**얼리스톱**

1. 모델 만들기

- 이전 시간 소스 복사 해서 데이터 입력

**from** **keras.preprocessing.image** **import** ImageDataGenerator

train = ImageDataGenerator().flow\_from\_directory(

'dog-vs-cat/train',

target\_size=(100, 100),

class\_mode='binary')

valid = ImageDataGenerator().flow\_from\_directory(

'dog-vs-cat/validation',

target\_size=(100, 100),

class\_mode='binary',

shuffle=False)

- 이전 시간 소스 복사해서 모델 생성

**from** **keras.layers** **import** Dense, Flatten

m1 = Sequential()

m1.add(Flatten(input\_shape=(100, 100, 3)))

m1.add(Dense(1, activation='sigmoid'))

m1.summary()

- 학습 시키기

**from** **keras.optimizers** **import** Adam, RMSprop

m1.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=Adam())

history1 = m1.fit\_generator(train, validation\_data=valid, epochs=5)

- 그래프 그려 보기(정확도, 손실)

Keras에서는 모델 학습을 위해 fit() 함수를 사용합니다. 이 때, 리턴값으로 학습 이력(History) 정보를 리턴합니다

* loss : 훈련 손실값
* acc : 훈련 정확도
* val\_loss : 검증 손실값
* val\_acc : 검증 정확도

%matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

acc = history1.history['acc']

val\_acc = history1.history['val\_acc']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'r', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

2. 얼리 스톱핑

- 앞의 소스로 model2와 history2만들기

**from** **keras.callbacks** **import** EarlyStopping

m3 = Sequential()

m3.add(Flatten(input\_shape=(100, 100, 3)))

m3.add(Dense(1, activation='sigmoid'))

m3.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=Adam())

앞의 모형의 문제는 에포크를 거듭하면서 성능이 향상되다가 다시 감소한다는 것이다. 기계학습 모형은 과도하게 학습을 하면 훈련용 데이터에 과적합되서 새로운 데이터에 대한 예측력이 떨어진다. 이를 방지하기 위해 얼리 스톱핑(early stopping)이라는 테크닉을 사용한다

history3 = m3.fit\_generator(train,

validation\_data=valid,

epochs=30,

callbacks=[

EarlyStopping(monitor = "val\_loss", patience=2)

])

Epoch 1/30

- 그래프 그려 보기

acc = history3.history['acc']

val\_acc = history3.history['val\_acc']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'r', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

CNN 합성곱 신경망

1. CNN 모형

- 데이터 불러오기

**from** **keras.preprocessing.image** **import** ImageDataGenerator

train = ImageDataGenerator(rescale=1.0/255).flow\_from\_directory(

'dog-vs-cat/train',

target\_size=(100, 100),

class\_mode='binary')

valid = ImageDataGenerator(rescale=1.0/255).flow\_from\_directory(

'dog-vs-cat/validation',

target\_size=(100, 100),

class\_mode='binary',

shuffle=False)

- 모형 만들기

Model과 레이어는 tensorflow를 붙여서 호출 해야 텐스 보드 정상 작동함

커널 32개, 커널 크기는 3, 풀링 2

**import keras**

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import \***

model1 = Sequential()

model1.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(100, 100, 3)))

model1.add(MaxPooling2D((2, 2)))

model1.add(Flatten())

model1.add(Dense(1, activation='sigmoid'))

model1.summary()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**from** telsorflow.**keras.optimizers** **import** Adam, RMSprop

model1.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=Adam())

- 실행 및 모미터링

**from** **keras.callbacks** **import** EarlyStopping, TensorBoard

history1 = model1.fit\_generator(

train, validation\_data=valid, epochs=30,

callbacks=[

EarlyStopping(monitor = "val\_loss", patience=2),

TensorBoard(log\_dir='log\_model1')

])

작업 디렉토리 명령창에 tensorboard --logdir=log\_model1이라고 입력하여 텐서보드를 실행하고 웹 브라우저 주소창에 [http://127.0.0.1:6006](http://127.0.0.1:6006/) 또는 [http://localhost:6006](http://localhost:6006/) 으로 텐서보드에 접속한다

**- 더 깊은 신경망**

model2 = Sequential()

model2.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(100, 100, 3)))

model2.add(MaxPooling2D((2, 2)))

model2.add(Conv2D(32, (3, 3), activation='relu'))

model2.add(MaxPooling2D((2, 2)))

model2.add(Flatten())

model2.add(Dense(1, activation='sigmoid'))

model2.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=Adam())

#텐스보드 모니터링 디렉토리 변경함

history2 = model2.fit\_generator(

train, validation\_data=valid, epochs=30,

callbacks=[

EarlyStopping(monitor = "val\_loss", patience=2),

TensorBoard(log\_dir='log\_model2')

])

**- 데이터 증강**

회전, 상하좌우로 이동, 기울이거나 확대 또는 뒤집는 방식으로 데이터를 증강한다.

img\_gen = ImageDataGenerator(

rescale=1./255,

rotation\_range=40, *# 40도까지 회전*

width\_shift\_range=0.2, *# 20%까지 좌우 이동*

height\_shift\_range=0.2, *# 20%까지 상하 이동*

shear\_range=0.2, *# 20%까지 기울임*

zoom\_range=0.2, *# 20%까지 확대*

horizontal\_flip=True, *# 좌우 뒤집기*

)

train\_ag = img\_gen.flow\_from\_directory(

'dog-vs-cat/train',

target\_size=(100, 100),

class\_mode='binary')

Found 2000 images belonging to 2 classes.

model3 = Sequential()

model3.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(100, 100, 3)))

model3.add(MaxPooling2D((2, 2)))

model3.add(Conv2D(32, (3, 3), activation='relu'))

model3.add(MaxPooling2D((2, 2)))

model3.add(Flatten())

model3.add(Dense(1, activation='sigmoid'))

model3.summary()

model3.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=Adam())

history3 = model3.fit\_generator(

train\_ag, validation\_data=valid, epochs=30,

callbacks=[

EarlyStopping(monitor = "val\_loss", patience=2),

TensorBoard(log\_dir='log\_model3')

])

**- 드롭아웃과 학습률 조정**

**from** **keras.layers** **import** Dropout

**from** **keras.callbacks** **import** ModelCheckpoint

model4 = Sequential()

model4.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(100, 100, 3)))

model4.add(MaxPooling2D((2, 2)))

model4.add(Conv2D(64, (3, 3), activation='relu'))

model4.add(MaxPooling2D((2, 2)))

model4.add(Conv2D(128, (3, 3), activation='relu'))

model4.add(MaxPooling2D((2, 2)))

model4.add(Flatten())

**model4.add(Dropout(0.5))**

model4.add(Dense(512, activation='relu'))

model4.add(Dense(1, activation='sigmoid'))

학습률을 0.0001로 낮춘다.

model4.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=RMSprop(lr=0.0001))

**- 체크 포인터**

가장 성능이 좋은 모델을 model4-00.hdf5와 같은 파일 명으로 저장한다.

history4 = model4.fit\_generator(

train, validation\_data=valid, epochs=30,

callbacks=[

ModelCheckpoint('model4-{epoch:02d}.hdf5', save\_best\_only=True),

TensorBoard(log\_dir='log\_model4')

])

**3. IMDB 분류**

우선 IMDB 리뷰 데이터를 받아오기 위한 datasets과 패딩을 위한 pad\_sequences를 임포트합니다.

**from** tensorflow.keras **import** datasets

**from** tensorflow.keras.preprocessing.sequence **import** pad\_sequences

최대 10,000개의 단어만을 허용하여 데이터를 받아옵니다.

vocab\_size = 10000

(X\_train, y\_train), (X\_test, y\_test) = datasets.imdb.load\_data(num\_words = vocab\_size)

X\_train을 상위 5개만 출력해봅시다.

**print**(**X\_train**[:5])

샘플은 이미 정수 인코딩까지 전처리가 된 상태입니다. 하지만 각 샘플들의 길이는 서로 다르죠? 패딩을 진행하여 모든 샘플들의 길이를 200으로 맞춥니다.

max\_len = 200

X\_train = pad\_sequences(X\_train, maxlen = max\_len)

X\_test = pad\_sequences(X\_test, maxlen = max\_len)

패딩이 되었는지 크기(shape)를 확인해봅시다.

**print**('X\_train의 크기(shape) :',X\_train.shape)

**print**('X\_test의 크기(shape) :',X\_test.shape)

훈련 데이터, 테스트 데이터 각 25,000 샘플이 전부 길이 200을 가지는 것을 확인할 수 있습니다. y\_train도 출력해봅시다.

**print**(**y\_train**[:5])

1과 0으로 구성된 것을 확인하였습니다. 이진 분류를 수행할 것이므로 레이블에는 더 이상 전처리를 할 것이 없습니다.

## 2. 1D CNN으로 IMDB 리뷰 분류하기

이제 IMDB 리뷰 분류를 위한 1D CNN 모델을 설계해봅시다. 우선 필요한 도구들을 임포트합니다. ->이미지 분류에서는 2D 사용

**from** tensorflow.keras.models **import** Sequential

**from** tensorflow.keras.layers **import** Embedding, Dropout, Conv1D, GlobalMaxPooling1D, Dense

**from** tensorflow.keras.callbacks **import** EarlyStopping, ModelCheckpoint

**from** tensorflow.keras.models **import** load\_model

모델을 설계합니다.

커널수는 256, 커널의 크기는 3을 사용합니다

embedding\_dim = 256

dropout\_ratio = 0.3

num\_filters = 256

kernel\_size = 3

hidden\_units = 128

model = Sequential()

model.add(Embedding(vocab\_size, embedding\_dim))

model.add(Dropout(dropout\_ratio))

model.add(Conv1D(num\_filters, kernel\_size, padding='valid', activation='relu'))

model.add(GlobalMaxPooling1D())

model.add(Dense(hidden\_units, activation='relu'))

model.add(Dropout(dropout\_ratio))

model.add(Dense(1, activation='sigmoid'))

1D 합성곱 연산을 수행하되, 커널수는 256, 커널의 크기는 3을 사용합니다. 그리고 GlobalMaxPooling1D를 사용하고, 두 개의 밀집층으로 은닉층과 출력층을 설계합니다.

검증 데이터의 손실(loss)이 증가하면, 과적합 징후이므로 검증 데이터 손실이 3회 증가하면 학습을 중단하는 조기 종료(EarlyStopping)를 사용합니다. 또한, ModelCheckpoint를 사용하여 검증 데이터의 정확도가 이전보다 좋아질 경우에만 모델을 저장하도록 합니다.

es = EarlyStopping(monitor = 'val\_loss', mode = 'min', verbose = 1, patience = 3)

mc = ModelCheckpoint('best\_model.h5', monitor = 'val\_acc', mode = 'max', verbose = 1, save\_best\_only = **True**)

model.compile(optimizer='adam', loss = 'binary\_crossentropy', metrics = ['acc'])

history = model.fit(X\_train, y\_train, epochs = 20, validation\_data = (X\_test, y\_test), callbacks=[es, mc])

저장된 모델을 로드하여 테스트 정확도를 확인합니다.

loaded\_model = load\_model('best\_model.h5')

**print**("\n 테스트 정확도: %.4f" % (loaded\_model.evaluate(X\_test, y\_test)[1]))

25000/25000 [==============================] - 3s 3ms/step - loss: 0.5373 - acc: 0.8873

테스트 정확도: 0.8873

테스트 데이터에서 88.73%의 정확도를 얻습니다.