**Coding과제 3**

목차

1. Fashion\_MNIST 모델 소스코드
2. 실행 결과

학번 20140269 이름 고혁훈



* 과제의 목적

뉴럴넷을 Mnist에 적용되었던 것을 다른 데이터 셋에 응용해본다.

Gray scale과 rgb scale을 둘 다 해본다.

Validation을 추가해본다. batch와 mini-batch training을 둘 다 해본다.

* 배운 점

첫 번째로, 뉴럴넷을 다른 gray scale 이미지인 fashion mnist data에 적용했을 때도 준수한 성능이 나온다. 그 모델이 잘 동작하는지 파악하려면, 우리는 unseen data에 대해 테스트를 해보아야 한다. 그래서 실제로 validation을 추가해서 모델을 학습해 보았다. 그러나 overfitting 관점에서 고려해볼 때 epoch에 늘어날 수록 training data를 잘 맞추지만 test data는 일정 수준이상 올라가지 않고 loss는 오히려 증가함을 알 수 있었다. 이는 모델을 학습할 때, training 성능이 좋아진다고 만사형통을 의미하는 것이 아님을 의미한다. 즉, model 학습에서 학습과정을 추적하는 것이 매우 중요함을 깨달았다.

두 번째로 데이터를 전체 넣고 돌리는 것과 mini-batch로 돌리는 것을 둘 다 실험해 보았는데, 전체에 비해 mini-batch의 학습 수렴 속도가 현저히 낮은 것을 알 수 있었다. 전체를 넣었을 때는 20epoch만에 train,test 성능이 벨런스를 이루었지만, mini-batch는 100 epoch가 지나도 train이 증가하고 test가 감소되는 구간을 찾지 못했다. 이는 mini-batch에 비해 데이터셋이 클수록 우리는 큰 에포크를 설정해야 한다는 것을 암시한다.

세 번째로, 컬러이미지는 채널이 3개라서 input\_shape이 매우 컸다. 그래서 메모리 오버플로우가 너무 많이 나서 채널을 극단적으로 줄여야 하는 이슈가 있었다. 그래도 모델을 돌려보니 컬러 이미지 또한 잘 작동하는 것을 알 수 있다. 그러나 메모리를 효율적으로 사용할 수 있는 모델 구조 개발이 필요할 것 같다. 너무 크면 학습자체를 돌릴 수가 없기 때문이다.

* 소스코드 – fashion - batch

1) 임포트

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, Softmax

import tensorflow\_datasets as tfds

import numpy as np

import tensorflow as tf

import pandas as pd

2) 데이터 load

# fashion data load

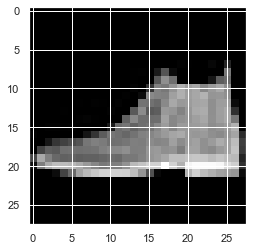
data = tf.keras.datasets.fashion\_mnist

(xt,yt),(xtt,ytt) = data.load\_data()

# 각 인덱스가 나타내는 라벨

labels=['Tshirt','Trouser','Pullover','Dress','coat','sandal','shirt','sneaker','bag','ankle book']

plt.imshow(xtt[0],cmap='gray') # index = 9



3) 데이터 preprocess

# 데이터를 normalize를 한다.

xt,xtt = xt/255, xtt/255

num\_class = len(np.unique(yt)) # layer 마지막 node를 구한다.

in\_shape = xt[0].reshape(-1).shape[0] # input shape을 구한다.

4) 모델생성

# layer는 되도록 많이 쌓아서 해볼 것이다. 그래서 차원에 맞게 model -

# layer를 자동생성해주는 코드를 만들었습니다.

layers = []

layers.append(Flatten(input\_shape=(28,28)))

temp = in\_shape

while(1):

temp = temp // (2\*\*2)

if temp < num\_class \* 2:

break

layer = tf.keras.layers.Dense(temp,activation='relu')

layers.append(layer)

layers.append(tf.keras.layers.Dense(num\_class,activation='softmax'))

# model loss와 optimizer, metric 장착

model = tf.keras.models.Sequential(layers)

model.compile(optimizer='adam',loss='SparseCategoricalCrossentropy',metrics=["accuracy"])

* 실행결과

1. 모델 summary

Model: "sequential"

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

Layer (type) Output Shape Param #

=================================================================

flatten (Flatten) (None, 784) 0

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

dense (Dense) (None, 196) 153860

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

dense\_1 (Dense) (None, 49) 9653

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

dense\_2 (Dense) (None, 10) 500

=================================================================

Total params: 164,013

Trainable params: 164,013

Non-trainable params: 0

1. epoch결과 일부

history= model.fit(xt,yt,epochs=100,validation\_data=(xtt, ytt))

Epoch 96/100

1875/1875 [==============================] - 2s 1ms/step - loss: 0.0505 - accuracy: 0.9808 - val\_loss: 0.8708 - val\_accuracy: 0.8913

Epoch 97/100

1875/1875 [==============================] - 2s 1ms/step - loss: 0.0547 - accuracy: 0.9800 - val\_loss: 0.8856 - val\_accuracy: 0.8952

Epoch 98/100

1875/1875 [==============================] - 2s 1ms/step - loss: 0.0504 - accuracy: 0.9812 - val\_loss: 0.8836 - val\_accuracy: 0.8913

Epoch 99/100

1875/1875 [==============================] - 2s 1ms/step - loss: 0.0506 - accuracy: 0.9810 - val\_loss: 0.9138 - val\_accuracy: 0.8905

Epoch 100/100

1875/1875 [==============================] - 2s 1ms/step - loss: 0.0489 - accuracy: 0.9814 - val\_loss: 0.8336 - val\_accuracy: 0.8934

1126/1126 [==============================] - 1s 1ms/step - loss: 2.8147 - accuracy: 0.9465

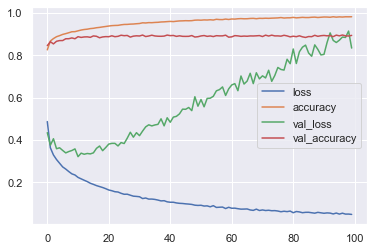
Epoch 68/100

1126/1126 [==============================] - 1s 1ms/step - loss: 2.6684 - accuracy: 0.9496

Epoch 69/100

1126/1126 [==============================] - 1s 1ms/step - loss: 2.6002 - accuracy: 0.9491

1. fit 결과 도표(x축은 epoch)



1. 예측결과

pred = np.argmax(model.predict(xtt[0:1]))

pred,ytt[0]

(9, 9)

- 소스코드 – mini-batch

history= model.fit(xt,yt,epochs=100,batch\_size=32,validation\_data=(xtt, ytt))



1875/1875 [==============================] - 2s 1ms/step - loss: 0.1276 - accuracy: 0.9531 - val\_loss: 0.3845 - val\_accuracy: 0.8867

Epoch 100/100

1875/1875 [==============================] - 2s 1ms/step - loss: 0.1258 - accuracy: 0.9537 - val\_loss: 0.3981 - val\_accuracy: 0.8864

* 소스코드 cycle\_gan – apple,orange

1) 임포트

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, Softmax

import tensorflow\_datasets as tfds

import numpy as np

import tensorflow as tf

import pandas as pd

2) 데이터 load

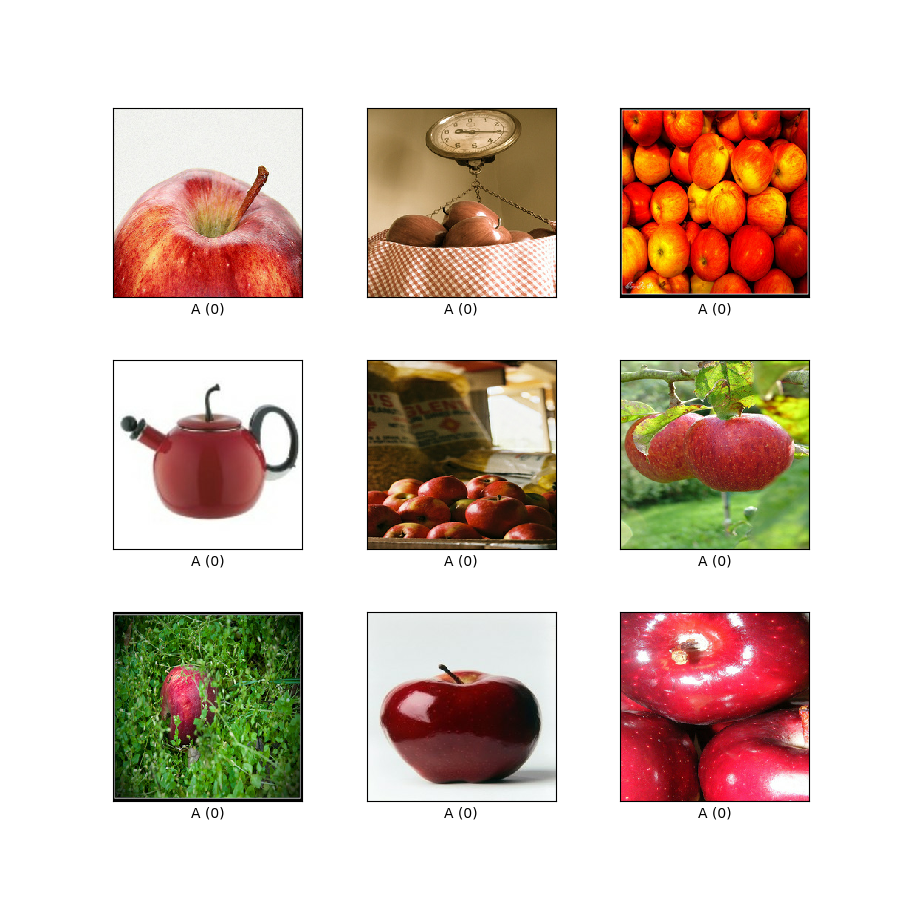
# cyclegan data load

dataset, metadata = tfds.load('cycle\_gan/apple2orange',

                              with\_info=True, as\_supervised=True)

a, b = dataset['trainA'], dataset['trainB']

ta, tb = dataset['testA'], dataset['testB']



3) 데이터 preprocess

# 불러온 데이터를 합쳐서 학습용을 tf.dataset으로 변환

# 변환할 때 normalize 적용

x = []

y = []

tx = []

ty = []

# apple ==0, orange ==1

for i in a:

    x.append(i[0] / 255)

    y.append(0)

for i in b:

    x.append(i[0]/ 255)

    y.append(1)

for i in ta:

    tx.append(i[0]/ 255)

    ty.append(0)

for i in tb:

    tx.append(i[0]/ 255)

    ty.append(1)

sample = next(iter(a))[0]

sample.shape

TensorShape([256, 256, 3])

x = tf.stack(x)

y = tf.stack(y)

tx = tf.stack(tx)

ty = tf.stack(ty)

4) 모델생성

# 모델을 초기화한다.

layers = []

layers.append(Flatten(input\_shape=(256,256,3)))

# 2048이상을 하면 메모리 오버플로우 발생

layers.append(tf.keras.layers.Dense(1024,activation='relu'))

layers.append(tf.keras.layers.Dense(512,activation='relu'))

layers.append(tf.keras.layers.Dense(128,activation='relu'))

# 클래스가 orange, apple이라 sigmoid 적용

layers.append(tf.keras.layers.Dense(1,activation='sigmoid'))

model = tf.keras.models.Sequential(layers)

#모델 컴파일

model.compile(optimizer='adam', loss='SparseCategoricalCrossentropy',metrics=["accuracy"])

* 실행결과

1. 모델 summary

Model: "sequential\_1"

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

Layer (type) Output Shape Param #

=================================================================

flatten\_1 (Flatten) (None, 196608) 0

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

dense\_2 (Dense) (None, 1024) 201327616

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

dense\_3 (Dense) (None, 512) 524800

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

dense\_4 (Dense) (None, 128) 65664

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

dense\_5 (Dense) (None, 2) 258

=================================================================

Total params: 201,918,338

Trainable params: 201,918,338

Non-trainable params: 0

1. epoch결과 일부

history= model.fit(x,y,epochs=100,validation\_data=(tx, ty))

Epoch 85/100

63/63 [==============================] - 83s 1s/step - loss: 0.0331 - accuracy: 0.9896 - val\_loss: 0.7885 - val\_accuracy: 0.8288

Epoch 86/100

63/63 [==============================] - 82s 1s/step - loss: 0.0351 - accuracy: 0.9871 - val\_loss: 0.7664 - val\_accuracy: 0.8366

Epoch 87/100

63/63 [==============================] - 84s 1s/step - loss: 0.0206 - accuracy: 0.9940 - val\_loss: 0.8474 - val\_accuracy: 0.8307

Epoch 88/100

63/63 [==============================] - 81s 1s/step - loss: 0.0227 - accuracy: 0.9930 - val\_loss: 0.7739 - val\_accuracy: 0.8385

텍스트이(가) 표시된 사진

자동 생성된 설명