配置、编译和安装该项目的说明信息

（安装过程）

1. **win10安装anaconda3，新建环境**。

**一、下载和安装**

1. 下载 Anaconda

到 清华开源软件镜像站 下载 Anaconda 安装包。

注：（1）什么是镜像站？

就是把另一个站点上服务器存储的东西全部复制到镜像站。清华大学、中国科学技术大学、阿里等都提供了开源软件镜像站，下载速度快。

（2）安装包名字后缀为 ‘-x86\_64’：适用于 Windows 64 位系统。后缀为 ‘-x86’: 适用于 Windows 32 位系统。不仅是 Anaconda，很多软件的安装包都是如此命名。

2. 安装时注意事项

（1）都选默认选项就行。例如，不要勾选“Add Anaconda to the system PATH environment variable”。

（2）如果自定义 Anaconda 的安装路径，这个文件夹的名字不要有空格或 Unicode 字符，例如：文件夹名 ’program files’ 有空格，不能用做安装文件夹。同时，留意一下安装路径，下一步要用到。

3. 安装完成后，手动配置环境变量

为什么要配置环境变量？

如果不配置，在命令行界面输入 conda 时（注：命令行的打开步骤：1.按 【Win】+【R】 键，或点击“开始”菜单旁边的搜索框；2.然后输入 cmd ），会显示：

'conda'不是内部或外部命令，也不是可运行的程序或批处理文件。

1

这种情况下，无法在命令行界面使用 conda 命令。怎么配置呢？

右键点击【此电脑】> 属性，打开【控制面板】（有的系统可能没有这一步，可跳过）> 高级系统设置 > 【环境变量】，选择【系统变量】中的 Path，添加路径。比如，我的 Anaconda 安装在 D:\anaconda，就添加以下几项：

D:\anaconda

D:\anaconda\Scripts

D:\anaconda\Library\bin

D:\anaconda\Library\mingw-w64\bin

你只要把上述路径中的 D:\anaconda 换成你的 Anaconda 安装路径即可。

**二、用命令行打开 Anaconda**

1. 快速打开 Anaconda

（1）首先要打开命令行界面：

方法1：“开始”菜单 > Anaconda3 (64-bit) > Anaconda3 Prompt 或 Anaconda3 Powershell Prompt 。

方法2：用 cmd (就是上文提到的命令行界面) 或 Windows Powershell 。

可能遇到的问题：

情况 1 ：打开 Anaconda3 Powershell Prompt 时，出现提示信息：“conda : 无法将“…\conda.exe”项识别为 cmdlet、函数、脚本文件或可运行程序的名称。请检查名称的拼写，如果包括路径，请确保路径正确，然后再试一次。”

原因：提示信息中提到 “…\ WindowsPowerShell\profile.ps1”，到这个文件夹中，打开 profile.ps1，会发现里面关联的 conda.exe 用的是以前安装 Anaconda 时的文件路径，因此无法正确关联到 conda.exe。

解决方法：修改 profile.ps1 中 conda.exe 的文件路径，改为本次安装的 Anaconda 所在文件夹。再次打开 Anaconda3 Powershell Prompt，正常显示。

情况 2 ：打开 Windows Powershell 时，出现提示信息：“无法加载文件 \…\WindowsPowerShell\profile.ps1，因为在此系统上禁止运行脚本。”

原因：这是因为 Windows 的执行策略（ExecutionPolicy）设置。为了防止 Powershell 运行未知脚本（带来计算机被远程控制的风险），默认设置为 Restricted，可在 Powershell 中输入 Get-ExecutionPolicy 查看。

解决方法：以管理员身份运行 Windows Powershell，然后把执行策略改为 RemoteSigned ，这样网络上带证书的脚本可运行，本地脚本也可以。这里有一个我个人的小建议：初学者还是不要随便修改执行策略，以免给电脑带来风险，反正 cmd 可以正常使用，作用是相同的，用 cmd 命令行即可。

（2）按上述任何一种方法打开命令行界面后：

可以直接在命令行运行语句：输入 python 可进入交互式界面，退出方式：输入exit() 或按【Ctrl】+【Z】键。或者输入 ipython，进入类似 Jupyter Notebook 的交互式界面，退出方式：输入 exit()。

或者运行 python 文件，输入 python filename.py。

或者运行 conda 命令。conda 既可以管理安装包，这方面它与 pip 类似；又可以管理虚拟环境，这方面它与 virtualenv 类似。

Powershell Prompt 与 Prompt 有什么不同吗？

（1）运行 Python 文件和 conda 命令，如：conda list，两者相同。

（2）Powershell Prompt 多一个 ‘PS’ 标识，这是 Powershell 的缩写。 Powershell Prompt 类似 linux 的命令行，可以运行一些 linux 常用命令，如： ls, pwd 等，而 Prompt 不可以。不过两者都可以使用 cd 变换文件夹目录。

2. 在特定文件夹中打开 Anaconda

打开 Anaconda Prompt 默认就是 C:\Users\user\_name，每次都需要切换文件路径到 Python 文件所在文件夹中（具体步骤可以看我之前的文章 Python入门：安装 Python 软件，Anaconda 中的例子），有没有办法能快速到达？

有。

方法一：如果你把 Python 文件都放在一个文件夹中，可以把打开时的初始文件路径设置为这个常用文件夹。

在“开始”菜单中，右键选择 Anaconda Prompt > 更多 > 打开文件位置。然后，在打开的文件夹中，再次右键点击 Anaconda Prompt，在菜单中选择【属性】，更改其中的【起始位置】栏，把常用文件夹的路径（在文件夹上方的地址栏可以看到，如下图所示）复制过去即可。下次再打开 Anaconda Prompt，文件路径就变成了这个常用文件夹。

方法二：打开 Python 文件所在文件夹，点击上方地址栏（如上图所示），输入 cmd 或 powershell，在当前目录运行命令行。

如果你安装了Git，在文件夹右键菜单中点击【Git Bash Here】 也可以在当前目录打开命令行界面。

<https://blog.csdn.net/applebear1123/article/details/124459816>

**三、创建环境**

1-创建环境

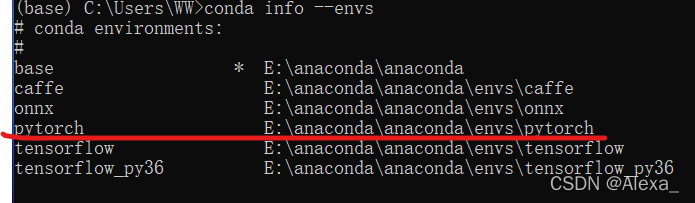
（默认你已经安装anaconda的基础上，以下操作均在anaconda prompt窗口进行命令行操作。）

创建环境，在这里创建一个pytorch版本的环境。使用python==3.7.4，取名为pytorch.

conda create -n pytorch python==3.7.4

查看是否创建成功。

conda info --envs

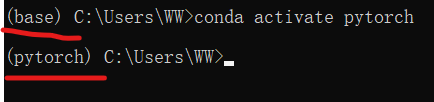


2-激活环境

下面激活环境，从base环境切换到pytorch环境中。

conda activate pytorch

如下图所示，可以看到切换成功。如果想要安装包的话，在这个环境下进行conda install 即可。



3-删除环境

有的时候为了安装某个适配的库需要好多个环境，看起来也比较麻烦，所以需要删除一些不需要的环境。

conda remove -n pytorch --all

4-退出环境

conda deactivate

或者

conda activate base

原文链接：https://blog.csdn.net/Alexa\_/article/details/123966189

1. **在新建环境里，添加镜像源。**

1.打开anaconda的prompt

如果不知道自己之前是否安装过多少镜像，可以用

conda config --show channels

显示镜像源，如果是新安装Anaconda则默认使用国外镜像源，它会显示

channels:

- defaults

如果想要删除重置，则执行

conda config --remove-key channels #删除所有的镜像源，恢复到默认

or

conda config --remove channels [urls] #删除指定的镜像源

2.在命令行下依次输入下面两行命令，配置国内镜像源：

conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/

conda config --set show\_channel\_urls yes

3.再输入以下命令验证一下是不是配置成功

conda config --show channels

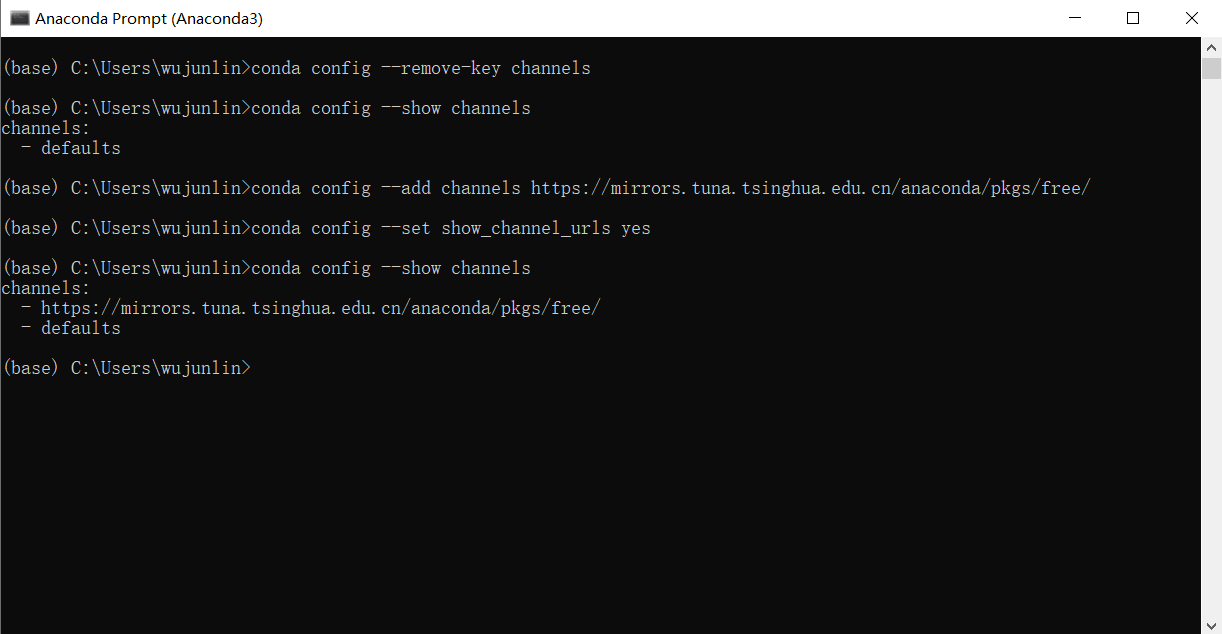
显示以下输出内容则设置成功

channels:

- https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/

- defaults

4.配置国内镜像源全部命令过程如下图所示



建议下面也全部添加，虽然有些可能用不到，但是实际包含比较冷门的包

conda config --add channels http://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/

conda config --add channels http://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/fastai/

conda config --set show\_channel\_urls yes

5.附，国内常用的Anaconda镜像：

http://mirrors.aliyun.com/pypi/simple/ #阿里

https://pypi.tuna.tsinghua.edu.cn/simple/ #清华

http://pypi.douban.com/ #豆瓣

http://pypi.mirrors.ustc.edu.cn/ #中国科学技术大学

https://blog.csdn.net/weixin\_49779629/article/details/124053811

1. **Win10anaconda安装TensorFlow-GPU pytorch GPu,比如添加第三方依赖库。**

1.打开Anaconda Prompt

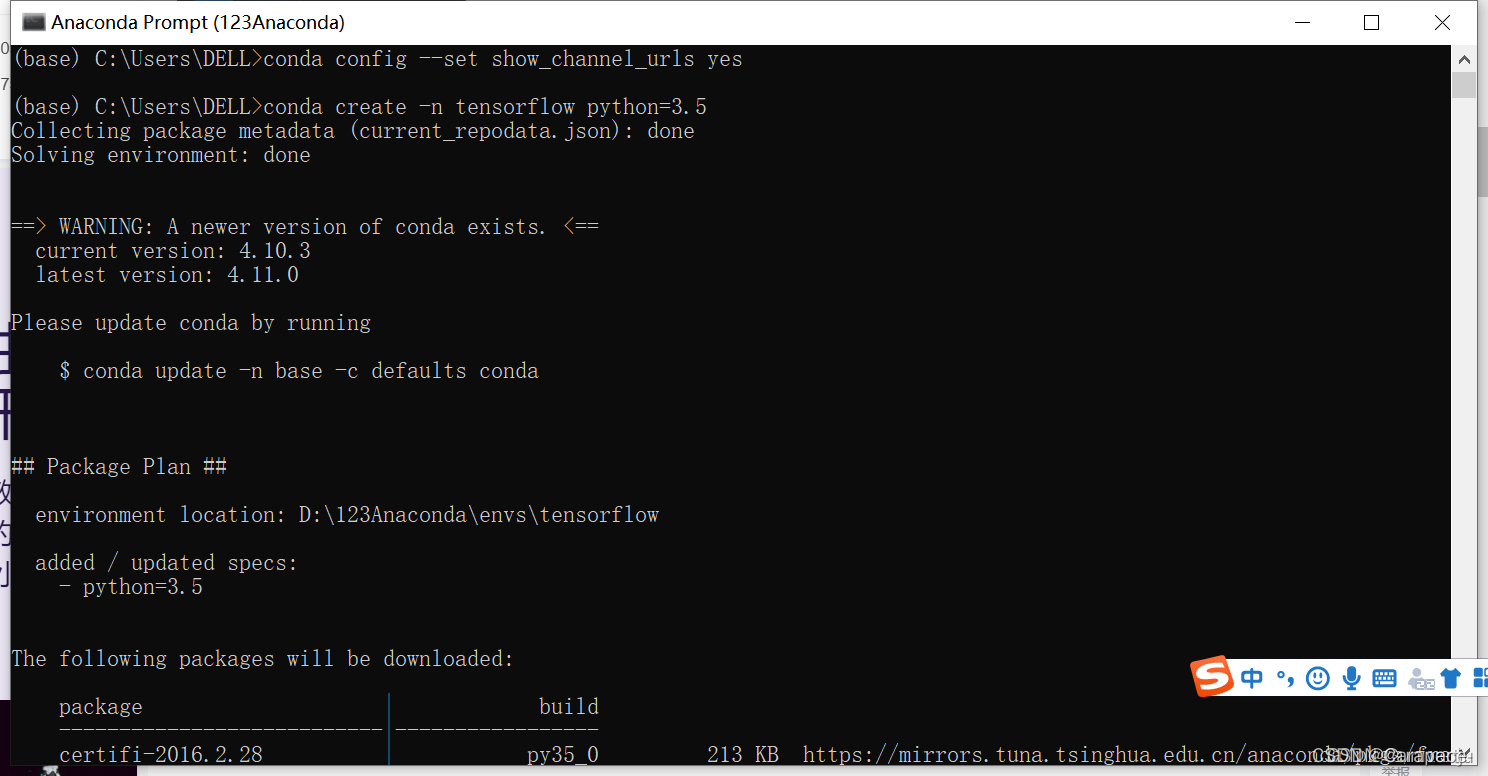
2.输入下面两行命令，打开清华镜像官网

conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/

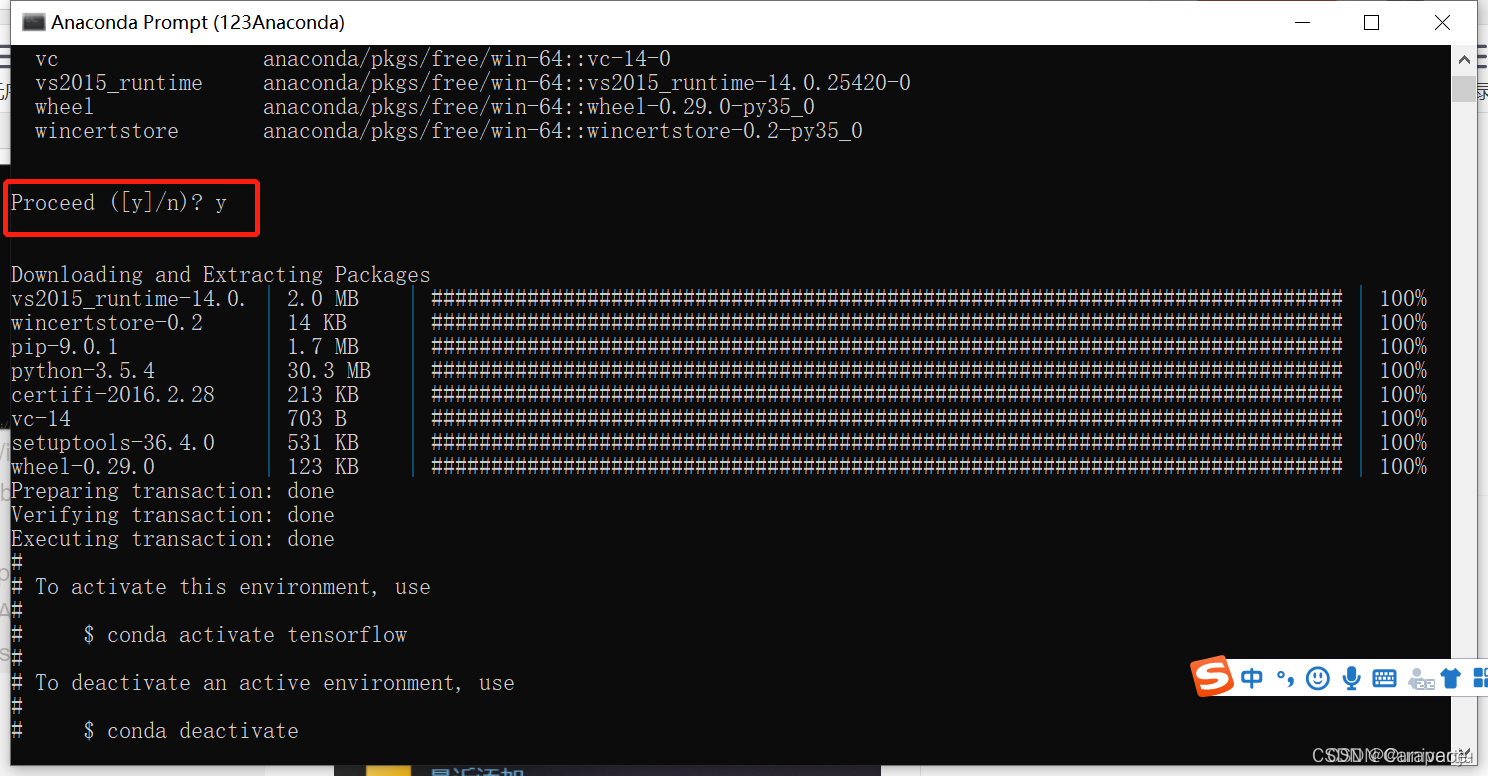
conda config --set show\_channel\_urls yes

3.输入下面命令，利用Anaconda创建一个python3.9的环境，环境名称为tensorflow（这里python版本根据自己的情况如下图）

conda create -n tensorflow python=3.9

如图所示：

4.输入y

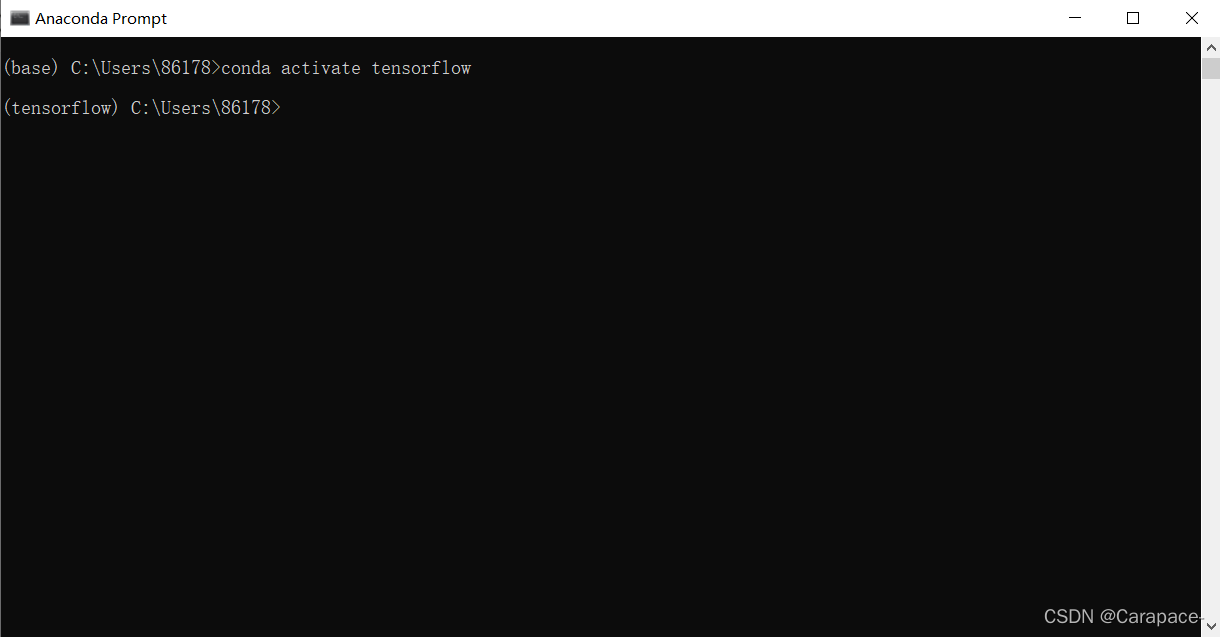


5.打开Anaconda Navigator，点击Environments，可以看到名称为tensorflow的环境已经创建好

6.上一步最后我们可以看到，启动Tensorflow，使用下面的命令行：

conda activate tensorflow

关闭tensorflow环境，使用下面的命令行：



7.我们在这里输入下面的命令，进入tensorflow环境

conda activate tensorflow

8.输入下面的命令行，安装cpu版本的tensorflow

conda install tensorflow

此时我们再次查看Anaconda Navigator中的Environments，可以发现tensorflow已经安装好了

1. **验证是否安装成功**。

https://blog.csdn.net/m0\_46299185/article/details/124280787

1. **运行代码。**

**程序主要源代码load\_and\_process.pyimport pandas as pd**

**import cv2**

**import numpy as np**

**from skimage.feature import local\_binary\_pattern**

**dataset\_path = 'fer2013/fer2013\_clean2.csv'**

**image\_size = (48, 48)**

**# settings for LBP**

**radius = 1 # LBP算法中范围半径的取值**

**n\_points = 8 \* radius # 领域像素点数**

**# 载入数据**

**def load\_fer2013():**

**data = pd.read\_csv(dataset\_path)**

**pixels = data['pixels'].tolist() # -转为列表类型**

**width, height = 48, 48**

**faces = [] # -创建faces列表**

**for pixel\_sequence in pixels:**

**face = [int(pixel) for pixel in pixel\_sequence.split(' ')]**

**face = np.asarray(face).reshape(width, height) # -将pixels项转为48x48的二维矩阵**

**face = cv2.resize(face.astype('uint8'), image\_size) # -将其中的数据转化为0-255之间的整数，并调整大小为64x64**

**# face = local\_binary\_pattern(face, n\_points, radius)**

**faces.append(face.astype('float32')) # -转化为float32类型**

**faces = np.asarray(faces)**

**faces = np.expand\_dims(faces, -1) # -在数组的最后维度增加一个维度**

**emotions = pd.get\_dummies(data['emotion']).values # -独热编码，将分类变量转换为虚拟/指标变量**

**return faces, emotions**

**# 将数据归一化**

**def preprocess\_input(x, v2=True):**

**x = x.astype('float32')**

**x = x / 255.0**

**if v2:**

**x = x - 0.5**

**x = x \* 2.0**

**return xcnn.pyfrom keras.layers import Activation, Convolution2D, Dropout, Conv2D, LocallyConnected2D**

**from keras.layers import AveragePooling2D, BatchNormalization**

**from keras.layers import GlobalAveragePooling2D**

**from keras.models import Sequential**

**from keras.layers import Flatten**

**from keras.models import Model**

**from keras.layers import Input**

**from keras.layers import MaxPooling2D**

**from keras.layers import SeparableConv2D**

**from keras import layers**

**from keras.regularizers import l2def mini\_XCEPTION(input\_shape, num\_classes, l2\_regularization=0.01):**

**regularization = l2(l2\_regularization)**

**# base**

**img\_input = Input(input\_shape)**

**x = Conv2D(8, (3, 3), strides=(1, 1), kernel\_regularizer=regularization,**

**use\_bias=False)(img\_input)**

**x = BatchNormalization()(x)**

**x = Activation('relu')(x)**

**x = Conv2D(8, (3, 3), strides=(1, 1), kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = Activation('relu')(x)**

**# module 1**

**residual = Conv2D(16, (1, 1), strides=(2, 2),**

**padding='same', use\_bias=False)(x)**

**residual = BatchNormalization()(residual)**

**x = SeparableConv2D(16, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = Activation('relu')(x)**

**x = SeparableConv2D(16, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)**

**x = layers.add([x, residual])**

**# module 2**

**residual = Conv2D(32, (1, 1), strides=(2, 2),**

**padding='same', use\_bias=False)(x)**

**residual = BatchNormalization()(residual)**

**x = SeparableConv2D(32, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = Activation('relu')(x)**

**x = SeparableConv2D(32, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)**

**x = layers.add([x, residual])**

**# module 3**

**residual = Conv2D(64, (1, 1), strides=(2, 2),**

**padding='same', use\_bias=False)(x)**

**residual = BatchNormalization()(residual)**

**x = SeparableConv2D(64, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = Activation('relu')(x)**

**x = SeparableConv2D(64, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)**

**x = layers.add([x, residual])**

**# module 4**

**residual = Conv2D(128, (1, 1), strides=(2, 2),**

**padding='same', use\_bias=False)(x)**

**residual = BatchNormalization()(residual)**

**x = SeparableConv2D(128, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = Activation('relu')(x)**

**x = SeparableConv2D(128, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)**

**x = layers.add([x, residual])**

**x = Conv2D(num\_classes, (3, 3),**

**# kernel\_regularizer=regularization,**

**padding='same')(x)**

**x = GlobalAveragePooling2D()(x)**

**output = Activation('softmax', name='predictions')(x)**

**model = Model(img\_input, output)**

**return modeldef my\_XCEPTION(input\_shape, num\_classes, l2\_regularization=0.01):**

**regularization = l2(l2\_regularization)**

**# entry**

**img\_input = Input(input\_shape)**

**x = Conv2D(8, (3, 3), strides=(1, 1), kernel\_regularizer=regularization,**

**padding='same', use\_bias=False)(img\_input)**

**x = BatchNormalization()(x)**

**x = Activation('relu')(x)**

**x = Conv2D(16, (3, 3), strides=(1, 1), kernel\_regularizer=regularization,**

**padding='same', use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = Activation('relu')(x)**

**# module 1**

**residual = Conv2D(32, (1, 1), strides=(2, 2),**

**padding='same', use\_bias=False)(x)**

**residual = BatchNormalization()(residual)**

**x = SeparableConv2D(32, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = Activation('relu')(x)**

**x = SeparableConv2D(32, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)**

**x = layers.add([x, residual])**

**# module 2**

**residual = Conv2D(64, (1, 1), strides=(2, 2),**

**padding='same', use\_bias=False)(x)**

**residual = BatchNormalization()(residual)**

**x = Activation('relu')(x)**

**x = SeparableConv2D(64, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = Activation('relu')(x)**

**x = SeparableConv2D(64, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)**

**x = layers.add([x, residual])**

**# module 3**

**residual = Conv2D(128, (1, 1), strides=(2, 2),**

**padding='same', use\_bias=False)(x)**

**residual = BatchNormalization()(residual)**

**x = Activation('relu')(x)**

**x = SeparableConv2D(128, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = Activation('relu')(x)**

**x = SeparableConv2D(128, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)**

**x = layers.add([x, residual])**

**# module 4**

**residual = Conv2D(256, (1, 1), strides=(2, 2),**

**padding='same', use\_bias=False)(x)**

**residual = BatchNormalization()(residual)**

**x = Activation('relu')(x)**

**x = SeparableConv2D(256, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = Activation('relu')(x)**

**x = SeparableConv2D(256, (3, 3), padding='same',**

**kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = MaxPooling2D((2, 2), strides=(2, 2), padding='same')(x)**

**x = layers.add([x, residual])**

**x = LocallyConnected2D(num\_classes, (2, 2), padding='valid',**

**# kernel\_regularizer=regularization,**

**use\_bias=False)(x)**

**x = BatchNormalization()(x)**

**x = Activation('relu')(x)**

**x = GlobalAveragePooling2D()(x)**

**output = Activation('softmax', name='predictions')(x)**

**model = Model(img\_input, output)**

**return modeltrain\_emotion\_classifier.py"""**

**Description: 训练人脸表情识别程序**

**"""**

**from keras.callbacks import CSVLogger, ModelCheckpoint, EarlyStopping**

**from keras.callbacks import ReduceLROnPlateau**

**from keras.preprocessing.image import ImageDataGenerator**

**from load\_and\_process import load\_fer2013**

**from load\_and\_process import preprocess\_input**

**from models.cnn import mini\_XCEPTION**

**from models.cnn import tiny\_XCEPTION**

**from models.cnn import my\_XCEPTION**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import confusion\_matrix**

**import itertools**

**import numpy as np**

**import matplotlib.pyplot as plt**

**# 参数**

**batch\_size = 32**

**num\_epochs = 10000**

**input\_shape = (48, 48, 1)**

**validation\_split = .2**

**verbose = 1**

**num\_classes = 7**

**patience = 40**

**base\_path = 'models/'**

**# 构建模型**

**# model = my\_XCEPTION(input\_shape, num\_classes)**

**model = mini\_XCEPTION(input\_shape, num\_classes)**

**# model = tiny\_XCEPTION(input\_shape, num\_classes)**

**model.compile(optimizer='adam', # 优化器采用adam**

**loss='categorical\_crossentropy', # 交叉熵损失函数**

**metrics=['accuracy'])**

**model.summary()**

**# 定义回调函数 Callbacks 用于训练过程**

**log\_file\_path = base\_path + '48.28mini\_emotion\_training.log'**

**csv\_logger = CSVLogger(log\_file\_path, append=True)**

**early\_stop = EarlyStopping('val\_loss', patience=patience) # patience次不变则停止训练**

**reduce\_lr = ReduceLROnPlateau('val\_loss', factor=0.1,**

**patience=int(patience/4),**

**verbose=1)**

**# 模型位置及命名**

**trained\_models\_path = base\_path + '48.28\_mini\_XCEPTION'**

**model\_names = trained\_models\_path + '.{epoch:02d}-{val\_acc:.2f}.hdf5'**

**# 定义模型权重位置、命名等**

**model\_checkpoint = ModelCheckpoint(model\_names,**

**'val\_loss', verbose=1,**

**save\_best\_only=True)**

**callbacks = [model\_checkpoint, csv\_logger, early\_stop, reduce\_lr]**

**# 载入数据集**

**faces, emotions = load\_fer2013()**

**faces = preprocess\_input(faces)**

**num\_samples, num\_classes = emotions.shape # -返回emotions维度大小的元组**

**# 划分训练、测试集**

**xtrain, xtest, ytrain, ytest = train\_test\_split(faces, emotions, test\_size=0.2, shuffle=True) # -20%的数据用于测试**

**# 图片产生器，在批量中对数据进行增强，扩充数据集大小**

**data\_generator = ImageDataGenerator(**

**featurewise\_center=False, # 布尔。将输入平均值设置为 0，按特征进行。**

**featurewise\_std\_normalization=False, # 布尔。按数据集的当前方向划分输入。**

**rotation\_range=10, # 随机旋转的度范围。**

**width\_shift\_range=0.1,**

**height\_shift\_range=0.1,**

**zoom\_range=.1,**

**horizontal\_flip=True) # 布尔。随机水平翻转输入。**

**# 利用数据增强进行训练**

**model.fit\_generator(data\_generator.flow(xtrain, ytrain, batch\_size),**

**steps\_per\_epoch=len(xtrain) / batch\_size,**

**epochs=num\_epochs,**

**verbose=1, callbacks=callbacks,**

**validation\_data=(xtest, ytest))**

**# 显示预测的混淆矩阵**

**# 计算预测**

**predictions = model.predict(xtest, batch\_size=32)**

**y\_pred = [np.argmax(probas) for probas in predictions]**

**y\_true = ytest.argmax(axis=-1)**

**class\_names = ('anger', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral')**

**def plot\_confusion\_matrix(cm, classes, title="Confusion matrix", cmap=plt.cm.Blues):**

**cm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]**

**# plt.figure(figsize=(10, 10))**

**plt.imshow(cm, interpolation="nearest", cmap=cmap)**

**plt.title(title)**

**plt.colorbar()**

**tick\_marks = np.arange(len(classes))**

**plt.xticks(tick\_marks, classes, rotation=45)**

**plt.yticks(tick\_marks, classes)**

**fmt = ".2f"**

**thresh = cm.max() / 2.**

**for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):**

**plt.text(j, i, format(cm[i, j], fmt),**

**horizontalalignment="center",**

**color="white" if cm[i, j] > thresh else "black")**

**plt.ylabel("True label")**

**plt.xlabel("Predicted label")**

**plt.tight\_layout()**

**# 计算混淆矩阵**

**cnf\_matrix = confusion\_matrix(y\_true, y\_pred)**

**np.set\_printoptions(precision=2)**

**plt.figure()**

**plot\_confusion\_matrix(cnf\_matrix, classes=class\_names, title="Normalized confusion matrix")**

**plt.show()facial\_expression\_recognition.py#! /usr/bin/env python**

**# -\*- encoding: UTF-8 -\*-**

**""" nao机器人表情识别程序 """**

**import qi**

**import argparse**

**import sys**

**import time**

**from PIL import Image**

**import socket**

**def main(session):**

**"""**

**此程序从机器人获取图片后传到服务端 nao\_Fer\_server/Fer\_server\_2.py ，识别表情后将结果传回此程序，机器人再做出反应**

**"""**

**# 唤醒机器人**

**# Get the service ALMotion.**

**motion\_service = session.service("ALMotion")**

**motion\_service.wakeUp()**

**# 机器人识别并跟踪人脸**

**face\_detection = session.service("ALFaceDetection")**

**memory = session.service("ALMemory")**

**tts = session.service("ALTextToSpeech")**

**# Get the services ALBasicAwareness**

**ba\_service = session.service("ALBasicAwareness")**

**ba\_service.setEnabled(True)**

**ba\_service.setStimulusDetectionEnabled("People", True)**

**ba\_service.setStimulusDetectionEnabled("Touch", False)**

**ba\_service.setStimulusDetectionEnabled("Sound", False)**

**ba\_service.setStimulusDetectionEnabled("Movement", False)**

**ba\_service.setTrackingMode("Head")**

**ba\_service.setEngagementMode("FullyEngaged")**

**# Subscribe to the ALFaceDetection proxy**

**# This means that the module will write in ALMemory with**

**# the given period below**

**period = 500**

**face\_detection.subscribe("Test\_Face", period, 0.0)**

**try:**

**while True:**

**time.sleep(5)**

**memValue = "FaceDetected"**

**val = memory.getData(memValue)**

**if val and isinstance(val, list) and len(val) == 5:**

**faceInfoArray = val[1]**

**print faceInfoArray**

**if faceInfoArray and len(faceInfoArray) == 2:**

**tts.say("你好")**

**time.sleep(1)**

**recognition()**

**# motion\_service.setIdlePostureEnabled("Body", True)**

**except KeyboardInterrupt:**

**face\_detection.unsubscribe("Test\_Face")**

**ba\_service.setEnabled(False)**

**motion\_service.rest()**

**def recognition():**

**# Get the service ALVideoDevice.**

**video\_service = session.service("ALVideoDevice")**

**resolution = 2 # 2:VGA,640x480像素 3:1280x960px 4:2560x1920px**

**colorSpace = 11 # RGB**

**videoClient = video\_service.subscribe("python\_client", resolution, colorSpace, 5)**

**t0 = time.time()**

**# Get a camera image.**

**# image[6] contains the image data passed as an array of ASCII chars.**

**naoImage = video\_service.getImageRemote(videoClient)**

**t1 = time.time()**

**# Time the image transfer.**

**print "acquisition delay ", t1 - t0**

**video\_service.unsubscribe(videoClient)**

**# Now we work with the image returned and save it as a PNG using ImageDraw**

**# package.**

**# Get the image size and pixel array.**

**imageWidth = naoImage[0]**

**imageHeight = naoImage[1]**

**array = naoImage[6]**

**image\_string = str(bytearray(array))**

**# Create a PIL Image from our pixel array.**

**im = Image.frombytes("RGB", (imageWidth, imageHeight), image\_string)**

**# Save the image.**

**im.save("camImage.png", "PNG")**

**# im.show()**

**print '照片已生成'**

**s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) # 创建 socket 对象**

**host = '192.168.1.9' # 获取本地主机名**

**port = 12345 # 设置端口号**

**s.connect((host, port))**

**s.send("照片已生成")**

**animSpeech = session.service("ALAnimatedSpeech")**

**emotion = s.recv(1024).decode()**

**if emotion == 'no face':**

**animSpeech.say(" ^start(animations/Stand/Exclamation/NAO/Center\_Neutral\_EXC\_04) 对不起，我没能检测到人脸"**

**" ^wait(animations/Stand/Exclamation/NAO/Center\_Neutral\_EXC\_04)")**

**elif emotion == 'angry':**

**animSpeech.say("^start(animations/Stand/Gestures/Shocked\_1) 你看起来很生气 ^wait(animations/Stand/Gestures/Shocked\_1)")**

**animSpeech.say("^start(animations/Stand/Gestures/CalmDown\_3) 请冷静下来 ^wait(animations/Stand/Gestures/CalmDown\_3)")**

**elif emotion == 'disgust':**

**animSpeech.say(" ^start(animations/Stand/Question/NAO/Center\_Neutral\_QUE\_01) 你的表情看起来是厌恶，我说的对吗？"**

**" ^wait(animations/Stand/Question/NAO/Center\_Neutral\_QUE\_01) ")**

**elif emotion == 'fear':**

**animSpeech.say(" ^start(animations/Stand/Question/NAO/Center\_Neutral\_QUE\_03) 你看起来很恐惧，怎么了？"**

**" ^wait(animations/Stand/Question/NAO/Center\_Neutral\_QUE\_03)")**

**elif emotion == 'happy':**

**animSpeech.say(" ^start(animations/Stand/Exclamation/NAO/Center\_Neutral\_EXC\_01) 你看起来很开心"**

**" ^wait(animations/Stand/Exclamation/NAO/Center\_Neutral\_EXC\_01) ")**

**animSpeech.say(" ^start(animations/Stand/Gestures/Happy\_1) 我也很开心 ^wait(animations/Stand/Gestures/Happy\_1) ")**

**elif emotion == 'sad':**

**animSpeech.say(" ^start(animations/Stand/Gestures/Sad\_2) 别伤心 ^wait(animations/Stand/Gestures/Sad\_2) ")**

**animSpeech.say(" ^start(animations/Stand/Gestures/AirJuggle\_1) 我可以让你高兴起来！ "**

**" ^wait(animations/Stand/Gestures/AirJuggle\_1) ")**

**elif emotion == 'surprise':**

**animSpeech.say(" ^start(animations/Stand/Gestures/Surprised\_1) 为什么你看起来这么惊讶？发生了什么？"**

**" ^wait(animations/Stand/Gestures/Surprised\_1) ")**

**elif emotion == 'neutral':**

**animSpeech.say(" ^start(animations/Stand/Affirmation/NAO/Center\_Neutral\_AFF\_04) 你的表情是中立的 "**

**" ^wait(animations/Stand/Affirmation/NAO/Center\_Neutral\_AFF\_04) ")**

**else:**

**animSpeech.say(" ^start(animations/Stand/Exclamation/NAO/Center\_Neutral\_EXC\_04) 对不起，我没能识别你的表情"**

**" ^wait(animations/Stand/Exclamation/NAO/Center\_Neutral\_EXC\_04)")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**parser = argparse.ArgumentParser()**

**parser.add\_argument("--ip", type=str, default="192.168.1.11",**

**help="Robot IP address. On robot or Local Naoqi: use '192.168.1.11'.")**

**parser.add\_argument("--port", type=int, default=9559,**

**help="Naoqi port number")**

**args = parser.parse\_args()**

**session = qi.Session()**

**try:**

**session.connect("tcp://" + args.ip + ":" + str(args.port))**

**except RuntimeError:**

**print ("Can't connect to Naoqi at ip \"" + args.ip + "\" on port " + str(args.port) + ".\n"**

**"Please check your script arguments. Run with -h option for help.")**

**sys.exit(1)**

**main(session)Fer\_server\_2.py# -\*- coding: UTF-8 -\*-**

**import socket**

**import cv2**

**import numpy as np**

**from keras.models import load\_model**

**from keras.preprocessing.image import img\_to\_array**

**face\_detection\_path = 'models/haarcascade\_frontalface\_default.xml'**

**fer\_model\_path = 'models/48.20\_my\_XCEPTION.42-0.69.hdf5'**

**# 载入人脸检测模型**

**face\_detection = cv2.CascadeClassifier(face\_detection\_path) # 级联分类器**

**# 载入人脸表情识别模型**

**emotion\_classifier = load\_model(fer\_model\_path, compile=False)**

**# 表情类别**

**EMOTIONS = ["angry", "disgust", "fear", "happy", "sad", "surprise", "neutral"]**

**s = socket.socket() # 创建 socket 对象**

**host = '192.168.1.9' # 获取本地主机名**

**port = 12345 # 设置端口**

**s.bind((host, port)) # 绑定端口**

**s.listen(5) # 等待客户端连接**

**print('开始监听端口：')**

**while True:**

**c, addr = s.accept() # 建立客户端连接**

**print('收到{}请求'.format(addr))**

**message = c.recv(1024)**

**if message.decode() == '照片已生成':**

**img = cv2.imread('../nao\_Facial\_expression\_recognition/camImage.png')**

**# cv2.imshow("Display window", img)**

**# cv2.waitKey(0)**

**gray = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY) # 转为灰度图**

**# cv2.imshow('Display window', gray)**

**# cv2.waitKey(0)**

**cv2.imwrite('photogray.png', gray)**

**# 检测人脸**

**faces = face\_detection.detectMultiScale(gray, scaleFactor=1.1,**

**minNeighbors=5, minSize=(30, 30),**

**flags=cv2.CASCADE\_SCALE\_IMAGE)**

**preds = [] # 预测的结果**

**label = None # 预测的标签**

**(fX, fY, fW, fH) = None, None, None, None # 人脸位置**

**if len(faces) > 0:**

**# 选择检测到的ROI最大的人脸**

**faces = sorted(faces, reverse=True, key=lambda x: (x[2] - x[0]) \* (x[3] - x[1]))[0]**

**(fX, fY, fW, fH) = faces**

**# 从灰度图中提取感兴趣区域（ROI），将其大小转换为48\*48 pixels，并为通过CNN的分类器准备ROI**

**roi = gray[fY - 5:fY + fH + 5, fX - 5:fX + fW + 5]**

**roi = cv2.resize(roi, (48, 48))**

**roi = roi.astype("float") / 255.0**

**roi = img\_to\_array(roi)**

**roi = np.expand\_dims(roi, axis=0)**

**# 用模型预测各分类的概率**

**preds = emotion\_classifier.predict(roi)[0]**

**# emotion\_probability = np.max(preds) # 最大的概率**

**label = EMOTIONS[preds.argmax()] # 选取最大概率的表情类**

**print(label)**

**c.send(bytes(label, 'utf-8'))**

**else:**

**c.send(b'no face')**

（注意：版本号与代码相对应，比如，表情识别TensorFlow-GPU要求Python3.6，这时候，如果想在同意该环境里面同时有pytorch和TensorFlow，就需要安装使用Python3.6的pytorch以及相关版本的CUDA、cudnn）