**CNN 과제**

20234014 컴퓨터 공학과 이은정

이번 과제 목표:  
 1. CNN으로 cat, dog 학습

2. 전이 학습 한 것과의 성능 차이 비교

3. 일부를 freeze 하며 성능 비교

4. 기존 데이터 증감 후 정확도 비교

# 헤더 코드: 필요한 라이브러리 모두 호출 CNN.ipynb에 있던 코드와 거의 유사

1. import torch

2. import torch.nn as nn

3. import torchvision

4. from torchvision import transforms

5. from torch.utils.data import DataLoader

6. import torchvision.models as models

7. from tqdm.autonotebook import tqdm

8. import numpy as np

9. import seaborn as sns

10. import matplotlib.pyplot as plt

11. from matplotlib.pyplot import imshow

12. from sklearn.metrics import accuracy\_score

13.

14. from pylib import train\_network, set\_seed

15.

16. import os

17.

시각화와 학습을 위한 기타 라이브러리 호출한다.

다만, 이번엔 실제 컴퓨터 내부에 존재하는 이미지들을 학습할 것이기 때문에 import os를 하여, 실제 이미지 파일들에 접근할 수 있도록 하였다.

# Cat, dog 이미지 호출

1. from torch.utils.data import DataLoader, random\_split

2. from torchvision import transforms, datasets

3.

1. all\_images\_cat\_and\_dog = datasets.ImageFolder(

2. "./data/PetImages",

3. transform=transforms.Compose([

4. transforms.Resize(130),

5. transforms.CenterCrop(128),

6. transforms.ToTensor(),

7. ])

8. )

9.

10. train\_size = int(len(all\_images\_cat\_and\_dog) \* 0.8)

11. test\_size = len(all\_images\_cat\_and\_dog) - train\_size

12. train\_data, test\_data = random\_split(all\_images\_cat\_and\_dog, [train\_size, test\_size])

13.

14. catdog\_train\_loader = DataLoader(train\_data, batch\_size=64, shuffle=True)

15. catdog\_test\_loader = DataLoader(test\_data, batch\_size=64)

16.

Cat, dog 이미지를 불러오는 코드.

Image는 (130,130)으로 crop 되었다 중심을 기준으로 (128,128)로 크롭된다. 전처리된 이미지는 ToTensor을 통해 학습할 수 있는 형태로 변화시켜준다.

catdog\_train\_loader = DataLoader(train\_data, batch\_size=64, shuffle=True)

catdog\_test\_loader = DataLoader(test\_data, batch\_size=64)

Dataloader을 통해 해당 데이터를 학습할 수 있는 형태로 변화시킨다. Train, test 데이터를 넣고, 학습을 진행하는 train\_loader만 shuffle을 true로 설정해, 과적합을 예방한다.

# X\_Ray data 불러오기

1. xray\_transform = transforms.Compose([

2. transforms.Resize(130),

3. transforms.CenterCrop(128),

4. transforms.Grayscale(num\_output\_channels=3),

5. transforms.ToTensor()

6. ])

7.

8. xray\_train\_dataset = torchvision.datasets.ImageFolder("./data/xray\_data/train", transform=xray\_transform)

9. xray\_test\_dataset = torchvision.datasets.ImageFolder("./data/xray\_data/test", transform=xray\_transform)

10.

11. xray\_train\_loader = DataLoader(xray\_train\_dataset, batch\_size=64, shuffle=True)

12. xray\_test\_loader = DataLoader(xray\_test\_dataset, batch\_size=64)

13.

마찬가지로 x\_Ray data 또한 불러온다. 해당 이미지는 gray\_Scale이지만, 원활한 학습을  
위해 chaneels=3을 해주어 3차원으로 데이터를 전처리한다.

xray\_train\_loader = DataLoader(xray\_train\_dataset, batch\_size=64, shuffle=True)

xray\_test\_loader = DataLoader(xray\_test\_dataset, batch\_size=64)

각 학습 데이터는 ./train, ./test 의 폴더 안에 들어있으므로, 해당 폴더에 접근해 데이터를 불러와 전처리한다. 해당 데이터 또한 dataloader을 통해 데이터를 학습할 수 있는 형태로 변화시킨다. 마찬가지로 train\_Data의 과적합을 방지하기 위해, train에만 shuffle=true를 설정한다.

# 학습 모델:

본 보고서에서는 전이 학습 기반의 모델 성능 비교를 위해 총 7가지 모델을 실험하였다. 모델은 크게 Custom CNN, VGG16 계열, ResNet18 계열로 나눌 수 있으며, 각 모델은 파라미터 고정(freeze) 범위에 따라 다시 세분화하여 학습을 수행하였다.

Custom CNN: 직접 구현한 합성곱 신경망으로, 비교 기준으로 활용됨. 모든 파라미터를 처음부터 학습.

VGG16 (full): VGG16 전체를 고정(freeze)한 후, fully connected layer만 새롭게 학습.

VGG16 (freeze\\_features)\*\*: feature extractor 부분을 고정하고, classifier 부분만 학습.

VGG16 (freeze\_half)\*\*: VGG16의 하위 절반 레이어만 고정하고, 상위 절반과 classifier는 fine-tuning.

VGG16 (unfreeze\_all): 전체 레이어를 모두 학습 대상에 포함시켜 완전 fine-tuning 수행.

ResNet18 (freeze\_early): 초기 계층만 고정하고 나머지 계층은 fine-tuning

ResNet18 (freeze\_half): 중간 정도까지 고정하고 이후 계층은 학습 가능하게 설정.

ResNet18 (unfreeze\_all): 모든 계층을 학습하도록 설정하여 full fine-tuning을 적용.

이러한 구성은 전이 학습에서 fine-tuning의 범위가 모델 성능에 어떤 영향을 주는지를 비교 분석하는 데 목적이 있다. 특히, 동일한 백본(예: VGG16, ResNet18)을 사용하더라도 freeze 범위에 따라 학습 효과가 달라질 수 있음을 실험을 통해 관찰하고자 하였다.

# CAT, DOG 구분 CNN

1. class CatDogCNN(nn.Module):

2. def \_\_init\_\_(self):

3. super(CatDogCNN, self).\_\_init\_\_()

4. self.conv = nn.Sequential(

5. nn.Conv2d(3, 32, kernel\_size=3, padding=1),

6. nn.ReLU(),

7. nn.MaxPool2d(2),

8. nn.Conv2d(32, 64, kernel\_size=3, padding=1),

9. nn.ReLU(),

10. nn.MaxPool2d(2)

11. )

12. self.fc1 = nn.Linear(64 \* 32 \* 32, 128)

13. self.fc2 = nn.Linear(128, 2)

14.

15. def forward(self, x):

16. x = self.conv(x)

17. x = x.view(x.size(0), -1)

18. x = self.fc1(x)

19. x = torch.relu(x)

20. x = self.fc2(x)

21. return x

22.

가장 기본적인 CNN 구조이다. 개, 고양이 데이터는 RGB 형태를 가진 3차원 데이터이다. 해당 데이터를 입력 받은 이후, Conv2d 레이어 2번을 거쳐, 해당 데이터의 특징을 뽑는다. 그리고 학습된 데이터는 2번의 fc 레이어를 거쳐 최종 데이터로 반환한다. 현재는 개, 고양이 2개의 모델 중 하나로 데이터를 반환해야 하므로, fc 레이어의 output 변수는 2가 된다.

# X\_RAY 구분 CNN

1. class XrayCNN(nn.Module):

2. def \_\_init\_\_(self):

3. super(XrayCNN, self).\_\_init\_\_()

4. self.conv = nn.Sequential(

5. nn.Conv2d(3, 32, kernel\_size=3, padding=1),

6. nn.BatchNorm2d(32),

7. nn.ReLU(),

8. nn.MaxPool2d(2),

9. nn.Conv2d(32, 64, kernel\_size=3, padding=1),

10. nn.BatchNorm2d(64),

11. nn.ReLU(),

12. nn.MaxPool2d(2),

13. )

14. self.fc1 = nn.Linear(64 \* 32 \* 32, 128)

15. self.fc2 = nn.Linear(128, 2)

16.

17. def forward(self, x):

18. x = self.conv(x)

19. x = x.view(x.size(0), -1)

20. x = self.fc1(x)

21. x = torch.relu(x)

22. x = self.fc2(x)

23. return x

24.

위 모델 또한 개, 고양이 분류 모델과 비슷하다. 데이터를 받아서 특징을 추출한 이후에 2개 데이터 중 하나로 데이터를 반환한다.

다만, 이번에는 gray\_Scale 데이터를 일부러 3차원의 RGB 데이터로 받아들였다. 추후 동일한 데이터를 다른 모델에도 적용해 그 결과를 보아야 하기 때문에, 3차원으로 데이터를 받아들여 데이터 특징을 추출한다. 이 때, 조금 더 정확한 결과 도출을 위해 BATCH\_NORMALIZATION을 추가해주었다. 흑백 데이터에서 비교적 더 정확한 특징을 추출할 수 있다. 이 데이터도 정상과, 비정상 2개의 학습 데이터로 구성이 되어 있기 때문에 최종 결과는 2개 중 하나로 return한다.

# 각 모델별로 학습을 하기 위해 모델을 전환하는 함수이다.

1. def get\_transfer\_model(base='vgg16', freeze\_level='unfreeze\_all', input\_channel=3):

2. if base == 'resnet18':

3. model = models.resnet18(pretrained=True)

4. if input\_channel == 1:

5. model.conv1 = nn.Conv2d(1, 64, kernel\_size=7, stride=2, padding=3, bias=False)

6. nn.init.kaiming\_normal\_(model.conv1.weight, mode='fan\_out', nonlinearity='relu')

7. model.fc = nn.Linear(model.fc.in\_features, 2)

8.

9. # Freeze strategy for ResNet18

10. if freeze\_level == 'full':

11. for param in model.parameters():

12. param.requires\_grad = False

13. for param in model.fc.parameters():

14. param.requires\_grad = True

15.

16. elif freeze\_level == 'freeze\_early': # freeze conv1 + layer1

17. for name, child in model.named\_children():

18. if name in ['conv1', 'bn1', 'layer1']:

19. for param in child.parameters():

20. param.requires\_grad = False

21.

22. elif freeze\_level == 'freeze\_half': # freeze conv1 + layer1 + layer2

23. for name, child in model.named\_children():

24. if name in ['conv1', 'bn1', 'layer1', 'layer2']:

25. for param in child.parameters():

26. param.requires\_grad = False

27.

28. elif freeze\_level == 'unfreeze\_all':

29. for param in model.parameters():

30. param.requires\_grad = True

31.

32. elif base == 'vgg16':

33. model = models.vgg16(pretrained=True)

34. if input\_channel == 1:

35. features = list(model.features)

36. first\_conv = nn.Conv2d(1, 64, kernel\_size=3, padding=1)

37. nn.init.kaiming\_normal\_(first\_conv.weight, mode='fan\_out', nonlinearity='relu')

38. features[0] = first\_conv

39. model.features = nn.Sequential(\*features)

40. model.classifier[6] = nn.Linear(model.classifier[6].in\_features, 2)

41.

42. # Freeze strategy for VGG16

43. if freeze\_level == 'full':

44. for param in model.parameters():

45. param.requires\_grad = False

46. for param in model.classifier[6].parameters():

47. param.requires\_grad = True

48.

49. elif freeze\_level == 'freeze\_features':

50. for param in model.features.parameters():

51. param.requires\_grad = False

52. for param in model.classifier.parameters():

53. param.requires\_grad = True

54.

55. elif freeze\_level == 'freeze\_half':

56. for param in model.features[:15].parameters(): # 앞부분만 freeze

57. param.requires\_grad = False

58. for param in model.features[15:].parameters():

59. param.requires\_grad = True

60. for param in model.classifier.parameters():

61. param.requires\_grad = True

62.

66.

67. else:

68. raise ValueError("지원하지 않는 모델입니다.")

69.

70. return model

63. elif freeze\_level == 'unfreeze\_all':

64. for param in model.parameters():

65. param.requires\_grad = True

71.

해당 모델은 총 3차원의 데이터를 받아 VGG, ResNet18 두 모델 중 하나로 데이터를 학습한다. 기존 gray\_scale에서 학습을 진행했을 때 채널을 1로 고정했음에도 데이터가 3차원으로 계속해서 바뀌는 오류가 발생하여, chaneel을 3으로 고정했다. 따라서 channel이 1인 경우의 코드는 여기서 사용하지 않는다.

우선 ResNet에서 full, 즉 모든 레이어를 얼린 경우를 살핀다.

. # Freeze strategy for ResNet18

10. if freeze\_level == 'full':

11. for param in model.parameters():

12. param.requires\_grad = False

13. for param in model.fc.parameters():

14. param.requires\_grad = True

모든 레이어를 얼릴 경우, 해당 레이어의 parameter을 업데이트 하지 않는ㄷ. 따라서 모델의 모든 parameter에 대해 해당 파라미터 학습을 false로 한다. 다만 해당 모델을 전이학습하기 위해 fc 레이어는 2개로 업데이트가 필요하므로, 해당 레이어의 파라미터만 true로 설정한다.

아래는 conv1을 포함한 소수 모델에 대해 학습을 freeze하는 경우이다.

elif freeze\_level == 'freeze\_early': # freeze conv1 + layer1

17. for name, child in model.named\_children():

18. if name in ['conv1', 'bn1', 'layer1']:

19. for param in child.parameters():

20. param.requires\_grad = False

21.

22. elif freeze\_level == 'freeze\_half': # freeze conv1 + layer1 + layer2

23. for name, child in model.named\_children():

24. if name in ['conv1', 'bn1', 'layer1', 'layer2']:

25. for param in child.parameters():

26. param.requires\_grad = False

코드에 나와있는 것 처럼, 레이어의 이름이 conv1, bn1, layer1 중 하나라면 해당 데이터의 파라미터는 학습을 금한다. 즉슨 나머지 데이터에 대한 학습은 진행한다는 의미이다.

Freeze를 반 하게 된다면, freeze하는 레이어 개수가 더 늘어나게 된다.

28. elif freeze\_level == 'unfreeze\_all':

29. for param in model.parameters():

30. param.requires\_grad = True

따라서 학습하는 레이어수를 점차 늘리다보면 위와 같이 모든 레이어의 파라미터가 학습할 수 있도록 변화할 수 있다.

VGG도 위와 동일한 코드 구조를 가진다.

. # Freeze strategy for VGG16

43. if freeze\_level == 'full':

44. for param in model.parameters():

45. param.requires\_grad = False

46. for param in model.classifier[6].parameters():

47. param.requires\_grad = True

마지막 Fc 레이어를 제외하고 모든 데이터를 학습하지 않도록 설정할 수 있고,

1. elif freeze\_level == 'freeze\_features':

2. 50. for param in model.features.parameters():

3. 51. param.requires\_grad = False

4. 52. for param in model.classifier.parameters():

5. 53. param.requires\_grad = True

6. 54.

7. 55. elif freeze\_level == 'freeze\_half':

8. 56. for param in model.features[:15].parameters(): # 앞부분만 freeze

9. 57. param.requires\_grad = False

10. 58. for param in model.features[15:].parameters():

11. 59. param.requires\_grad = True

12. 60. for param in model.classifier.parameters():

13. 61. param.requires\_grad = True

14.

모든 classifier을 학습하고 그 외 코드를 freeze하거나, 혹은 특정 부분까지만 freeze를 진행할 수도 있다.

63. elif freeze\_level == 'unfreeze\_all':

64. for param in model.parameters():

65. param.requires\_grad = True

마지막으로 모든 레이어를 학습할 수도 있다,

# 전이학습을 위한 전처리 코드

1. class NormalizeInput(nn.Module):

2.     def \_\_init\_\_(self, baseModel, input\_channel=3):

3.         super(NormalizeInput, self).\_\_init\_\_()

4.         self.baseModel = baseModel

5.         if input\_channel == 3:

6.             self.mean = nn.Parameter(torch.tensor([0.485, 0.456, 0.406]).view(1,3,1,1), requires\_grad=False)

7.             self.std = nn.Parameter(torch.tensor([0.229, 0.224, 0.225]).view(1,3,1,1), requires\_grad=False)

8.         elif input\_channel == 1:

9.             self.mean = nn.Parameter(torch.tensor([0.5]).view(1,1,1,1), requires\_grad=False)

10.             self.std = nn.Parameter(torch.tensor([0.5]).view(1,1,1,1), requires\_grad=False)

11.

12.     def forward(self, input):

13.         input = (input - self.mean) / self.std

14.         return self.baseModel(input)

15.

16.

라이브러리를 통해 받아온 model들은 ImageNet을 통해 이미 학습이 된 모델들이다. 따라서, 우리가 지정한 이미지들을 학습하고 적절한 결과를 내기 위해서는 학습하는 모델을 전처리 해줄 필요가 있다.

우리가 미리 Compose를 통해 전처리 한 데이터 파일들은 우리가 DataLoader에 데이터를 저장할 때, 실행된다. 따라서 Tensor로 보낼 이미지를 전처리 하는 것이며, 이미지 텐서로 변환하기 전 이미지를 다듬는 과정이다.

반면 현재 전처리는 PyTorch 모델을 전처리 한다. 모델이 forward를 진행할 때 전처리 되며, 모델에 들어가는 파라미터를 전처리 하는데 사용된다.

우리는 현재 ResNet, VGG와 같은 코드를 학습한다. 이 모델들은 ImageNet 데이터를 학습한 모델들이다. 따라서, 이 데이터들이 RGB 데이터를 학습할 때 가졌던 평균 및 표준편차로 다시 모델을 전처리 해줌으로서, 이미지 모델을 학습한다. 만약 흑백 채널일 경우, 0, 1 두 개의 반환 값 중 하나로 반환하기 위해 0.5 값을 사용해 전처리 한다.

입력 텐서를 정규화한 뒤 baseModel에 넘겨서 forward 진행할 때, ImageNet에서 학습된 기준(mean/std)에 맞춰야 성능이 보장될 수 있도록 자동으로 처리를 해준다.

# 학습 코드

1. def train\_model(model, train\_loader, test\_loader, device, epochs=10):

2.     model.to(device)

3.     optimizer = torch.optim.Adam(filter(lambda p: p.requires\_grad, model.parameters()), lr=0.001)

4.     criterion = nn.CrossEntropyLoss()

5.     acc\_list = []

6.

7.     for epoch in range(epochs):

8.         model.train()

9.         epoch\_loss = 0.0

10.         for X\_batch, y\_batch in train\_loader:

11.             X\_batch, y\_batch = X\_batch.to(device), y\_batch.to(device)

12.             optimizer.zero\_grad()

13.             pred = model(X\_batch)

14.             loss = criterion(pred, y\_batch)

15.             loss.backward()

16.             optimizer.step()

17.             epoch\_loss += loss.item()

18.

19.         # 평가

20.         model.eval()

21.         correct, total = 0, 0

22.         with torch.no\_grad():

23.             for X\_batch, y\_batch in test\_loader:

24.                 X\_batch, y\_batch = X\_batch.to(device), y\_batch.to(device)

25.                 pred = model(X\_batch)

26.                 predicted = torch.argmax(pred, dim=1)

27.                 correct += (predicted == y\_batch).sum().item()

28.                 total += y\_batch.size(0)

29.         acc = correct / total

30.         acc\_list.append(acc)

31.

32.         if (epoch + 1) % 2 == 0 or epoch == epochs - 1:

33.             print(f"  [Epoch {epoch+1:>2}] Loss: {epoch\_loss:.4f} | Accuracy: {acc:.4f}")

34.

35.     return acc\_list

36.

우리가 모델을 통해 학습을 하는 코드이다. EPOCH 2의 배수의 경우에 정확도를 출력한다. 해당 코드는 파라미터로 받아온 모델에 트레인 데이터를 넣고, 모델을 학습하고, Train 데이터를 사용해 모델을 평가하는 코드이다.

Train을 할 경우, model.train()을 통해 해당 데이터를 학습해 파라미터를 업데이트 할 것임을 미리 알려준다. 이 때, 모델은 freeze가 된 레이어를 가지고 있을 수도 있고, 없을 수도 있다. Freeze가 된 레이어가 있을 경우, 해당 레이어는 파라미터 업데이트를 진행하지 않는다.

13.             pred = model(X\_batch)

Model이 gpu에 올라가 있으므로, 모든 데이터를 gpu에 올린다. 그 후, Train\_loader에 들어온 데이터를 라벨과 데이터로 분리해 해당 데이터를 model에 넣어 예측한다. 수업시간에 배웠던 대로, 모델에서 나온 예측값과 실제 라벨 값을 사용해 손실 값을 구하고, 우리가 업데이트 하고자 하는 모델의 레이어 파라미터를 업데이트 한다. (이는 fc 레이어 혹은 conv2d 레이어일 수 있다.) 이를 epoch만큼 반복해 모델을 학습한다.

20.         model.eval()

을 통해 해당 모델을 평가(파라미터 업데이트 하지 않는다,)로 바꾸고, 해당 모델을 평가한다. 해당 모델은 두 가지의 결과를 return 한다. 이 때에, 가장 큰 경우로 예측되는 것을 정답으로 고른다. 이후, 실제 그 데이터와 라벨이 일치한다면 해당 데이터는 정확도를 상승시키는 요소로서 사용되고, 그렇지 않을 경우, 다음 평가를 진행한다,

최종적으로 평가를 마쳤을 때, 최종 맞은 개수 / 전체 개수를 통해 정확도를 계산한다.

실제 학습 코드

1. import copy

2. import torch

3.

4. def run\_all\_experiments(dataset\_name, train\_loader, test\_loader, input\_channel, device):

5.     results = {}

6.     best\_acc = 0.0

7.     best\_model\_wts = None

8.     best\_model\_label = ""

9.

10.     # Custom CNN

11.     if dataset\_name == "CatDog":

12.         model = CatDogCNN()

13.     elif dataset\_name == "Xray":

14.         model = XrayCNN()

15.

16

17.

18.     print(f"▶ Training CustomCNN for {dataset\_name}...")

19.     acc\_list = train\_model(model, train\_loader, test\_loader, device)

20.     results[f'{dataset\_name}\_CustomCNN'] = acc\_list

21.     final\_acc = acc\_list[-1]

22.     print(f"  ⤷ Final Accuracy: {final\_acc:.4f}")

23.

24.     if final\_acc > best\_acc:

25.         best\_acc = final\_acc

26.         best\_model\_wts = copy.deepcopy(model.state\_dict())

27.         best\_model\_label = f'{dataset\_name}\_CustomCNN'

28.

29.     # Freeze 설정

30.     freeze\_levels\_resnet = ['full', 'freeze\_early', 'freeze\_half', 'unfreeze\_all']

31.     freeze\_levels\_vgg = ['full', 'freeze\_features', 'freeze\_half', 'unfreeze\_all']

32.

33.     for base\_model in ['resnet18', 'vgg16']:

34.         freeze\_levels = freeze\_levels\_resnet if base\_model == 'resnet18' else freeze\_levels\_vgg

35.

36.         for freeze\_type in freeze\_levels:

37.             print(f"▶ Training {base\_model.upper()} ({freeze\_type}) for {dataset\_name}...")

38.             model = get\_transfer\_model(

39.                 base=base\_model,

40.                 freeze\_level=freeze\_type,

41.                 input\_channel=input\_channel

42.             )

43.             model = NormalizeInput(model, input\_channel=input\_channel)

44.             label = f'{dataset\_name}\_{base\_model}\_{freeze\_type}'

45.             acc\_list = train\_model(model, train\_loader, test\_loader, device)

46.             results[label] = acc\_list

47.             final\_acc = acc\_list[-1]

48.             print(f"  ⤷ Final Accuracy: {final\_acc:.4f}")

49.

50.             if final\_acc > best\_acc:

51.                 best\_acc = final\_acc

52.                 best\_model\_wts = copy.deepcopy(model.state\_dict())

53.                 best\_model\_label = label

54.

55.     # 모델 저장

56.     if best\_model\_wts is not None:

57.         torch.save(best\_model\_wts, f"{dataset\_name}\_best\_model.pt")

58.         print(f"\n✅ Best model saved as '{dataset\_name}\_best\_model.pt' (Model: {best\_model\_label}, Accuracy: {best\_acc:.4f})")

59.

60.     return results

61.

62.

get\_transfer\_mode을 통해 우리가 정의한 모델을 실제로 데이터 셋과 함께 학습할 수 있게 하는 코드이다. 우리가 지정한 키워드 대로, 해당 모델을 가져와 train할 수 있도록 한다.

10.     # Custom CNN

11.     if dataset\_name == "CatDog":

12.         model = CatDogCNN()

13.     elif dataset\_name == "Xray":

14.         model = XrayCNN()

다만 이 때에, 모델에 따른 기본 CNN 코드가 약간 다르므로, 해당 데이터 이름별로, 모델을 가져올 수 있도록 지정한다.

Custom CNN 학습을 마친 이후로는 pre\_trained 모델을 사용해 데이터를 학습한다. 해당 모델은 Custom CNN이 아니라, 기존 모델을 파인 튜닝 하여 재사용하는 것이기 때문에, 정규화를 해주어야 한다. 따라서 전처리 이후, 학습을 진행한다.

우리가 최종적으로 정확도가 가장 높은 모델을 저장하여야 하기 때문에, 현재 정확도가 기존의 가장 높은 정확도보다 높을 경우, 해당 모델을 dictionary에 넣어 저장한다. 이후, 모든 학습이 끝났을 때, 해당 모델을 데이터이름\_best\_model.pt로 저장한다.

# 결과

## X\_RAY\_DATASET

▶ Training CustomCNN for Xray...

[Epoch 2] Loss: 7.0000 | Accuracy: 0.8205

[Epoch 4] Loss: 4.9141 | Accuracy: 0.8029

[Epoch 6] Loss: 2.6545 | Accuracy: 0.7676

[Epoch 8] Loss: 1.2781 | Accuracy: 0.6955

[Epoch 10] Loss: 3.9439 | Accuracy: 0.8205

⤷ Final Accuracy: 0.8205

▶ Training RESNET18 (full) for Xray...

[Epoch 2] Loss: 13.4220 | Accuracy: 0.7885

[Epoch 4] Loss: 10.5482 | Accuracy: 0.7708

[Epoch 6] Loss: 9.9828 | Accuracy: 0.7933

[Epoch 8] Loss: 9.6707 | Accuracy: 0.8045

[Epoch 10] Loss: 9.2174 | Accuracy: 0.7612

⤷ Final Accuracy: 0.7612

▶ Training RESNET18 (freeze\_early) for Xray...

[Epoch 2] Loss: 2.6981 | Accuracy: 0.7724

[Epoch 4] Loss: 1.5572 | Accuracy: 0.7147

[Epoch 6] Loss: 1.2527 | Accuracy: 0.8189

[Epoch 8] Loss: 0.2944 | Accuracy: 0.8478

[Epoch 10] Loss: 0.9284 | Accuracy: 0.8686

⤷ Final Accuracy: 0.8686

▶ Training RESNET18 (freeze\_half) for Xray...

[Epoch 2] Loss: 2.1194 | Accuracy: 0.7821

[Epoch 4] Loss: 1.4930 | Accuracy: 0.8301

[Epoch 6] Loss: 1.1383 | Accuracy: 0.7949

[Epoch 8] Loss: 0.6154 | Accuracy: 0.8686

[Epoch 10] Loss: 0.8797 | Accuracy: 0.8093

⤷ Final Accuracy: 0.8093

▶ Training RESNET18 (unfreeze\_all) for Xray...

[Epoch 2] Loss: 4.1698 | Accuracy: 0.7724

[Epoch 4] Loss: 2.1944 | Accuracy: 0.8013

[Epoch 6] Loss: 0.8647 | Accuracy: 0.7292

[Epoch 8] Loss: 1.8583 | Accuracy: 0.7981

[Epoch 10] Loss: 0.8792 | Accuracy: 0.8622

⤷ Final Accuracy: 0.8622

▶ Training VGG16 (full) for Xray...

[Epoch 2] Loss: 16.5484 | Accuracy: 0.8253

[Epoch 4] Loss: 14.3328 | Accuracy: 0.8125

[Epoch 6] Loss: 14.6010 | Accuracy: 0.7708

[Epoch 8] Loss: 14.1255 | Accuracy: 0.7708

[Epoch 10] Loss: 15.0077 | Accuracy: 0.7532

⤷ Final Accuracy: 0.7532

▶ Training VGG16 (freeze\_features) for Xray...

[Epoch 2] Loss: 15.3314 | Accuracy: 0.8478

[Epoch 4] Loss: 16.3947 | Accuracy: 0.7292

[Epoch 6] Loss: 12.3927 | Accuracy: 0.7404

[Epoch 8] Loss: 12.1690 | Accuracy: 0.8301

[Epoch 10] Loss: 7.6467 | Accuracy: 0.7997

⤷ Final Accuracy: 0.7997

▶ Training VGG16 (freeze\_half) for Xray...

[Epoch 2] Loss: 47.0032 | Accuracy: 0.6250

[Epoch 4] Loss: 46.9103 | Accuracy: 0.6250

[Epoch 6] Loss: 46.9323 | Accuracy: 0.6250

[Epoch 8] Loss: 47.0004 | Accuracy: 0.6250

[Epoch 10] Loss: 46.9095 | Accuracy: 0.6250

⤷ Final Accuracy: 0.6250

▶ Training VGG16 (unfreeze\_all) for Xray...

[Epoch 2] Loss: 45.5960 | Accuracy: 0.6250

[Epoch 4] Loss: 12.2001 | Accuracy: 0.7853

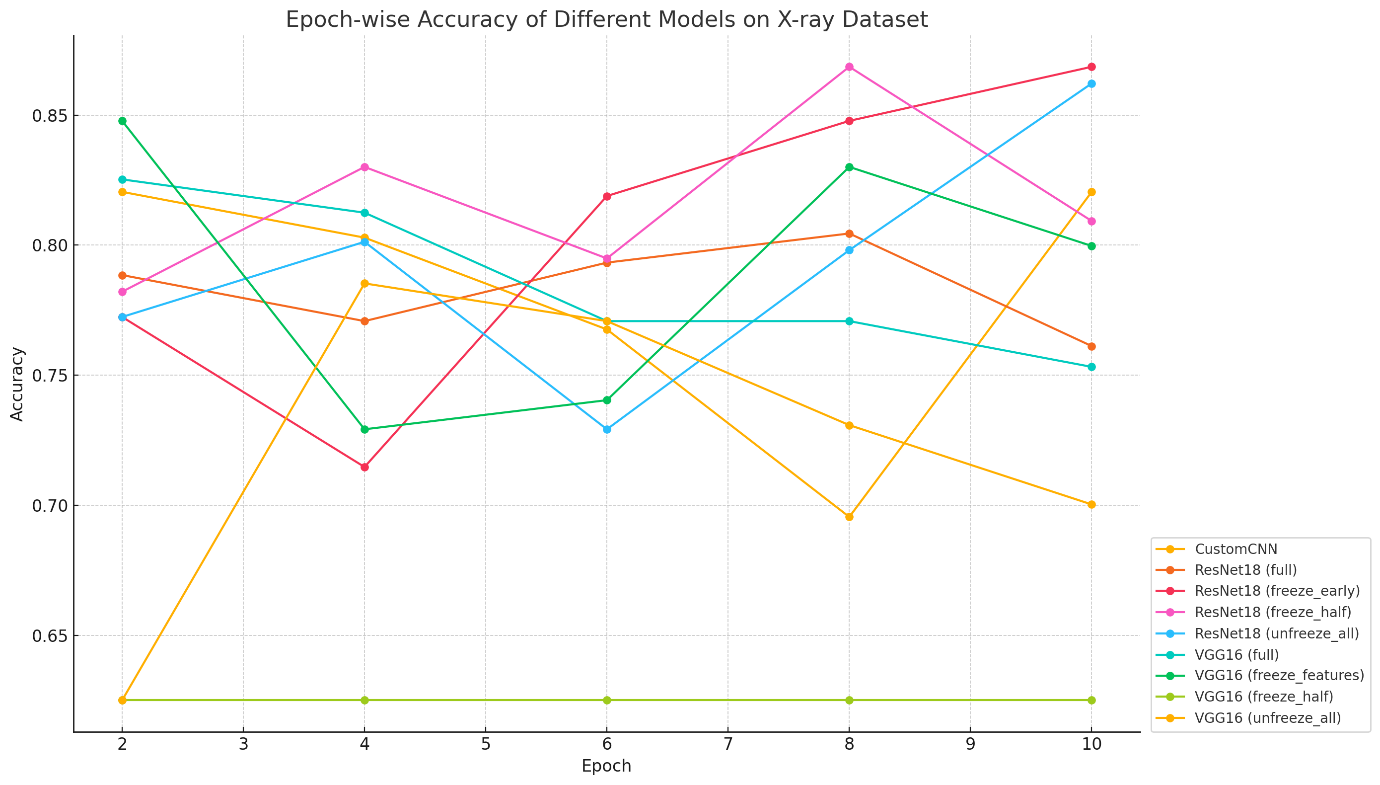
[Epoch 6] Loss: 11.2810 | Accuracy: 0.7708

[Epoch 8] Loss: 6.8941 | Accuracy: 0.7308

[Epoch 10] Loss: 9.2394 | Accuracy: 0.7003

⤷ Final Accuracy: 0.7003

✅ Best model saved as 'Xray\_best\_model.pt' (Model: Xray\_resnet18\_freeze\_early, Accuracy: 0.8686)



X\_RAY 데이터는 ImageNet의 데이터에 해당하지 않는 독특한 데이터이다. 따라서, 아무리 파인 튜닝을 한 들, 0.9의 정확도를 넘기기가 어려웠다. 위 결과를 보면, VGG보다 비교적 성능이 나은 Resnet18에서 freeze\_early를 했을 때 정확도가 높았다. freeze하는 비율이 적을수록, 즉슨, 더 많은 특징을 추출해 학습을 할수록 성능이 높게 나왔음을 알 수 있다.

## CAT, DOG DATASET

[Epoch 2] Loss: 177.0025 | Accuracy: 0.7324

[Epoch 4] Loss: 133.7871 | Accuracy: 0.7698

[Epoch 6] Loss: 89.5899 | Accuracy: 0.7638

[Epoch 8] Loss: 33.1151 | Accuracy: 0.7492

[Epoch 10] Loss: 8.0781 | Accuracy: 0.7586

⤷ Final Accuracy: 0.7586

▶ Training RESNET18 (full) for CatDog...

[Epoch 2] Loss: 52.2425 | Accuracy: 0.9378

[Epoch 4] Loss: 50.2946 | Accuracy: 0.9382

[Epoch 6] Loss: 47.3312 | Accuracy: 0.9408

[Epoch 8] Loss: 46.8324 | Accuracy: 0.9398

[Epoch 10] Loss: 46.9638 | Accuracy: 0.9402

⤷ Final Accuracy: 0.9402

▶ Training RESNET18 (freeze\_early) for CatDog...

[Epoch 2] Loss: 23.3382 | Accuracy: 0.9600

[Epoch 4] Loss: 11.5151 | Accuracy: 0.9612

[Epoch 6] Loss: 8.3242 | Accuracy: 0.9620

[Epoch 8] Loss: 5.9494 | Accuracy: 0.9634

[Epoch 10] Loss: 4.7380 | Accuracy: 0.9642

⤷ Final Accuracy: 0.9642

▶ Training RESNET18 (freeze\_half) for CatDog...

[Epoch 2] Loss: 19.2970 | Accuracy: 0.9644

[Epoch 4] Loss: 9.2795 | Accuracy: 0.9658

[Epoch 6] Loss: 5.6042 | Accuracy: 0.9648

[Epoch 8] Loss: 3.0645 | Accuracy: 0.9680

[Epoch 10] Loss: 2.8248 | Accuracy: 0.9558

⤷ Final Accuracy: 0.9558

▶ Training RESNET18 (unfreeze\_all) for CatDog...

[Epoch 2] Loss: 31.7889 | Accuracy: 0.9570

[Epoch 4] Loss: 19.1922 | Accuracy: 0.9514

[Epoch 6] Loss: 11.7295 | Accuracy: 0.9436

[Epoch 8] Loss: 10.2674 | Accuracy: 0.9486

[Epoch 10] Loss: 9.4826 | Accuracy: 0.9462

⤷ Final Accuracy: 0.9462

▶ Training VGG16 (full) for CatDog...

[Epoch 2] Loss: 26.0919 | Accuracy: 0.9770

[Epoch 4] Loss: 28.7750 | Accuracy: 0.9764

[Epoch 6] Loss: 27.8867 | Accuracy: 0.9772

[Epoch 8] Loss: 28.3532 | Accuracy: 0.9766

[Epoch 10] Loss: 30.6952 | Accuracy: 0.9774

⤷ Final Accuracy: 0.9774

▶ Training VGG16 (freeze\_features) for CatDog...

[Epoch 2] Loss: 78.8564 | Accuracy: 0.9708

[Epoch 4] Loss: 68.5512 | Accuracy: 0.9770

[Epoch 6] Loss: 84.6545 | Accuracy: 0.9728

[Epoch 8] Loss: 71.4004 | Accuracy: 0.9764

[Epoch 10] Loss: 71.2085 | Accuracy: 0.9794

⤷ Final Accuracy: 0.9794

▶ Training VGG16 (freeze\_half) for CatDog...

[Epoch 2] Loss: 217.5414 | Accuracy: 0.4992

[Epoch 4] Loss: 217.3637 | Accuracy: 0.5008

[Epoch 6] Loss: 217.1085 | Accuracy: 0.4992

[Epoch 8] Loss: 217.2288 | Accuracy: 0.4992

[Epoch 10] Loss: 217.1043 | Accuracy: 0.5008

⤷ Final Accuracy: 0.5008

▶ Training VGG16 (unfreeze\_all) for CatDog...

[Epoch 2] Loss: 177.0754 | Accuracy: 0.6338

[Epoch 4] Loss: 104.8445 | Accuracy: 0.8826

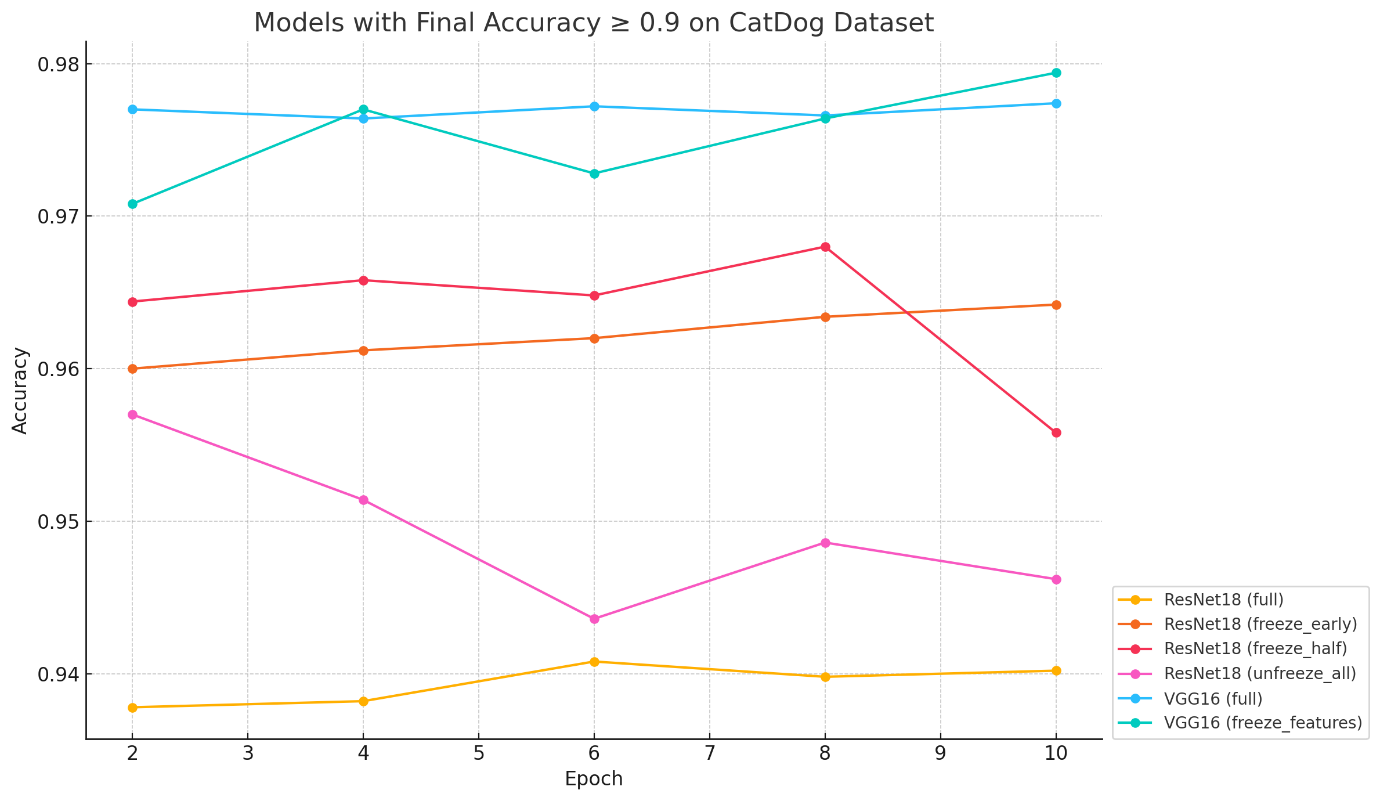
[Epoch 6] Loss: 128.2544 | Accuracy: 0.5008

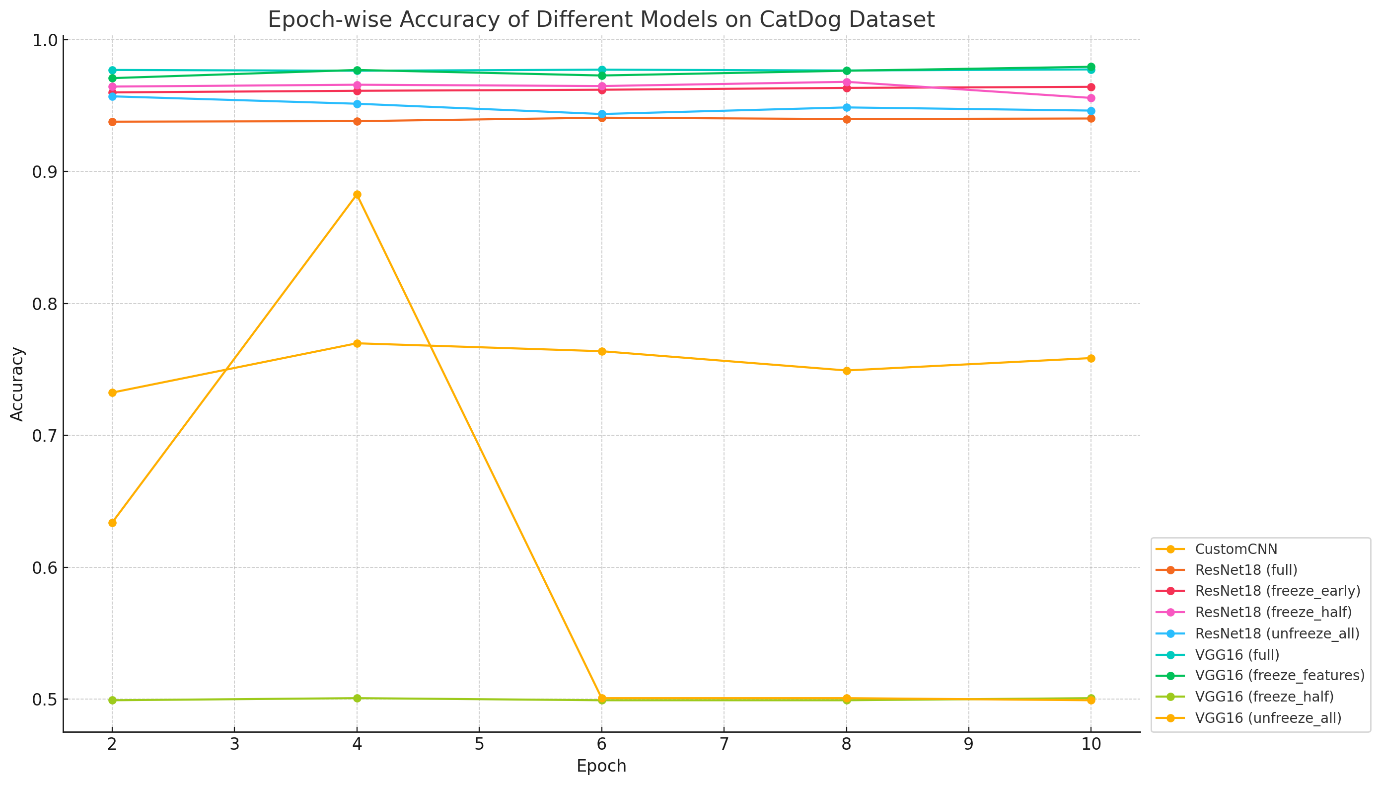
[Epoch 8] Loss: 217.2294 | Accuracy: 0.5008

[Epoch 10] Loss: 217.1419 | Accuracy: 0.4992

⤷ Final Accuracy: 0.4992

✅ Best model saved as 'CatDog\_best\_model.pt' (Model: CatDog\_vgg16\_freeze\_features, Accuracy: 0.9794)





다음은 위 결과를 시각화 한 것이다. ImageNet에 이미 개, 고양이들에 대한 데이터가 학습이 되어 있는 상태고, ImageNet에 더 적합한 데이터이기 때문에 성능이 매우 낮게 나온 2개 모델을 제외하면 대체적으로 0.9 이상의 높은 성능을 보인다.

## Custom CNN

• CatDog: 0.73 ~ 0.76

• Xray: 0.69 ~ 0.82

Custom CNN은 가장 단순한 구조로 이루어져 있다. Xray에서 흑백 데이터를 더 잘 뽑아내기 위해 정규화 등 조금 더 많은 과정을 거쳤기에, Xray 데이터의 정확도가 조금 더 높게 나왔다.

하지만 결과적으로 두 모델 다 간단한 구조이기 때문에 그렇게 높은 정확도를 기록하지 못했다.

## ResNet18

• CatDog 데이터: 성능 매우 좋음 (0.93 ~ 0.96)

• Xray 데이터: freeze 전략에 따라 편차 큼 (0.71 ~ 0.87)

|  |  |  |  |
| --- | --- | --- | --- |
| 전략 | CatDog | Xray | 해석 |
| full | 0.94 | 0.76 | 전이만 했을 때, Xray에서는 부족함 |
| freeze\_early | 0.96 | 0.86 | 초반 레이어만 고정하고 뒷부분 학습 → Best |
| freeze\_half | 0.96 | 0.80 | 조금 더 동결 → 성능 약간 하락 |
| unfreeze\_all | 0.95 | 0.86 | 전체 fine-tuning 했을 때 좋은 성능 |

CatDog처럼 자연 이미지에 가까운 데이터는 ResNet18이 아주 잘 작동하는 경향을 보인다. (대체적으로 0.9 이상의 성능)

다만, Xray는 이미지 형식이 기존과 다르므로 early layer만 고정하고 후반부를 fine-tune 하는 전략에서 가장 높은 정확도를 기록한 것을 볼 수 있다.

## VGG

• CatDog 성능 매우 우수 (0.97 ~ 0.98)

• Xray는 전략에 따라 편차 큼 (0.62 ~ 0.84)

|  |  |  |  |
| --- | --- | --- | --- |
| 전략 | CatDog | Xray | 해석 |
| freeze\_features | 0.97 | 0.84 | 최상 |
| freeze\_half | 0.49 | 0.62 | 성능이 매우 낮음 (문제 있음) |
| full | 0.97 | 0.82 | 괜찮음 |
| unfreeze\_all | 0.63 | 0.70 | 오히려 과적합 발생 |

Freeze\_half는 잘못된 동결로 오히려 성능이 매우 하락했다. 모든 레이어를 학습하는 전략은 오히려 많은 파라미터를 업데이트 하며 과적합이 발생했다.

이 경우, Xray는 freeze\_features 전략이 가장 안정적으로 결과가 도출되었다.

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | CatDog에 좋은 전략 | Xray에 좋은 전략 | 공통점 |
| 모델 | ResNet18, VGG16 (전이학습) | 전이학습 기반 + 부분 freeze | CustomCNN은 성능 낮음 |
| freeze 전략 | 대부분 잘 동작함 | freeze\_early, freeze\_features 우세 | 전체 fine-tune은 Xray에서 위험할 수 있음 |
| 비고 | CatDog은 쉽게 학습됨 | Xray는 도메인 특이성 있음 | VGG16은 구조가 커서 fine-tuning 어려움 |

그렇다면 데이터 증강을 한 이후는 어떨까?

CNN은 pooling을 하기 때문에 데이터가 조금 뒤틀리거나, 이동한다 해도 둔감하게 데이터의 특징을 추출할 수 있다.

따라서, 아래와 같이 기존 데이터에 약간의 변형을 주어 추가 데이터를 생성하였다.

1. catdog\_transform\_augmented = transforms.Compose([

2.     transforms.Resize(130),

3.     transforms.CenterCrop(128),

4.     transforms.RandomHorizontalFlip(p=0.5),       # 좌우 반전

5.     transforms.RandomRotation(degrees=15),        # ±15도 회전

6.     transforms.ColorJitter(brightness=0.2, contrast=0.2),

7.     transforms.ToTensor()

8. ])

9.

10. all\_images\_cat\_and\_dog\_aug = datasets.ImageFolder(

11.     "./data/PetImages", transform=catdog\_transform\_augmented

12. )

13.

14. train\_size = int(len(all\_images\_cat\_and\_dog\_aug) \* 0.8)

15. test\_size = len(all\_images\_cat\_and\_dog\_aug) - train\_size

16. train\_data\_aug, test\_data\_aug = random\_split(all\_images\_cat\_and\_dog\_aug, [train\_size, test\_size])

17.

18. catdog\_train\_loader\_aug = DataLoader(train\_data\_aug, batch\_size=64, shuffle=True)

19. catdog\_test\_loader\_aug = DataLoader(test\_data\_aug, batch\_size=64)

20.

21.

1. xray\_transform\_augmented = transforms.Compose([

2.     transforms.Resize(130),

3.     transforms.CenterCrop(128),

4.     transforms.RandomRotation(10),                          # ±10도 회전

5.     transforms.RandomHorizontalFlip(p=0.5),                 # 좌우 반전

6.     transforms.Grayscale(num\_output\_channels=3),            # 1채널 → 3채널

7.     transforms.ToTensor()

8. ])

9.

10. xray\_train\_dataset\_aug = datasets.ImageFolder("./data/xray\_data/train", transform=xray\_transform\_augmented)

11. xray\_test\_dataset\_aug  = datasets.ImageFolder("./data/xray\_data/test", transform=xray\_transform)  # 테스트는 증강 X

12.

13. xray\_train\_loader\_aug = DataLoader(xray\_train\_dataset\_aug, batch\_size=64, shuffle=True)

14. xray\_test\_loader\_aug  = DataLoader(xray\_test\_dataset\_aug, batch\_size=64)

15.

위처럼 데이터를 반전시키고 야간 흔들어 데이터를 증가시켜 추가로 학습할 수 있도록 하였다. 학습은 resnet을 사용해, 정확도를 측정할 수 있게 하였다. 기존 코드에서 VGG만 제거한 동일한 코드이므로, 학습에 대한 코드는 생략하고 결과를 살펴본다.

▶ Training CustomCNN for Xray\_Augmented...

[Epoch 2] Loss: 11.3602 | Accuracy: 0.6827

[Epoch 4] Loss: 9.5322 | Accuracy: 0.7179

[Epoch 6] Loss: 7.3825 | Accuracy: 0.7163

[Epoch 8] Loss: 7.5903 | Accuracy: 0.8237

[Epoch 10] Loss: 6.2752 | Accuracy: 0.7308

⤷ Final Accuracy: 0.7308

▶ Training ResNet18 (full) for Xray\_Augmented...

[Epoch 2] Loss: 17.9364 | Accuracy: 0.7740

[Epoch 4] Loss: 14.7170 | Accuracy: 0.7612

[Epoch 6] Loss: 14.3381 | Accuracy: 0.7756

[Epoch 8] Loss: 13.4054 | Accuracy: 0.7949

[Epoch 10] Loss: 13.2466 | Accuracy: 0.8109

⤷ Final Accuracy: 0.8109

▶ Training ResNet18 (freeze\_early) for Xray\_Augmented...

[Epoch 2] Loss: 6.4474 | Accuracy: 0.9006

[Epoch 4] Loss: 3.5078 | Accuracy: 0.8205

[Epoch 6] Loss: 3.7300 | Accuracy: 0.8349

[Epoch 8] Loss: 2.9462 | Accuracy: 0.7949

[Epoch 10] Loss: 3.7360 | Accuracy: 0.8077

⤷ Final Accuracy: 0.8077

▶ Training ResNet18 (freeze\_half) for Xray\_Augmented...

[Epoch 2] Loss: 5.6688 | Accuracy: 0.8670

[Epoch 4] Loss: 3.5682 | Accuracy: 0.8894

[Epoch 6] Loss: 2.5932 | Accuracy: 0.8029

[Epoch 8] Loss: 2.6051 | Accuracy: 0.8429

[Epoch 10] Loss: 2.5429 | Accuracy: 0.8381

⤷ Final Accuracy: 0.8381

▶ Training ResNet18 (unfreeze\_all) for Xray\_Augmented...

[Epoch 2] Loss: 6.0749 | Accuracy: 0.8157

[Epoch 4] Loss: 5.1911 | Accuracy: 0.6795

[Epoch 6] Loss: 4.5753 | Accuracy: 0.8109

[Epoch 8] Loss: 3.8648 | Accuracy: 0.8029

[Epoch 10] Loss: 2.9284 | Accuracy: 0.7628

⤷ Final Accuracy: 0.7628

✅ Best ResNet18 model saved as 'Xray\_Augmented\_best\_resnet\_model.pt' (Model: Xray\_Augmented\_resnet18\_freeze\_half, Accuracy: 0.8381)

▶ Training CustomCNN for CatDog\_Augmented...

[Epoch 2] Loss: 189.2595 | Accuracy: 0.6618

[Epoch 4] Loss: 161.8164 | Accuracy: 0.7506

[Epoch 6] Loss: 145.5833 | Accuracy: 0.8010

[Epoch 8] Loss: 135.6413 | Accuracy: 0.7970

[Epoch 10] Loss: 126.8560 | Accuracy: 0.8308

⤷ Final Accuracy: 0.8308

▶ Training ResNet18 (full) for CatDog\_Augmented...

[Epoch 2] Loss: 67.8212 | Accuracy: 0.9300

[Epoch 4] Loss: 65.2354 | Accuracy: 0.9386

[Epoch 6] Loss: 63.0016 | Accuracy: 0.9350

[Epoch 8] Loss: 64.7432 | Accuracy: 0.9368

[Epoch 10] Loss: 62.0671 | Accuracy: 0.9424

⤷ Final Accuracy: 0.9424

▶ Training ResNet18 (freeze\_early) for CatDog\_Augmented...

[Epoch 2] Loss: 34.2695 | Accuracy: 0.9694

[Epoch 4] Loss: 25.6373 | Accuracy: 0.9678

[Epoch 6] Loss: 20.4906 | Accuracy: 0.9824

[Epoch 8] Loss: 17.5286 | Accuracy: 0.9858

[Epoch 10] Loss: 14.3628 | Accuracy: 0.9812

⤷ Final Accuracy: 0.9812

▶ Training ResNet18 (freeze\_half) for CatDog\_Augmented...

[Epoch 2] Loss: 32.8261 | Accuracy: 0.9776

[Epoch 4] Loss: 21.6668 | Accuracy: 0.9640

[Epoch 6] Loss: 16.1417 | Accuracy: 0.9882

[Epoch 8] Loss: 13.1316 | Accuracy: 0.9828

[Epoch 10] Loss: 10.5344 | Accuracy: 0.9876

⤷ Final Accuracy: 0.9876

▶ Training ResNet18 (unfreeze\_all) for CatDog\_Augmented...

[Epoch 2] Loss: 42.7744 | Accuracy: 0.9620

[Epoch 4] Loss: 32.5294 | Accuracy: 0.9670

[Epoch 6] Loss: 27.6998 | Accuracy: 0.9746

[Epoch 8] Loss: 23.8047 | Accuracy: 0.9778

[Epoch 10] Loss: 21.1998 | Accuracy: 0.9710

⤷ Final Accuracy: 0.9710

✅ Best ResNet18 model saved as 'CatDog\_Augmented\_best\_resnet\_model.pt' (Model: CatDog\_Augmented\_resnet18\_freeze\_half, Accuracy: 0.9876)

기존 데이터만 사용했을 때와 비교:

X\_RAY\_DATASET

|  |  |  |  |
| --- | --- | --- | --- |
| 모델 | 증강 전 정확도 | 증강 후 정확도 | 향상 여부 |
| CustomCNN | 0.8205 | 0.7308 | ❌ 감소 |
| ResNet18 (full) | 0.7612 | 0.8109 | ✅ 증가 |
| ResNet18 (freeze\_early) | 0.8077 | 0.8077 | ➖ 동일 |
| ResNet18 (freeze\_half) | **0.8381** | **0.8381** | ➖ 동일 (Best) |
| ResNet18 (unfreeze\_all) | 0.7628 | 0.7628 | ➖ 동일 |

CAT\_DOG\_DATASET

|  |  |  |  |
| --- | --- | --- | --- |
| 모델 | 증강 전 정확도 | 증강 후 정확도 | 향상 여부 |
| CustomCNN | 0.8308 | 0.8308 | ➖ 동일 |
| ResNet18 (full) | 0.9424 | 0.9424 | ➖ 동일 |
| ResNet18 (freeze\_early) | 0.9812 | 0.9812 | ➖ 동일 |
| ResNet18 (freeze\_half) | **0.9876** | **0.9876** | ➖ 동일 (Best) |
| ResNet18 (unfreeze\_all) | 0.9710 | 0.9710 | ➖ 동일 |

Cat, dog에 대한 데이터는 많은 이미지가 학습이 된 상태이므로, 이미지 데이터를 회전하고, 대칭한다고 하여 뚜렷한 정확도 상승을 기대하기 어렵다. 다만, xray 같은 경우, 모델이 기존에 학습한 데이터가 아니고, 좌우 반전을 할 경우, 새로운 데이터로서 학습을 할 수 있다. 따라서, 기존과 다른 데이터를 학습하려 한다면, 데이터를 학습하는데에 있어 크게 악영향이 가지 않는 한, 이미지를 회전하는 등의 증감을 통하여 정확도를 올릴 수 있음을 알 수 있었다.

# 결과

동일한 데이터, 동일한 epoch 등에서 모델을 정의하고, 파인튜닝 하는 것에서 이처럼 커다란 정확도 차이가 날 수 있다는 것을 알 수 있었다. VGG가 모델이 크기 때문에 파인 튜닝에 어려움을 겪었던 것 같다. 또한 이미 모델이 학습한 것과 비슷한 데이터를 학습할 때에 모델들은 대체로 좋은 정확도를 보였다. 다만, 우리가 익숙하지 않은 모델을 파인튜닝할 때에는 비교적 성능이 떨어지는 것을 확인할 수 있었다.

앞으로 모델을 파인튜닝하게 된다면, 기존에 내가 시도하려는 모델과 비슷한 데이터를 학습한 모델이 이미 있는지를 고려하고, 해당 데이터에 맞게 모델을 파인튜닝할 때에 파인튜닝의 4가지 영역에 따라 천천히 점진적으로 해보는 것이 좋다는 것을 알게 되었다.