<인공지능개론>

-과제 2-

소프트웨어학과
2019313065 박다영

<코드 설명>

1.

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

먼저 구글 드라이브에 이미지 데이터를 저장한 뒤, 드라이브를 mount 하여 colab에 이미지 데이터들을 불러온다.

2.

```
img_dir = './drive/MyDrive/Dog_breeds_Dataset'
categories = os.listdir(img_dir)
nb_classes =  len(categories)
print(nb_classes)
print(categories[3])
```

데이터가 잘 들어왔는지 확인해본다.

3.

<Data preprocessing>

```
from PIL import Image
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Convolution2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array,
load_img

image_w = 128
image_h = 128
pixels = image_w * image_h * 3

X = []
Y = []
for idx, cat in enumerate(categories):
    label = [0 for i in range(nb_classes)]
    label[idx] = 1
    image_dir = img_dir + "/" + cat
```

```
    files = glob.glob(image_dir+"/*.jpg")
    for i, f in enumerate(files):
        img = Image.open(f)
        img = img.convert("RGB")
        img = img.resize((image_w, image_h))
        data = np.asarray(img)
        X.append(data)
        Y.append(label)

X = np.array(X)
Y = np.array(Y)

X_train, X_test, y_train, y_test = \
    train_test_split(X, Y, test_size=0.3, shuffle=True,)
xy = (X_train, X_test, y_train, y_test)
np.save(".data.npy", xy)
print("ok,", len(Y))
X_train = X_train.astype("float") / 256
X_test  = X_test.astype("float")  / 256
```

이미지의 크기가 모두 다르기 때문에 모든 이미지를 128*128 크기로 resize 해주고 학습에 쉽게 사용할 수 있도록 Numpy 배열 형식으로 변환하여 저장해준다. 변수 X에는 실제 이미지 데이터를, 변수 Y에는 이미지가 어떤 것을 나타내는지 설명하는 레이블 데이터를 넣어준다. train_test_split 함수를 통해 데이터의 70%는 training set, 30%는 Validation Set으로 나누어 저장한다.

4.

```
print('X_train shape:', X_train.shape)
print('X_test shape', X_test.shape)
```

training set과 validation set의 shape를 프린트해본다. 70%, 30%로 잘 나뉜 것을 확인할 수 있다.

5.

```
model = Sequential()
model.add(Convolution2D(32, (3, 3), padding='same', input_shape=X_train.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Convolution2D(64, (3, 3), padding='same', data_format='channels_last'))
model.add(Activation('relu'))
model.add(Convolution2D(64, 3, 3))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))

model.compile(loss='binary_crossentropy',
    optimizer='rmsprop',
    metrics=['accuracy'])

results = model.fit(X_train, y_train, batch_size=16, epochs=30, validation_data=(X_test, y_test))

score = model.evaluate(X_test, y_test)
print('loss=', score[0])
print('accuracy=', score[1])
```

모델을 학습시키는데 keras 모델을 사용하였다.

먼저 첫 번째 convolution layer에 32개의 필터를 적용하고, 커널의 행과 열을 3으로 정해주었다. padding = “same”으로 하여 출력 이미지의 크기를 입력 이미지의 크기와 동일하게 해준다(데이터가 없는 경우 zero-padding을 통해 데이터의 손실을 막아준다). input shape는 앞에서 저장했던 X_train의 shape로 정해준다. 활성 함수로 relu 함수를 사용하고, pooling size가 (2,2)인 max pooling 방식을 사용하여 해당 윈도우에 들어가는 요소들의 값 중 최댓값을 뽑아내어 저장한다. 모델의 성능을 높이기 위해 dropout도 적용하였다.
두 번째 convolution layer는 64개의 필터를 적용하고 relu 활성함수를 사용한다.
세 번째 convolution layer 역시 64개의 필터를 적용하고 pooling과 dropout을 적용한다.
Flatten layer을 통해 다차원 배열을 1차원 배열로 펼쳐주고 dense layer로 입력과 출력을 연결해준 후, softmax 함수를 적용한다.
이렇게 CNN 모델을 구현하고 compile method를 통해 학습방식에 대한 환경설정을 해준다.

정규화기(optimizer)는 훈련과정 즉, 최적화 알고리즘 설정을 의미하며 이를 rmsprop로 설정한다. 손실함수는 모델이 최적화에 사용되는 목적함수로, 이를 binary_crossentropy로 설정하여 주고 마지막으로 훈련을 모니터링 하기 위해 사용되는 평가지표(metric)을 accuracy로 설정하여 준다.

모델을 학습하고 그 결과를 result에 저장하여 준다.

6.

```
model.summary()
```

모델을 한눈에 보여주면 다음과 같다.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 128, 128, 32)      896
_________________________________________________________________
activation (Activation)      (None, 128, 128, 32)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 64, 64, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 64)        18496
_________________________________________________________________
activation_1 (Activation)    (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 21, 21, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 10, 10, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3277312
_________________________________________________________________
activation_2 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 25)                12825
_________________________________________________________________
activation_3 (Activation)    (None, 25)                0
=================================================================
```

7.

```
import matplotlib.pyplot as plt

plt.plot(results.history['accuracy'])
plt.plot(results.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(results.history['loss'])
plt.plot(results.history['val_loss'])
plt.title('model loss')
```

```
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

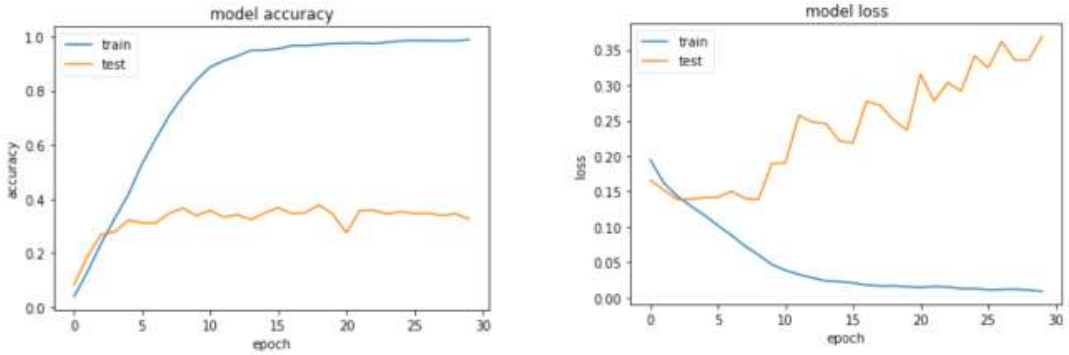
training set과 validation set에 대한 accuracy와 loss 값을 그래프로 나타낸다.

8.

```
model.save("my_model")
```

모델을 저장해준다.

<실행 결과>



epoch = 30으로 모델을 학습시켰더니 다음과 같이 training set에 대한 정확도는 높게 나왔으나, validation set에 대한 정확도가 30% 정도에서 오르지 않는 것을 볼 수 있다. overfitting 문제인 듯하여 epoch를 낮춰보았지만, 정확도가 더 낮아지는 결과가 나왔다. 따라서 데이터양이 부족하여 나타나는 문제라고 판단하여 data augmentation을 통해 데이터를 추가하는 방법을 도입해보았다.

<코드 설명>

```
datagen = ImageDataGenerator(
          rotation_range=40,
          horizontal_flip=True,
          fill_mode='nearest')

img_dir = './drive/MyDrive/Dog_breeds_Dataset'
categories = os.listdir(img_dir)
nb_classes =  len(categories)

for idx, cat in enumerate(categories):

    label = [0 for i in range(nb_classes)]
    label[idx] = 1
```

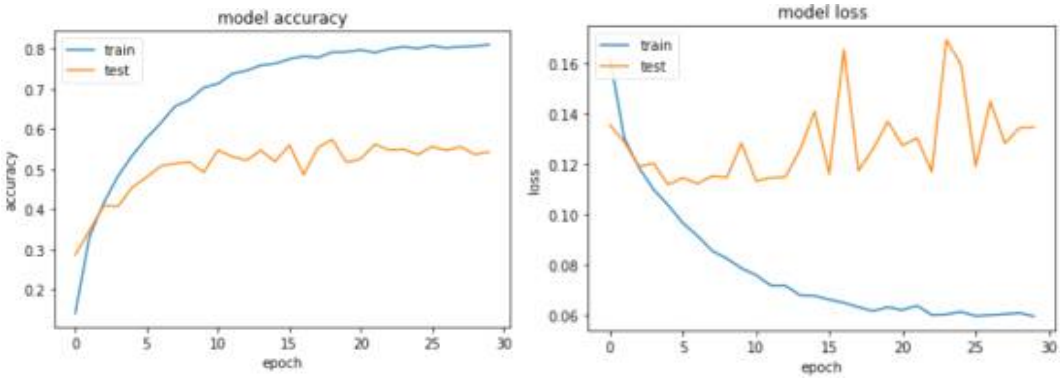
```
    image_dir = img_dir + "/" + cat
    files = glob.glob(image_dir+"/*.jpg")
    for i, f in enumerate(files):
        img = Image.open(f)
        x = img_to_array(img)
        x = x.reshape((1,) + x.shape)

        i = 0
        for batch in datagen.flow(x, batch_size=1,
                                  save_to_dir=img_dir + "/" + categories[idx],
save_prefix='aug', save_format='jpg'):
            i += 1
            if i > 3:
                break
```

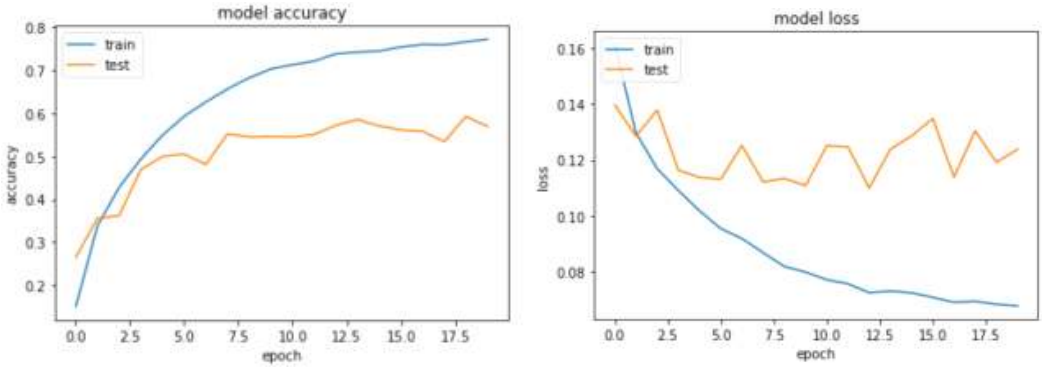
3번 코드 앞에 이렇게 데이터의 양을 늘려주는 data augmentation code를 추가해준다. 'rotation_range'를 통해 이미지를 회전시키고, 'horizontal_flip'을 True로 설정하여 50%의 확률로 이미지를 수평으로 뒤집어준다. 'fill_mode'를 통해 빈 공간이 생기면 이를 채워준다. 이미지 파일을 열어 이렇게 새로운 이미지들을 생성하고, 이를 원래 이미지가 해당하는 카테고리의 폴더에 저장한다. 이를 통해 3000개 정도였던 이미지를 훨씬 많은 양으로 늘려줄 수 있었다.

<실행 결과>

data augmentation 과정을 1번 iterate하고 epoch를 30으로 학습하였을 때 결과는 다음과 같다.



이렇게 validation data의 accuracy가 50% 정도로 올라간 것을 알 수 있다. 하지만 여전히 loss가 수렴하지 않는 것을 볼 수 있다. 따라서 augmentation 과정을 2번 iterate 하여 데이터의 양을 조금 더 늘리고 epoch = 20으로 학습을 진행하여 보았다. 그 결과는 다음과 같다.



이렇듯 validation data의 accuracy는 60% 정도로 올랐으나 loss는 여전히 수렴하지 않는 것을 알 수 있다. 이를 해결하기 위해서는 더 다양한 방법의 augmentation을 사용하여 더 많고 다양한 데이터를 확보하거나 epoch을 조절하여 overfitting 또는 underfitting을 방지하는 등의 방법을 사용해야 할 것이다.