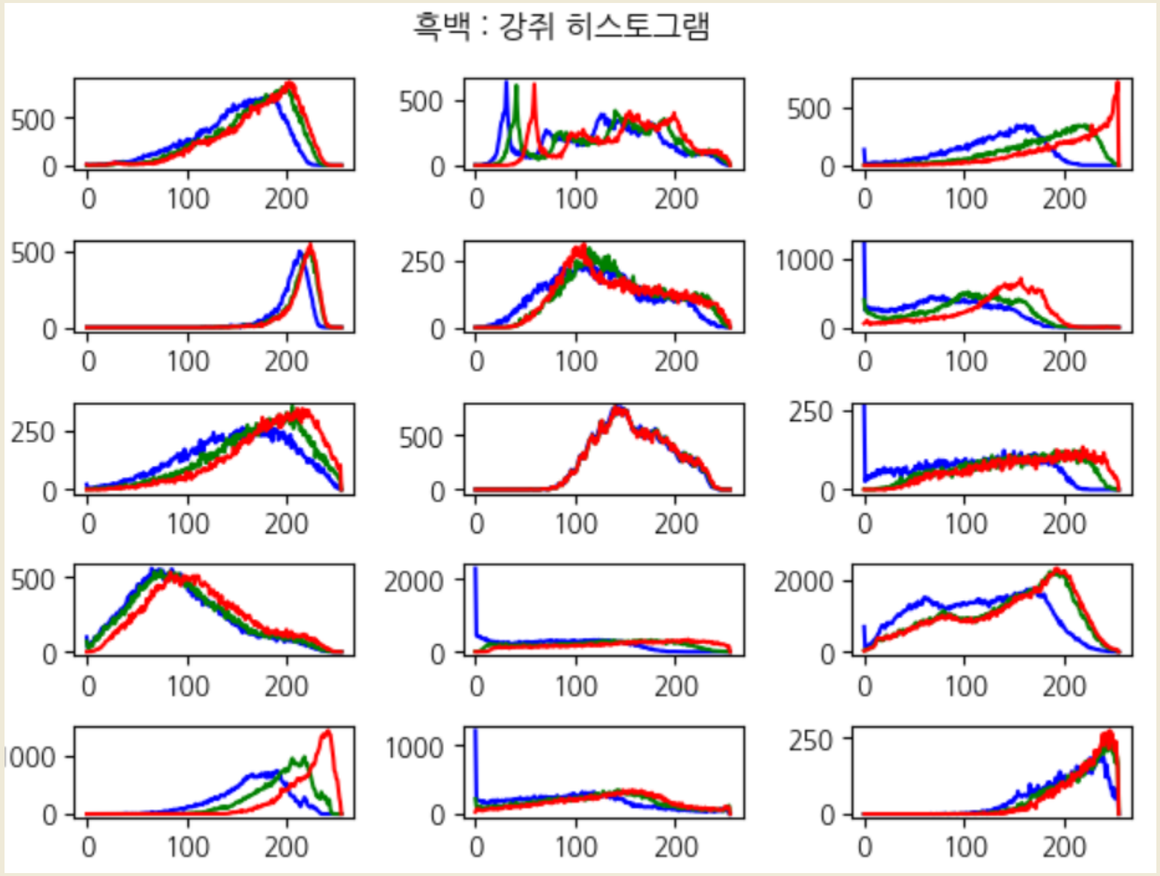


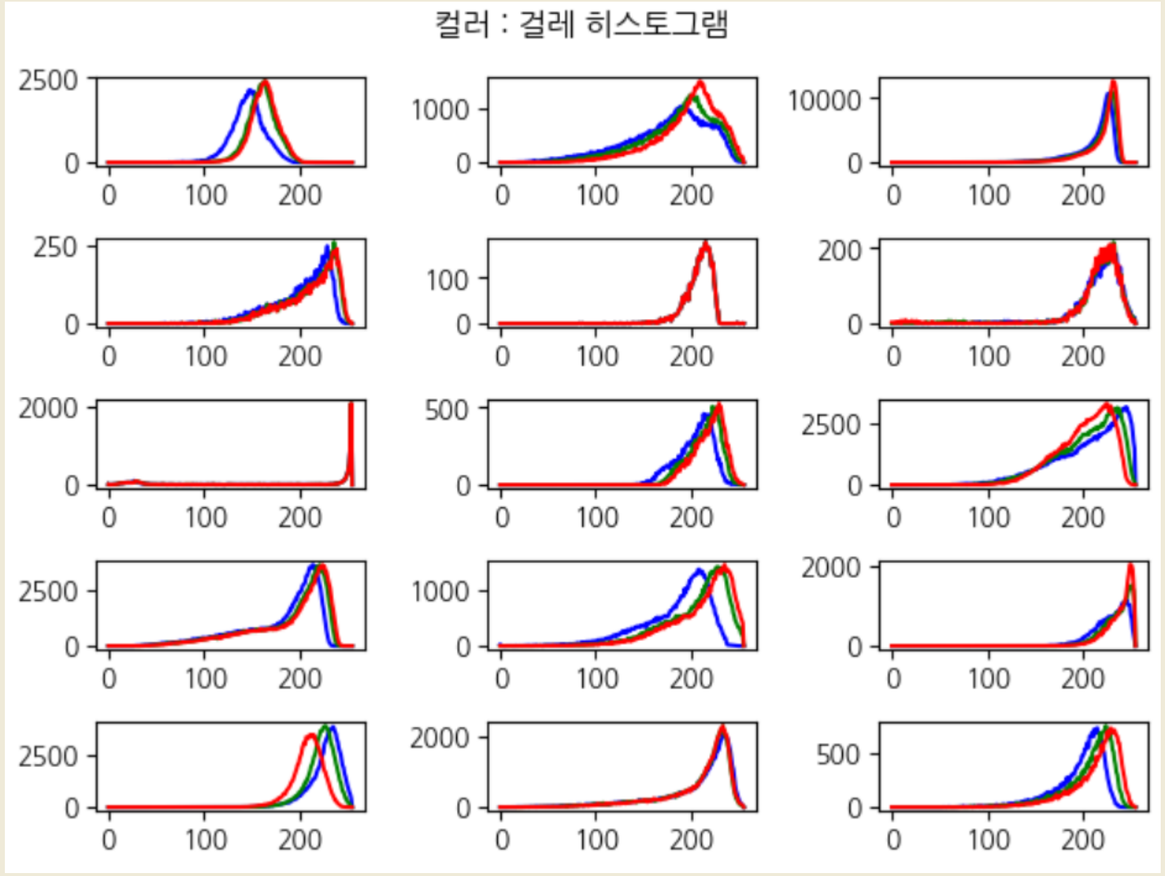
강아지 사진이 화소 분포 범위가 넓다

컬러 히스토그램

KDT

mop vs mop_dog





강아지 사진이 rgb 분포 차이가 크다!

배운 거 최대로 활용

아마 외부생활로 인한 털 색깔의 불균형함 => 오츠의 알고리즘으로 이진화

```python
def OTSU(filename):
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    _, t_130 = cv2.threshold(img, 130, 255, cv2.THRESH_BINARY)
    t, t_otsu = cv2.threshold(img, -1, 255,  cv2.THRESH_BINARY | cv2.THRESH_OTSU)
    imgs = {'Original': img, 't:130':t_130, 'otsu:%d'%t: t_otsu}
    for i , (key, value) in enumerate(imgs.items()):
        plt.subplot(1, 3, i+1)
        plt.title(key)
        plt.imshow(value, cmap='gray')
        plt.xticks([]); plt.yticks([])
    plt.show()

OTSU('./data/mop_dog/0.jpg')
```

OTSU 의 알고리즘

KDT

mop vs mop_dog

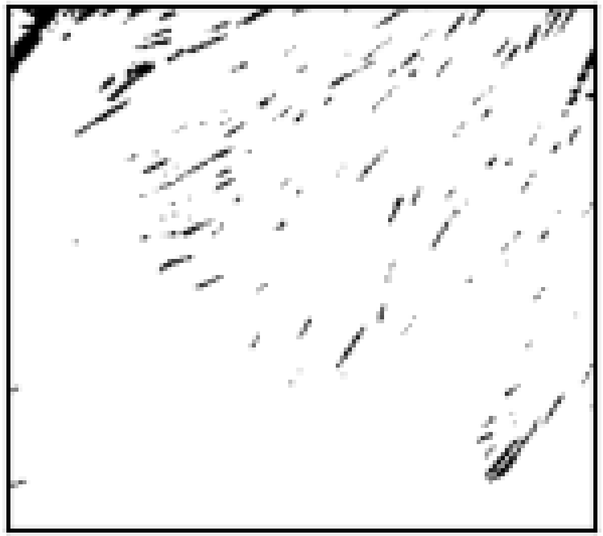
Original

t:130

otsu:154

Original

t:130



otsu:159

적응형 스레시홀드로 계산

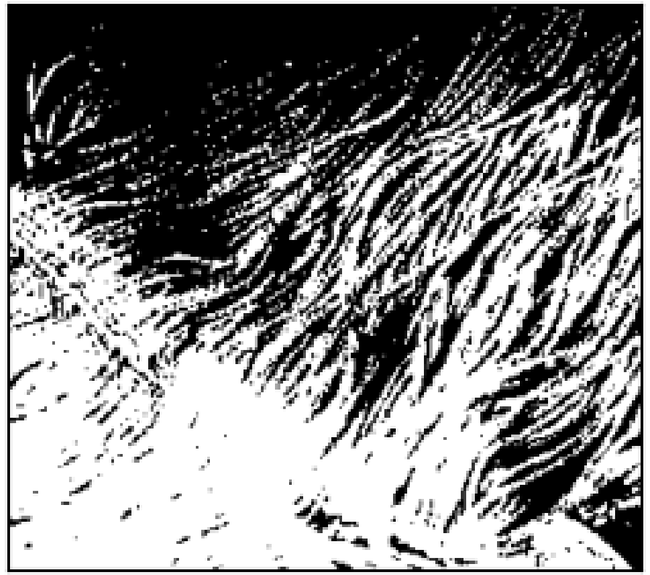

```
# 적응형 스레시홀드로 계산
def AG(filename) :
    blk_size = 9          # 블럭 사이즈
    C = 5                 # 차감 상수
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE) # 그레이 스케일로  읽기

    ret, th1 = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
    th3 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, \
                                cv2.THRESH_BINARY, blk_size, C)
    imgs = {'Global-Otsu:%d'%ret:th1,\
            'Adapted-Gaussian': th3}
    for i, (k, v) in enumerate(imgs.items()):
        plt.subplot(1,2,i+1)
        plt.title(k)
        plt.imshow(v,'gray')
        plt.xticks([]),plt.yticks([])

    plt.show()
# Adapted-Gaussian으로 더욱 정밀한 질감을 나타낸다!
AG('./data/mop/1.jpg')
```

적응형 스레시홀드로 계산

KDT

mop vs mop_dog

Global-Otsu:159

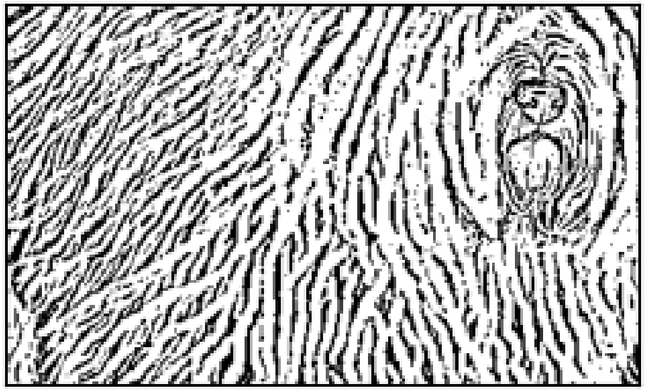


Adapted-Gaussian

OTSU < ADAPTED GAUSSIAN

Global-Otsu:154

Adapted-Gaussian



적응형 스레시홀드로 계산

KDT

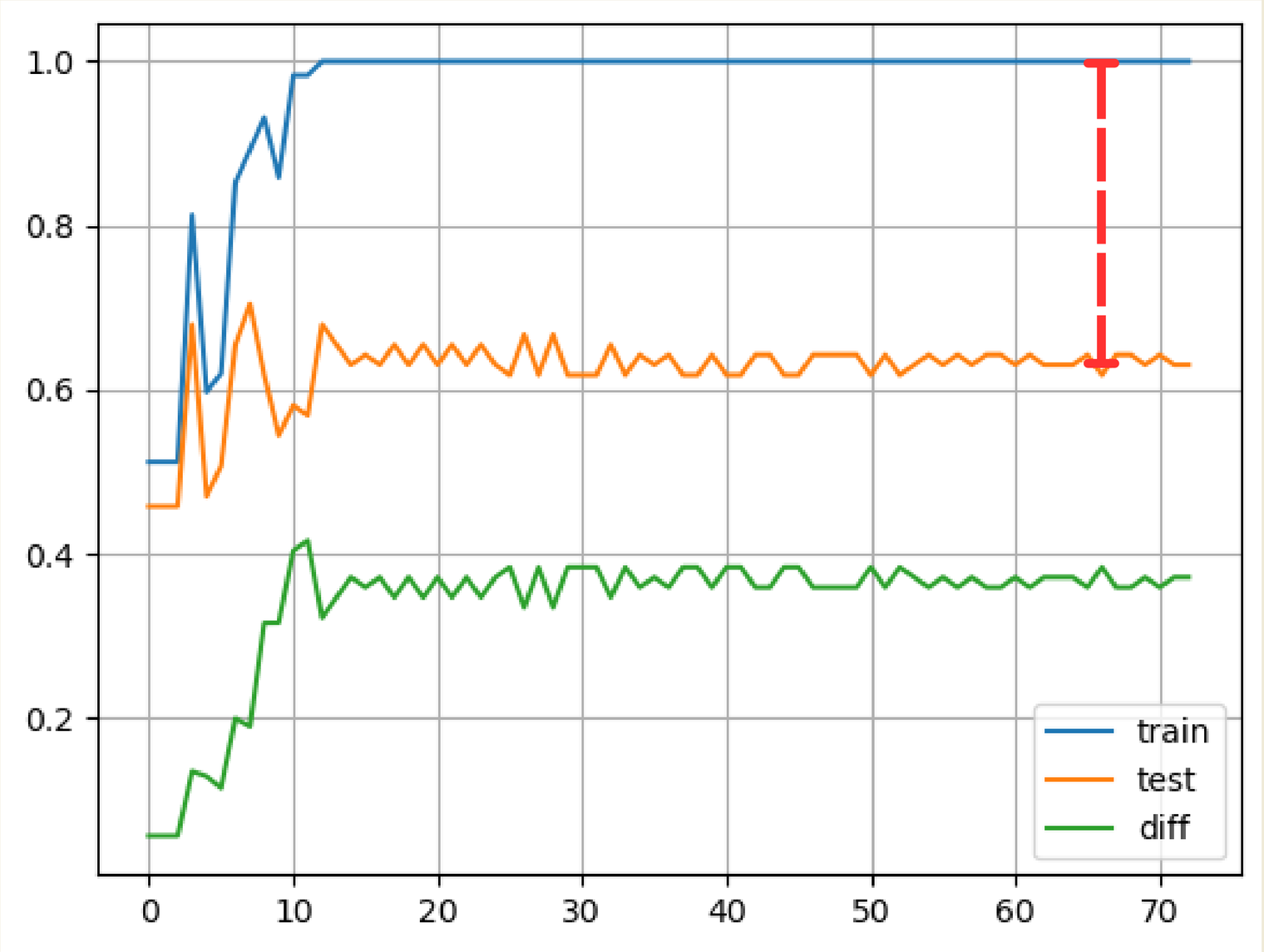
mop vs mop_dog

```python
def apply_adaptive_threshold(img):
    gray_img = cv2.cvtColor(np.array(img), cv2.COLOR_RGB2GRAY)
    th_img = cv2.adaptiveThreshold(gray_img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
    return Image.fromarray(th_img)

def apply_clahe_and_threshold(img):
    img_yuv = cv2.cvtColor(np.array(img), cv2.COLOR_RGB2YUV)
    img_clahe = img_yuv.copy()
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
    img_clahe[:,:,0] = clahe.apply(img_clahe[:,:,0])
    img_clahe = cv2.cvtColor(img_clahe, cv2.COLOR_YUV2BGR)

    return apply_adaptive_threshold(img_clahe)

# 이미지 변환 연결하기
transform = transforms.Compose([
    transforms.Resize((50, 50)),
    transforms.Lambda(apply_clahe_and_threshold),
    transforms.ToTensor()
])
```

KDT

적응형 THRESHOLD

mop vs mop_dog



TRAIN 인식 뛰어남

BUT, 과대적합...

+DROPOUT, BATCHNORM

KDT

mop vs mop_dog

```
class Mob_Dog(nn.Module):
    def __init__(self):
        super(Mob_Dog, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 9 * 9, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 2)
        self.dropout = nn.Dropout(p=0.5)  # Dropout 추가
        self.batchnorm1 = nn.BatchNorm2d(6)  # Batch Normalization 추가
        self.batchnorm2 = nn.BatchNorm2d(16)  # Batch Normalization 추가
        self.batchnorm3 = nn.BatchNorm1d(120)  # Batch Normalization 추가
        self.batchnorm4 = nn.BatchNorm1d(84)  # Batch Normalization 추가

    def forward(self, x):
        x = self.pool(F.relu(self.batchnorm1(self.conv1(x))))
        x = self.pool(F.relu(self.batchnorm2(self.conv2(x))))  # Batch Normalization 추가

        x = x.view(-1, 16 * 9 * 9)
        x = F.relu(self.batchnorm3(self.fc1(x)))  # Batch Normalization 추가
        x = F.relu(self.batchnorm4(self.fc2(x)))  # Batch Normalization 추가
        x = self.dropout(x)  # Dropout 추가
        x = self.fc3(x)
        return x
```

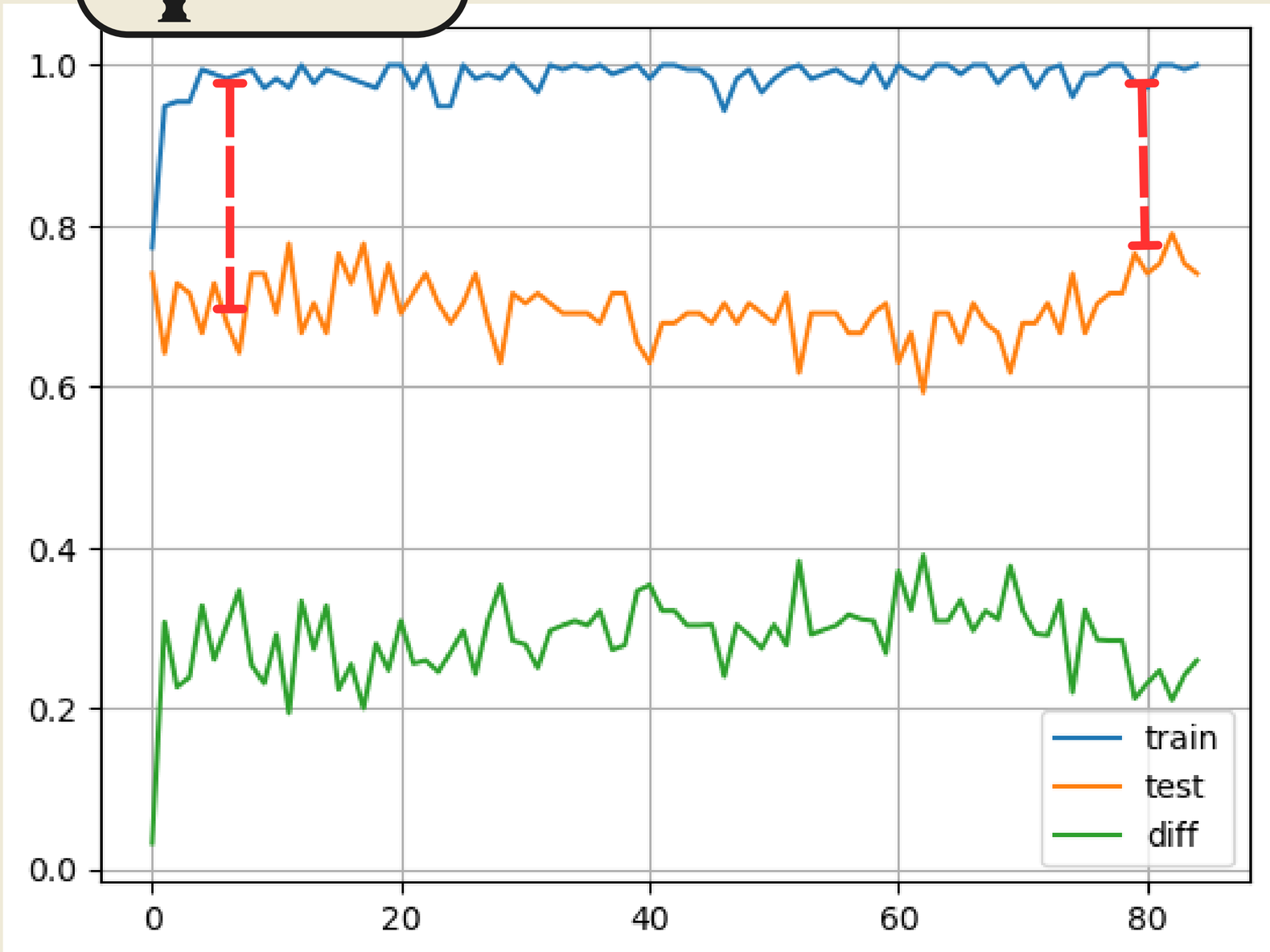
BATCHNORM X 2

DROPOUT X 1

+DROPOUT, BATCHNORM

KDT

mop vs mop_dog



과적합이 덜해짐

결과물

KDT

mop vs mop_dog

09_result_model.ipynb

+ apply_adaptive_threshold(img)
+ apply_clahe_and_threshold(img)
+class : Mob_Dog(nn.Module)
+ get_accuracy(loader, model)

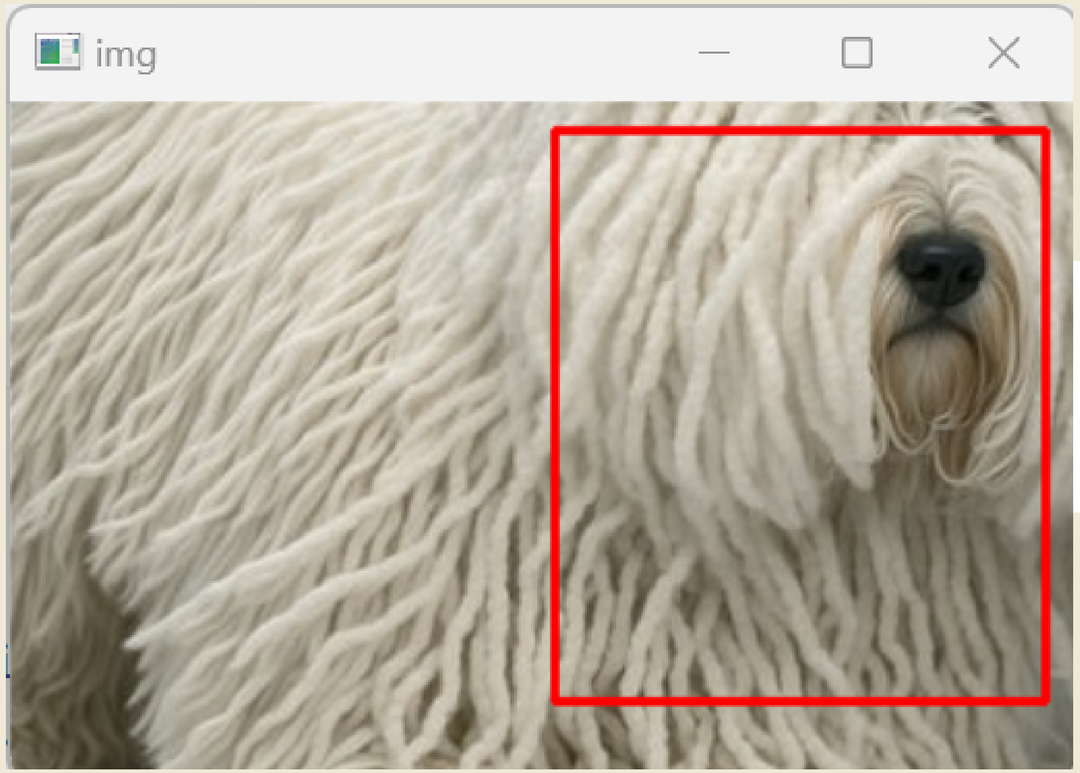
```
from dog_class import Mob_Dog
from dog_class import apply_clahe_and_threshold
from dog_class import apply_adaptive_threshold
from dog_class import get_accuracy
```

```
transform = transforms.Compose([
    transforms.Resize((50, 50)),
    transforms.Lambda(apply_clahe_and_threshold),
    transforms.ToTensor()
])
```

0.0s

```
# 모델 객체 생성, 옵티마이저 생성
model = torch.load('model.pth') # model.pth , first_model.pth, resnet18.pth
model.eval()
```

```python
img = cv2.imread("test.jpg")
img_pil = to_pil_image(img)

transformed_img = transform(img_pil)
transformed_img = transformed_img.unsqueeze(0)

with torch.no_grad():
    output = model(transformed_img)
    _, predicted = torch.max(torch.softmax(output, dim=1).data, 1)
    print(f"{torch.softmax(output, dim=1).data[0][predicted].item():.2f} 확률로 {'강쥐!!!' if predicted.item() else '大걸레'}입니다!")
```




```
0.93 확률로 강쥐!!!입니다!
```

총 비교

	CNN	*resnet18*	gray+ CNN	*gray+ threshold+* CNN
train_acc	91%	93%	100%	99%
test_acc	90%	89%	64%	77%

서비스 구현

```
                cv2.imshow( 'img' , img)
                print("좌측 상단에서 우측 하단으로 영역을 드래그 하세요.")


try :
    img = cv2.imread("./add_data/mop_dog/0.jpg")
    cv2.imshow('img', img)
    cv2.setMouseCallback('img', onMouse, param="test.jpg")
    cv2.waitKey()
    cv2.destroyAllWindows()
except :
    pass
```

✓ 3.0s

```
x:146, y:18, w:193, h:143
croped.
```

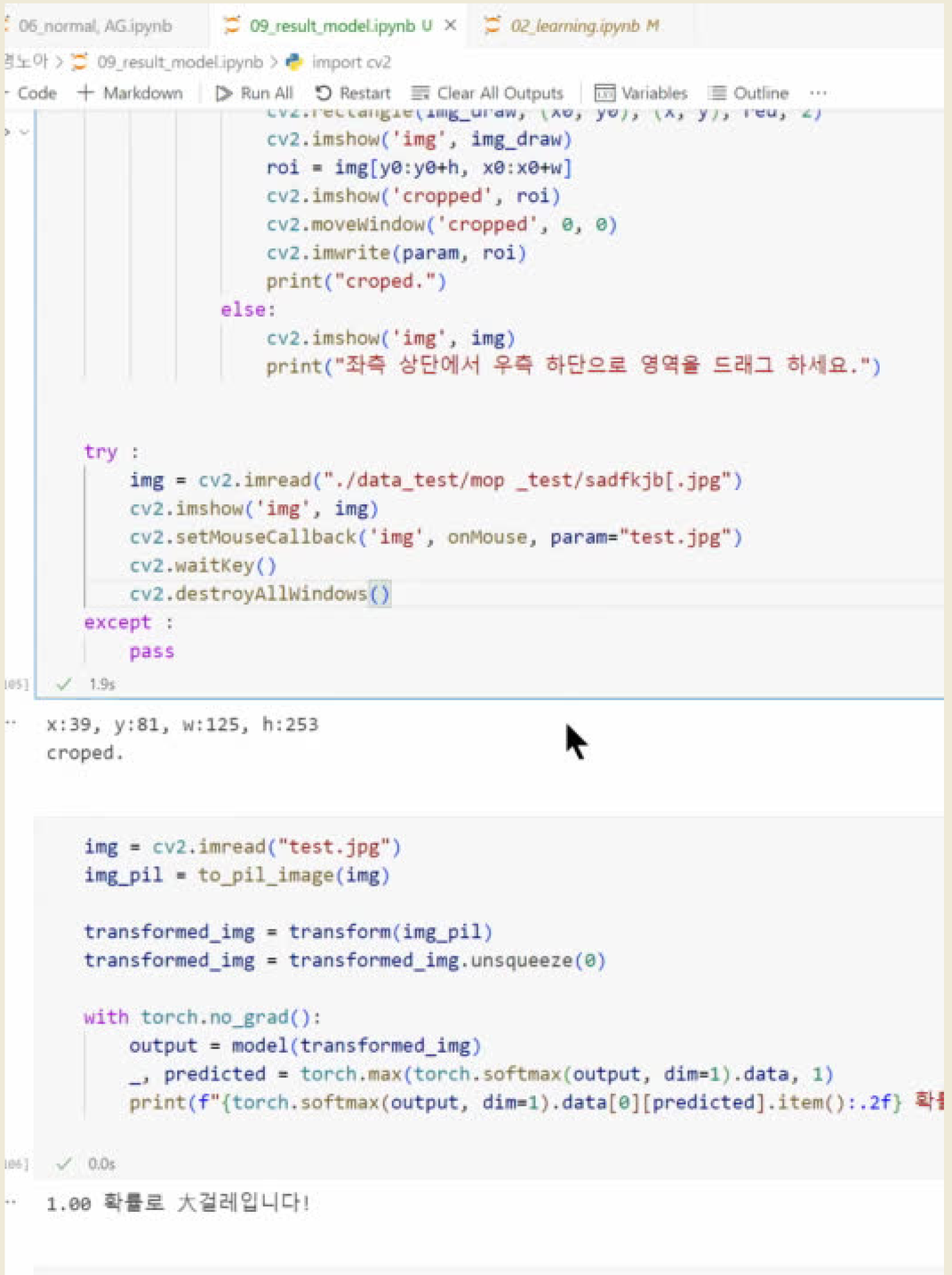
(function) unsqueeze: Any

```
img = cv2.imread("test.jpg")
img_pil = to_pil_image(img)

transformed_img = transform(img_pil)
transformed_img = transformed_img.unsqueeze(0)

with torch.no_grad():
    output = model(transformed_img)
    _, predicted = torch.max(torch.softmax(output, dim=1).data, 1)
    print(f"{torch.softmax(output, dim=1).data[0][predicted].item():.2f} 확률.
```

✓ 0.0s

```
1.00 확률로 강쥐!!!입니다!
```

서비스 구현


06_normal, AG.ipynb
09_result_model.ipynb U
02_learning.ipynb M
Code
Markdown
Run All
Restart
Clear All Outputs
Variables
Outline
cv2.imshow('img', img_draw)
roi = img[y0:y0+h, x0:x0+w]
cv2.imshow('cropped', roi)
cv2.moveWindow('cropped', 0, 0)
cv2.imwrite(param, roi)
print("croped.")
else:
cv2.imshow('img', img)
print("좌측 상단에서 우측 하단으로 영역을 드래그 하세요.")
try :
img = cv2.imread("./data_test/mop _test/sadfkjb[.jpg")
cv2.imshow('img', img)
cv2.setMouseCallback('img', onMouse, param="test.jpg")
cv2.waitKey()
cv2.destroyAllWindows()
except :
pass
1.9s
x:39, y:81, w:125, h:253
croped.
img = cv2.imread("test.jpg")
img_pil = to_pil_image(img)
transformed_img = transform(img_pil)
transformed_img = transformed_img.unsqueeze(0)
with torch.no_grad():
output = model(transformed_img)
_, predicted = torch.max(torch.softmax(output, dim=1).data, 1)
print(f"{torch.softmax(output, dim=1).data[0][predicted].item():.2f} 확
0.0s
1.00 확률로 大걸레입니다!

TEAM CONCLUSION

CNN

3채널(GBR) 데이터 : 확실히 높은 성능을 뽑아낼 수 있다
1채널(gray) 데이터 : threshold를 주어 학습 모델을 더 잘 학습시킬 수 있다 (과대적합 발생 우려 있음)

RESNET

RESNET 으로 할 시, 과대적합 우려가 있다
과대적합 발생 시, dropout + batchnorm이 큰 도움이 됨

감사합니다.