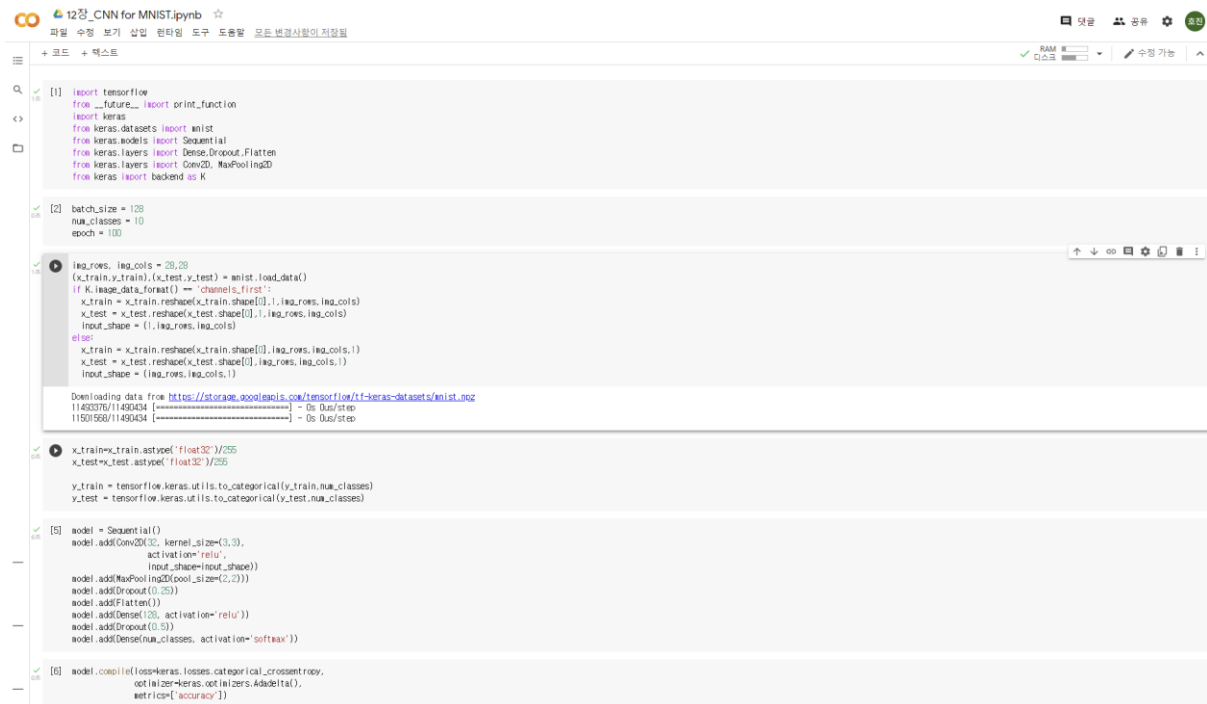


6주차 실습과제

2016314786 김호진

[12장 실습 – CNN for MNIST]



```
[1] import tensorflow
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

[2] batch_size = 128
num_classes = 10
epoch = 100

[3] img_rows, img_cols = 28, 28
(x_train, y_train), (x_test, y_test) = mnist.load_data()
if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist_noz
11493376/11490434 [=====] - 0s 0us/step
11501568/11490434 [=====] - 0s 0us/step

[4] x_train=x_train.astype('float32')/255
x_test=x_test.astype('float32')/255

y_train = tensorflow.keras.utils.to_categorical(y_train,num_classes)
y_test = tensorflow.keras.utils.to_categorical(y_test,num_classes)

[5] model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
activation='relu',
input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

[6] model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adadelta(),
metrics=['accuracy'])
```

```
[7] model.fit(x_train,y_train,
batch_size=batch_size,
epochs = epoch,
verbose=1)

score=model.evaluate(x_test,y_test,verbose=0)
print("Test loss : ",score[0])
print("Test acc : ",score[1])

469/469 [-----] - 2s 46s/step - loss: 0.5306 - accuracy: 0.8391
Epoch 73/100
469/469 [-----] - 2s 46s/step - loss: 0.5311 - accuracy: 0.8448
Epoch 74/100
469/469 [-----] - 2s 46s/step - loss: 0.5253 - accuracy: 0.8438
Epoch 75/100
469/469 [-----] - 2s 46s/step - loss: 0.5201 - accuracy: 0.8448
Epoch 76/100
469/469 [-----] - 2s 46s/step - loss: 0.5132 - accuracy: 0.8481
Epoch 77/100
469/469 [-----] - 2s 46s/step - loss: 0.5123 - accuracy: 0.8478
Epoch 78/100
469/469 [-----] - 2s 46s/step - loss: 0.5130 - accuracy: 0.8469
Epoch 79/100
469/469 [-----] - 2s 46s/step - loss: 0.5057 - accuracy: 0.8522
Epoch 80/100
469/469 [-----] - 2s 46s/step - loss: 0.5016 - accuracy: 0.8518
Epoch 81/100
469/469 [-----] - 2s 46s/step - loss: 0.5049 - accuracy: 0.8499
Epoch 82/100
469/469 [-----] - 2s 46s/step - loss: 0.4991 - accuracy: 0.8520
Epoch 83/100
469/469 [-----] - 2s 46s/step - loss: 0.4988 - accuracy: 0.8512
Epoch 84/100
469/469 [-----] - 2s 46s/step - loss: 0.4950 - accuracy: 0.8551
Epoch 85/100
469/469 [-----] - 2s 46s/step - loss: 0.4968 - accuracy: 0.8531
Epoch 86/100
469/469 [-----] - 2s 46s/step - loss: 0.4930 - accuracy: 0.8542
Epoch 87/100
469/469 [-----] - 2s 46s/step - loss: 0.4852 - accuracy: 0.8540
Epoch 88/100
469/469 [-----] - 2s 46s/step - loss: 0.4857 - accuracy: 0.8564
Epoch 89/100
469/469 [-----] - 2s 46s/step - loss: 0.4748 - accuracy: 0.8583
Epoch 90/100
469/469 [-----] - 2s 46s/step - loss: 0.4785 - accuracy: 0.8569
Epoch 91/100
469/469 [-----] - 2s 46s/step - loss: 0.4786 - accuracy: 0.8591
Epoch 92/100
469/469 [-----] - 2s 46s/step - loss: 0.4697 - accuracy: 0.8611
Epoch 93/100
469/469 [-----] - 2s 46s/step - loss: 0.4624 - accuracy: 0.8555
Epoch 94/100
469/469 [-----] - 2s 46s/step - loss: 0.4739 - accuracy: 0.8555
Epoch 95/100
469/469 [-----] - 2s 46s/step - loss: 0.4742 - accuracy: 0.8594
Epoch 96/100
469/469 [-----] - 2s 46s/step - loss: 0.4643 - accuracy: 0.8632
Epoch 97/100
469/469 [-----] - 2s 46s/step - loss: 0.4712 - accuracy: 0.8608
Epoch 98/100
469/469 [-----] - 2s 46s/step - loss: 0.4629 - accuracy: 0.8630
Epoch 99/100
469/469 [-----] - 2s 46s/step - loss: 0.4659 - accuracy: 0.8612
Epoch 100/100
469/469 [-----] - 2s 46s/step - loss: 0.4677 - accuracy: 0.8594
Test loss : 0.31788003070904
Test acc : 0.9154003237598

[8] model2 = Sequential()
model2.add(Conv2D(32, kernel_size=(3,3),
activation='relu',
input_shape=input_shape))
model2.add(Conv2D(32,(3,3),activation='relu'))
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Dropout(0.25))
model2.add(Flatten())
model2.add(Dense(128, activation='relu'))
model2.add(Dropout(0.5))
model2.add(Dense(num_classes, activation='softmax'))

[9] model2.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adadelta(),
metrics=['accuracy'])

model2.fit(x_train,y_train,
batch_size=batch_size,
epochs = epoch,
verbose=1)

score=model2.evaluate(x_test,y_test,verbose=0)
print("Test loss : ",score[0])
print("Test acc : ",score[1])

469/469 [-----] - 3s 66s/step - loss: 0.4457 - accuracy: 0.8649
Epoch 73/100
469/469 [-----] - 3s 66s/step - loss: 0.4449 - accuracy: 0.8618
Epoch 74/100
469/469 [-----] - 3s 66s/step - loss: 0.4587 - accuracy: 0.8601
Epoch 75/100
469/469 [-----] - 3s 66s/step - loss: 0.4407 - accuracy: 0.8657
Epoch 76/100
469/469 [-----] - 3s 66s/step - loss: 0.4325 - accuracy: 0.8694
Epoch 77/100
469/469 [-----] - 3s 66s/step - loss: 0.4445 - accuracy: 0.8651
Epoch 78/100
469/469 [-----] - 3s 66s/step - loss: 0.4320 - accuracy: 0.8682
Epoch 79/100
469/469 [-----] - 3s 66s/step - loss: 0.4323 - accuracy: 0.8693
Epoch 80/100
469/469 [-----] - 3s 66s/step - loss: 0.4344 - accuracy: 0.8697
Epoch 81/100
469/469 [-----] - 3s 66s/step - loss: 0.4222 - accuracy: 0.8740
Epoch 82/100
469/469 [-----] - 3s 66s/step - loss: 0.4259 - accuracy: 0.8728
Epoch 83/100
469/469 [-----] - 3s 66s/step - loss: 0.4241 - accuracy: 0.8722
Epoch 84/100
469/469 [-----] - 3s 66s/step - loss: 0.4252 - accuracy: 0.8715
Epoch 85/100
469/469 [-----] - 3s 66s/step - loss: 0.4211 - accuracy: 0.8742
Epoch 86/100
469/469 [-----] - 3s 66s/step - loss: 0.4212 - accuracy: 0.8739
Epoch 87/100
469/469 [-----] - 3s 66s/step - loss: 0.4181 - accuracy: 0.8739
Epoch 88/100
469/469 [-----] - 3s 66s/step - loss: 0.4153 - accuracy: 0.8754
Epoch 89/100
469/469 [-----] - 3s 66s/step - loss: 0.4089 - accuracy: 0.8747
Epoch 90/100
469/469 [-----] - 3s 66s/step - loss: 0.4094 - accuracy: 0.8781
Epoch 91/100
469/469 [-----] - 3s 66s/step - loss: 0.4087 - accuracy: 0.8738
Epoch 92/100
469/469 [-----] - 3s 66s/step - loss: 0.4051 - accuracy: 0.8772
Epoch 93/100
469/469 [-----] - 3s 66s/step - loss: 0.4123 - accuracy: 0.8731
Epoch 94/100
469/469 [-----] - 3s 66s/step - loss: 0.3995 - accuracy: 0.8789
Epoch 95/100
469/469 [-----] - 3s 66s/step - loss: 0.4039 - accuracy: 0.8763
Epoch 96/100
469/469 [-----] - 3s 66s/step - loss: 0.3969 - accuracy: 0.8782
Epoch 97/100
469/469 [-----] - 3s 66s/step - loss: 0.4039 - accuracy: 0.8781
Epoch 98/100
469/469 [-----] - 3s 66s/step - loss: 0.3951 - accuracy: 0.8788
Epoch 99/100
469/469 [-----] - 3s 66s/step - loss: 0.3953 - accuracy: 0.8833
Epoch 100/100
469/469 [-----] - 3s 66s/step - loss: 0.3944 - accuracy: 0.8818
Test loss : 0.2391183078299032
Test acc : 0.930199807357788
```

[13장 실습 - RNN]

파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

+ 코드 + 텍스트

RAM 디스크

수정 가능

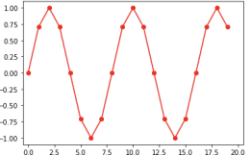
코딩

실행

공유

설정

```
[1] import numpy as np
from matplotlib import pyplot as plt
s = np.sin(2 * np.pi * 0.125 * np.arange(20))
plt.plot(s, 'ro-')
plt.xlim(-0.5, 20.5)
plt.ylim(-1.1, 1.1)
plt.show()
```



```
[2] from scipy.linalg import toeplitz
S = np.fliplr(toeplitz(np.r_[s[-1], np.zeros(s.shape[0] - 2)], s[::-1]))
S[:5, :3]
```

```
array([[ 0.00000000e+00,  7.07106781e-01,  1.00000000e+00],
       [ 7.07106781e-01,  1.00000000e+00,  7.07106781e-01],
       [ 1.00000000e+00,  7.07106781e-01,  1.22464680e-16],
       [ 7.07106781e-01,  1.22464680e-16, -7.07106781e-01],
       [ 1.22464680e-16, -7.07106781e-01, -1.00000000e+00]])
```

```
[3] X_train = S[:, :, :3][:, :, np.newaxis]
Y_train = S[:, :, 3]
X_train.shape, Y_train.shape
```

```
((18, 3, 1), (18,))
```

```
from keras.models import Sequential
from keras.layers import SimpleRNN, Dense

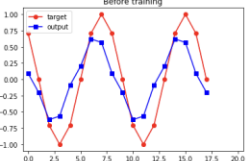
np.random.seed(0)
model = Sequential()
model.add(SimpleRNN(10, input_shape=(3,)))
model.add(Dense(1, activation='linear'))
model.compile(loss='mse', optimizer='sgd')
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 10)	120
dense (Dense)	(None, 1)	11

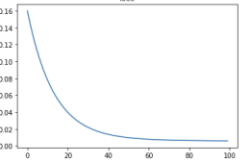
Total params: 131
Trainable params: 131
Non-trainable params: 0

```
plt.plot(Y_train, 'ro-', label='target')
plt.plot(model.predict(X_train[:, :, :]), 'bs-', label='output')
plt.xlim(-0.5, 20.5)
plt.ylim(-1.1, 1.1)
plt.legend()
plt.title('Before training')
plt.show()
```

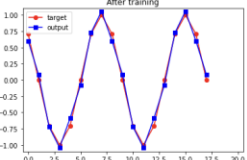


```
[6] history = model.fit(X_train, Y_train, epochs=100, verbose=0)
```

```
plt.plot(history.history['loss'])
plt.title('loss')
plt.show()
```



```
plt.plot(Y_train, 'ro-', label='target')
plt.plot(model.predict(X_train[:, :, :]), 'bs-', label='output')
plt.xlim(-0.5, 20.5)
plt.ylim(-1.1, 1.1)
plt.legend()
plt.title('After training')
plt.show()
```



[13장 실습 - GRU + LSTM]

13강_GRU+LSTM.ipynb

파일 수정 보기 삽입 런타임 도구 도움말 모든 보기사항이 적절함

+ 코드 + 텍스트

RAM 디스크

수정 가능

[1]

```
import numpy as np
from matplotlib import pyplot as plt
s = np.sin(2 * np.pi * 0.125 * np.arange(20))
from scipy.linalg import toeplitz
S = np.fliplr(toeplitz(np.r_[s[-1], np.zeros(s.shape[0] - 2)], s[1:-1]))

X_train = S[1:-1, :3]
Y_train = S[1:-1, 3]
```

[2]

```
from keras.models import Sequential
from keras.layers import LSTM, Dense

np.random.seed(0)
model = Sequential()
model.add(LSTM(10, input_shape=(3,)))
model.add(Dense(1, activation='linear'))
model.compile(loss='mse', optimizer='sgd')
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 10)	480
dense (Dense)	(None, 1)	11

Total params: 491
Trainable params: 491
Non-trainable params: 0

[3]

```
history = model.fit(X_train, Y_train, epochs = 100, verbose=0)

plt.plot(history.history['loss'])
plt.title('loss')
plt.show()
```

loss

[4]

```
plt.plot(Y_train, 'ro-', label='target')
plt.plot(model.predict(X_train[:, :3]), 'bo-', label='output')
plt.xlim(-0.5, 20.5)
plt.ylim(-1.1, 1.1)
plt.legend()
plt.title('After training')
plt.show()
```

After training

[5]

```
history = model.fit(X_train, Y_train, epochs = 1000, verbose=0)
```

[6]

```
plt.plot(Y_train, 'ro-', label='target')
plt.plot(model.predict(X_train[:, :3]), 'bo-', label='output')
plt.xlim(-0.5, 20.5)
plt.ylim(-1.1, 1.1)
plt.legend()
plt.title('After training')
plt.show()
```

After training

[7]

```
history = model.fit(X_train, Y_train, epochs = 2000, verbose=0)
```

[8]

```
plt.plot(Y_train, 'ro-', label='target')
plt.plot(model.predict(X_train[:, :3]), 'bo-', label='output')
plt.xlim(-0.5, 20.5)
plt.ylim(-1.1, 1.1)
plt.legend()
plt.title('After training')
plt.show()
```

After training

1

from keras.models import Sequential
from keras.layers import GRU, Dense

np.random.seed(0)
model = Sequential()
model.add(GRU(10, input_shape=(3,1)))
model.add(Dense(1, activation='linear'))
model.compile(loss='mse', optimizer='sgd')
model.summary()

Model: "sequential_1"

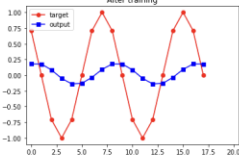
Layer (type)	Output Shape	Param #
gru (GRU)	(None, 10)	390
dense_1 (Dense)	(None, 1)	11

Total params: 400
Trainable params: 400
Non-trainable params: 0

[10] history = model.fit(X_train,Y_train, epochs = 100, verbose=0)

11

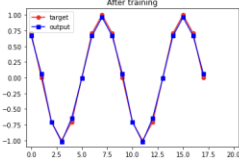
plt.plot(Y_train, 'ro-', label='target')
plt.plot(model.predict(X_train[:, :]), 'bs-', label='output')
plt.xlim(-0.5,20.5)
plt.ylim(-1,1.1)
plt.legend()
plt.title('After training')
plt.show()

After training


[12] history = model.fit(X_train,Y_train, epochs = 1000, verbose=0)

13

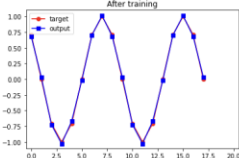
plt.plot(Y_train, 'ro-', label='target')
plt.plot(model.predict(X_train[:, :]), 'bs-', label='output')
plt.xlim(-0.5,20.5)
plt.ylim(-1,1.1)
plt.legend()
plt.title('After training')
plt.show()

After training


[14] history = model.fit(X_train,Y_train, epochs = 2000, verbose=0)

15

plt.plot(Y_train, 'ro-', label='target')
plt.plot(model.predict(X_train[:, :]), 'bs-', label='output')
plt.xlim(-0.5,20.5)
plt.ylim(-1,1.1)
plt.legend()
plt.title('After training')
plt.show()

After training


[13장 실습 – RNN for Reuter dataset]

13장 RNN for Reuter dataset.ipynb

파일 수정 보기 삽입 런타임 도구 도움말 코드 보기가 없음이 저장됨

+ 코드 + 텍스트

RAM 8.0GB
CPU 100%

수정 가능

1

from tensorflow.keras.datasets import reuters

import matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
(X_train, Y_train), (X_test, Y_test) = reuters.load_data(num_words=1000, test_split=0.2)

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/reuters.npz>
213636/210940 [=====] - 0s 0us/step
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/reuters.py:143: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. This behavior will change so that this code will raise an exception instead. See https://www.tensorflow.org/api_guides/python/ndarray for more details.
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/reuters.py:144: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. This behavior will change so that this code will raise an exception instead. See https://www.tensorflow.org/api_guides/python/ndarray for more details.
x_train, y_train = np.array(xs[1:100]), np.array(labels[1:100])
x_test, y_test = np.array(xs[100:]), np.array(labels[100:])

2

print('Train data : {}'.format(len(X_train)))
print('Test data : {}'.format(len(X_test)))
num_classes = max(Y_train) + 1
print('class : {}'.format(num_classes))

Train data : 6982
Test data : 2246
class : 46

3

word_index = reuters.get_word_index()

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/reuters_word_index.npz
557056/550378 [=====] - 0s 0us/step

4

index_word={}
for key,value in word_index.items():
 index_word[value]=key

from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, LSTM, Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

5

max_len = 100
X_train = pad_sequences(X_train, maxlen=max_len)
X_test = pad_sequences(X_test, maxlen=max_len)

Y_train = to_categorical(Y_train)
Y_test = to_categorical(Y_test)

6

model = Sequential()
model.add(Embedding(1000,120))
model.add(LSTM(120))
model.add(Dense(45, activation='softmax'))
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=1)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', save_best_only=True)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])

7

history=model.fit(X_train, Y_train, batch_size=128, epochs=30, callbacks=[es,mc], validation_split=0.2)

Epoch 1/30
5/57 [=====] - 9s 49ms/step - loss: 2.6798 - acc: 0.3428 - val_loss: 2.4099 - val_acc: 0.3450
Epoch 2/30
5/57 [=====] - 2s 39ms/step - loss: 2.3994 - acc: 0.3784 - val_loss: 2.0910 - val_acc: 0.4863
Epoch 3/30
5/57 [=====] - 2s 35ms/step - loss: 2.0279 - acc: 0.4061 - val_loss: 1.9791 - val_acc: 0.4964
Epoch 4/30
5/57 [=====] - 2s 35ms/step - loss: 1.9162 - acc: 0.5241 - val_loss: 2.0862 - val_acc: 0.4574
Epoch 5/30
5/57 [=====] - 2s 35ms/step - loss: 1.8494 - acc: 0.5260 - val_loss: 1.8143 - val_acc: 0.5570
Epoch 6/30
5/57 [=====] - 2s 35ms/step - loss: 1.7005 - acc: 0.5676 - val_loss: 1.7081 - val_acc: 0.5968
Epoch 7/30
5/57 [=====] - 2s 35ms/step - loss: 1.6100 - acc: 0.5852 - val_loss: 1.6076 - val_acc: 0.5838
Epoch 8/30
5/57 [=====] - 2s 34ms/step - loss: 1.5892 - acc: 0.5954 - val_loss: 1.5969 - val_acc: 0.5954
Epoch 9/30
5/57 [=====] - 2s 34ms/step - loss: 1.4960 - acc: 0.6209 - val_loss: 1.5882 - val_acc: 0.6110
Epoch 10/30
5/57 [=====] - 2s 34ms/step - loss: 1.4230 - acc: 0.6383 - val_loss: 1.5104 - val_acc: 0.6255
Epoch 11/30
5/57 [=====] - 2s 34ms/step - loss: 1.4316 - acc: 0.6395 - val_loss: 1.4568 - val_acc: 0.6333
Epoch 12/30
5/57 [=====] - 2s 34ms/step - loss: 1.3238 - acc: 0.6647 - val_loss: 1.4169 - val_acc: 0.6522
Epoch 13/30
5/57 [=====] - 2s 34ms/step - loss: 1.2569 - acc: 0.6814 - val_loss: 1.3750 - val_acc: 0.6561
Epoch 14/30
5/57 [=====] - 2s 33ms/step - loss: 1.2012 - acc: 0.6938 - val_loss: 1.3521 - val_acc: 0.6617
Epoch 15/30
5/57 [=====] - 2s 33ms/step - loss: 1.1538 - acc: 0.7061 - val_loss: 1.3477 - val_acc: 0.6717
Epoch 16/30
5/57 [=====] - 2s 33ms/step - loss: 1.1198 - acc: 0.7141 - val_loss: 1.3175 - val_acc: 0.6739
Epoch 17/30
5/57 [=====] - 2s 33ms/step - loss: 1.0800 - acc: 0.7205 - val_loss: 1.3064 - val_acc: 0.6828
Epoch 18/30
5/57 [=====] - 2s 33ms/step - loss: 1.0287 - acc: 0.7374 - val_loss: 1.2885 - val_acc: 0.6817
Epoch 19/30
5/57 [=====] - 2s 33ms/step - loss: 1.1130 - acc: 0.7098 - val_loss: 1.2864 - val_acc: 0.6828
Epoch 20/30
5/57 [=====] - 2s 33ms/step - loss: 0.9826 - acc: 0.7449 - val_loss: 1.2635 - val_acc: 0.6900
Epoch 21/30
5/57 [=====] - 2s 33ms/step - loss: 0.9284 - acc: 0.7628 - val_loss: 1.2944 - val_acc: 0.6900
Epoch 22/30
5/57 [=====] - 2s 33ms/step - loss: 0.8943 - acc: 0.7704 - val_loss: 1.2873 - val_acc: 0.7001
Epoch 23/30
5/57 [=====] - 2s 33ms/step - loss: 0.8650 - acc: 0.7772 - val_loss: 1.2950 - val_acc: 0.6889
Epoch 24/30
5/57 [=====] - 2s 33ms/step - loss: 0.8432 - acc: 0.7832 - val_loss: 1.2811 - val_acc: 0.6978
Epoch 25/30
5/57 [=====] - 2s 33ms/step - loss: 0.7876 - acc: 0.7997 - val_loss: 1.2772 - val_acc: 0.7017
Epoch 26/30
5/57 [=====] - 2s 33ms/step - loss: 0.7673 - acc: 0.8017 - val_loss: 1.3642 - val_acc: 0.6917
Epoch 27/30
5/57 [=====] - 2s 33ms/step - loss: 0.7417 - acc: 0.8118 - val_loss: 1.2964 - val_acc: 0.7084
Epoch 28/30
5/57 [=====] - 2s 33ms/step - loss: 0.6982 - acc: 0.8217 - val_loss: 1.3017 - val_acc: 0.7056
Epoch 00028: early stopping

8

loaded = load_model('best_model.h5')
print('테스트 정확도 : %.4f' % (loaded.evaluate(X_test, Y_test)[1]))

71/71 [=====] - 1s 11ms/step - loss: 1.3247 - acc: 0.6941
테스트 정확도 : 0.6941

9

epochs = range(1, len(history.history['acc']) + 1)
plt.plot(epochs, history.history['loss'])
plt.plot(epochs, history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

10

model loss