1. 提取特征

```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:


get_ipython().system('unzip /home/aistudio/data/data41960/dddd.zip -d
/home/aistudio/work/')


# In[3]:


import os

path = "/home/aistudio/work/"
os.chdir(path)

print(os.getcwd())

get_ipython().system('nvidia-smi')


import os
import wave
import librosa
import numpy as np
from tqdm import tqdm
import pickle as pkl
import librosa.display
from sklearn.preprocessing import normalize
import matplotlib.pyplot as plt
# import librosa.display



# In[ ]:


import os
import wave
import librosa
import numpy as np
from tqdm import tqdm
import pickle as pkl
import librosa
from sklearn.preprocessing import normalize
```

```
def extract_logmel(y, sr, size=3):
    """
    extract log mel spectrogram feature
    :param y: the input signal (audio time series)
    :param sr: sample rate of 'y'
    :param size: the length (seconds) of random crop from original audio, default
as 3 seconds
    :return: log-mel spectrogram feature
    """
    # normalization
    y = y.astype(np.float32)
    normalization_factor = 1 / np.max(np.abs(y))
    y = y * normalization_factor

    # random crop
    if len(y) <= size * sr:
        new_y = np.zeros((size * sr+1, ))
        new_y[:len(y)] = y
        y = new_y

    start = np.random.randint(0, len(y) - size * sr)
    y = y[start: start + size * sr]

    # extract log mel spectrogram #####
    melspectrogram = librosa.feature.melspectrogram(
        y=y, sr=sr, n_fft=2048, hop_length=1024, n_mels=60)
    logmelspec = librosa.power_to_db(melspectrogram)

    return logmelspec

def get_wave_norm(file):
    data, framerate = librosa.load(file, sr=22050)
    return data, framerate


LABELS = ['awake', 'diaper', 'hug', 'hungry', 'sleepy', 'uncomfortable']
N_CLASS = len(LABELS)


# In[ ]:


import os
import wave
import librosa
import numpy as np
from tqdm import tqdm
import pickle as pkl
import librosa
from sklearn.preprocessing import normalize

################################################    提取mfcc特征
```

```
def extract_mfcc(y, sr, size=3):
    """
    extract log mel spectrogram feature
    :param y: the input signal (audio time series)
    :param sr: sample rate of 'y'
    :param size: the length (seconds) of random crop from original audio, default
as 3 seconds
    :return: log-mel spectrogram feature
    """
    # normalization
    y = y.astype(np.float32)
    normalization_factor = 1 / np.max(np.abs(y))
    y = y * normalization_factor

    # random crop
    if len(y) <= size * sr:
        new_y = np.zeros((size * sr+1, ))
        new_y[:len(y)] = y
        y = new_y

    start = np.random.randint(0, len(y) - size * sr)
    y = y[start: start + size * sr]

    # extract log mel spectrogram #####
    mfcc = librosa.feature.mfcc(
        y=y, sr = sr, hop_length = 1024,n_mfcc=60
    )
    # melspectrogram = librosa.feature.melspectrogram(
    #     y=y, sr=sr, n_fft=2048, hop_length=1024, n_mels=60)
    # logmelspec = librosa.power_to_db(melspectrogram)

    return mfcc

def get_wave_norm(file):
    data, framerate = librosa.load(file, sr=22050)
    return data, framerate


LABELS = ['awake', 'diaper', 'hug', 'hungry', 'sleepy', 'uncomfortable']
N_CLASS = len(LABELS)

#

file_glob = []
DATA_DIR = './train'



data = []

file_glob = []

for i, cls_fold in enumerate(os.listdir(DATA_DIR)):
```

```
        cls_base = os.path.join(DATA_DIR, cls_fold)
        lbl = cls_fold

        files = os.listdir(cls_base)
        print('{} train num:'.format(lbl), len(files))
        for pt in files:
            file_pt = os.path.join(cls_base, pt)
            file_glob.append((file_pt, LABELS.index(lbl)))


print('done.')
print(len(file_glob))

data = []

for file, lbl in tqdm(file_glob):
    try:
        raw, sr = get_wave_norm(file)
    except Exception as e:
        print(e, file)
    feature = extract_mfcc(y=raw, sr=sr, size=15) #########  3
    y = np.zeros(N_CLASS)
    y[lbl] = 1
    data.append((feature, y))


with open('./data_mfcc_60.pkl', 'wb') as f:
    pkl.dump(data, f)

del data


# In[ ]:


# logmel

file_glob = []
DATA_DIR = './train'



data = []

file_glob = []

for i, cls_fold in enumerate(os.listdir(DATA_DIR)):

        cls_base = os.path.join(DATA_DIR, cls_fold)
        lbl = cls_fold

        files = os.listdir(cls_base)
```

```
        print('{} train num:'.format(lbl), len(files))
        for pt in files:
            file_pt = os.path.join(cls_base, pt)
            file_glob.append((file_pt, LABELS.index(lbl)))


print('done.')
print(len(file_glob))

data = []

for file, lbl in tqdm(file_glob):
    try:
        raw, sr = get_wave_norm(file)
    except Exception as e:
        print(e, file)
    feature = extract_logmel(y=raw, sr=sr, size=15) #########  3
    y = np.zeros(N_CLASS)
    y[lbl] = 1
    data.append((feature, y))


with open('./data.pkl', 'wb') as f:
    pkl.dump(data, f)

del data


# In[ ]:


import os
import wave
import numpy as np
import pickle as pkl

train_x = []
train_y = []

LABELS = ['awake', 'diaper', 'hug', 'hungry', 'sleepy', 'uncomfortable']
N_CLASS = len(LABELS)

with open('./data_mfcc_60.pkl', 'rb') as f:
    raw_data = pkl.load(f)

np.random.seed(5)
np.random.shuffle(raw_data)

print(raw_data[2][0].shape)

train_data = raw_data[:-50]
valid_data = raw_data[-50:]
```

```
# In[ ]:


print(len(train_data))
print(len(valid_data))
print(train_data[0][0].shape)


# In[ ]:


import numpy as np
import paddle as paddle
import paddle.fluid as fluid
from PIL import Image
import matplotlib.pyplot as plt
import os
import math

def reader_createor(data):
    def reader():
        for i in  range(len(data)):
            x = np.expand_dims(data[i][0].T, axis=0)
            y = np.argmax(data[i][1])
            if not np.random.randint(0, 2):
                noise = np.random.rand(x.shape[0], x.shape[1], x.shape[2]) * 0.08 *
x - 0.04
                x += noise
            yield x, y
    return reader


train_reader = paddle.batch(
    paddle.reader.shuffle(
        reader=reader_createor(train_data),buf_size=100
    ), batch_size=64
)

valid_reader = paddle.batch(
    paddle.reader.shuffle(
        reader=reader_createor(valid_data),buf_size=100
    ), batch_size=64
)

print('done.')


# In[ ]:


class MyNet():
    def __init__(self, is_train=True):
```

```
        self.is_train = is_train
        self.weight_decay = 1e-4

    def net(self, input, class_dim):

        depth = [3, 3, 3, 3, 3]
        num_filters = [16, 16, 32, 32, 64]

        conv = self.conv_bn_layer(
            input=input, num_filters=16, filter_size=3, act='elu')
        conv = fluid.layers.pool2d(
            input=conv,
            pool_size=3,
            pool_stride=2,
            pool_padding=1,
            pool_type='max')

        for block in range(len(depth)):
            for i in range(depth[block]):
                conv = self.bottleneck_block(
                    input=conv,
                    num_filters=num_filters[block],
                    stride=2 if i == 0 and block != 0 else 1)
                conv = fluid.layers.batch_norm(input=conv)
        print(conv.shape)
        pool = fluid.layers.pool2d(
            input=conv, pool_size=2, pool_type='max', global_pooling=False)

        pool = fluid.layers.conv2d(
            input=pool, num_filters=32, filter_size=[3, 1], stride=[2, 1],
act='elu')
        print(pool.shape)
        pool = fluid.layers.flatten(pool)
        pool = fluid.layers.dropout(pool, dropout_prob=0.5)
        net = fluid.layers.fc(input=pool,
                              size=128,
                              act="elu"
                              )
        print(net.shape)
        stdv = 1.0 / math.sqrt(pool.shape[1] * 1.0)

        out = fluid.layers.fc(input=net,
                              size=class_dim,
                              act="softmax",
                              param_attr=fluid.param_attr.ParamAttr(
                                  initializer=fluid.initializer.Uniform(-stdv,
stdv),

 regularizer=fluid.regularizer.L2Decay(self.weight_decay))
                              )
        return out
```

```
    def conv_bn_layer(self,
                      input,
                      num_filters,
                      filter_size,
                      stride=1,
                      groups=1,
                      act=None,
                      bn_init_value=1.0):
        conv = fluid.layers.conv2d(
            input=input,
            num_filters=num_filters,
            filter_size=filter_size,
            stride=stride,
            padding=(filter_size - 1) // 2,
            groups=groups,
            act=None,
            bias_attr=False,

 param_attr=fluid.ParamAttr(regularizer=fluid.regularizer.L2Decay(self.weight_decay
)))
        return fluid.layers.batch_norm(
                input=conv, act=act, is_test=not self.is_train,
                param_attr=fluid.ParamAttr(
                    initializer=fluid.initializer.Constant(bn_init_value),
                    regularizer=None))

    def shortcut(self, input, ch_out, stride):
        ch_in = input.shape[1]
        if ch_in != ch_out or stride != 1:
            return self.conv_bn_layer(input, ch_out, 1, stride)
        else:
            return input

    def bottleneck_block(self, input, num_filters, stride):
        conv0 = self.conv_bn_layer(
            input=input, num_filters=num_filters, filter_size=1, act='relu')
        conv1 = self.conv_bn_layer(
            input=conv0,
            num_filters=num_filters,
            filter_size=3,
            stride=stride,
            act='relu')
        conv2 = self.conv_bn_layer(
            input=conv1, num_filters=num_filters * 4, filter_size=1, act=None,
bn_init_value=0.0)

        short = self.shortcut(input, num_filters * 4, stride)

        return fluid.layers.elementwise_add(x=short, y=conv2, act='relu')


# In[ ]:
```

```
# 定义输入输出层
image = fluid.layers.data(name='image', shape=[1, 323, 60], dtype='float32')  # 单通道，28*28像素值
label = fluid.layers.data(name='label', shape=[1], dtype='int64')              # 图片标签


# In[ ]:


# 获取分类器
model = MyNet()
out = model.net(input=image, class_dim=6) # class_dim = 分类的数目

# 获取损失函数和准确率函数
cost = fluid.layers.cross_entropy(input=out, label=label)
avg_cost = fluid.layers.mean(cost)
acc = fluid.layers.accuracy(input=out, label=label)

# 定义优化方法
optimizer = fluid.optimizer.AdamOptimizer(learning_rate=2e-4)   #使用Adam算法进行优化
opts = optimizer.minimize(avg_cost)


# In[ ]:


# 定义一个使用CPU的解析器
model_save_dir = "/home/aistudio/work/mfcc.inference.model"

place = fluid.CUDAPlace(0)
exe = fluid.Executor(place)
exe.run(fluid.default_startup_program())
# fluid.io.load_params(executor=exe, dirname=model_save_dir,
#                      main_program=None)
feeder = fluid.DataFeeder(place=place, feed_list=[image, label])


# In[ ]:


def draw_train_process(title,iters,costs,accs,label_cost,lable_acc):
    plt.title(title, fontsize=24)
    plt.xlabel("iter", fontsize=20)
    plt.ylabel("cost/acc", fontsize=20)
    plt.plot(iters, costs,color='red',label=label_cost)
    plt.plot(iters, accs,color='green',label=lable_acc)
    plt.legend()
    plt.grid()
    plt.show()

all_train_iter=0
```

```
all_train_iters=[]
all_train_costs=[]
all_train_accs=[]


# In[ ]:


from tqdm import tqdm

EPOCH_NUM=20  # 调参 训练轮数 20

for pass_id in range(EPOCH_NUM):
    # 进行训练
    for data in tqdm(train_reader()):                          #遍历train_reader
        train_cost, train_acc = exe.run(program=fluid.default_main_program(),#运行主
程序
                                        feed=feeder.feed(data),              #给模型
喂入数据
                                        fetch_list=[avg_cost, acc])          #fetch
误差、准确率

        all_train_iter=all_train_iter+1
        all_train_iters.append(all_train_iter)
        all_train_costs.append(train_cost[0])
        all_train_accs.append(train_acc[0])

    print('Pass:%d, Cost:%0.5f, Accuracy:%0.5f' %
                  (pass_id, np.mean(train_cost), np.mean(train_acc)))

    # 进行测试
    test_accs = []
    test_costs = []
    #每训练一轮 进行一次测试
    for batch_id, data in enumerate(valid_reader()):                          #遍历
test_reader
        test_cost, test_acc = exe.run(program=fluid.default_main_program(), #执行训
练程序
                                      feed=feeder.feed(data),                #喂入数
据
                                      fetch_list=[avg_cost, acc])            #fetch
误差、准确率
        test_accs.append(test_acc[0])                                        #每个
batch的准确率
        test_costs.append(test_cost[0])                                      #每个
batch的误差

    # 求测试结果的平均值
    test_cost = (sum(test_costs) / len(test_costs))                          #每轮的
平均误差
    test_acc = (sum(test_accs) / len(test_accs))                             #每轮的
平均准确率
    print('Test:%d, Cost:%0.5f, Accuracy:%0.5f' % (pass_id, test_cost, test_acc))
```

```
    #保存模型
    # 如果保存路径不存在就创建
    if not os.path.exists(model_save_dir):
        os.makedirs(model_save_dir)
    print ('save models to %s' % (model_save_dir))
    fluid.io.save_inference_model(model_save_dir,  #保存推理model的路径
                                  ['image'],       #推理 (inference) 需要 feed 的数据
                                  [out],       #保存推理 (inference) 结果的 Variables
                                  exe)             #executor 保存 inference model
draw_train_process("training",all_train_iters,all_train_costs,all_train_accs,"train
ning cost","trainning acc")


# In[ ]:


import os
import wave
import librosa
import numpy as np
from tqdm import tqdm
import pickle as pkl
from sklearn.preprocessing import normalize

def extract_logmel (y, sr,size=3):
    """
    extract log mel spectrogram feature
    : param y: the input signal (audio time series)
    : param sr: sample rate of 'y'
    : param size: the length (seconds) of random crop from original audio, default
as 3 seconds
    """
    # normalization
    y = y.astype(np.float32)
    normalization_factor = 1 / np.max(np.abs(y))
    y = y * normalization_factor

    if len(y) <= size * sr:
        new_y = np.zeros((size * sr + 1,))
        new_y[:len(y)] = y
        y = new_y

    start = np.random.randint(0,len(y)-size*sr)  # 随机选取一个开始点
    y = y[start : start + size * sr]               # 随机截取一下 y

    melspectrogram = librosa.feature.melspectrogram(y = y,
                                                    sr = sr,
                                                    n_fft = 2048,
                                                    hop_length = 1024,
                                                    n_mels = 60)

    logmelspec = librosa.power_to_db(melspectrogram)
```

```
    return logmelspec

def extract_mfcc (y, sr,size=3):
    """
    extract log mel spectrogram feature
    : param y: the input signal (audio time series)
    : param sr: sample rate of 'y'
    : param size: the length (seconds) of random crop from original audio, default
as 3 seconds
    """
    # normalization
    y = y.astype(np.float32)
    normalization_factor = 1 / np.max(np.abs(y))
    y = y * normalization_factor

    if len(y) <= size * sr:
        new_y = np.zeros((size * sr + 1,))
        new_y[:len(y)] = y
        y = new_y

    start = np.random.randint(0,len(y)-size*sr)  # 随机选取一个开始点
    y = y[start : start + size * sr]               # 随机截取一下 y

    mfcc = librosa.feature.mfcc(y = y,
                                                sr = sr,
                                                n_fft = 2048,
                                                hop_length = 1024,
                                                n_mfcc = 60)


    return mfcc

def get_wave_norm(file):
    data, framerate = librosa.load(file, sr = 22050)
    return data,framerate

LABELS = ['awake', 'diaper', 'hug', 'hungry', 'sleepy', 'uncomfortable']
DATA_DIR = './train'
DATA_DIR = './test'

file_glob = []
# data = []
data = {}

# for
files = os.listdir(DATA_DIR)
print('test num:' , len(files))
for pt in files:
    file_pt = os.path.join(DATA_DIR,pt)
    file_glob.append(file_pt)                #

print("done")
```

```
print(len(file_glob))

for fileone in tqdm(file_glob):
    try:
        raw,sr = get_wave_norm(fileone)
    except Exception as e:
        print(e,fileone)
    feature = extract_mfcc(y = raw, sr = sr, size = 15)           # 15 s 是不对的 提
取的特征不一样
    # y = np.zeros(len(LABELS))
    # y[lbl] = 1
    basename = os.path.basename(fileone)
    data[basename] = feature
    # data.append((feature,y))

with open('./data_test_mfcc_60.pkl', 'wb') as f:
    pkl.dump(data,f)

del data


# In[ ]:


test_x = []
test_y = []

with open ('./data_test.pkl','rb') as f:
    raw_data = pkl.load(f)

# print(raw_data[0][0].shape)

test_data = raw_data

print (len(test_data))
print (type(test_data))


# In[ ]:


test_x = []
test_y = []

with open ('./data_test_mfcc_60.pkl','rb') as f:
    raw_data = pkl.load(f)

# print(raw_data[0][0].shape)

test_data = raw_data

print (len(test_data))
print (type(test_data))
```

```
# In[ ]:


infer_exe = fluid.Executor(place)
#声明一个新的作用域
inference_scope = fluid.core.Scope()


# In[ ]:


import os
import wave
import numpy as np
import pickle as pkl

from tqdm import tqdm
import pandas as pd

LABELS = ['awake', 'diaper', 'hug', 'hungry', 'sleepy', 'uncomfortable']
N_CLASS = len(LABELS)

with open('./data_test.pkl', 'rb') as f:
    raw_data = pkl.load(f)

feeder = fluid.DataFeeder(place=place, feed_list=[image])

result = {'id': [], 'label': []}

# model_save_dir = "/home/aistudio/data/hand.inference.model"
#运行时中的所有变量都将分配给新的scope
with fluid.scope_guard(inference_scope):
    #获取训练好的模型
    #从指定目录中加载模型
    [inference_program,                                                    #推理Program
     feed_target_names,                                                    #是一个str列表，它
包含需要在推理 Program 中提供数据的变量的名称。
     fetch_targets] = fluid.io.load_inference_model(model_save_dir,#fetch_targets:
是一个列表，从中我们可以得到推断结果。model_save_dir：模型保存的路径
                                                    infer_exe)     #infer_exe: 运行
inference model的 executor

    for key, value in tqdm(raw_data.items()):
    # for key, value in tqdm(raw_data):

        # x = np.expand_dims(np.array(value), axis=1)
        x = np.expand_dims(np.array(value).T,axis = 0)
        x = np.expand_dims(x,axis = 0)

        y = infer_exe.run(program=inference_program,          #运行推测程序
                  feed={feed_target_names[0]: x},             #喂入要预测的img
```

```
                    fetch_list=fetch_targets)[0]                    #得到推测结果,
        if len(y) == 0:
            print(key)
        else:
            y = np.mean(y, axis=0)
            y = np.argmax(y)
            pred = LABELS[y]
        key = os.path.basename(key)
        result['id'].append(key)
        result['label'].append(pred)

result = pd.DataFrame(result)
result.to_csv('./submission_mfcc.csv', index=False)


# In[ ]:


import matplotlib.pyplot as plt
import librosa.display
import librosa

audio_path = './train/uncomfortable/uncomfortable_1.wav'
x , sr = librosa.load(audio_path)
print(len(x))
x = x[:10000]
X,_ = librosa.stft(x)

y, sr = librosa.load(librosa.util.example_audio_file())
D = librosa.stft(y)
magnitude, phase = librosa.magphase(D)

Xdb = magnitude

plt.figure(figsize=(14, 5))
librosa.display.specshow(Xdb, cmap='Reds',sr=sr, x_axis='time', y_axis='hz')

plt.ylim([0,4000])
plt.colorbar()
# plt.savefig('3.png')
plt.show()


# In[14]:


y, sr = librosa.load('./train/hungry/hungry_64.wav',sr=16000,duration=10)
# y = y[:16000 ]
D = librosa.stft(y)
# magnitude, phase = librosa.magphase(D)
Xdb = librosa.amplitude_to_db(abs(D))
# Xdb = D
plt.figure(figsize=(14, 5))
```

```
librosa.display.specshow(Xdb,sr=sr, x_axis='time', y_axis='hz')
plt.ylim([0,8000])
plt.colorbar()
# plt.savefig('3.png')
plt.show()
plt.clf()


# In[ ]:


y, sr = librosa.load('./train/uncomfortable/uncomfortable_3.wav')
y = librosa.stft(y)
y = librosa.amplitude_to_db(abs(y))
y = librosa.db_to_power(y)


plt.figure(figsize=(14, 5))
librosa.display.waveplot(y,sr=sr)
plt.show()


# In[ ]:


y, sr = librosa.load('./train/uncomfortable/uncomfortable_3.wav')
y = librosa.stft(y,n_fft=2048)
y = librosa.amplitude_to_db(abs(y))
# y = librosa.db_to_power(y)

print(y.shape)



# In[10]:


import numpy as np
import wave
import matplotlib.pyplot as plt
wlen=1024
inc=256
f = wave.open(r"./train/hungry/hungry_64.wav", "rb")
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
str_data = f.readframes(nframes)
wave_data = np.fromstring(str_data, dtype=np.short)
raw = wave_data
wave_data = wave_data*1.0/(max(abs(wave_data)))
print(wave_data[:10])

y,sr = librosa.load("./train/hungry/hungry_64.wav")
raw = y
```

```
print("wavedata--:",len(wave_data))
print("y-----:",len(y))
y = y*1.0/(max(abs(y)))
print("wavedata--:",type(wave_data))
wave_data = y*1.0/(max(abs(y)))
print("wavedata--:",len(wave_data))

signal_length=len(wave_data) #信号总长度
if signal_length<=wlen: #若信号长度小于一个帧的长度，则帧数定义为1
        nf=1
else: #否则，计算帧的总长度
        nf=int(np.ceil((1.0*signal_length-wlen+inc)/inc))
print(nf)


pad_length=int((nf-1)*inc+wlen) #所有帧加起来总的铺平后的长度
zeros=np.zeros((pad_length-signal_length,)) #不够的长度使用0填补，类似于FFT中的扩充数组操
作
pad_signal=np.concatenate((wave_data,zeros)) #填补后的信号记为pad_signal

indices=np.tile(np.arange(0,wlen),(nf,1))+np.tile(np.arange(0,nf*inc,inc),
(wlen,1)).T  #相当于对所有帧的时间点进行抽取，得到nf*nw长度的矩阵
print(indices[:2])


indices=np.array(indices,dtype=np.int32) #将indices转化为矩阵
frames=pad_signal[indices] #得到帧信号
windown=np.hanning(wlen)
d=np.zeros(nf)
x=np.zeros(nf)
time = np.arange(0,nf) * (inc*1.0/framerate)
for i in range(0,nf):              #############
        a=frames[i:i+1]
        b = a[0] * windown
        c=np.square(b)
        d[i]=np.sum(c)

d = d*1.0/(max(abs(d)))
print(d)


plt.figure(figsize=(10,4))
plt.plot(time,d,c="g")
plt.grid()
plt.show()


# In[13]:


print(indices.shape)  # 2254,1024

max_power = max(d)
```

```
wave_raw_data = []
wave_data = raw
# indices 每一行都是一帧，有1024个，帧移为 256

for i in range(0,nf):
    if(d[i] *5 < max_power):
        end = indices[i][0]+1024
        break
tag = 0
for i in range(0,nf):
    if(d[i] *5 < max_power): #删除这帧
        if(indices[i][0] > end):
            wave_raw_data.extend(wave_data[indices[i][0]:indices[i][-1]+1])
            end = indices[i][0]+1024
        elif(indices[i][0] < end):
            wave_raw_data.extend(wave_data[end+1:indices[i][-1]+1])
            end = indices[i][0]+1024
        tag = tag + 1
print(len(wave_data))
print(tag)
print((wave_raw_data[:10]))
wave_raw_data = np.array(wave_raw_data)
# wave_raw_data = wave_raw_data*1.0/(max(abs(wave_raw_data)))
D = librosa.stft(wave_raw_data)

# D = librosa.stft(y)
# magnitude, phase = librosa.magphase(D)
Xdb = librosa.amplitude_to_db(abs(D))
# Xdb = D
plt.figure(figsize=(14, 5))
librosa.display.specshow(Xdb,sr=sr, x_axis='time', y_axis='hz')
plt.ylim([0,8000])
plt.colorbar()
plt.savefig('./PNG/3.png')
plt.show()


# In[14]:


print(len(wave_raw_data))
print (tag,nf)


# In[ ]:


a = []
b = [1,23,4,4,42]
c = [[1,2,3],[3,4,5]]
a.extend(b[c[0][0]:c[0][-1]])

print(a)
```

```
# In[ ]:


import numpy as np
import wave
import matplotlib.pyplot as plt

path = './train/uncomfortable/uncomfortable_3.wav'
def get_png(path,tag=False,pngname=None,idx=None):
    wlen=1024
    inc=256

    y,sr = librosa.load(path)
    raw = y

    wave_data = y*1.0/(max(abs(y)))


    signal_length=len(wave_data) #信号总长度
    if signal_length<=wlen: #若信号长度小于一个帧的长度，则帧数定义为1
        nf=1
    else: #否则，计算帧的总长度
        nf=int(np.ceil((1.0*signal_length-wlen+inc)/inc))



    pad_length=int((nf-1)*inc+wlen) #所有帧加起来总的铺平后的长度
    zeros=np.zeros((pad_length-signal_length,)) #不够的长度使用0填补，类似于FFT中的扩充数
组操作
    pad_signal=np.concatenate((wave_data,zeros)) #填补后的信号记为pad_signal

    indices=np.tile(np.arange(0,wlen),(nf,1))+np.tile(np.arange(0,nf*inc,inc),
(wlen,1)).T  #相当于对所有帧的时间点进行抽取，得到nf*nw长度的矩阵



    indices=np.array(indices,dtype=np.int32) #将indices转化为矩阵
    frames=pad_signal[indices] #得到帧信号
    windown=np.hanning(wlen)
    d=np.zeros(nf)
    x=np.zeros(nf)
    time = np.arange(0,nf) * (inc*1.0/framerate)
    for i in range(0,nf):              #############
        a=frames[i:i+1]
        b = a[0] * windown
        c=np.square(b)
        d[i]=np.sum(c)

    d = d*1.0/(max(abs(d)))
```

```
    max_power = max(d)
    wave_raw_data = []
    wave_data = raw
# indices 每一行都是一帧，有1024个，帧移为 256

    for i in range(0,nf):
        if(d[i] *5 < max_power):
            end = indices[i][0]+1024
            break

    for i in range(0,nf):
        if(d[i] *5 < max_power): #删除这帧
            if(indices[i][0] > end):
                wave_raw_data.extend(wave_data[indices[i][0]:indices[i][-1]+1])
                end = indices[i][0]+1024
            elif(indices[i][0] < end):
                wave_raw_data.extend(wave_data[end+1:indices[i][-1]+1])
                end = indices[i][0]+1024


    wave_raw_data = np.array(wave_raw_data)

    if tag is True:
        D = librosa.stft(wave_raw_data)


        Xdb = librosa.amplitude_to_db(abs(D))

        plt.figure(figsize=(14, 5))
        librosa.display.specshow(Xdb,sr=sr, x_axis='time', y_axis='hz')
        plt.ylim([0,8000])
        plt.colorbar()
        if pngname is None:
            Dir = './PNG'
            name = str(idx) + '.png'
        else:
            Dir = './PNG/' + pngname
            name = pngname + '_' + str(idx) + '.png'
        save_path = os.path.join(Dir,name)
        plt.savefig(save_path)


    else:
        # plt.clf()
        for i in range(0,len(wave_raw_data)//22050):
            start = i*22050
            end = start + 22050
            D = librosa.stft(wave_raw_data[start:end])


            Xdb = librosa.amplitude_to_db(abs(D))

            plt.figure(figsize=(14, 5))
```

```
            librosa.display.specshow(Xdb,sr=sr, x_axis='time', y_axis='hz')
            plt.ylim([0,8000])
            plt.colorbar()

            if pngname is None:
                Dir = './PNG'
                name = str(i) + '.png'
            else:
                Dir = './PNG/' + pngname
                name = pngname + '_' + str(i) + '.png'
            save_path = os.path.join(Dir,name)
            plt.savefig(save_path)









# In[ ]:


file_glob = []
DATA_DIR = './train'



data = []

file_glob = []

for i, cls_fold in enumerate(os.listdir(DATA_DIR)):

        cls_base = os.path.join(DATA_DIR, cls_fold)
        lbl = cls_fold                                          # 文件夹名

        files = os.listdir(cls_base)
        print('{} train num:'.format(lbl), len(files))
        for pt in files:                                        # pt是文件名
       hungry_0.wav
            file_pt = os.path.join(cls_base, pt)                # 每一个文件地址
            # file_glob.append((file_pt, LABELS.index(lbl)))
            # get_png(path = file_pt, tag=True, name = pt.trim('_')[0])
            print(pt)


# In[6]:


# path = './train/hungry/hungry_0.wav'
# get_png(path,tag=True)

file_glob = []
```

```
DATA_DIR = './train'



data = []

file_glob = []

for i, cls_fold in enumerate(os.listdir(DATA_DIR)):

        cls_base = os.path.join(DATA_DIR, cls_fold)
        lbl = cls_fold                                          # 文件夹名

        files = os.listdir(cls_base)
        print('{} train num:'.format(lbl), len(files))

        for pt in files:                                        # pt是文件名
       hungry_0.wav
            if pt.split('_')[0] != 'hug':
                continue
            file_pt = os.path.join(cls_base, pt)                # 每一个文件地址
            # file_glob.append((file_pt, LABELS.index(lbl)))

            get_png(path = file_pt, tag=True, pngname = pt.split('_')[0],idx =
pt.split('_')[1].split('.')[0])
            print(pt,"done")
            plt.close()
            plt.clf()


# for file, lbl in tqdm(file_glob):
#     try:
#         raw, sr = get_wave_norm(file)
#     except Exception as e:
#         print(e, file)
#     feature = extract_logmel(y=raw, sr=sr, size=15) #########  3
#     y = np.zeros(N_CLASS)
#     y[lbl] = 1
#     data.append((feature, y))


# In[ ]:


file_glob = []
DATA_DIR = './PNG'



data = []

file_glob = []
```

```
for i, cls_fold in enumerate(os.listdir(DATA_DIR)):

        cls_base = os.path.join(DATA_DIR, cls_fold)
        lbl = cls_fold                                          # 文件夹名

        files = os.listdir(cls_base)
        print('{} train num:'.format(lbl), len(files))

        for pt in files:                                        # pt是文件名
       hungry_0.wav
            # if pt.split('_')[0] != 'awake':
            #     continue
            file_pt = os.path.join(cls_base, pt)                # 每一个文件地址
            file_glob.append((file_pt, LABELS.index(lbl)))

            # get_png(path = file_pt, tag=True, pngname = pt.split('_')[0],idx =
pt.split('_')[1].split('.')[0])
            # print(pt,"done")
            # plt.close()
            # plt.clf()


for file, lbl in tqdm(file_glob):
    try:
        raw, sr = get_wave_norm(file)
    except Exception as e:
        print(e, file)
    feature = extract_logmel(y=raw, sr=sr, size=15) #########  3
    y = np.zeros(N_CLASS)
    y[lbl] = 1
    data.append((feature, y))

with open('./data.pkl', 'wb') as f:
    pkl.dump(data, f)


# In[40]:


from PIL import Image

path = './PNG/awake/awake_9.png'
im = Image.open(path).convert('L')
raw = im
im = im.resize((28, 28), Image.ANTIALIAS)
im = np.array(im).reshape(1, 1, 28, 28).astype(np.float32)
im = im / 255.0 * 2.0 - 1.0
plt.imshow(raw)
plt.show()
```