```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from __future__ import print_function
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from time import time
#np.random.seed(1337)  # for reproducibility

from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Convolution1D, MaxPooling1D
from keras.utils import np_utils
from keras.callbacks import TensorBoard


# set parameters:
test_dim = 499
maxlen = 100
nb_filter = 512
filter_length_1 = 10
filter_length_2 = 5
hidden_dims = 750
nb_epoch = 12
nb_classes = 2
split_ratio = 0.15

print('Loading data...')

X = np.load('/content/drive/My Drive/Colab Notebooks/data/numpy_vectors/x_test_mfcc_500_50:50_samples_sliced_out.npy')
```

```
y = np.load('/content/drive/My Drive/Colab Notebooks/data/numpy_vectors/y_label_500_50:50_samples_sliced_out.npy')
print(X.shape)
print(y.shape)
```

```
Loading data...
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-6-4680f96d35c4> in <module>()
     29 print('Loading data...')
     30
---> 31 X = np.load('/content/drive/My Drive/Colab Notebooks/data/numpy_vectors/x_test_mfcc_500_50:50_samples_sliced_
     32 y = np.load('/content/drive/My Drive/Colab Notebooks/data/numpy_vectors/y_label_500_50:50_samples_sliced_out.
     33 print(X.shape)

/usr/local/lib/python3.6/dist-packages/numpy/lib/npyio.py in load(file, mmap_mode, allow_pickle, fix_imports, encodin
    426         own_fid = False
    427     else:
--> 428         fid = open(os_fspath(file), "rb")
    429         own_fid = True
    430

FileNotFoundError: [Errno 2] No such file or directory: '/content/drive/My Drive/Colab Notebooks/data/numpy_vectors/x
```

SEARCH STACK OVERFLOW

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split_ratio)

xts = X_train.shape
#X_train = np.reshape(X_train, (xts[0], xts[1], 1))
xtss = X_test.shape
#X_test = np.reshape(X_test, (xtss[0], xtss[1], 1))
yts = y_train.shape
#y_train = np.reshape(y_train, (yts[0], 1))
ytss = y_test.shape
#y_test = np.reshape(y_test, (ytss[0], 1))

print(len(X_train), 'train sequences')
```

```
print(len(X_train), 'train sequences')
print(len(X_test), 'test sequences')

Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)

# print('Pad sequences (samples x time)')
# X_train = sequence.pad_sequences(X_train, maxlen=maxlen)
# X_test = sequence.pad_sequences(X_test, maxlen=maxlen)
# print('X_train shape:', X_train.shape)
# print('X_test shape:', X_test.shape)
```

```
1273 train sequences
225 test sequences
```

```
for batch_size in range(10, 11, 5):
    print('Build model...')
    model = Sequential()

    # we start off with an efficient embedding layer which maps
    # our vocab indices into embedding_dims dimensions
    # model.add(Embedding(max_features, embedding_dims, input_length=maxlen))
    # model.add(Dropout(0.25))

    # we add a Convolution1D, which will learn nb_filter
    # word group filters of size filter_length:
    model.add(Convolution1D(nb_filter=nb_filter,
                            filter_length=filter_length_1,
                            input_shape=(test_dim, 13),
                            border_mode='valid',
                            activation='relu'
                            ))
    # we use standard max pooling (halving the output of the previous layer):
    model.add(BatchNormalization())

    model.add(Convolution1D(nb_filter=nb_filter,
```

```
                        filter_length=5,
                        border_mode='valid',
                        activation='relu'
                        ))

model.add(BatchNormalization())

model.add(MaxPooling1D(pool_length=2))

model.add(Convolution1D(nb_filter=nb_filter,
                        filter_length=25,
                        border_mode='same',
                        activation='relu'
                        ))

model.add(BatchNormalization())

model.add(MaxPooling1D(pool_length=2))

model.add(Convolution1D(nb_filter=nb_filter,
                        filter_length=50,
                        border_mode='same',
                        activation='relu'
                        ))

model.add(BatchNormalization())

model.add(MaxPooling1D(pool_length=2))

model.add(Convolution1D(nb_filter=nb_filter,
                        filter_length=2,
                        border_mode='same',
                        activation='relu'
                        ))

model.add(BatchNormalization())
```

```
model.add(MaxPooling1D(pool_length=2))

# We flatten the output of the conv layer,
# so that we can add a vanilla dense layer:
model.add(Flatten())

# We add a vanilla hidden layer:
# model.add(Dense(hidden_dims))
model.add(Dropout(0.25))
# model.add(Activation('relu'))

model.add(Dense(1000))
model.add(Activation('relu'))
model.add(Dense(750))
model.add(Activation('relu'))
model.add(Dense(50))
model.add(Activation('relu'))
# We project onto a single unit output layer, and squash it with a sigmoid:
model.add(Dense(nb_classes))
model.add(Activation('softmax'))

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

print("model/split = {} <> batchsize = {}".format(split_ratio, batch_size))
tensorboard = TensorBoard(log_dir="logs/split_{}_batchsize_{}".format(split_ratio, batch_size))

model.fit(X_train, Y_train, batch_size=batch_size,
        nb_epoch=10,  verbose=1, callbacks=[tensorboard]  )

# model.save('model_hin_tel_38_samples.h5')

y_preds = model.predict(X_test)
for i in range(len(y_preds)):
    print(y_preds[i], y_test[i])

score = model.evaluate(X_test, Y_test, verbose=1)
print(score)
```

```
      print("\n*********************************\n")

  # print(classification_report(Y_test, Y_preds))
```

```
[9.9999595e-01 4.1118506e-06] 0
[9.999999e-01 7.172379e-08] 0
[0. 1.] 1
[8.55177e-13 1.00000e+00] 1
[1.0000000e+00 6.0110055e-12] 0
[1.0000000e+00 1.1615431e-12] 0
[1.0000000e+00 7.7009417e-17] 0
[0.00489223 0.99510777] 1
[4.1049173e-30 1.0000000e+00] 1
[1.000000e+00 7.806691e-15] 0
[1.0000000e+00 2.4161398e-16] 0
[0. 1.] 1
[1.000000e+00 3.507439e-16] 0
[1.0000000e+00 3.0289424e-11] 0
[0. 1.] 1
[1.0000000e+00 1.7882132e-11] 0
[9.999981e-01 1.854361e-06] 0
[0. 1.] 1
[0. 1.] 1
[0. 1.] 1
[0. 1.] 1
[1.000000e+00 3.966624e-13] 0
[1.0000000e+00 4.9043398e-11] 0
[1.0000000e+00 3.0563344e-11] 0
[1.000000e+00 7.388705e-11] 0
[0. 1.] 1
[0. 1.] 1
[1.000000e+00 4.969569e-10] 0
[1.0000000e+00 3.3645626e-09] 0
[1.0000000e+00 5.3188794e-19] 0
[9.99895215e-01 1.04796905e-04] 0
[1.6291619e-04 9.9983704e-01] 1
[1.0000000e+00 1.2124434e-09] 0
[0. 1.] 1
[0. 1.] 1
[1.000000e+00 7.202074e-15] 0
[1.0000000e+00 3.8066872e-08] 0
[0. 1.] 1
```

```
[1.000000e+00 7.141009e-16] 0
[1.0000000e+00 5.2745325e-10] 0
[9.9999928e-01 6.6868074e-07] 0
[1.000000e+00 5.826321e-16] 0
[0.49737185 0.50262815] 0
[1.0000000e+00 1.4455808e-15] 0
[1.8994878e-19 1.0000000e+00] 1
[0. 1.] 1
[0. 1.] 1
[0. 1.] 1
[0. 1.] 1
[1.0000000e+00 6.6683902e-12] 0
[0. 1.] 1
[1.000000e+00 2.604208e-11] 0
[0. 1.] 1
[1.0000000e+00 1.2759153e-15] 0
[1.0000000e+00 5.6002547e-16] 0
225/225 [==============================] - 1s 5ms/step
[0.03600539289628759, 0.9911111111111112]

**********
```