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

# In[ ]:


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


# In[ ]:


import os

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

print(os.getcwd())

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


# In[6]:


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):                    # 提取特征
    """
    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):                                # 读取语音
    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'

for i, cls_fold in tqdm(enumerate(LABELS)):

    cls_base = os.path.join(DATA_DIR, cls_fold)
    files = os.listdir(cls_base)
    print('{} train num:'.format(cls_fold), len(files))
    for pt in files:
        file_pt = os.path.join(cls_base, pt)
        file_glob.append((file_pt, LABELS.index(cls_fold)))  # file_glob 是数据集（地址，
类型）

print('done.')

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('{} {} num:'.format(lbl, type_fold), len(files))
        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)
    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[7]:


file_glob = []
DATA_DIR = './train'

for i, cls_fold in tqdm(enumerate(LABELS)):

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

print('done.')

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('{} {} num:'.format(lbl, type_fold), len(files))
        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)
    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[8]:


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.pkl', 'rb') as f:
    raw_data = pkl.load(f)

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

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

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

```
# In[9]:


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


# In[10]:


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[11]:


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[12]:


# 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[13]:


# 获取分类器
model = MyNet()
out = model.net(input=image, class_dim=N_CLASS)

# 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[14]:


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

place = fluid.CUDAPlace(0) # place = fluid.CPUPlace()
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[15]:


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[16]:


from tqdm import tqdm

EPOCH_NUM=20  # 调参 训练轮数

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,"trainning
cost","trainning acc")


# In[17]:


import os
import wave
import librosa
import numpy as np
from tqdm import tqdm
import pickle as pkl
import matplotlib.pyplot as plt

DATA_DIR = './test'

file_glob = []


def track_features(y, sr):

    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=33)
    feature_0 = mfcc.T

    return feature_0


def get_wave_norm(file):
    with wave.open(file, 'rb') as f:
        params = f.getparams()
        nchannels, sampwidth, framerate, nframes = params[:4]
        data = f.readframes(nframes)
    data = np.fromstring(data, dtype=np.int16)
    # data = data * 1.0 / max(abs(data))
    return data, framerate


seg = 250000

data_f = {}

for file in tqdm(os.listdir(DATA_DIR)):
    data_x = []
    raw, sr = get_wave_norm(os.path.join(DATA_DIR, file))
    length = raw.shape[0]
    for i in range((length//seg)*2+1):
        start = i * int(seg // 2)
        end = start + seg
```

```
        if end > length:
            end = length
        x = np.zeros(seg)
        x[start-end:] = raw[start:end]
        r = track_features(x, sr)
        data_x.append(r)
    data_f[file] = data_x


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

for key, value in data_f.items():
    print(key, len(value))


# In[32]:



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

DATA_DIR = './test'


file_glob = []

files = os.listdir(DATA_DIR)
for pt in files:
    file_pt = os.path.join(DATA_DIR, pt)
    file_glob.append(file_pt)

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


data = {}
for file 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)
    # y = np.zeros(N_CLASS)
    # y[lbl] = 1
    data[file] = feature


with open('./data_test2.pkl', 'wb') as f:
    pkl.dump(data, f)

del data                                    # 删除data


# In[33]:


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


# In[34]:
```

```
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_test2.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)

        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]

        result['id'].append(key)
        result['label'].append(pred)

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