+Assignment #1. Principles of AI Learning

2024년 3월 20일

XXXXX (학번)

이름

학과

이메일

|  |
| --- |
| 요약 – 이미지 분류 작업을 위해 LeNet-5 구조를 기반으로 한 컨볼루션 신경망(CNN)을 구현한 것이며, 주어진 고양이와 개 이미지를 분류하는 문제에 적합하게 설계되었다. |
| Keywords: Classification |

# Introduction

이미지 분류 작업은 주어진 이미지를 정해진 카테고리로 분류하는 과정을 의미한다. 이 과제는 컴퓨터 비전에서 중요한 문제로, 예를 들어 고양이와 개 사진을 구분하거나 손글씨 숫자를 인식하는 등 다양한 응용이 있다. 최근 몇 년간 딥러닝과 인공지능 기술의 발전으로 이미지 분류 작업의 성능이 크게 향상되었으며, 이미지 분류 학습 모델을 만들 때 효율을 향상시키기 위한 방안에 대해 모색하고자 한다.

# Related Theories

## Augment

기본적인 분류를 진행하기에 앞서 Augmentation을 각각의 이미지에 적용한다면 데이터셋의 개수는 두배로 늘어나며, 이에 대한 학습 데이터의 양이 증가함에 따라 보다 학습 능률을 높일 수 있는 방안 중 하나이다. 이 때 우리는 이미지를 어떤 방식으로 Augmentation 할 지에 대한 3가지 증강 상황에 대해 부여받았으며, 차례로 행하고자 한다.

# 

# 그림 1. 원본 이미지

### Flip

반전은 중심을 지나는 한 직선을 기점으로 이미지를 뒤집을 수 있는 데, 흔히 생각할 수 있는 반전 기법은 상하 반전, 좌우 반전, 대각 반전 등이 있다. 우리는 상하 반전과 좌우 반전을 수행하고자 하며, 그림 1의 이미지를 기준으로 수행한 결과는 다음과 같다.

 

**그림 2.** 상하 및 좌우 반전 수행 결과

### Rotation

회전은 중심을 기점으로 이미지를 회전할 수 있는 데, 90도, 180도, 270도의 각도가 흔히 사용되고 있으며, 우리는 원점 대칭인 180과, 45도 회전을 수행하고자 한다. 마찬가지로 그림 1의 이미지를 기준으로 수행한 결과는 다음과 같다.

 

**그림 3.** 원점 대칭 및 45도 회전 수행 결과

# Implementations

증강 1, 2, 3의 전체적 코드는 이미지 처리 과정을 제외하고 모두 동일하게 제작하였다. 코드에서 이미지를 처리할 때, Part 2를 고려하여 Image Data Generator 라이브러리를 사용하였으며, LeNet-5를 이용하여 모델 네트워크를 구성하였다. 코드는 다음과 같다.

### Augment

import os

import shutil

from PIL import Image

from tqdm import tqdm

from tensorflow.keras import layers, models

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.preprocessing.image import ImageDataGenerator

필요한 라이브러리들을 불러오는 코드.

def augment\_image(image):

augmented\_images = []

augmented\_images.append(image)

augmented\_images.append(image.transpose(Image.FLIP\_TOP\_BOTTOM))

augmented\_images.append(image.transpose(Image.FLIP\_LEFT\_RIGHT))

augmented\_images.append(image.rotate(180))

augmented\_images.append(image.rotate(45, expand=True))

return augmented\_images

이미지 변환을 실행하는 코드. 받아온 이미지를 5번 줄부터 각각 상하 대칭, 좌우 대칭, 원점 대칭, 45도 회전하여 원본 이미지와 함께 리스트에 추가하여 리스트를 반환한다.

def save\_augmented\_images(src\_dir, dest\_dir):

if not os.path.exists(dest\_dir):

os.makedirs(dest\_dir)

images = os.listdir(src\_dir)

for img\_name in tqdm(images):

img\_path = os.path.join(src\_dir, img\_name)

img = Image.open(img\_path)

augmented\_images = augment\_image(img)

for i, aug\_img in enumerate(augmented\_images):

new\_img\_name = f"{os.path.splitext(img\_name)[0]}\_aug{i}.jpg"  
 aug\_img.save(os.path.join(dest\_dir, new\_img\_name))

이미지를 변환하여 저장하는 함수. 이미지 소스 경로와 저장할 경로를 받아 존재하지 않을 경우 저장할 경로 폴더를 생성하고, 소스 경로 내 모든 이미지들을 열어 앞서 설명한 augment\_image 함수를 각각에 이미지에 수행하여 저장 경로에 저장한다.

def copy\_files(file\_list, src\_dir, dst\_dir):  
 for file\_name in file\_list:  
 shutil.copy(os.path.join(src\_dir, file\_name), os.path.join(dst\_dir, file\_name))

파일 리스트, 소스 경로, 저장 경로를 입력받아 shutil 라이브러리를 활용하여 해당 저장 경로에 파일을 복사한다.

cat\_original\_dir = 'training\_set/cats'  
dog\_original\_dir = 'training\_set/dogs'  
  
cat\_dir = 'data/cats'  
dog\_dir = 'data/dogs'  
  
train\_dir = 'data/train'  
validation\_dir = 'data/validation'  
test\_dir = 'data/test'  
  
for dir\_path in [train\_dir, validation\_dir, test\_dir]:  
 if not os.path.exists(dir\_path):  
 os.makedirs(os.path.join(dir\_path, 'cats'))  
 os.makedirs(os.path.join(dir\_path, 'dogs'))  
  
save\_augmented\_images(cat\_original\_dir, cat\_dir)  
save\_augmented\_images(dog\_original\_dir, dog\_dir)  
  
cat\_files = os.listdir(cat\_dir)  
dog\_files = os.listdir(dog\_dir)  
  
train\_cats, test\_cats = train\_test\_split(cat\_files, test\_size=0.2, random\_state=42)  
train\_dogs, test\_dogs = train\_test\_split(dog\_files, test\_size=0.2, random\_state=42)  
  
train\_cats, val\_cats = train\_test\_split(train\_cats, test\_size=0.25, random\_state=42) # 0.25 x 0.8 = 0.2  
train\_dogs, val\_dogs = train\_test\_split(train\_dogs, test\_size=0.25, random\_state=42)

코드의 메인 부분. 제공받은 폴더를 통해 각각 원본 이미지 폴더 경로를 변수에 저장하고, 변환하여 저장해 사용할 경로를 변수로 저장하고, 학습, 검증, 테스트에 사용할 분할된 이미지들을 저장할 폴더의 경로를 각각 변수로 생성한다.

고양이와 강아지 이미지들에 대해서 위에서 정의한 함수들을 사용하여 이미지를 변환하고, 파일 리스트로 호출하여 학습, 테스트, 검증 데이터 이미지로 사용을 위해 분할한다. 이때 분할 비율은 학습 : 테스트 = 8 :2 이며, 학습 : 검증 = 75 : 25로 설정하였다.

### Model

def create\_lenet5():  
 model = models.Sequential()  
 model.add(layers.Conv2D(6, (5, 5), activation='relu', input\_shape=(32, 32, 3)))  
 model.add(layers.AveragePooling2D(pool\_size=(2, 2)))  
  
 model.add(layers.Conv2D(16, (5, 5), activation='relu'))  
 model.add(layers.AveragePooling2D(pool\_size=(2, 2)))  
  
 model.add(layers.Flatten())  
 model.add(layers.Dense(120, activation='relu'))  
 model.add(layers.Dense(84, activation='relu'))  
 model.add(layers.Dense(1, activation='sigmoid'))

LeNet-5 모델을 생성하는 함수. 시퀀셜 모델을 생성하여 LeNet-5 구조의 네트워크를 구성하기 위해 생성한 시퀀셜 모델에 레이어를 추가하여 모델을 반환한다.

copy\_files(train\_cats, cat\_dir, os.path.join(train\_dir, 'cats'))  
copy\_files(val\_cats, cat\_dir, os.path.join(validation\_dir, 'cats'))  
copy\_files(test\_cats, cat\_dir, os.path.join(test\_dir, 'cats'))  
  
copy\_files(train\_dogs, dog\_dir, os.path.join(train\_dir, 'dogs'))  
copy\_files(val\_dogs, dog\_dir, os.path.join(validation\_dir, 'dogs'))  
copy\_files(test\_dogs, dog\_dir, os.path.join(test\_dir, 'dogs'))  
  
train\_datagen = ImageDataGenerator(rescale=1. / 255,  
 rotation\_range=20,  
 width\_shift\_range=0.2,  
 height\_shift\_range=0.2,  
 shear\_range=0.2,  
 zoom\_range=0.2,  
 horizontal\_flip=True)  
  
test\_datagen = ImageDataGenerator(rescale=1. / 255)  
  
train\_generator = train\_datagen.flow\_from\_directory(  
 train\_dir,  
 target\_size=(32, 32),  
 batch\_size=32,  
 class\_mode='binary')  
  
validation\_generator = test\_datagen.flow\_from\_directory(  
 validation\_dir,  
 target\_size=(32, 32),  
 batch\_size=32,  
 class\_mode='binary')

test\_generator = test\_datagen.flow\_from\_directory(  
 test\_dir,  
 target\_size=(32, 32),  
 batch\_size=32,  
 class\_mode='binary')

model = create\_lenet5()  
  
model.compile(optimizer='adam',  
 loss='binary\_crossentropy',  
 metrics=['accuracy'])  
  
history = model.fit(  
 train\_generator,  
 steps\_per\_epoch=train\_generator.samples // train\_generator.batch\_size,  
 epochs=30,  
 validation\_data=validation\_generator,  
 validation\_steps=validation\_generator.samples // validation\_generator.batch\_size)  
  
test\_loss, test\_accuracy = model.evaluate(test\_generator, steps=test\_generator.samples // test\_generator.batch\_size)  
print(f'Test accuracy: {test\_accuracy:.4f}')  
print(f'Test loss: {test\_loss:.4f}')

변환된 이미지가 저장된 폴더들로부터 분할할 폴더에 각각 복사하고, Image Data Generator를 사용하여 학습 데이터를 전처리한다. 이미지 픽셀 값을 정규화 하고, 여러 옵션을 사용하여 학습될 이미지를 증강 변화 시킬수 있으며, 과적합을 방지한다.

또한 디렉토리에서 데이터를 로드할 때, 이미지의 크기를 모델에 맞게 변경하여 로드하고, 고양이와 강아지의 이진 분류이므로 binary 클래스 모드 옵션을 사용한다.

앞에서 정의한 함수를 통해 모델을 생성하여 컴파일하고, 생성한 제너레이터들로 학습 및 검증을 수행하고, 결과를 출력한다. 결과는 다음과 같다.

# Evaluations

### 학습 수행 및 테스트 결과 - 증강 #1

# 

# 그림 4. 증강 #1 - 테스트 정확도 및 테스트 손실

### 학습 수행 및 테스트 결과 - 증강 #2

# 

# 그림 5. 증강 2 - 테스트 정확도 및 테스트 손실

### 학습 수행 및 테스트 결과 - 증강 #3

# 

# 그림 6. 증강 3 - 테스트 정확도 및 테스트 손실

# Conclusions

반전 및 회전 등의 증강을 수행하여 데이터셋을 학습시켜 테스트한 결과, 큰 수치적 증가를 보기는 힘들었으나, 데이터의 양이 증가함에 따라 점진적으로 정확도가 증가하고 로스가 줄어드는 모습을 볼 수 있었다. LeNet-5의 구성 및 학습률, epoch 등의 하이퍼파라미터를 수정하여 수행해보았으나 해당 정확도가 가장 높은 정확도를 보였기에 현재 모델을 채택하여 수행하였다.