2018년 1학기 컴퓨터언어학 기말 프로젝트

한국어 뉴스 기사 분류

언어학과 노하경

Environment: Amazon Web Services EC2 Deep Learning AMI (Ubuntu) Version 10.0 (r4.8xlarge instance)

In [1]:
```
# Import dependencies

import sys
from os import listdir
import time
import re
import numpy as np
import codecs
import collections
import hanja
from konlpy.tag import Twitter
from konlpy.tag import Komoran
from konlpy.tag import Kkma
import keras
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, Activation, Flatten
from keras.layers import Conv1D, GlobalAveragePooling1D, MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing.sequence import pad_sequences
from keras.models import model_from_json
import csv
import matplotlib.pyplot as plt

# fix random seed for reproducibility
np.random.seed(5)
np.set_printoptions(suppress=True)
```

```
Using TensorFlow backend.
```

1. Konlpy 태그 모듈 선택

Kkma

오류 예시: ['남비난', '비난', '미정상회담', '정상', '회담', '맥스', '맥스선더', '태영호', '영호', '17', '11', '11일', '2주간', '주간']

특징: 중복 처리되는 단어들 많음, 숫자와 날짜 남아있음 (twitter, komoran은 제거함)

Hannanum

오류 예시: ['【서울=뉴시스】김성진', '22일']

특징: 일부 특수문자 처리 안됨, 숫자와 날짜 남아있음 (twitter, komoran은 제거함)

Twitter

예시: ['연일', '남비', '한미정상회담', '경고', '메시지', '발신', '맥스', '선더', '영호', '불만', '표시', '최종', '조율', '의제', '반발', '북중', '관계', '개선', '시진핑', '의중', '반영', '서울', '뉴시스', '김성진', '기자', '북한', '이틀', '대남', '대미', '비난', '의도', '대해', '관심', '북한', '물론', '대미', '관계', '조금', '무게', '모양새', '한미', '정상회담', '비난', '수위', '한국', '미국', '각각', '불만', '풀이', '북한', '지난', '한미', '공군', '연합', '훈련', '맥스', '선더', '자신', '최고', '존엄', '영호', '주영', '북한', '대사관', '공사', '대해', '불만', '출하', '남북', '고위', '회담', '무기한', '연기', '통보', '김계관', '외무성', '부상', '담화', '통해', '포기', '보상', '검증', '비핵화', '미국', '리선', '조국', '평화통일', '위원회', '위원장', '조선', '중앙', '통신', '통해', '남조선', '당국', '괴이', '논리', '조선반도', '평화', '화해', '흐름', '가로막', '장애물', '하나', '북침', '전쟁', '연습', '합리화', '비방', '중상', '지속', '철면피', '파렴치', '극치', '비난', '위원장', '북남', '고위', '회담', '중지', '엄중', '사태', '남조선', '정권', '다시', '마주', '라며', '차후', '북남관계', '방향', '전적', '남조선', '당국', '행동', '리선', '위원장', '발언', '표면', '판문점', '선언', '정신', '부합', '즉각', '회담', '우리', '정부', '성명', '대한', '반발', '전날', '맥스', '선더', '훈련', '공사', '대해', '경고', '메시지', '발신', '방치', '정부', '대한', '유감', '불만', '표시', '전문가', '북한', '일련', '행보', '최종', '비핵화', '담판', '벌이', '미국', '방점', '홍민', '통일', '연구원', '북한', '연구실', '북한', '최종', '미국', '라며', '미국', '조절', '의제', '방식', '문제', '제기', '실장', '자신', '통제', '이야기', '최대한', '폼페이', '조율', '이야기', '표현', '라며', '자신', '선의', '비핵화', '취지', '왜곡', '때문', '정도', '정리', '입장', '다음주', '개최', '한미', '정상회담', '목표', '비난', '수위', '가능성', '제기', '자칫', '한미', '구도', '시도', '분석', '지난', '북중', '정상회담', '이후', '북중', '관계', '중국', '의중', '가능성', '제기', '지난', '김정은', '북한', '국무위원', '중국', '외교부', '평양', '면담', '중국', '다롄', '시진핑', '중국', '국가주석', '정상회담', '지난', '주석', '도널드', '트럼프', '미국', '대통령', '전화통화', '미국', '북한', '합리', '안보', '우려', '바란', '기도', '중국', '북한', '모두', '견제', '주한', '미군', '군사', '동맹', '미국', '핵우산', '제공', '염두', '발언', '트럼프', '대통령', '백악관', '옌스', '스톨', '베르크', '북대서양', '조약', '기구', '사무', '총장', '만난', '자리', '주석', '김정은', '영향', '가능성', '매우', '면서', '무슨', '일이', '기도', '신범철', '아산', '정책', '연구원', '안보', '통일', '센터', '북중', '관계', '정세', '변화', '따라서', '북한', '입장', '약간', '비핵화', '협상', '미국', '강도', '북한', '나름', '목소리', '위해', '남북관계', '역시', '자신', '불만', '다만', '북한', '협상', '위해', '행보', '풀이', '한편', '경색', '국면', '남북', '대화', '북한', '지난', '주간', '계획', '맥스', '선더', '훈련', '명분', '남북', '고위', '회담', '만큼', '한미', '정상회담', '맥스', '선더', '훈련', '종료', '다시', '재개', '관측', '이후', '바로', '북미', '정상회담', '때문', '판문점', '선언', '개최', '합의', '남북', '장성', '회담', '개최', '전망']

특징: 속도는 빠르나, komoran에 비해 분석되는 어휘 수가 적음

Komoran

예시: ['연일', '비난', '한미정상회담', '경고', '메시지', '발신', '맥스', '선더', '영호', '불만', '표시', '의제', '반발', '북중', '관계', '개선', '시진핑', '의중', '반영', '서울', '뉴시스', '김성진', '기자', '북한', '이틀', '대남', '대미', '비난', '의도', '관심', '북한', '대미', '관계', '무게', '모양새', '한미 정상회담', '비난', '수위', '한국', '미국', '불만', '표출', '풀이', '북한', '미 공군', '연합', '훈련', '맥스', '선더', '자신', '최고', '존엄', '영호', '주영', '북한', '대사관', '공사', '불만', '표출', '남북', '고위급', '회담', '무기', '연기', '통보', '김계관', '외무성', '부상', '담화', '포기', '보상', '완전', '검증', '비핵화', '주장', '미국', '규탄', '조국평화통일위원회', '위원장', '조선중앙통신', '남조선', '당국', '논리', '조선반도', '평화', '화해', '흐름', '장애물', '제거', '전쟁', '연습', '합리', '비방', '중상', '지속', '철면피', '파렴치', '극치', '비난', '위원장', '북남', '고위급', '회담', '중지', '사태', '해결', '남조선', '정권', '차후', '북남', '관계', '방향', '전적', '남조선', '당국', '행동', '여하', '위원장', '발언', '표면', '판문점', '선언', '정신', '부합', '회담', '정부', '성명', '반발', '전날', '맥스', '선더', '훈련', '공사', '경고', '메시지', '발신', '방치', '정부', '유감', '불만', '표시', '전문가', '북한', '일련', '행보', '결국', '최종', '비핵화', '담판', '미국', '방점', '분석', '통일연구원', '북한', '연구실장', '북한', '최종', '미국', '미국', '조절', '의제', '방식', '문제', '제기', '분석', '실장', '자신', '통제', '이야기', '최대한', '폼페이', '오와', '조율', '이야기', '불편', '표현', '자신', '선의', '비핵화', '취지', '왜곡', '불편', '때문', '정도', '정리', '입장', '다음', '개최', '한미 정상회담', '목표', '비난', '수위', '제기', '한미', '구도', '시도', '분석', '북중', '정상회담', '이후', '북중', '관계', '개선', '중국', '의중', '반영', '제기', '김정은', '북한', '국무', '위원장', '중국', '외교부장', '평양', '면담', '중국', '다롄', '중국 국가주석', '정상회담', '주석', '도널드 트럼프', '미국 대통령', '전화통', '미국', '북한', '합리', '안보', '우려', '고려', '중국', '북한', '양측', '견제', '주한미군', '군사', '동맹', '미국', '핵우산', '제공', '염두', '발언', '트럼프', '대통령', '백악관', '옌스', '스톨', '베르크', '북대서양조약기구', '사무총장', '자리', '주석', '김정은', '영향', '신범철', '아산정책연구원', '안보', '통일', '센터', '북중', '관계', '정세', '변화', '북한', '입장', '비핵화', '협상', '미국', '강도', '북한', '나름', '목소리', '이야기', '남북관계', '자신', '불만', '표현', '북한', '협상', '전개', '행보', '풀이', '한편', '경색', '국면', '남북', '대화', '북한', '주간', '계획', '맥스', '선더', '훈련', '명분', '남북', '고위급', '회담', '연기', '만큼', '한미 정상회담', '맥스', '선더', '훈련', '종료', '재개', '관측', '이후', '북미', '정상회담', '때문', '판문점', '선언', '개최', '합의', '남북', '장성', '회담', '개최', '전망']

특징: 4개 모듈 중에서는 정확도가 높다고 판단. '조국평화통일위원회'와 같은 복합 단어를 개별 토큰으로 분석. 이 프로젝트에서는 기사별 특징을 보다 효율적으로 표현할 수 있을 것으로 판단하여 최종적으로 komoran 태그를 사용하였습니다.

In [2]:
```
# Create konlpy tags
twitter = Twitter()
komoran = Komoran()
kkma = Kkma()
```

2. Functions

전처리 프로세스 및 모델 특징

1. 특수문자는 '…'과 '·'만 제거. 나머지는 konlpy에서 처리.
2. 괄호 안에 있는 정보는 부가적인 내용으로 판단되어 제거. (부가설명, 이름, 나이, 한자 병기 등)
3. 2번 처리 후 남아있는 한자는 한글로 번역
4. konlpy 모듈(komoran)로 형태소 분석 및 명사 추출. - 명사를 기준으로 학습
5. 중복되는 어휘, 1음절 토큰 제거
6. 폴더 안에 있는 모든 기사들을 토큰화 하고, 최종적인 vocab 사전으로 저장. (단어 : 인덱스)
7. 2회 미만 출현 단어들은 무시.
8. 전처리 중에 분석된 기사들은 (재분석하는 과정을 없애기 위해) 별도의 사전 객체로 저장. (파일 인덱스[예: 0001] : 토큰화된 기사)
9. 어휘 사전(vocab), 토큰화된 기사들 사전, 총 어휘 수 반환
10. 160: 40 으로 트레이닝, 테스트 데이터 생성.
11. 자료를 불러올 때 편하게 파일 인덱스를 별도의 리스트로 정리: 트레이닝: [000~159], 테스트:[160~199]
12. Zero padding 적용. 가장 긴 기사의 길이로 통일.
13. CNN 레이어 입력을 위해 3차원으로 자료 변환.
14. 임베딩은 사용하지 않음.

In [3]:
```
#=====================================================FUNCTIONS=============================================================

def one_hot_encoding(x, output_size):
    encoded_data = np.zeros((len(x), output_size))
    for i in range(len(x)):
        encoded_data[i][x[i]] = 1
    return encoded_data

# Generate labels set for training and test data
# labels_count: how many categories?
# set_size: how many items are allocated for training or test data set
def generate_label_set(labels_count, set_size, one_hot):
    label_set = list()
    for i in range(labels_count):
        for x in range (set_size):
            label_set.append(i)
    if one_hot == True:
        one_hot_data = one_hot_encoding(label_set, labels_count)
        return one_hot_data
    else:
        return np.array(label_set) # No one-hot-encoding

# Preprocess raw korean article text file. Input- file name, konlpy package name, minimum syllables, remove dupl
def kr_text_preprocess(text, packageName, minSyllables, shrink):
    raw_file = codecs.open(text, "r", encoding='utf-8', errors='ignore').read()
    remove_spChar = re.sub(r'[…·]', ' ', raw_file) # Remove special characters
    remove_par = re.sub(r'\([^)]*\)', "", remove_spChar) # Remove contents inside parenthesis
```

```
# Creating np ndarray data (indexes[1~159]&[160~199], categories count(8), dictionary with tokenized articles, m
def create_np_dataset(index_list, categories, articles_dict, words_dic, max_vocab_count, max_length):
    python_list = []
    for category in range(categories):
        for i in index_list:
            data_index = str(category) + '{:03d}'.format(i)
            encoded_text = article_encoder(articles_dict[data_index], words_dic, max_vocab_count)
            if len(encoded_text) != max_length:
                difference = max_length - len(encoded_text)
                padding = np.zeros(difference).tolist() # Padding with zero, post
                padded_text = encoded_text + padding
                python_list.append(padded_text)
            else:
                python_list.append(encoded_text)

    output_data = np.array(python_list)
    return output_data

def create_dataset_nonEncoded(index_list, categories, articles_dict, max_length):
    python_list = []
    for category in range(categories):
        for i in index_list:
            data_index = str(category) + '{:03d}'.format(i)
            article_length = len(articles_dict[data_index])
            if article_length != max_length:
                difference = max_length - article_length
                padding = np.zeros(difference).tolist() # Padding with zero, post
                padded_text = articles_dict[data_index] + padding
                python_list.append(padded_text)
            else:
                python_list.append(articles_dict[data_index])
    return np.array(python_list)


def create_prediction_data(article, dictionary, max_count, max_length):
    encoded_text = article_encoder_prediction(article, dictionary, max_count)
    padded_text = list()
    if len(encoded_text) != max_length:
        difference = max_length - len(encoded_text)
        padding = np.zeros(difference).tolist() # Padding with zero, post
        padded_text = encoded_text + padding
    output_data = np.array(padded_text)
    return output_data

# Prediction data
def article_encoder_prediction(article, vocab_dictionary, max_count):
    article_encoded = list()
    for word in range(1, len(article)):
        word_index = vocab_dictionary.get(article[word])
        if word_index == None:
            continue
        else:
            normalize = word_index / max_count
            article_encoded.append(normalize)

    return article_encoded

# Plot Loss History
class PlotLosses(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.i = 0
        self.x = []
        self.losses = []
        self.val_losses = []

        self.fig = plt.figure()

        self.logs = []

    def on_epoch_end(self, epoch, logs={}):

        self.logs.append(logs)
        self.x.append(self.i)
        self.losses.append(logs.get('loss'))
        self.val_losses.append(logs.get('val_loss'))
        self.i += 1

        clear_output(wait=True)
        plt.plot(self.x, self.losses, label="loss")
        plt.plot(self.x, self.val_losses, label="val_loss")
        plt.legend()
        plt.show();
```

```
# Show image
def display_image_in_actual_size(im_path):

    dpi = 80
    im_data = plt.imread(im_path)
    height, width, depth = im_data.shape

    # What size does the figure need to be in inches to fit the image?
    figsize = width / float(dpi), height / float(dpi)

    # Create a figure of the right size with one axes that takes up the full figure
    fig = plt.figure(figsize=figsize)
    ax = fig.add_axes([0, 0, 1, 1])

    # Hide spines, ticks, etc.
    ax.axis('off')

    # Display the image.
    ax.imshow(im_data, cmap='gray')

    plt.show()

# Import CSV as dict
def import_csv(filename):
    vocab_dict = {}
    with open(filename, mode='r') as infile:
        reader = csv.reader(infile)
        for row in reader:
            vocab_dict[row[0]] = int(row[1])

    return vocab_dict

# Export CSV
def csv_exporter(filename, dictionary):
    name = filename + '.csv'
    with open(name, 'w') as f:
        w = csv.writer(f)
        w.writerows(dictionary.items())

article_categories = {0:'정치', 1:'경제', 2:'사회', 3:'생활/문화', 4:'세계', 5:'기술/IT', 6:'연예', 7:'스포츠'}
```

3. 변수들

In [4]:
```
#=====================================================VARIABLES=====================================================
module_tag = 'komoran'
categories_count = 8
directory = 'newsData'
num_of_articles_in_cat = 200
split = 0.8 # Split rate
ignore_less = 2 # Ignore words with occurence less than n
encode_one_hot = True
export_csv_file = True
modelfile = "model.png"
```

4. 뉴스 기사 분석 및 전처리

- CSV 파일로 vocab 저장

In [5]:
```
#=====================================================PREPROCESSING=====================================================
# Pre-process text and generate vocab

# Timer
start_time = time.clock()
print("Pre-processing articles and creating vocabulary set. It can take a while depending on number of files. \n
articles, words_dictioinary, word_count = create_dictionary_and_backup_articles(directory, categories_count, ign

# Export vocab as csv
if export_csv_file == True:
    csv_exporter('vocab', words_dictioinary)

print ('Dictionary creation finished!')
if export_csv_file == True:
    print ('CSV file saved!')
print ('Total words in dictionary: ', word_count)
print ("Processing time: ", "{0:.2f}".format(time.clock() - start_time), " seconds")
```

```
Pre-processing articles and creating vocabulary set. It can take a while depending on number of files.
Please be patient...
```

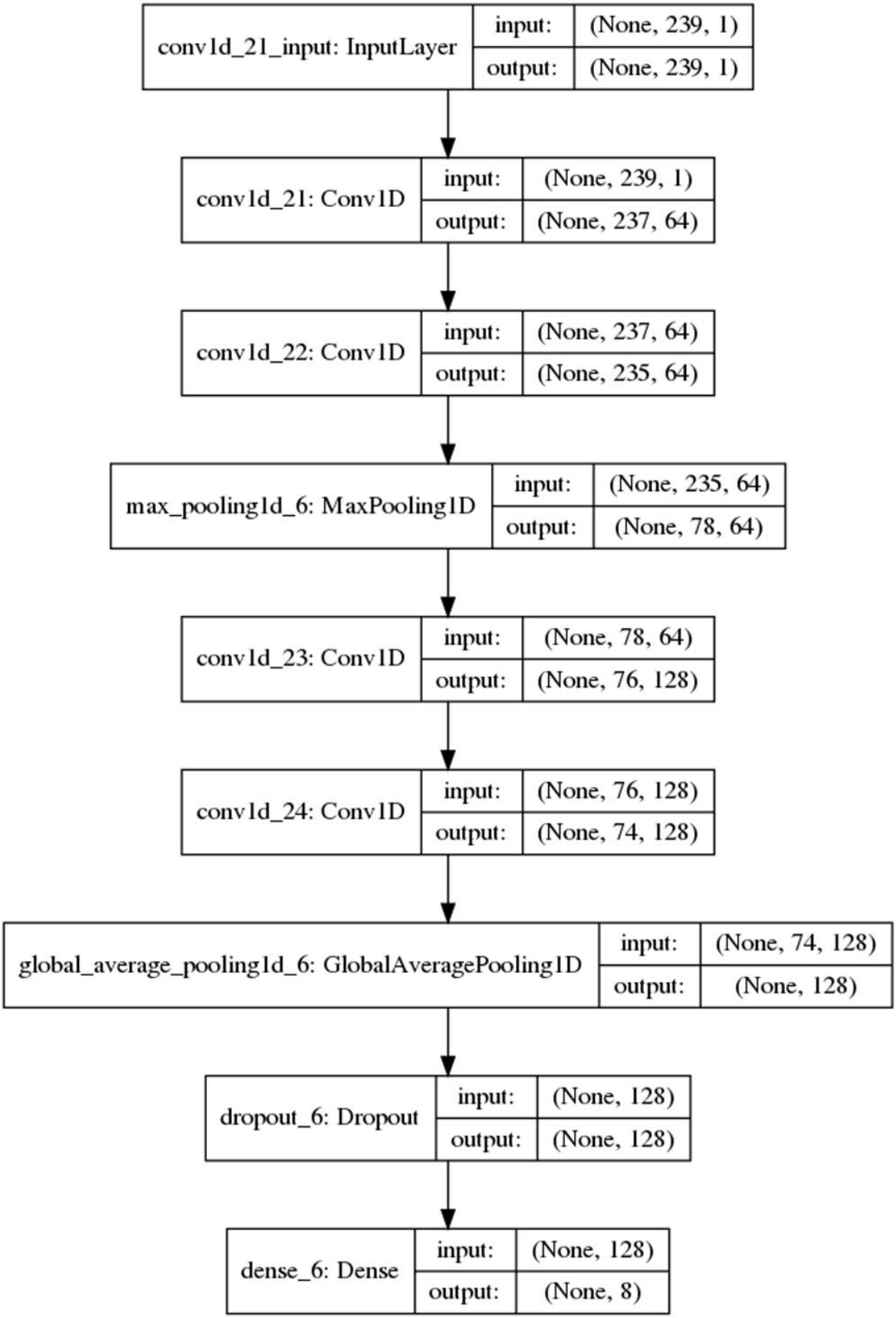
```
Processing articles in category 정치 complete!
Processing articles in category 경제 complete!
Processing articles in category 사회 complete!
Processing articles in category 생활/문화 complete!
Processing articles in category 세계 complete!
Processing articles in category 기술/IT complete!
Processing articles in category 연예 complete!
Processing articles in category 스포츠 complete!
Dictionary creation finished!
CSV file saved!
Total words in dictionary:  11496
Processing time:  41.20  seconds
```

5. Training 및 Test 데이터 생성

In [6]:
```
#====================================================TRAINING====================================================

# Generate t_train and t_test
t_train = generate_label_set(categories_count, int(num_of_articles_in_cat * split), encode_one_hot)
t_test = generate_label_set(categories_count, int(num_of_articles_in_cat - num_of_articles_in_cat * split), enco


# Create training and test set data (File name indexes for training and test data)
training_indexes = np.arange(num_of_articles_in_cat * split).astype(np.int32).tolist()
test_indexes = np.arange(num_of_articles_in_cat * split, num_of_articles_in_cat).astype(np.int32).tolist()


# Get the length of the longest article
article_lengths = []
for k, v in articles.items():
    length = len(v)
    article_lengths.append(length)
max_length = max(article_lengths)


# Create train data
#x_train_nonEncoded = create_dataset_nonEncoded(training_indexes, categories_count, articles, max_length)
#x_test_nonEncoded = create_dataset_nonEncoded(test_indexes, categories_count, articles, max_length)

x_train = create_np_dataset(training_indexes, categories_count, articles, words_dictioinary, word_count, max_len
x_test = create_np_dataset(test_indexes, categories_count, articles, words_dictioinary, word_count, max_length)

x_train_3d = np.expand_dims(x_train, axis=2)
x_test_3d = np.expand_dims(x_test, axis=2)

print('x_train, x_test shape:', x_train_3d.shape, x_test_3d.shape)
print('t_train, t_test shape:', t_train.shape, t_test.shape)
print('\n')
print('Sample x_train data: \n', x_train[0])

# Neural Net
print('\nLets start training network\n')
```

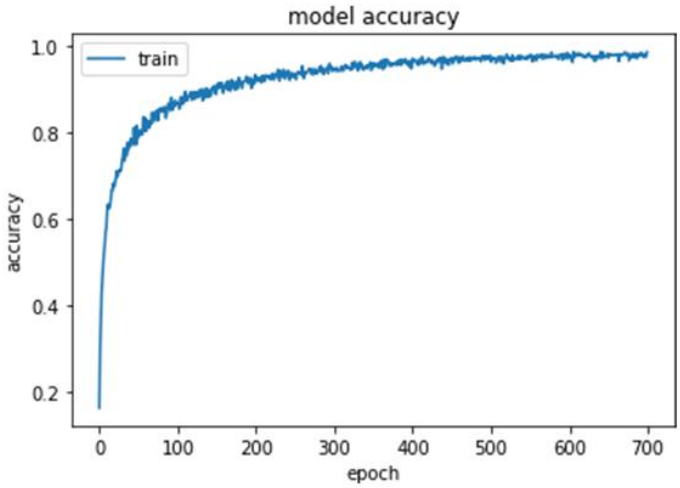
```
x_train, x_test shape: (1280, 239, 1) (320, 239, 1)
t_train, t_test shape: (1280, 8) (320, 8)


Sample x_train data: 
 [0.12152053 0.06724078 0.0401009  0.12160752 0.1216945  0.02757481
 0.12178149 0.0164405  0.00608907 0.12186848 0.12195546 0.12204245
 0.10325331 0.00643702 0.01096033 0.10342728 0.00869868 0.12030271
 0.06149965 0.00243563 0.08063674 0.12212944 0.1203897  0.00695894
 0.12221642 0.12230341 0.1223904  0.1019485  0.02444328 0.01122129
 0.02026792 0.12247738 0.04358038 0.06263048 0.00704593 0.02870564
 0.12256437 0.12265136 0.02061587 0.12273834 0.03262004 0.12282533
 0.00408838 0.12291232 0.00121781 0.01731037 0.1229993  0.12308629
 0.12317328 0.02174669 0.0138309  0.07054628 0.09977383 0.12326026
 0.04105776 0.12117258 0.12334725 0.12343424 0.12352122 0.05367084
 0.02244259 0.01417884 0.10125261 0.00226166 0.00513222 0.12360821
 0.05810717 0.0079158  0.1236952  0.12378219 0.12386917 0.12395616
 0.04279749 0.03175017 0.00556715 0.10829854 0.12404315 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.        ]

Lets start training network
```

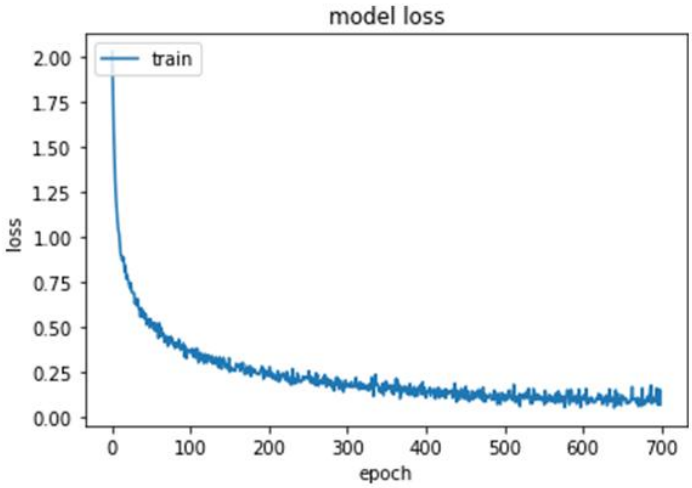
6. 모델 생성 및 학습


conv1d_21_input: InputLayer
input: (None, 239, 1)
output: (None, 239, 1)
conv1d_21: Conv1D
input: (None, 239, 1)
output: (None, 237, 64)
conv1d_22: Conv1D
input: (None, 237, 64)
output: (None, 235, 64)
max_pooling1d_6: MaxPooling1D
input: (None, 235, 64)
output: (None, 78, 64)
conv1d_23: Conv1D
input: (None, 78, 64)
output: (None, 76, 128)
conv1d_24: Conv1D
input: (None, 76, 128)
output: (None, 74, 128)
global_average_pooling1d_6: GlobalAveragePooling1D
input: (None, 74, 128)
output: (None, 128)
dropout_6: Dropout
input: (None, 128)
output: (None, 128)
dense_6: Dense
input: (None, 128)
output: (None, 8)

In [17]:
```
epochs_total = 700

# Model
model = Sequential()
model.add(Conv1D(64, 3, activation='relu', input_shape=(max_length, 1)))
model.add(Conv1D(64, 3, activation='relu'))
model.add(MaxPooling1D(3))
model.add(Conv1D(128, 3, activation='relu'))
model.add(Conv1D(128, 3, activation='relu'))
model.add(GlobalAveragePooling1D())
model.add(Dropout(0.5))
model.add(Dense(categories_count, activation='softmax'))

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

print(model.summary())
#plot_model(model, to_file=modelfile, show_shapes=True, show_layer_names=True)

history = model.fit(x_train_3d, t_train, batch_size=16, epochs=epochs_total)

# Show model graph
display_image_in_actual_size(modelfile)

# 학습과정 살펴보기
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
#plt.plot(history.history['val_acc'])
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(history.history['loss'])
#plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

score = model.evaluate(x_test_3d, t_test, batch_size=16)
print("Accuracy: %.2f%%" % (score[1]*100))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_41 (Conv1D)           (None, 237, 64)           256
_________________________________________________________________
conv1d_42 (Conv1D)           (None, 235, 64)           12352
_________________________________________________________________
max_pooling1d_11 (MaxPooling (None, 78, 64)            0
_________________________________________________________________
conv1d_43 (Conv1D)           (None, 76, 128)           24704
_________________________________________________________________
conv1d_44 (Conv1D)           (None, 74, 128)           49280
_________________________________________________________________
global_average_pooling1d_11  (None, 128)               0
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 8)                 1032
=================================================================
Total params: 87,624
Trainable params: 87,624
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/700
1280/1280 [==============================] - 2s 1ms/step - loss: 2.0279 - acc: 0.1633
Epoch 2/700
1280/1280 [==============================] - 1s 794us/step - loss: 1.8371 - acc: 0.3008
Epoch 3/700
1280/1280 [==============================] - 1s 813us/step - loss: 1.5890 - acc: 0.3742
........
Epoch 699/700
1280/1280 [==============================] - 1s 815us/step - loss: 0.1570 - acc: 0.9734
Epoch 700/700
1280/1280 [==============================] - 1s 755us/step - loss: 0.0736 - acc: 0.9844
```






```
320/320 [==============================] - 0s 852us/step
Accuracy: 94.69%
```

7. Training trials

10 trials 700 epochs each

- Try 1: 94.06%
- Try 2: 90.62%
- Try 3: 94.69%
- Try 4: 92.19%
- Try 5: 93.44%
- Try 6: 94.69%
- Try 7: 93.44%
- Try 8: 92.81%
- Try 9: 94.06%
- Try 10: 90.31%

평균 성능: 93.03%

평균 성능: 93.03%

In [22]:
```
scores_list = [94.06, 90.62, 94.69, 92.19, 93.44, 94.69, 93.44, 92.81, 94.06, 90.31]
print ("Average performance: ", "{:.2f}".format(sum(scores_list) / len(scores_list)), "%")
```

```
Average performance:  93.03 %
```

8. Saving model to file

In [11]:
```
# serialize model to JSON
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

9. 새로운 기사를 모델로 예측

- 바로 입력: String 으로 입력하기 위해 약간의 전처리 필요 (따옴표, 엔터 제거)
- 텍스트 파일 입력: 'test_article.txt' 파일에 기사 원문 저장

In [12]:
```
#sample_prediction_article = ""
```

In [23]:
```
#prediction_preprocess = kr_string_preprocess(sample_prediction_article, module_tag, 2, True)
prediction_preprocess = kr_text_preprocess('test_article.txt', module_tag, 2, True)
prediction_data = create_prediction_data(prediction_preprocess, words_dictioinary, word_count, max_length)
prediction_data_reshape = np.reshape(prediction_data, [1, max_length])
prediction_data_3d = np.expand_dims(prediction_data_reshape, axis=2)

print("Prediction demo\n")
print(prediction_preprocess)
print(prediction_data_3d.shape)
```

```
Prediction demo

['소나기', '불안정', '정상회담', '정보', '기상', '미터', '동반', '강원', '날씨', '서울', '정도', '남부', '경북', '수도권', '충
청', '광주', '동해', '오늘', '호남 지방', '영동', '북미', '지역', '동해안', '대기', '대전', '남해안', '최고', '번개', '예보',
'한때', '내륙', '파도', '주의', '천둥', '안팎', '영남 지방', '싱가포르', '너울', '내일', '예상', '우박', '기록', '물결', '지방',
'곳곳', '오후', '바다', '구름', '기온', '한낮']
(1, 239, 1)
```

In [24]:
```
prediction_test = model.predict(prediction_data_3d)
prediction_index = np.argmax(prediction_test)
print("Predicted category: ", article_categories[prediction_index])
print("Accuracy: ", "{:.2%}".format(prediction_test[0][prediction_index]), '\n\n')

for i in range(categories_count):
    if i != prediction_index:
        print("Category: ", article_categories[i], "- ", "{:.2%}".format(prediction_test[0][i]))
```

```
Predicted category:  생활/문화
Accuracy:  99.46%


Category:  정치 -  0.00%
Category:  경제 -  0.00%
Category:  사회 -  0.00%
Category:  세계 -  0.52%
Category:  기술/IT -  0.01%
Category:  연예 -  0.02%
Category:  스포츠 -  0.00%
```

감사합니다