```python
# DongKyu Kim
# ECE 471 CGML Assignment 5
# CIFAR-10
# Professor Curro

# library imports
import tarfile
import csv
import io
import numpy as np
import tensorflow as tf
from keras import optimizers
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import one_hot
from keras.models import Sequential
from keras.layers import Embedding, GlobalAveragePooling1D, Dense
from keras.layers import BatchNormalization, Conv1D, Dropout, MaxPooling1D
from keras.callbacks import LearningRateScheduler

# Parameters
num_class = 4
num_words = 50000
r = 0.95 # Ratio of training data otu of training + validation
learning_rate = 3e-2
epochs = 10
batch_size = 128;

# Useful Functions
def loaddata(name):
    csv_file = io.StringIO(name.read().decode(’ascii’))
    x=[]
    y=[]
    for row in csv.reader(csv_file):
        y.append(int(row[0])-1) # -1 is here so that classes are from 0 to 3
        x.append([str(row[1]),str(row[2])])
    return np.array(x),np.array(y)

def one_hot_encoding(data,numclass):
    targets = np.array(data).reshape(-1)
    return np.eye(numclass)[targets]

def encode_pad_data(data,num_words):
    out = []
    for i in data:
        out.append(np.concatenate((one_hot(i[0],num_words),
            one_hot(i[1],num_words)),axis=None))
    out = np.array(out)
    out = pad_sequences(out,padding=’post’)
    return out

def lr_adaptive(self,epoch):
    lr = learning_rate
    if epoch > 5:
        lr = learning_rate/10
    elif epoch > 10:
        lr = learning_rate/20
    return lr


# Data Import
fid = tarfile.open(’ag-news-csv.tar.gz’,’r:gz’)
x_train,y_train = loaddata(fid.extractfile(’ag_news_csv/train.csv’))
x_test,y_test = loaddata(fid.extractfile(’ag_news_csv/test.csv’))
```

```
y_test = one_hot_encoding(y_test,num_class)
y_train = one_hot_encoding(y_train,num_class)

# Data Encoding
# I decided to use title + description as one feature.
encode_pad_train = encode_pad_data(x_train,num_words)
encode_pad_test = encode_pad_data(x_test,num_words)

mask = np.full(encode_pad_train.shape[0],True)
mask[np.random.choice(encode_pad_train.shape[0],
    int(encode_pad_train.shape[0]*(1-r)), replace=False)] = False
encode_pad_val = encode_pad_train[~mask]
encode_pad_train = encode_pad_train[mask]
y_val = y_train[~mask]
y_train = y_train[mask]

# Model
model = Sequential()
model.add(Embedding(num_words,32))
model.add(Conv1D(64,2,activation=’relu’))
model.add(BatchNormalization())
model.add(GlobalAveragePooling1D())
model.add(Dense(256,activation=’relu’))
model.add(Dropout(0.6))
model.add(Dense(num_class,activation=’softmax’))
# model.summary()
model.compile(optimizer=optimizers.Adam(learning_rate),
    loss=’categorical_crossentropy’,metrics=[’accuracy’])
model.fit(encode_pad_train,y_train,epochs=epochs,batch_size=1024,
    validation_data=(encode_pad_val,y_val),
    callbacks=[LearningRateScheduler(lr_adaptive)],verbose=2)

scores = model.evaluate(encode_pad_test,y_test,verbose=2)
print(’Test loss: ’, scores[0])
print(’Test accuracy:’,scores[1])

# I started with a simple model with two dense layers. It achieved a fairly
# good result of about 86% validation accuracy. I was curious as to how
# effective convolutional networks would be, and surprisingly, it did not
# help that much. I tried a deeper network, upto depth 12, but going deeper
# did not seem to help at all. I played around with the length of the
# embedding vector, width and depth of conv-nets, different activations
# but nothing really improved the result. The state of the art from the paper
# is 92.7%. My final configuration gives results that are in the range of
# validation accuracy of 88% to 91%.
# The final test result that I did as I realized I don’t have any strong
# strategy of improving the model further, came out to be 89.9%.
```