

✓ CNN 추가 실습_이미지분류(cat vs dog)



✓ 1.환경준비

✓ (1) 사전 준비사항

- 1) 구글 드라이브에 폴더 생성
 - 홈 밑에 **images** 폴더 생성
- 2) 다운받은 파일들을 **images**에 복사해 넣기

✓ (2) 라이브러리 로딩

```
import cv2
from google.colab.patches import cv2_imshow

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random as rd
import zipfile
import os

from sklearn.model_selection import train_test_split
from sklearn.metrics import *
from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential, load_model
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from keras.backend import clear_session
from keras.optimizers import Adam
```

- 함수 만들기

```
# 학습곡선 함수
def dl_history_plot(history):
    plt.figure(figsize=(10,6))
    plt.plot(history['loss'], label='train_err', marker = '.')
    plt.plot(history['val_loss'], label='val_err', marker = '.')

    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend()
    plt.grid()
    plt.show()
```

✓ (3) 구글드라이브 마운트

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
# 구글드라이브 홈 밑에 images 폴더를 생성해야 합니다!!!
path = '/content/drive/MyDrive/program/DeepLearning/images'
```

✓ 2.이미지 전처리

- 이미지를 전처리하는 작업은 꽤 오래 걸리는 작업입니다.
- 아래 절차와 코드가 있지만, 중간 결과물을 이용하여 진행해 보도록 하겠습니다.

✓ (1) 데이터 둘러보기

- 데이터 목록 읽어오기

```
dogs_path = path + '/dogs'
fileList = os.listdir(dogs_path)
```

- 이미지 파일 둘러보기

```
n = rd.randrange(0, len(fileList))
fileList[n]
file_r = dogs_path + '/' + fileList[n]
img = cv2.imread(file_r)
print('이미지 size : ', img.shape)
cv2.imshow(img)
```

✓ (2) 이미지 전처리

- 아래 두가지 요건에 맞춰 전처리를 진행합니다.
 - 흑백이미지
 - 150 * 150 크기
- 아래 코드는 시간이 많이 소요되므로 이미 수행한 이미지를 사용합니다.
- 실제 데이터를 가지고 테스트할 때는 전처리를 수행해보게 됩니다.

✓ 1) 파일 크기, 흑백 맞춰서 저장하기

```

# # 전처리된 이미지 폴더 생성
# new_path = path + '/cats2'
# os.makedirs(new_path, exist_ok=True)

# path_r = '/content/drive/MyDrive/images/cats'
# fileList = os.listdir(path_r)

# for idx, file in enumerate(fileList) :
#     file_r = path_r + '/' + file
#     img = cv2.imread(file_r, cv2.IMREAD_GRAYSCALE) # 읽을때, 흑백으로 변환
#     img = cv2.resize(img, (150, 150)) # 크기 맞추기
#     file_w = new_path + '/' + str(idx) + '.jpg'
#     cv2.imwrite(file_w, img) # cats2에 저장
#     print(idx)

# # 전처리된 이미지 폴더 생성
# new_path = path + '/dogs2'
# os.makedirs(new_path, exist_ok=True)

# path_r = '/content/drive/MyDrive/images/dogs'
# fileList = os.listdir(path_r)

# for idx, file in enumerate(fileList) :
#     file_r = path_r + '/' + file
#     img = cv2.imread(file_r, cv2.IMREAD_GRAYSCALE)
#     img = cv2.resize(img, (150, 150))
#     file_w = new_path + '/' + str(idx) + '.jpg'
#     cv2.imwrite(file_w, img)
#     print(idx)

```

✓ 2) 하나의 데이터셋으로 만들기

- 약 2 ~ 5분 소요

```

# # cat2
# path = '/content/drive/MyDrive/images/cats2/'
# fileList = os.listdir(path)
# cat_file = len(fileList)

# for idx, file in enumerate(fileList) :
#     file_name = path + '/' + file
#     img = cv2.imread(file_name, cv2.IMREAD_GRAYSCALE)
#     if idx == 0 :
#         x = img.reshape(1,150,150)
#     else :
#         x = np.concatenate((x, img.reshape(1,150,150)), axis=0)

# # dog2
# path = '/content/drive/MyDrive/images/dogs2/'
# fileList = os.listdir(path)
# dog_file = len(fileList)

# for idx, file in enumerate(fileList) :
#     file_name = path + '/' + file
#     img = cv2.imread(file_name, cv2.IMREAD_GRAYSCALE)
#     x = np.concatenate((x, img.reshape(1,150,150)), axis=0)

# y = np.array([0] * cat_file + [1] * dog_file)
# x.shape, y.shape

```

✓ (3) 이미지 데이터셋 업로드

- 이미지 파일이 아닌 데이터셋으로 부터 시작시 아래 코드 실행
 - 제공받은 두 파일을 구글 드라이브에 업로드한 후 아래 코드 실행
 - x_catdog6000.pkl, y_catdog6000.pkl

```
import joblib # 파이썬으로 만든 오브젝트를 저장하거나 로딩할 수 있는 라이브러리.
```

```
# 여러분의 구글 경로
```

```
path = '/content/drive/MyDrive/program/DeepLearning/images/'
```

```
# 로딩하기
```

```
x_filename = path + 'x_catdog6000.pkl'
```

```
y_filename = path + 'y_catdog6000.pkl'
```

```
x = joblib.load(x_filename)
```

```
y = joblib.load(y_filename)
```

```
x.shape, y.shape
```

```
((6000, 150, 150), (6000,))
```

- 전처리된 데이터 사진 둘러보기

```
# 아래 숫자(n)을 바꿔봅시다.
```

```
n = 100
```

```
print(x[n])
```

```
cv2_imshow(x[n])
```

```
[[186 174 168 ... 117 124 110]
 [172 167 167 ... 121 125 114]
 [170 174 175 ... 119 117 109]
 ...
 [ 90  88  92 ...  97 103  98]
 [ 95  92 101 ... 101 103 101]
 [ 91  98  96 ... 102  97  97]]
```



✓ 3.모델링

✓ (1) 데이터 전처리

- 데이터 분할

```
x_train, x_val, y_train, y_val = train_test_split(x,y, test_size = 2000)
```

```
x_train.shape, x_val.shape
```

```
((4000, 150, 150), (2000, 150, 150))
```

- **[중요!]모델 입력(input_shape)에 맞게 차원 조절**
- 기존 : (건수, 가로픽셀, 세로픽셀)
- 조정 :
 - CNN 레이어 input_shape = (가로픽셀, 세로픽셀, 컬러)
 - 그러므로 데이터셋은 (건수, 가로픽셀, 세로픽셀, 컬러)

```
x_train = x_train.reshape(-1, 150, 150, 1)
```

```
x_val= x_val.reshape(-1, 150, 150, 1)
```

```
# 생성된 데이터셋의 구조(shape)
```

```
x_train.shape, x_val.shape
```

```
((4000, 150, 150, 1), (2000, 150, 150, 1))
```

- Scaling : Min-Max

- 0-255 값으로 되어 있는 데이터를 0-1사이 값으로 변환
- x_train, x_val를 그냥 255로 나누면 됨

```
x_train = x_train / 255.
x_val = x_val / 255.
```

✓ (2) 모델링

- 아래 구조와 동일하거나 조금 변형해서 모델링을 수행해 봅시다.

```
clear_session()

model = Sequential([Conv2D(32, kernel_size=3, input_shape=(150, 150, 1), activation='relu'),
                    MaxPooling2D(pool_size=2),
                    Conv2D(64, kernel_size=3, activation='relu'),
                    MaxPooling2D(pool_size=2),
                    Conv2D(128, kernel_size=3, activation='relu'),
                    MaxPooling2D(pool_size=2),
                    Conv2D(256, kernel_size=3, activation='relu'),
                    MaxPooling2D(pool_size=2),
                    Flatten(),
                    Dense(512, activation='relu'),
                    Dense(1, activation='sigmoid')
                    ])

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	320
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 256)	295168
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 256)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 512)	6423040
dense_1 (Dense)	(None, 1)	513

```
=====
Total params: 6811393 (25.98 MB)
Trainable params: 6811393 (25.98 MB)
Non-trainable params: 0 (0.00 Byte)
=====
```

```
# GPU 버전에서 약 2 분이내 소요(GPU를 쓸 수 없다면 1시간 이상 소요ㅠㅠ)
model.compile(optimizer=Adam(0.0001), loss='binary_crossentropy')
history = model.fit(x_train, y_train, epochs = 30, batch_size = 64, validation_split=0.2).history
```

```
Epoch 2/30
50/50 [=====] - 2s 46ms/step - loss: 0.6748 - val_loss: 0.6606
Epoch 3/30
50/50 [=====] - 2s 44ms/step - loss: 0.6395 - val_loss: 0.6514
Epoch 4/30
50/50 [=====] - 2s 44ms/step - loss: 0.6000 - val_loss: 0.6047
Epoch 5/30
```

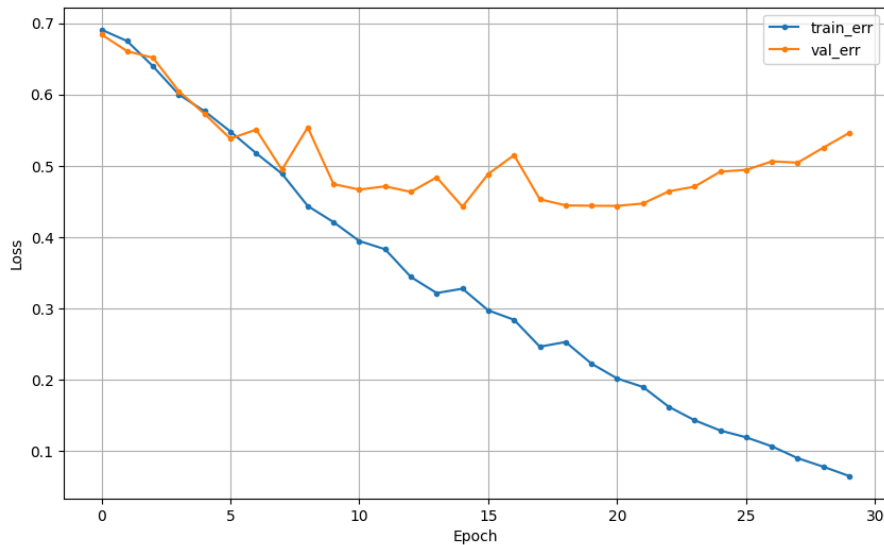
```

50/50 [=====] - 2s 44ms/step - loss: 0.3402 - val_loss: 0.3301
Epoch 7/30
50/50 [=====] - 2s 44ms/step - loss: 0.5177 - val_loss: 0.5508
Epoch 8/30
50/50 [=====] - 2s 49ms/step - loss: 0.4890 - val_loss: 0.4946
Epoch 9/30
50/50 [=====] - 2s 45ms/step - loss: 0.4438 - val_loss: 0.5539
Epoch 10/30
50/50 [=====] - 2s 47ms/step - loss: 0.4212 - val_loss: 0.4745
Epoch 11/30
50/50 [=====] - 2s 44ms/step - loss: 0.3947 - val_loss: 0.4667
Epoch 12/30
50/50 [=====] - 2s 44ms/step - loss: 0.3831 - val_loss: 0.4713
Epoch 13/30
50/50 [=====] - 2s 44ms/step - loss: 0.3443 - val_loss: 0.4635
Epoch 14/30
50/50 [=====] - 2s 49ms/step - loss: 0.3218 - val_loss: 0.4839
Epoch 15/30
50/50 [=====] - 2s 44ms/step - loss: 0.3279 - val_loss: 0.4427
Epoch 16/30
50/50 [=====] - 2s 44ms/step - loss: 0.2975 - val_loss: 0.4886
Epoch 17/30
50/50 [=====] - 2s 44ms/step - loss: 0.2843 - val_loss: 0.5150
Epoch 18/30
50/50 [=====] - 2s 47ms/step - loss: 0.2466 - val_loss: 0.4531
Epoch 19/30
50/50 [=====] - 2s 48ms/step - loss: 0.2532 - val_loss: 0.4446
Epoch 20/30
50/50 [=====] - 2s 47ms/step - loss: 0.2229 - val_loss: 0.4441
Epoch 21/30
50/50 [=====] - 2s 44ms/step - loss: 0.2021 - val_loss: 0.4439
Epoch 22/30
50/50 [=====] - 2s 44ms/step - loss: 0.1902 - val_loss: 0.4472
Epoch 23/30
50/50 [=====] - 2s 44ms/step - loss: 0.1625 - val_loss: 0.4643
Epoch 24/30
50/50 [=====] - 2s 44ms/step - loss: 0.1435 - val_loss: 0.4709
Epoch 25/30
50/50 [=====] - 2s 45ms/step - loss: 0.1291 - val_loss: 0.4919
Epoch 26/30
50/50 [=====] - 2s 50ms/step - loss: 0.1196 - val_loss: 0.4943
Epoch 27/30
50/50 [=====] - 2s 44ms/step - loss: 0.1070 - val_loss: 0.5062
Epoch 28/30
50/50 [=====] - 2s 44ms/step - loss: 0.0904 - val_loss: 0.5045
Epoch 29/30
50/50 [=====] - 2s 47ms/step - loss: 0.0783 - val_loss: 0.5256
Epoch 30/30
50/50 [=====] - 2s 44ms/step - loss: 0.0654 - val_loss: 0.5462

```

- 학습결과 그래프

```
dl_history_plot(history)
```



✓ (3) 모델 로딩

- 만약 학습시간이 너무 오래 걸린다면, 아래 코드로 이미 만든 모델을 로딩합니다.
- 로딩하는 모델은 더 많은 데이터로 학습하였습니다.

```
# model = load_model(path + 'model.h5')
```

- 예측 및 평가

```
# 예측
```

```
pred = model.predict(x_val)
pred = np.where(pred > 0.5, 1, 0)
```

```
# 평가
```

```
print(f'Accuracy : {accuracy_score(y_val, pred)}')
print('-' * 60)
print('<< Confusion Matrix >>\n')
print(confusion_matrix(y_val, pred))
print('-' * 60)
print(classification_report(y_val, pred))
```

```
63/63 [=====] - 1s 12ms/step
```

```
Accuracy : 0.8215
```

```
<< Confusion Matrix >>
```

```
[[898 113]
 [244 745]]
```

```
-----
              precision    recall  f1-score   support

     0       0.79      0.89      0.83       1011
     1       0.87      0.75      0.81       989

 accuracy          0.82       2000
  macro avg       0.83      0.82      0.82       2000
  weighted avg    0.83      0.82      0.82       2000
-----
```

✓ 4.틀린 이미지 찾아보기

- 아래 코드는 이미지 확인을 위해 실행하는 데에 사용합니다.

```

idx = (y_val != pred.reshape(2000,))
x_val_wr = x_val[idx]
y_val_wr = y_val[idx]
pred_wr = pred[idx].reshape(-1,)

x_val_wr = x_val_wr.reshape(-1,150,150)

idx = rd.sample(range(x_val_wr.shape[0]),16)
x_temp = x_val_wr[idx]
y_temp = y_val_wr[idx]
p_temp = pred_wr[idx]
class_name = ['cat', 'dog']

plt.figure(figsize=(8, 8))
for i in range(16):
    plt.subplot(4,4,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(1-x_temp[i], cmap=plt.cm.binary)
    plt.xlabel(f'act : {class_name[y_temp[i]]}, pred : {class_name[p_temp[i]]}')
plt.tight_layout()
plt.show()

```

✓ 5.인터넷에서 이미지를 찾아 테스트해봅시다.

(1) 이미지 다운받아 구글드라이브에 올리기

- 인터넷에서 고양이/강아지 이미지 몇 개를 다운 받습니다.
- 콜랩, 왼쪽 탭 중 파일 선택하고
- 이미지를 드래그 & 드롭으로 업로드


✓ (2) 이미지 전처리

```

# 아래 file_name을 여러분이 업로드한 파일 이름으로 수정합니다.
file_name = 'dog1.png'

# 업로드한 이미지 확인하기
img = cv2.imread(file_name)
print(img.shape)
cv2.imshow(img)

```


 (960, 960, 3)

```
# 이미지 전처리
# 흑백으로 다시 로딩
img = cv2.imread(file_name, cv2.IMREAD_GRAYSCALE)

# 사이즈 조정
img = cv2.resize(img, (150, 150))          # 크기 맞추기
print(img.shape)
cv2_imshow(img)
```

(150, 150)



```
class_name = ['cat', 'dog']

# 입력데이터 형식을 맞추기
test_num = img.reshape(1,150,150,1)

# 예측하기(예측된 값이 2차원이어서, 1차원 변환)
test_pred = model.predict(test_num).reshape(-1,)

# 예측결과 0,1 추출 : pred[0]
# float 타입을 int(정수)로 변환 : int(pred[0])
# class 이름으로 변환
class_name[int(test_pred[0])]

# 아래 file_name을 여러분이 업로드한 파일 이름으로 수정합니다.
file_name = 'cat1.png'

# 업로드한 이미지 확인하기
img = cv2.imread(file_name)
print(img.shape)
cv2.imshow(img)
```

(667, 1000, 3)

