



中山大學
SUN YAT-SEN UNIVERSITY

人机交互技术

3.编码作业-视觉

学 院 名 称 : 数据科学与计算机学院

专业 (班级) : 17 软件工程 1 班

学 生 姓 名 : 曾 峥

学 号 : 17343006

时 间 : 2020 年 7 月 4 日

目录

1. 实验介绍.....	3
1.1 作业内容.....	3
1.2 实验环境.....	3
1.3 项目结构.....	4
2. UI 页面.....	5
2.1 环境配置.....	5
2.2 UI 界面实现.....	5
2.3 UI 逻辑实现.....	8
2.3.1 自定义槽函数.....	8
2.3.2 选择模型.....	9
2.3.3 选择图片.....	9
2.3.4 摄像头实时识别.....	11
2.4 主函数运行.....	12
3. 表情识别.....	13
3.1 实验原理.....	13
3.2 数据集介绍.....	14
3.3 mini_XCEPTION 网络.....	15
3.4 加载数据集.....	16
3.5 搭建模型并保存.....	19
3.6 数据增强.....	24
3.7 批量训练.....	25
3.8 测试.....	27
3.9 实验过程.....	29
3.10 算法评估.....	31
3.11 封装成类.....	31
4. 心得体会.....	35
4.1 Python 知识积累.....	35
4.2 PyQt5-界面程序开发.....	35
4.2.1 环境配置.....	35
4.2.2 配置时的 bug.....	36
4.2.3 Pyqt 界面 UI 与逻辑分离.....	38
4.2.4 零碎知识点.....	38
4.3 表情识别存在的问题与发展前景.....	39
4.4 改进.....	40
5. 参考文献和资料.....	41
6. 代码附录.....	42

1.实验介绍

1.1 作业内容

- 1、编程实现HCI 交互软件的各个页面，对关键代码进行说明和注释
- 2、编程实现视觉交互或语音交互的人工智能算法和程序部分，完成模型训练，并对关键代码进行说明和注释
- 3、要求：提交word 或者pdf 电子文档，并附上代码。要求图文并茂，逻辑清晰，格式规整。每个部分各50 分。
- 4、deadline：2020.7.17

1.2 实验环境

Win 10 , Pycharm.2020.1.3 community , python 3.8 , Anaconda3

库	版本	功能
Keras	2.2.4	搭建模型；数据加强；回调函数
PyQt	5.11.3	UI 界面
pyqt5-tools	5.15.0.1.7	
Pandas	0.24.2	数据加载、处理
Scikit-learn	0.21.2	划分训练集、测试集；计算混淆矩阵
Imutils	0.5.2	处理图像（如缩放）用于 UI 显示
Opencv-python	4.10.25	图像处理；载入人脸模型

打开 Pycharm 底/右端的的 Terminal 终端输入，运行程序。

```
(base) E:\zeng\HCI\Emotion Recognition>python runMain.py
Using TensorFlow backend.
libpng warning: iCCP: known incorrect sRGB profile
```

1.3 项目结构

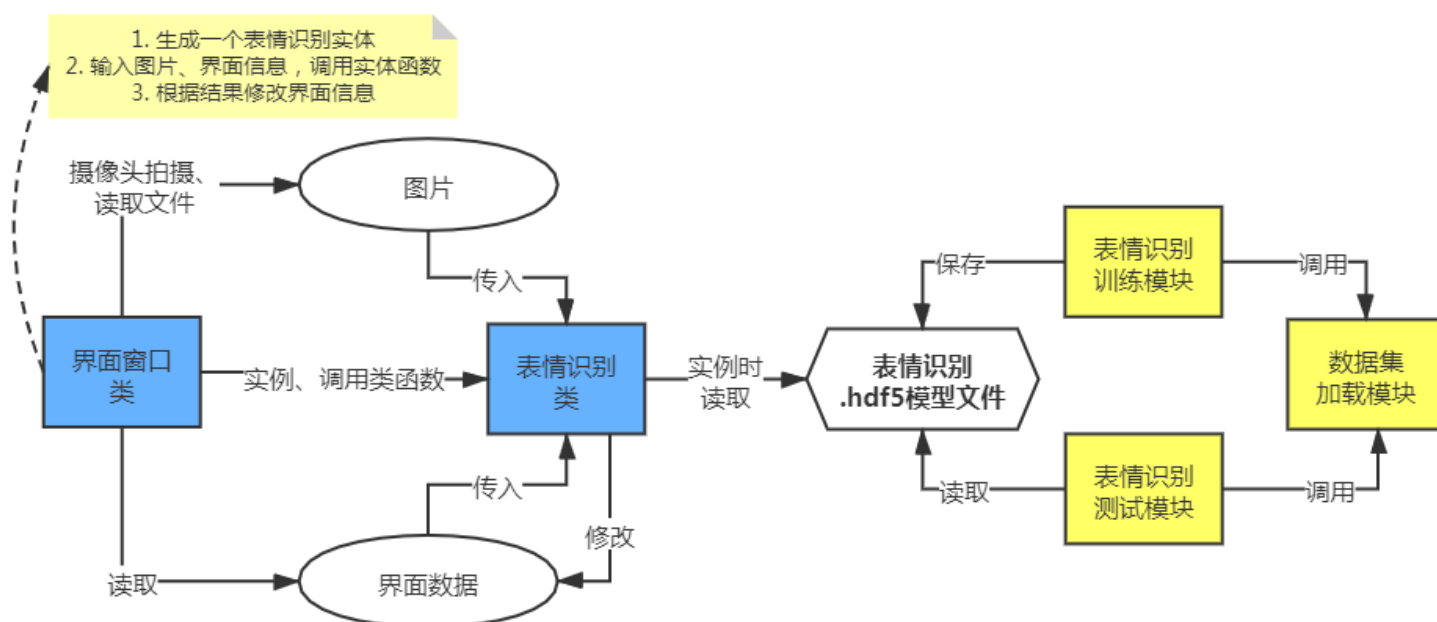
整个项目的核心代码为表情识别。

在搭建模型、训练数据之后，存储模型。

外部 UI 与用户交互，内部调用模型计算结果。

所有程序完成之后，打包成 `.exe` 文件。

整个项目的大概流程如下所示：



2. UI 页面

伪算法:

1. 建立一个 Window 类，负责软件的外观；
 1. 用 pyqt5 designer 画出一个界面
 2. 个性化一些控件的名称，后续需要在逻辑层面中修改数据，反馈给用户
 3. 建立 qrc 资源文件，负责界面的图片需要
 4. Style 表格，用于美化界面
2. 建立一个 MainWindow 类，继承 window 类，负责软件的逻辑实现；
 1. 在自定义槽函数（软件的逻辑）中，调用软件的核心算法，即表情识别；
 1. 选择表情识别需要的模型文件
 2. 开启摄像头进行实时识别
 3. 选择本地照片进行识别
 2. 得到结果后，修改界面，反馈给用户，实现人机交互，包括：
 1. 在原图中将人脸框出来，并写上识别结果
 2. 运行时间、表情识别概率信息、最终识别结果

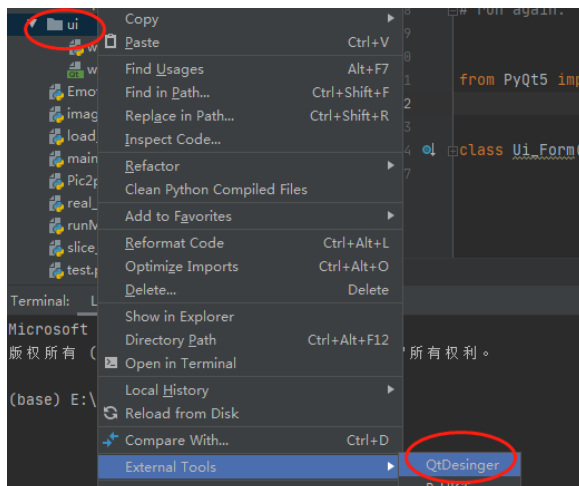
2.1 环境配置

pip install --index-url <https://pypi.douban.com/simple> pyqt5 PyQt5-tools

```
Installing collected packages: PyQt5-sip, pyqt5, python-dotenv, PyQt5-tools
Successfully installed PyQt5-sip-12.8.0 PyQt5-tools-5.15.0.1.7 pyqt5-5.15.0 python-dotenv-0.14.0
```

2.2 UI 界面实现

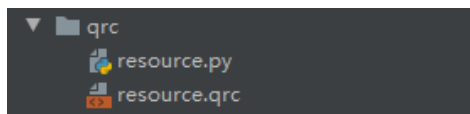
- 在 UI 文件夹中，新建一个 UI 文件，启动 pyqt5 designer → 进入界面之后 → create Widget



- .qrc 代码格式如下：

```
<!DOCTYPE RCC>
<RCC version="1.0">
<qresource>
    <file>../images/ui/background.png</file>
    .....
    <file>../images/ui/init.png</file>
</qresource>
</RCC>
```

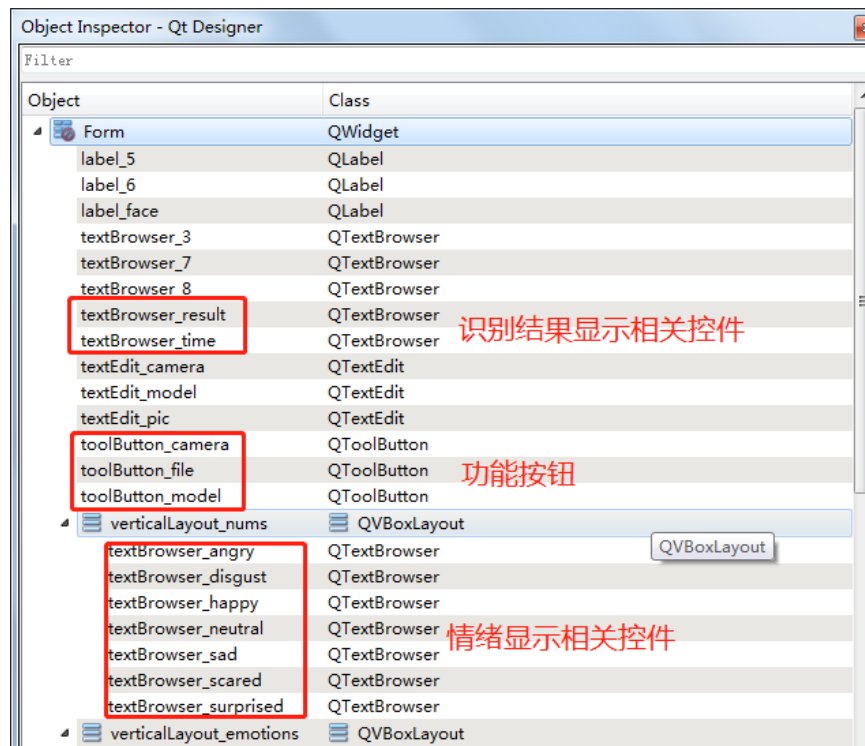
右键选中 resource.qrc → External Tools → PyRcc ，就会生成对应的 resource.py.



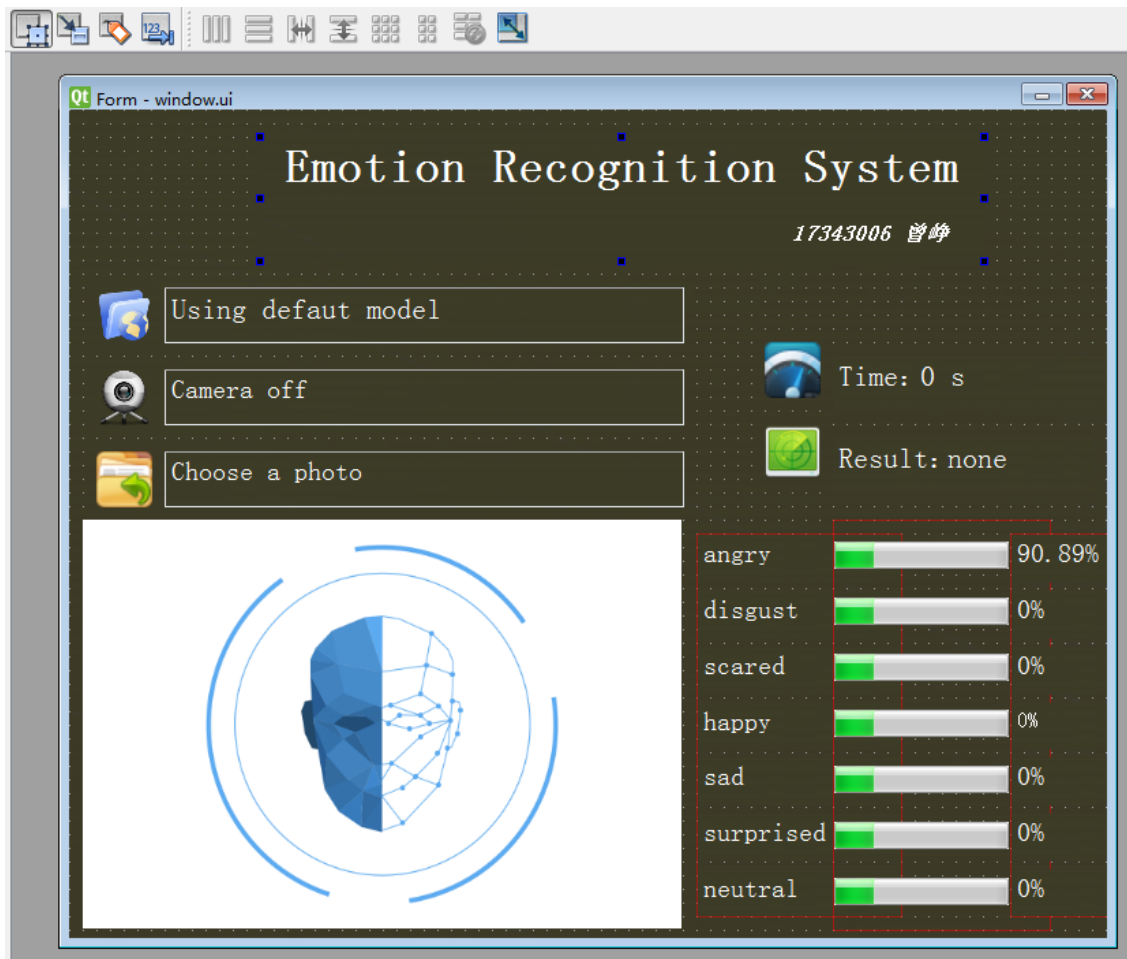
后续如果需要改变 UI 界面的图片时，引入这个模块，就可以对其进行操作。

```
import sys
sys.path.append('../') # 使用模块和这个模块不在同一个目录下
import qrc.resource # dirName.fileName
```

- 对一些控件进行命名

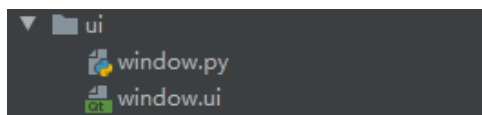


- 最后，绘制出一个 UI 界面如下：



- 将这个 window.ui 文件转化为.py 文件供逻辑层面使用。

右击 window.ui → External Tools → PyUIC5 :



- 生成的 window.py 结构如下：

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(761, 606)
        Form.setStyleSheet("background-image: url(../background.png); \n"
        .....
    def retranslateUi(self, Form):
        .....
```

2.3 UI 逻辑实现

要响应事件操作，往往会将相应的槽函数写在 ui 的 py 文件中，这样，界面和逻辑开发就混合在一起了，每一次的 ui 更新都会伴随着转换后 py 文件的槽函数的添加修改，及其不方便，造成效率低下。为了将二者剥离，另外新建一个 MainWindow 类，继承上面那个类。另外可以自定义一些槽函数，与用户进行交互。

```
from ui.window import Ui_Form # ui 文件夹下的 window 模块，分离的 UI 界面

class MainWindow(QMainWindow, Ui_Form):
    def __init__(self, parent=None):
        # super(MainWindow, self).__init__(parent)
        QMainWindow.__init__(self)
        Ui_Form.__init__(self)
        self.setupUi(self)
        self.retranslateUi(self)

        self.slot_init() # 自定义槽函数集合
        //.....一些其他初始化操作
```

2.3.1 自定义槽函数

为了与用户进行交互，自定义槽函数集合如下：

```
def slot_init(self): # 定义槽函数
    self.toolButton_model.clicked.connect(self.choose_model)
    self.toolButton_camera.clicked.connect(self.button_open_camera_click)
    self.timer_camera.timeout.connect(self.show_camera)
    self.toolButton_file.clicked.connect(self.choose_pic)

def choose_model(self): # 选择训练好的模型文件
    ...

def button_open_camera_click(self): # 开启摄像头，进行实时识别
    ...

def choose_pic(self): # 选择本地图片进行识别
    ...
```

特定的控件被点击触发之后，就会调用响应的函数，比如，用户点击了 `toolButton_model` 之后，

就会调用 `def choose_model()` 函数，用户进行模型选择。

2.3.2 选择模型

这个功能是可以选择测试表情的模型，如果不选择的话，就使用默认模型。

伪算法：

1. 一些界面变化操作，比如将识别结果刷新为初始状态、关闭摄像头以及摄像头定时器；
2. 使用 `QFileDialog` 调用文件选择对话框，选择需要的模型文件；
3. 记录这个模型文件的路径，并在 UI 界面上显示选择状态；
4. 选择结束之后，界面恢复正常。

关键代码如下：

```
def choose_model(self):
    # 一些界面变化、关闭摄像头等操作
    .....
    # 调用文件选择对话框
    fileName_choose, filetype = QFileDialog.getOpenFileName(self,
                                                            "Select a file...", getcwd(), # 定位起始路径
                                                            "Model File (*.hdf5)") # 文件类型，只接受.hdf5 模型文件

    # 显示提示信息
    if fileName_choose != '':
        self.model_path = fileName_choose
        self.textEdit_model.setText(fileName_choose)
    else:
        self.textEdit_model.setText('Using default model')

    # 恢复界面
    .....
```

2.3.3 选择图片

本地选择一个图片，并调用表情识别的接口，进行识别，将得到的结果显示在 UI 界面中。

伪算法:

1. 同上, 一些界面的处理;
2. 同上, 使用文件选择对话框; 选择一张图片并读取;
3. 调用封装好的识别 API 生成一个模型对象
4. 计时并开始模型预测
5. 调用 `def showResult()` 显示结果

代码如下:

```
def choose_pic(self):
    # 界面处理
    .....
    # 使用文件选择对话框选择图片
    fileName_choose, filetype = QFileDialog.getOpenFileName(
        self, "Choose a photo...", self.path, # 起始路径
        "图片(*.jpg;*.jpeg;*.png)") # 文件类型
    self.path = fileName_choose # 保存路径
    if fileName_choose != '':
        self.textEdit_pic.setText(fileName_choose)
        self.label_face.setText('Analysing...\n\nleading')
        QtWidgets.QApplication.processEvents()
        # 生成模型对象
        self.emotion_model = Emotion_Rec(self.model_path)

        # 计时并开始模型预测
        QtWidgets.QApplication.processEvents()
        time_start = time.time()
        results, result = self.emotion_model.run(image, self.label_face)
        time_end = time.time()
        # 显示结果
        self.showResult(results, result, time_end - time_start)

    else:
        # 选择取消, 恢复界面状态
        .....
```

用到了辅助函数 `def showResult()`, 这会在 UI 中显示各个表情识别的概率以及识别结果, 代码如下:

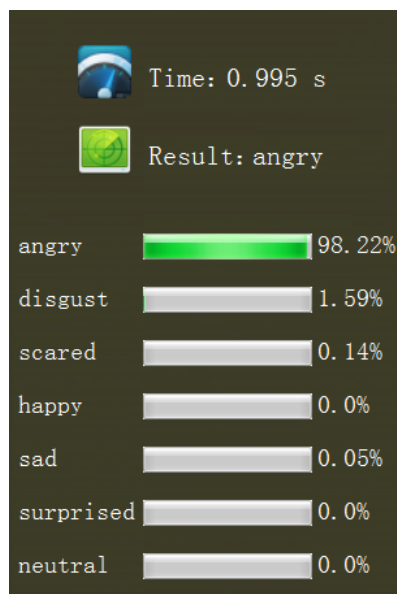
```
def showResult(self, results=None, result='none', totalTime=0.0):
    if results is None:
        for emotion in self.EMOTIONS:
            bar_widget = self.findChild(name='progressBar_' + emotion)
            bar_widget.setValue(0)
            text_widget = self.findChild(name='textBrowser_' + emotion)
            text_widget.setText(str(0) + '%')
```

```

else:
    self.textBrowser_result.setText(result)
    self.textBrowser_time.setText(str(round(totalTime, 3)) + ' s')
    for (i, (emotion, prob)) in enumerate(results):
        # widget = QProgressBar(self)
        # widget.setObjectName(emotion)
        bar_widget = self.findChild(name='progressBar_' + emotion)
        bar_widget.setValue(prob * 100)
        text_widget = self.findChild( name='textBrowser_' + emotion)
        prob = round(prob * 100, 2)
        text_widget.setText(str(prob) + '%')

```

效果如下：



2.3.4 摄像头实时识别

摄像头实时识别

伪算法：

1. 同上，一些界面变化操作
2. 开启摄像头、定时器
3. 每隔一段时间拍摄，拍摄的照片处理逻辑同上

代码如下：

```

def button_open_camera_click(self):
    if self.timer_camera.isActive() == False: # 检查定时状态
        flag = self.cap.open(self.CAM_NUM) # 检查相机状态
        if flag == False: # 相机打开失败提示

```

```

        msg = QtWidgets.QMessageBox.warning(self, u"Warning",
                                             u"请检测相机与电脑是否连接正确! ",
                                             buttons=QtWidgets.QMessageBox.Ok,
                                             defaultButton=QtWidgets.QMessageBox.Ok)

    else:
        # 准备运行识别程序
        self.textEdit_pic.setText('No photo is choosed')
        QtWidgets.QApplication.processEvents()
        self.textEdit_camera.setText('Camera on...')
        self.label_face.setText('Analysing...\n\nleading')
        # 新建对象
        self.emotion_model = Emotion_Rec(self.model_path)
        QtWidgets.QApplication.processEvents()
        # 打开定时器
        self.timer_camera.start(30)
    else:
        # 定时器未开启, 界面回复初始状态
        self.timer_camera.stop()
        self.cap.release()
        .....

```

2.4 主函数

代码如下：

```

from ui_mainWindow import MainWindow
from sys import argv, exit
from PyQt5.QtWidgets import QApplication

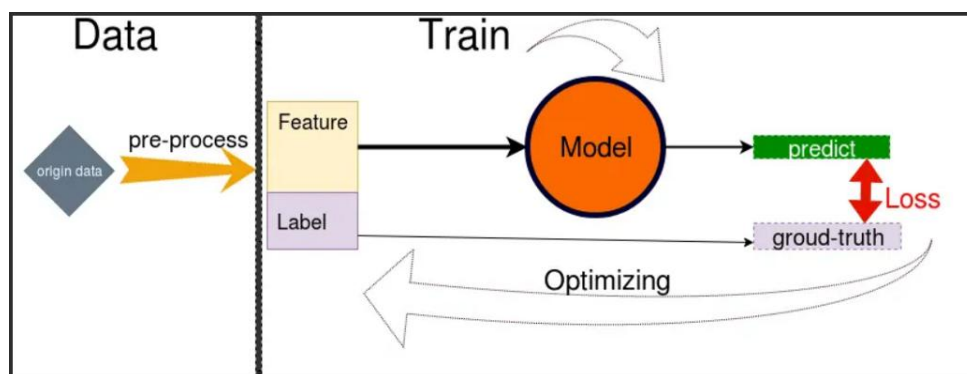
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
warnings.filterwarnings(action='ignore')

if __name__ == '__main__':
    app = QApplication(argv)
    window = MainWindow()
    window.show()
    exit(app.exec_())

```

3. 表情识别

3.1 实验原理



(图像来自网络)

因此，有监督学习的算法一般如下：

伪算法：

1. 加载数据集，并做预处理。
2. 预处理后的数据分为 feature 和 label 两部分，feature 送到模型里面，label 被当做 ground-truth。
3. model 接收 feature 作为 input，并通过一系列运算，向外输出 predict。
4. 通过以 predict 和 predict 为变量，建立一个损失函数 Loss，Loss 的函数值是为了表示 predict 与 ground-truth 之间的差距。
5. 建立 Optimizer 优化器，优化的目标就是 Loss 函数，让它的取值尽可能最小，loss 越小代表 Model 预测的准确率越高。
6. Optimizer 优化过程中，Model 根据规则改变自身参数的权重，这是个反复循环和持续的过程，直到 loss 值趋于稳定，不能在取得更小值。

- 计算维度变化公式：

$$W_2 = (W_1 - F + 2P) / S + 1 \quad (\text{式2})$$

$$H_2 = (H_1 - F + 2P) / S + 1 \quad (\text{式3})$$

在上面两个公式中， W_2 是卷积后 Feature Map 的宽度； W_1 是卷积前图像的宽度； F 是 filter 的宽度； P 是 Zero Padding 数量，Zero Padding 是指在原始图像周围补几圈 0，如果 P 的值是 1，那么就补 1 圈 0； S 是步幅； H_2 是卷积后 Feature Map 的高度； H_1 是卷积前图像的宽度。**式 2** 和 **式 3** 本质上是一样的。

本项目中，搭建一个 CNN 模型对数据集进行训练、分类，主要步骤如下：

伪算法：

1. 加载数据集；
2. 建立神经网络模型；
3. 训练模型，保存模型参数；
4. 预测。

3.2 数据集介绍

本次项目中，进行表情识别采用的是 [FER2013 数据集](#)。

FER2013 数据集是 Kaggle 人脸表情分析比赛提供的一个数据集。该数据集含 28709 张训练样本(训练图)，3859 张验证数据集(公开测试图)和 3859 张测试样本(私有测试图)，共 35887 张包含生气、厌恶、恐惧、高兴、悲伤、惊讶和正常七种类别的图像，图像分辨率为 48×48 。

该数据库是 2013 年 Kaggle 比赛的数据，由于这个数据库大多是从网络爬虫下载的，存在一定的误差性。图像大都在平面和非平面上有旋转，并且很多图像都有手、头发和围巾等的遮挡物的遮挡。这个数据库的人为准确率是 $65\% \pm 5\%$ 。CVS 格式数据集如下：

	A	B	C	D
1	emotion	pixels	Usage	
2		0 70 80 82 72 58 58 60 63 54 58 60	Training	
3		0 151 150 147 155 148 133 111 140	Training	
4		2 231 212 156 164 174 138 161 173	Training	
5		4 24 32 36 30 32 23 19 20 30 41 21	Training	
6		6 4 0 0 0 0 0 0 0 0 0 3 15 23	Training	
7		2 55 55 55 55 55 54 60 68 54 85 15	Training	
8		4 20 17 19 21 25 38 42 42 46 54 56	Training	
9		3 77 78 79 79 78 75 60 55 47 48 58	Training	
10		3 85 84 90 121 101 102 133 153 153	Training	
32295		4 178 176 172 173 173 174 176 173	PublicTest	
32296		3 25 34 42 44 42 47 57 59 59 58 54	PublicTest	
32297		4 255 255 255 255 255 255 255 255	PublicTest	
32298		4 33 25 31 36 36 42 69 103 132 163	PublicTest	
32299		4 61 63 59 75 151 159 166 161 143	PublicTest	
32300		0 170 118 101 88 88 75 78 82 66 74	PrivateTest	
32301		5 7 5 8 6 7 3 2 6 5 4 4 5 7 5 5 5	PrivateTest	
32302		6 232 240 241 239 237 235 246 117	PrivateTest	
32303		4 200 197 149 139 156 89 111 58 62	PrivateTest	
32304		2 40 28 33 56 45 33 31 78 152 194	PrivateTest	
32305		0 138 142 66 80 87 92 97 99 88 73	PrivateTest	
32306		4 72 66 66 69 62 51 57 60 56 66 63	PrivateTest	

每行 pixels 下有 48*48 个数据，以空格进行分隔，即图片的像素灰度值数据。

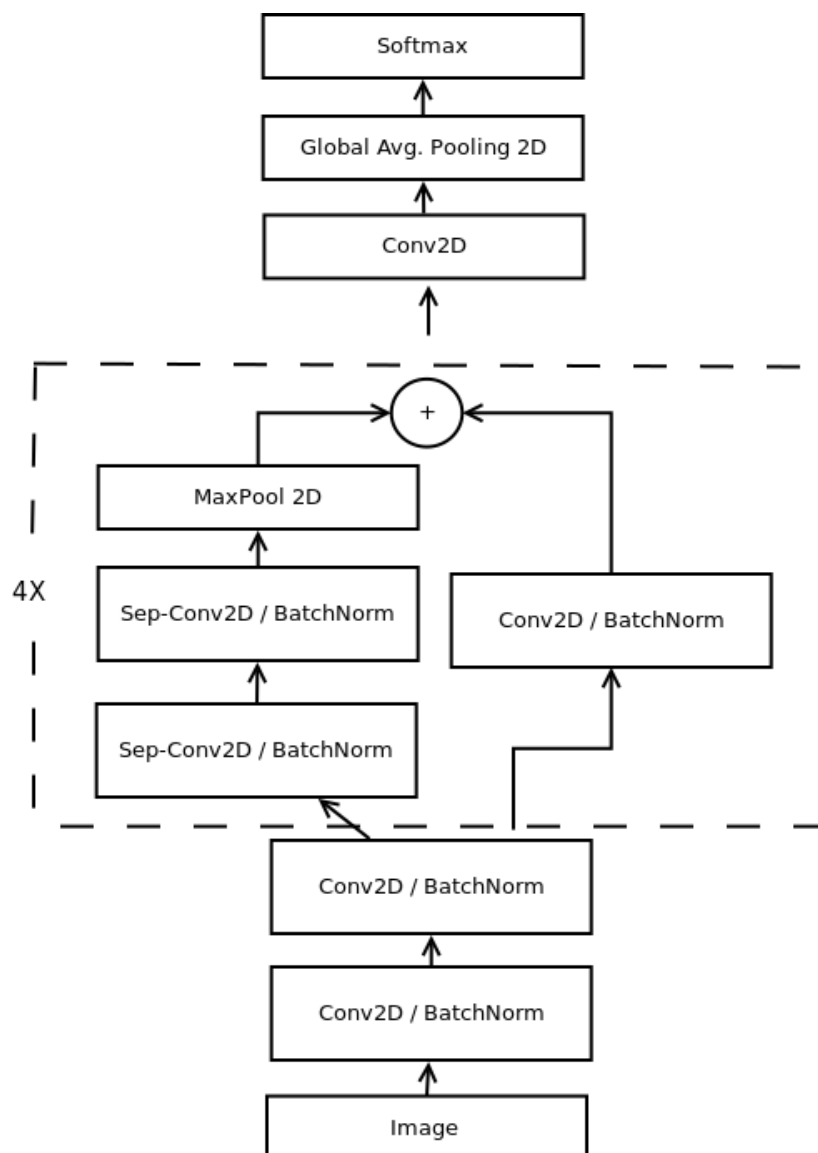
emotion 有 0-6，分别代表以下意思：‘anger’,‘disgust’,‘fear’,‘happy’,‘neutral’,‘sad’,‘surprised’

整个数据集被分为 Training、PrivateTest、PublicTest 三个集合，分别对应训练集，验证集，测试集。

3.3 mini_XCEPTION 网络

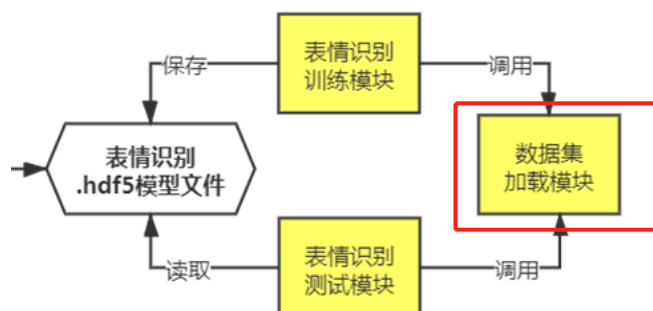
采用了 CNN 主流框架 mini-XCEPTON，来自论文 *Real-time Convolutional Neural Networks for*

Emotion and Gender Classification，结构如下：



3.4 加载数据集

即整个流程中的加载数据集模块：



加载数据集、数据预处理（归一化）代码如下：


```

def load_fer2013():
    """加载数据集
        :return: faces: 表情数据
                emotions: 情绪标签
    """
    data = pd.read_csv(dataset_path, nrows=1000)
    pixels = data['pixels'].tolist()
    width, height = 48, 48
    faces = []

    for pixel_sequence in pixels:
        face = [int(pixel) for pixel in pixel_sequence.split(' ')]
        face = np.asarray(face).reshape(width, height)
        face = cv2.resize(face.astype('uint8'), image_size)
        faces.append(face.astype('float32'))

    faces = np.asarray(faces)
    faces = np.expand_dims(faces, -1)

    # 可视化
    emotions = np.array(data['emotion'])
    visualizeImage(faces, emotions)

    emotions = pd.get_dummies(data['emotion']).as_matrix()
    return faces, emotions

# 预处理：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 x

```

数据集加载时，可视化如下：



载入数据后将数据集划分为训练集和测试集，在程序中调用上面的函数代码如下：

```
from load_and_process import load_fer2013
from load_and_process import preprocess_input
from sklearn.model_selection import train_test_split

# 载入数据集，预处理
faces, emotions = load_fer2013()
faces = preprocess_input(faces)
num_samples, num_classes = emotions.shape

# 划分训练、测试集
xtrain, xtest, ytrain, ytest = train_test_split(faces, emotions, test_size=0.2,
shuffle=True)
```

其中，`train_test_split` 是 `sklearn` 里的随机划分训练集和测试集函数，功能是从样本中随机的按比

例选取 `train data` 和 `test data`，常用于交叉验证。参考[官方文档中](#)，本项目中形式如下：

```
X_train, X_test, y_train, y_test =
cross_validation.train_test_split(train_data, train_target, test_size=0.4,
random_state=0)
```

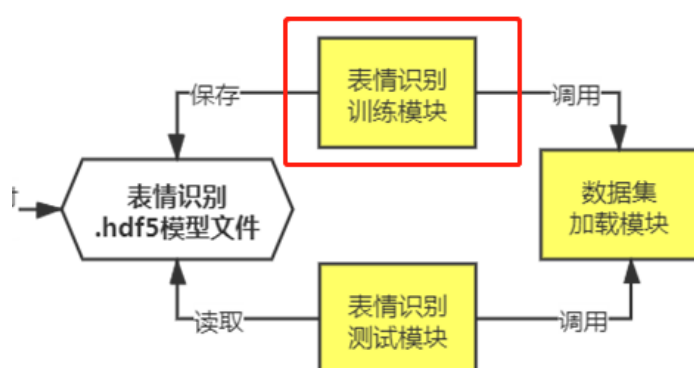
参数解释如下：

参数	含义
Train_data	所要划分的样本特征集

Train_target	所要划分的样本结果
Test_size (float/int)	样本占比，如果是整数的话就是样本的数量
Random_state	随机数的种子（该组随机数的编号），本项目中使用默认的随机数生成器，因此没有用这个参数，默认为 none
Shuffle (bool)	是否在拆分前重组数据顺序，默认为 true。如果 shuffle=False，则 stratify 必须为 None。

3.5 搭建模型并保存

模块位置如下：



由于考虑到代码可读性，本项目中使用 **keras** 框架搭建网络模型，并保存此 **hdf5** 模型。网络如下：

```

from keras.layers import Activation, Convolution2D, Dropout, Conv2D
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 l2

def mini_XCEPTION(input_shape, num_classes, l2_regularization=0.01):
    regularization = l2(l2_regularization)
    """
    卷积层 Conv2d(),
    线性整流单元也就是激活函数 Activation('relu'),
    池化层 MaxPooling2D
  
```

全连接层 `torch.nn.Linear()`

BatchNormalizations: 加快收敛速度的方法—批标准化,
使得数据在进行 **ReLU** 之前不会因为数据过大而导致网络性能的不稳定

```
"""

# 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
# .....
# module 3
# .....

# 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 model

```

其中，除了主要的卷积函数，还有激活函数和标准化快速收敛模块。如下：

- **滑动窗卷积层：Conv2d()**参数如下：

参数	含义
Filters	卷积核的数目（即输出的维度）
kernel_size	卷积核的宽度和长度。例如 kernel_size=(3, 2)表示 3X2 的卷积核，如果宽和高相同，可以只用一个数字表示
strides	卷积每次移动的步长，如为单个整数，则表示在各个空间维度的相同步长。
padding	补 0 策略，为“valid”，“same”。“ valid ”代表只进行有效的卷积，即对边界数据不处理。“ same ”代表保留边界处的卷积结果，通常会导致输出 shape 与输入 shape 相同。
kernel_regularizer	施加在权重上的正则项，为 Regularizer 对象
use_bias	布尔值，是否使用偏置项

- **可分离卷积层：SeparableConvolution2D()**是传统卷积的一种变体，计算更快，它执行深度空间卷积，然后是逐点卷积，它将得到的输出通道混合在一起。参数方面，用法与传统卷积差不多，具体区别可参考 [here](#)，参数含义如下：

参数	含义
filters	卷积核的数目（即输出的维度）
kernel_size	卷积核的宽度和长度. 例如 kernel_size=(3, 2)表示 3X2 的卷积核，如果宽和高相同，可以只用一个数字表示
strides	卷积每次移动的步长，如为单个整数，则表示在各个空间维度的相同步长。
padding	补 0 策略，为“valid”，“same”。“ valid ”代表只进行有效的卷积，即对边界数据不处理。“ same ”代表保留边界处的卷积结果，通常会导致输出 shape 与输入 shape 相同。
kernel_regularizer	施加在权重上的正则项，为 Regularizer 对象
use_bias	布尔值，是否使用偏置项

- **池化层：MaxPool2d()**参数如下：

参数	含义
pool_size	整数或长为 2 的整数 tuple，代表在两个方向（竖直，水平）上的下采样因子，如取（2，2）将使图片在两个维度上均变为原长的一半。为整数意为各个维度值相同且为该数字。
Strides	整数或长为 2 的整数 tuple，或者 None，步长值。
padding	'valid'或者'same'

- **激活函数：Activation()**：

relu	整流线性单元。使用默认值时，它返回逐元素的 $\max(x, 0)$ 。
softmax	Softmax 激活函数。对输入数据的最后一维进行归一化

- **标准化快速收敛 BatchNormalization()**，该层在每个 batch 上将前一层的激活值重新规范化，即使得其输出数据的均值接近 0，其标准差接近 1。作用如下：（1）加速收敛 （2）控制过拟合，可以减少或不用 Dropout 和正则 （3）降低网络对初始化权重不敏感 （4）允许使用较大的学习率。
- 于是，在训练模型之前，就可以直接调用 CNN.py 中的 **mini_XCEPTION** 函数，

```
# 构建模型并 compile 编译配置参数，最后输出网络摘要
model = mini_XCEPTION(input_shape, num_classes)
model.compile(optimizer='adam', # 优化器采用 adam
              loss='categorical_crossentropy', # 多分类的对数损失函数
              metrics=['accuracy'])
model.summary()
```

输出如下：

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 48, 48, 1)	0	
conv2d_1 (Conv2D)	(None, 46, 46, 8)	72	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 46, 46, 8)	32	conv2d_1[0][0]
activation_1 (Activation)	(None, 46, 46, 8)	0	batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 44, 44, 8)	576	activation_1[0][0]
batch_normalization_2 (BatchNor	(None, 44, 44, 8)	32	conv2d_2[0][0]
activation_2 (Activation)	(None, 44, 44, 8)	0	batch_normalization_2[0][0]

// Layer1...(模型文本有点长，已经保存在 model.txt 中)

// ...

// Layer4...

conv2d_7 (Conv2D)	(None, 3, 3, 7)	8071	add_4[0][0]
global_average_pooling2d_1 (Glo	(None, 7)	0	conv2d_7[0][0]
predictions (Activation)	(None, 7)	0	global_average_pooling2d_1[0][0]
Total params: 58,423			
Trainable params: 56,951			
Non-trainable params: 1,472			

- 现在对模型进行保存。代码中设置了训练时的结果输出，在训练结束后会将训练的模型保存为 hdf5 文件到自己指定的文件夹下，由于数据量大模型的训练时间会比较长，建议使用 GPU 加速。

```

num_classes = 7
patience = 50
base_path = 'models/emotion_models/'

# 构建模型并compile编译配置参数，最后输出网络摘要
model = mini_XCEPTION(input_shape, num_classes)
model.compile(optimizer='adam', # 优化器采用adam
              loss='categorical_crossentropy', # 多分类的对数损失函数
              metrics=['accuracy'])
model.summary()

# 定义回调函数 Callbacks 用于训练过程
log_file_path = base_path + '_emotion_training.log'
csv_logger = CSVLogger(log_file_path, append=False)
early_stop = EarlyStopping('val_loss', patience=patience)
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 + '_mini_XCEPTION'
model_names = trained_models_path + '_.{epoch:02d}-{val_acc:.2f}.hdf5'

```

3.6 数据增强

神经网络的训练需要大量的数据，数据的量决定了网络模型可以达到的高度，网络模型尽量地逼近这个高度。所以，载入数据之后，数据训练之前，可以对数据进行增强。为了防止网络过快地过拟合，可以人为的做一些图像变换，例如翻转，旋转，切割等。上述操作称为数据增强。数据操作还有另一大好处是扩大数据库的数据量，使得训练的网络鲁棒性更强。

例子如下，一张猫的照片，经过数据增强之后，可以生成多张不同的图片，用于网络训练：



“为了尽量利用我们有限的训练数据，我们将通过一系列随机变换堆数据进行提升，这样我们的模型将看不到任何两张完全相同的图片，这有利于我们抑制过拟合，使得模型的泛化能力更好。在 Keras 中，这个步骤可以通过 `keras.preprocessing.image.ImageGenerator` 来实现，这个类使你可以：在训练过程中，设置要施行的随机变换通过 `.flow` 或 `.flow_from_directory(directory)` 方法实例化一个针对图像 batch 的生成器，这些生成器可以被用作 keras 模型相关方法的输入，如 `fit_generator`，`evaluate_generator` 和 `predict_generator`。” ——[Keras 官方文档](#)

Keras 提供的图像增强 (Image Augmentation) API 简单而强大。它 `ImageDataGenerator` 类，用于定义图像数据准备和增强的配置。主要包括以下功能：

- 图像随机旋转(rotate)
- 图像随机偏移(shift)
- 图像随机推移错切(shear)

- 图像随机翻转(flip)
- Sample-wise 图像像素标准化
- Feature-wise 图像像素标准化
- ZCA 白化转换
- 图像张量维度的重排序
- 存储增强图像数据

本项目中，定义一个数据增强器的相关代码如下：

```
# 图片产生器，在批量中对数据进行增强，扩充数据集大小
from keras.preprocessing.image import ImageDataGenerator

data_generator = ImageDataGenerator(
    featurewise_center=False,
    featurewise_std_normalization=False,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=.1,
    horizontal_flip=True)
```

代码中相关参数含义如下：

参数	含义
featurewise_center	将输入数据的均值设置为 0，逐特征进行。
samplewise_std_normalization	布尔值。将每个输入除以其标准差。
rotation_range	整数。随机旋转的度数范围。
width_shift_range	浮点数、一维数组或整数。float: 如果 <1，则是除以总宽度的值，或者如果 >=1，则为像素值。
height_shift_range	同上。
zoom_range	浮点数 或 [lower, upper]。随机缩放范围。如果是浮点数，[lower, upper] = [1-zoom_range, 1+zoom_range]。
horizontal_flip	布尔值。随机水平翻转。

3.7 批量训练

现在利用刚刚定义的图片增强器来进行数据集的批量训练，还是用 Keras 的函数，相关代码如下：

```
# 参数：每个 batch 的采样本数、训练轮数、patience
batch_size = 32
num_epochs = 10000
patience = 50

# 利用数据增强器进行训练
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))
```

以上代码中设置了训练时的结果输出，在训练结束后会将训练的模型保存为 hdf5 文件到自己指定的文件夹下，由于数据量大模型的训练时间会比较长，建议使用 GPU 加速。

简单测试了一下，训练过程如下：

```
Epoch 1/100

1/25 [>.....] - ETA: 4:02 - loss: 2.4097 - acc: 0.1250
2/25 [=>.....] - ETA: 2:13 - loss: 2.2768 - acc: 0.1875
3/25 [==>.....] - ETA: 1:33 - loss: 2.2346 - acc: 0.1875
4/25 [===>.....] - ETA: 1:12 - loss: 2.1941 - acc: 0.2031
5/25 [====>.....] - ETA: 59s - loss: 2.1762 - acc: 0.1938
6/25 [=====>.....] - ETA: 50s - loss: 2.2267 - acc: 0.1823
7/25 [=====>.....] - ETA: 43s - loss: 2.2071 - acc: 0.1875
8/25 [=====>.....] - ETA: 37s - loss: 2.1717 - acc: 0.1953
9/25 [=====>.....] - ETA: 33s - loss: 2.1544 - acc: 0.1910
10/25 [=====>.....] - ETA: 29s - loss: 2.1510 - acc: 0.1875
11/25 [=====>.....] - ETA: 26s - loss: 2.1579 - acc: 0.1847
12/25 [=====>.....] - ETA: 23s - loss: 2.1543 - acc: 0.1797
13/25 [=====>.....] - ETA: 21s - loss: 2.1399 - acc: 0.1899
14/25 [=====>.....] - ETA: 18s - loss: 2.1379 - acc: 0.1920
15/25 [=====>.....] - ETA: 16s - loss: 2.1195 - acc: 0.1979
16/25 [=====>.....] - ETA: 14s - loss: 2.1113 - acc: 0.2031
17/25 [=====>.....] - ETA: 12s - loss: 2.1119 - acc: 0.2004
18/25 [=====>.....] - ETA: 10s - loss: 2.1222 - acc: 0.1979
19/25 [=====>.....] - ETA: 9s - loss: 2.1300 - acc: 0.1990
20/25 [=====>.....] - ETA: 7s - loss: 2.1295 - acc: 0.2000
21/25 [=====>.....] - ETA: 6s - loss: 2.1175 - acc: 0.1994
22/25 [=====>....] - ETA: 4s - loss: 2.1119 - acc: 0.2031
23/25 [=====>...] - ETA: 2s - loss: 2.1124 - acc: 0.1997
```

```

24/25 [=====>...] - ETA: 1s - loss: 2.0973 - acc: 0.2018
25/25 [=====] - 39s 2s/step - loss: 2.1031 - acc: 0.1975
- val_loss: 2.3473 - val_acc: 0.1650

Epoch 00001: val_loss improved from inf to 2.34729, saving model to
models/emotion_models/_mini_XCEPTION.01-0.17.hdf5
...
Epoch 00002: val_loss improved from 2.34729 to 2.08542, saving model to
models/emotion_models/_mini_XCEPTION.02-0.17.hdf5
...
Epoch 00003: val_loss did not improve from 2.08542
...

```

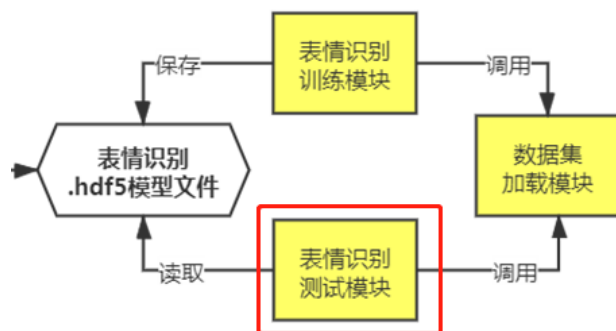
批量训练中，如果训练效果提升，会保存这个新模型。我们使用准确率最高的模型即可，生成文件如下：

名称	训练批次	修改日期	类型	大小
_emotion_training		2020/7/15 21:36	文本文档	6 KB
_mini_XCEPTION.01-0.17.hdf5		2020/7/15 20:49	HDF5 文件	877 KB
_mini_XCEPTION.02-0.17.hdf5		2020/7/15 20:49	HDF5 文件	877 KB
_mini_XCEPTION.05-0.24.hdf5		2020/7/15 20:51	HDF5 文件	877 KB
_mini_XCEPTION.06-0.28.hdf5		2020/7/15 20:51	HDF5 文件	877 KB
_mini_XCEPTION.08-0.27.hdf5		2020/7/15 20:52	HDF5 文件	877 KB
_mini_XCEPTION.11-0.29.hdf5		2020/7/15 20:54	HDF5 文件	877 KB
_mini_XCEPTION.19-0.35.hdf5		2020/7/15 20:57	HDF5 文件	877 KB
_mini_XCEPTION.22-0.31.hdf5		2020/7/15 20:59	HDF5 文件	877 KB
_mini_XCEPTION.23-0.38.hdf5		2020/7/15 20:59	HDF5 文件	877 KB
_mini_XCEPTION.27-0.33.hdf5		2020/7/15 21:01	HDF5 文件	877 KB
_mini_XCEPTION.29-0.35.hdf5		2020/7/15 21:03	HDF5 文件	877 KB
_mini_XCEPTION.41-0.39.hdf5		2020/7/15 21:09	HDF5 文件	877 KB
_mini_XCEPTION.54-0.43.hdf5		2020/7/15 21:15	HDF5 文件	877 KB

上面的模型正确率不高，因为用 CPU 训练模型跑得慢，所以只用了数据包的前 1000 张图片，训练批次暂定了 100。

3.8 测试

模型位置如下：



为了检验这个模型的能力，在训练完成后需要进行测试，然后生成混淆矩阵（**confusion matrix**）评估这个模型。

混淆矩阵的每一列代表了预测类别，每一列的总数表示预测为该类别的数据的数目；每一行代表了数据的真实归属类别，每一行的数据总数表示该类别的数据实例的数目。每一列中的数值表示真实数据被预测为该类的数目。举例如下：

如有 150 个样本数据，这些数据分成 3 类，每类 50 个。分类结束后得到的混淆矩阵为：

		预测		
		类 1	类 2	类 3
实际	类1	43	5	2
	类2	2	45	3
	类3	0	1	49

每一行之和为 50，表示 50 个样本；

第一行说明类 1 的 50 个样本有 43 个分类正确，5 个错分为类 2，2 个错分为类 3。

伪算法：

1. 加载数据集；
2. 加载网络模型；
3. 利用训练好的模型对测试集进行预测；
4. 计算混淆矩阵；
5. 可视化输出。

3.9 实验过程

这个卷积神经网络算法模型简单，数据集挺大，加上用了数据增强器，所以用 cpu 训练的时候，进行了两次模型的训练，然后用两个模型分别做一次预测，两次实验参数如下：




参数	Model1	Model2
batch_size（每个 batch 的采样本数）	32	32
nrows（所用数据集数量）	1000	35000+（全部）
num_epochs（训练集训练次数）	100	200

- 训练 1 部分过程截图如下：

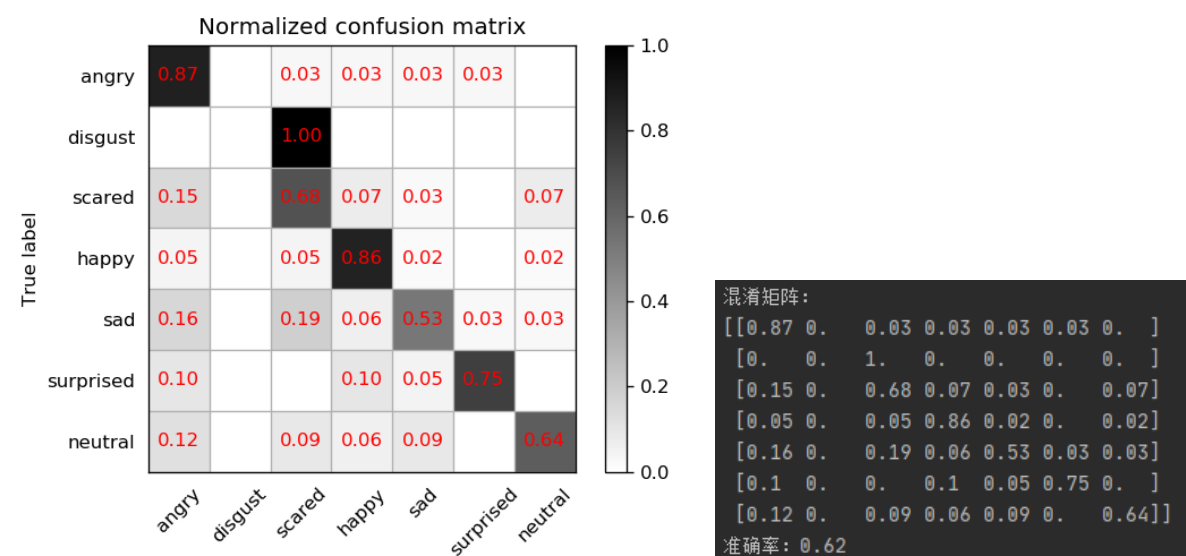
```
Epoch 1/100

1/25 [>.....] - ETA: 4:57 - loss: 3.2218 - acc: 0.0938
2/25 [=>.....] - ETA: 2:41 - loss: 2.9708 - acc: 0.0625
3/25 [==>.....] - ETA: 1:53 - loss: 2.8257 - acc: 0.0625
4/25 [===>.....] - ETA: 1:27 - loss: 2.6518 - acc: 0.0859
5/25 [====>.....] - ETA: 1:11 - loss: 2.5584 - acc: 0.1062
6/25 [=====>.....] - ETA: 59s - loss: 2.5024 - acc: 0.1354
7/25 [=====>.....] - ETA: 51s - loss: 2.4509 - acc: 0.1518
8/25 [======>.....] - ETA: 44s - loss: 2.3904 - acc: 0.1641
9/25 [======>.....] - ETA: 39s - loss: 2.3737 - acc: 0.1562
10/25 [======>.....] - ETA: 34s - loss: 2.3294 - acc: 0.1625
11/25 [======>.....] - ETA: 30s - loss: 2.3087 - acc: 0.1619
```

生成的最终模型文件：

 _mini_XCEPTION.29-0.35.hdf5	2020/7/15 21:03	HDF5 文件	877 KB
 _mini_XCEPTION.41-0.39.hdf5	2020/7/15 21:09	HDF5 文件	877 KB
 _mini_XCEPTION.54-0.43.hdf5	2020/7/15 21:15	HDF5 文件	877 KB

预测时生成的混淆矩阵：



准确率为：0.62，可以看到不同的表情识别的效果不一样，angry 效果最好，disgust 效果最低。这

与训练的数据集、训练批次有关。

- 训练 2 过程部分截图如下：

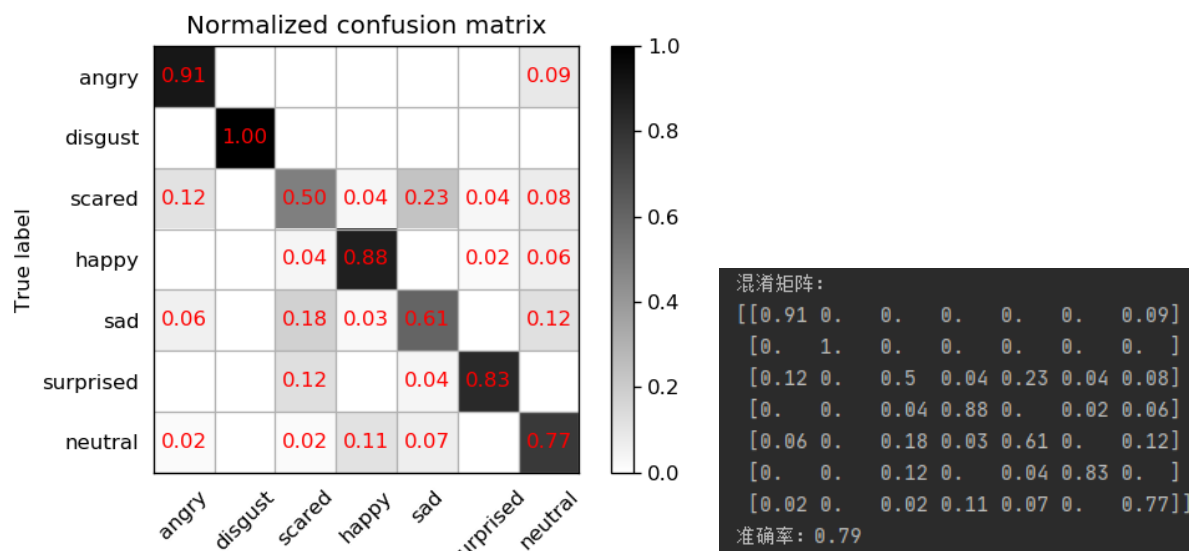
```
Epoch 1/10000

1/897 [.....] - ETA: 3:39:44 - loss: 2.9124 - acc: 0.0625
2/897 [.....] - ETA: 2:02:40 - loss: 2.6062 - acc: 0.0938
3/897 [.....] - ETA: 1:28:30 - loss: 2.6214 - acc: 0.0729
4/897 [.....] - ETA: 1:10:59 - loss: 2.4594 - acc: 0.1250
5/897 [.....] - ETA: 1:00:02 - loss: 2.4245 - acc: 0.1375
6/897 [.....] - ETA: 52:36 - loss: 2.3942 - acc: 0.1458
7/897 [.....] - ETA: 47:10 - loss: 2.4012 - acc: 0.1562
8/897 [.....] - ETA: 43:10 - loss: 2.4115 - acc: 0.1641
9/897 [.....] - ETA: 40:00 - loss: 2.3941 - acc: 0.1667
10/897 [.....] - ETA: 37:36 - loss: 2.4254 - acc: 0.1594
11/897 [.....] - ETA: 35:37 - loss: 2.4070 - acc: 0.1648
12/897 [.....] - ETA: 33:52 - loss: 2.3858 - acc: 0.1615
```

最终生成的模型文件（别人训练 1000 批次的模型）：

 mini_XCEPTION.54-0.43.hdf5	2020/7/15 21:15	HDF5 文件	877 KB
 _mini_XCEPTION.102-0.66.hdf5	2020/7/13 9:48	HDF5 文件	853 KB

预测时生成的混淆矩阵：



准确率为：0.79，所有的情绪都能识别出来，scared 识别率最低，只有 50%。

3.10 算法评估

实验表现总结如下：

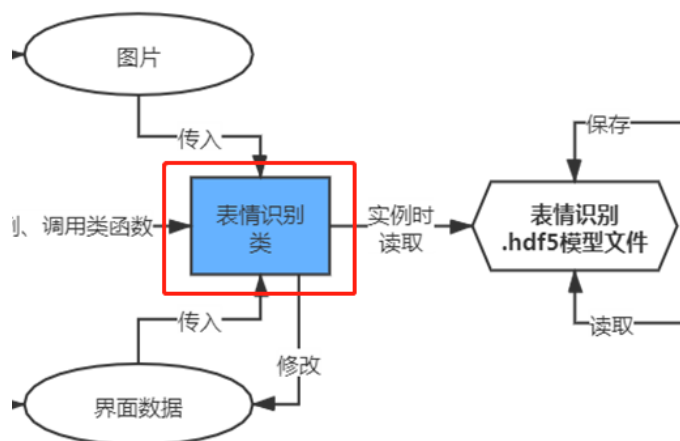
参数	Model1	Model2
数据集加载时间	4s	124s
单批训练平均时间（per epoch）	24s	1100s
训练总时间	2400s	预计 61 天
测试正确率	62%	79%

可见，模型的正确率与数据息息相关，数据量越大，训练批次越高，越有可能达到一个更高的正确率。

另外，如果有条件还是用 GPU 训练，时间会比 CPU 快太多了。我用 CPU 训练时的模型，部分/小数据集还好，1/2 小时之内能跑完，但是用全部数据进行训练就很慢。

3.11 封装成类

UI 页面和核心表情识别都有了，现在定义一个类，传入图片（摄像头/本地读取）以及 UI 界面数据，这个类的函数可以识别这张图像，然后将 UI 界面数据修改，呈现给用户。即这一部分：



当然，这图不完整，因为还涉及到人脸识别 RIO 部分，将照片的人脸切出来之后才是表情识别。

封装如下：

```

import ...
from keras.models import load_model
from load_and_process import preprocess_input

default_emotion_model_path =
'models/emotion_models/_mini_XCEPTION.102-0.66.hdf5' # 默认模型路径

class Emotion_Rec:
    def __init__(self, model_path=None):
        ...
    def run(self, frame_in, label_face):
        # frame_in 摄像画面或图像
        # label_face 用于人脸显示画面的 label 对象
        ...
  
```

总体来说是两个函数：初始化以及识别函数。

- **def __init__()**：相当于构造函数，实例化这个类时，可以输入一个模型路径，那么这个类就会调用

Keras 的 API，载入表情识别模型。如果不输入的话，就使用默认模型路径。代码如下：

```

def __init__(self, model_path=None):
    # 载入数据和图片的参数
    detection_model_path =
    'haarcascade_files/haarcascade_frontalface_default.xml'

    if model_path == None: # 若未指定路径，则使用默认模型
  
```



```
emotion_model_path = default_emotion_model_path
else:
    emotion_model_path = model_path

# 载入人脸检测模型
self.face_detection = cv2.CascadeClassifier(detection_model_path) # 级联分类器

# 载入人脸表情识别模型
self.emotion_classifier = load_model(emotion_model_path, compile=False)
```

- **def run():** 外部调用这个类时进行表情识别

- input：一张待识别的图片 `frame_in`，以及控件 `label_face` 作为图片显示的容器。
- output：所有表情的识别概率 `results`，以及最终识别结果 `result`。

参数	含义	数据格式
frame_in	摄像画面或图像	图像数据
label_face	用于人脸显示画面的 label 对象	UI 控件，人脸框图像容器
Results	所有表情以及对应的识别概率	Dict(string, float)
Result	最终的识别结果， 概率最高的表情	String

代码步骤如下：

伪算法:

1. 图像预处理：大小调节、灰度化；
2. 加载人脸识别模型，检测人脸；
3. 加载表情识别模型；
4. ROI 面积排序，对于最大/全部人脸：
 1. ROI 面积调节成模型输入大小
 2. 表情识别
 3. 圈出人脸区域并标出识别结果，图片显示到 UI 中
5. 返回所有表情的概率以及最终结果

代码如下：

[illegible]

```

flags=cv2.CASCADE_SCALE_IMAGE)

preds = [] # 预测的结果
label = None # 预测的标签
frameClone = frame.copy() # 复制画面

if len(faces) > 0:
    # 根据 ROI 大小将检测到的人脸排序，面积从小到大
    faces = sorted(faces, reverse=False, key=lambda x: (x[2] - x[0]) * (x[3]
- x[1]))

    for i in range(len(faces)): # 遍历每张检测到的人脸，默认识别全部人脸
        fX, fY, fW, fH) = faces[i] # 人脸位置

        # 从灰度图中提取感兴趣区域 (ROI)，将其大小转换为与模型输入相同的尺寸，并为通过
        CNN 的分类器准备 ROI
        roi = gray[fY:fY + fH, fX:fX + fW]
        roi = cv2.resize(roi, self.emotion_classifier.input_shape[1:3])
        roi = preprocess_input(roi)
        roi = img_to_array(roi)
        roi = np.expand_dims(roi, axis=0)

        # 用模型预测各分类的概率
        preds = self.emotion_classifier.predict(roi)[0]
        # emotion_probability = np.max(preds) # 最大的概率
        label = self.EMOTIONS[preds.argmax()] # 选取最大概率的表情类

        # 圈出人脸区域并显示识别结果
        cv2.putText(frameClone, label, (fX, fY - 10),
                     cv2.FONT_HERSHEY_TRIPLEX, 0.4, (0, 0, 255), 1)
        cv2.rectangle(frameClone, (fX, fY), (fX + fW, fY + fH), (0, 0, 255), 1)

    # 调整画面大小与界面相适应
    # 439 * 299
    frameClone = cv2.resize(frameClone, (420, 280))

    # 在 Qt 界面中显示人脸
    show = cv2.cvtColor(frameClone, cv2.COLOR_BGR2RGB)
    showImage = QtGui.QImage(show.data, show.shape[1], show.shape[0],
        QtGui.QImage.Format_RGB888)
    label_face.setPixmap(QtGui.QPixmap.fromImage(showImage))
    QtWidgets.QApplication.processEvents()

    return zip(self.EMOTIONS, preds), label

```

4. 心得体会

4.1 Python 知识积累

- Python 安装特定版本的 Opencv：例如 `pip install opencv-python==3.4.3.18`
- Python , Pycharm , Anaconda 区别与联系，参考 [here](#)
- Pip 命令下载库的速度太慢时，**read timed out** 错误时，可以用国内的镜像，参考 [here](#)

清华：`-i https://pypi.tuna.tsinghua.edu.cn/simple`

例如：`pip install -i https://pypi.tuna.tsinghua.edu.cn/simple opencv-python==4.1.0.25`

`pip install --index-url https://pypi.douban.com/simple PyQt5-tools`

- 简单理解混淆矩阵，参考 [here](#)
- Python3.7 中 time 模块的 `time()`、`perf_counter()`和 `process_time()`的区别，参考 [here](#)
- Sklearn 拆分训练、测试数据：`train_test_split()`，参考 [here](#)
- Python 项目管理的利器：虚拟环境 `venv` 的使用,[here](#)
- 导入上级目录的模块，[here](#)

```
import sys
sys.path.append('../')
import qrc.resource          #dirName.fileName
```

4.2 PyQt5-界面程序开发

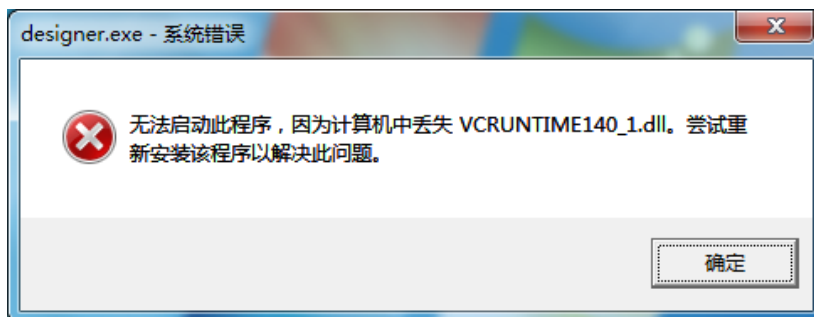
4.2.1 环境配置

环境配置参考 [here](#)，使用教程参考 [here](#)，一些库的作用：

工具	功能	其他参考博文
qtdesigner	用于设计和修改 ui 界面文件	Here
pyuic	用于将 ui 文件转化为 py 文件	
pyrcc5	用于将 qrc 文件转化为 py 文件	
pyi-makespec	用于创建 spec 文件	
pyinstaller	基于 spec 文件打包生成可执行程序	Python 脚本打包成.exe 文件，参考 here 、 here

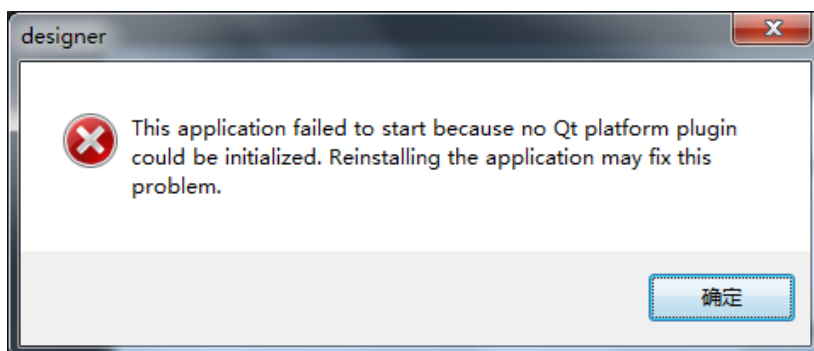
4.2.2 配置时的 bug

- QT 启动出现 During startup program exited with code 0xc0000135 错误，是缺少.dll 文件，[here](#)



- **解决办法：**下载对应的 dll 文件，解压到 C:\Windows\System32 文件夹下。

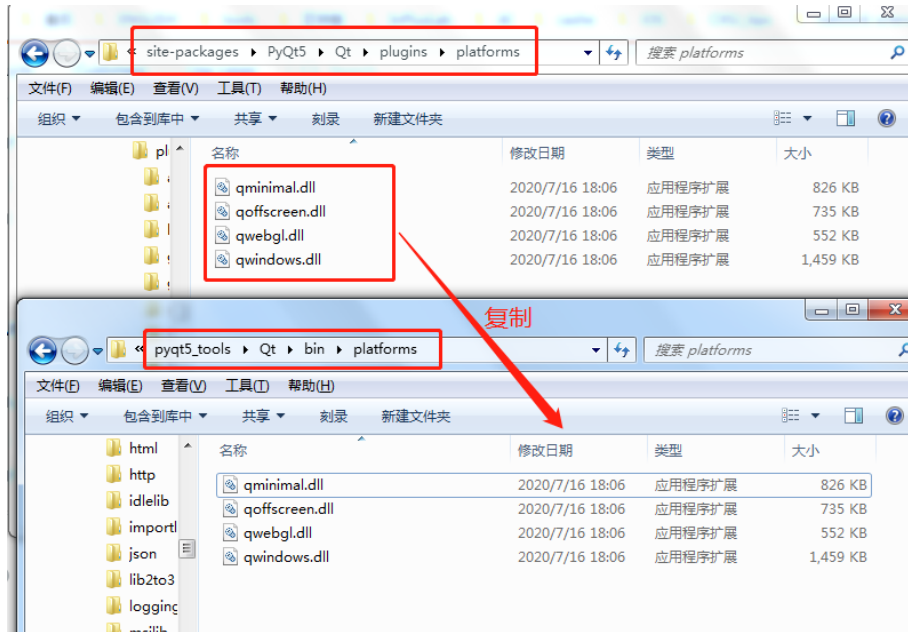
- QT 启动说是缺少 Qt platform plugin,参考 [here](#)



- **解决办法 1：**

1. Go to => *Python38>lib>site-packages>PyQt5>Qt>plugins*
2. In plugins copy platform folder
3. After that go to *Python38>lib>site-packages>PyQt5_tools>Qt>bin*
4. paste folder here . Do copy and replace.

This will surely work.. Now you can use designer tool go and do some fun with python.



➤ 解决办法 2 :

Try running it using command: `pyqt5designer`
It should set all the paths for libraries.

这个方法也可以， qtdesigner 可以启动，但是命令行会出现一些信息：

```
QT_DEBUG_PLUGINS is not set
QT_PLUGIN_PATH: ;d:\anaconda3\lib\site-packages\pyqt5_tools\Qt\plugins
PYQT5TOOLS_TEST_PATH is not set
```

● 界面添加自定的图像

{如在 qt designer 中修改某个东西的图标 icon，不能直接使用图片，要将图片路径写成 qrc 文件形式才能使用}

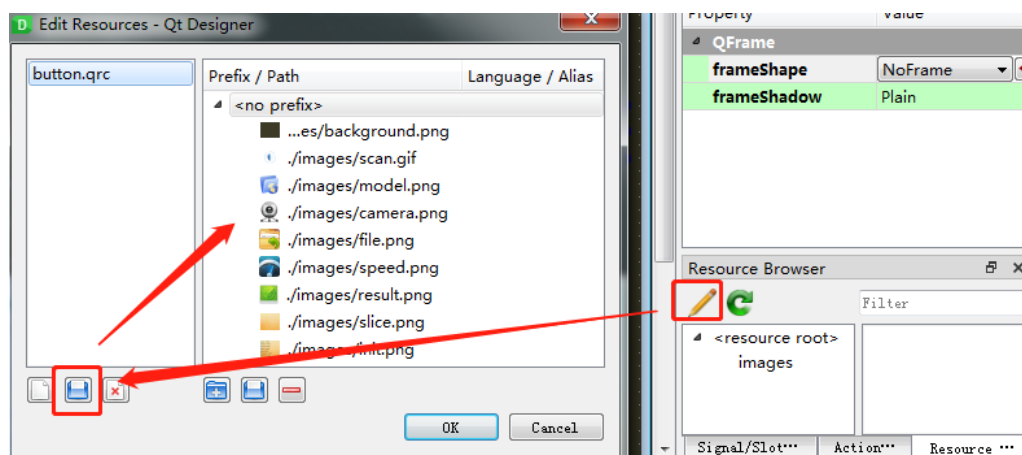
➤ 解决办法：配置图标 qrc 资源文件

button.qrc 文件内容如下：引入图片路径即可

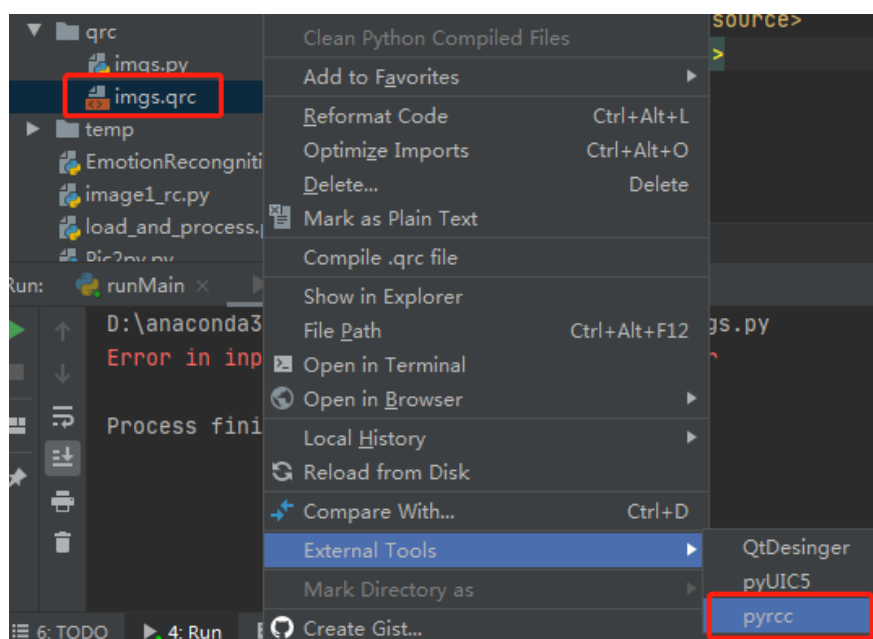
```
<!DOCTYPE RCC>
<RCC version="1.0">
<qresource>
  <file>./images/background.png</file>
  <file>./images/init.png</file>
```

```
</qresource>
</RCC>
```

然后将 qrc 文件加入 qt designer resources (右下角) 中



最后将 qrc 资源文件转为 py 文件。



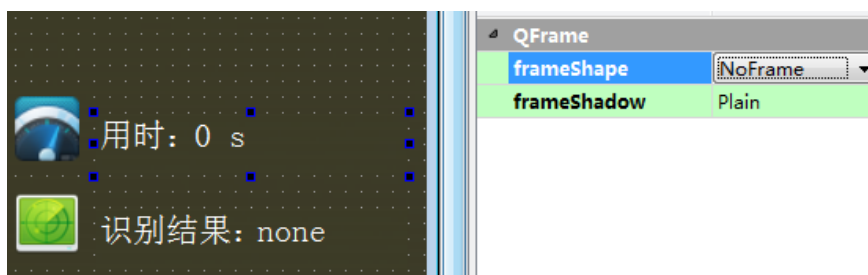
4.2.3 Pyqt 界面 UI 与逻辑分离

参考教程 [here](#) , 项目 [here](#)

4.2.4 零碎知识点

- QTextEdit 无边框设计

选中 QTextEdit 控件，找到 QFrame 里面的 frameShape 属性，将属性设置为 NoFrame



- 文件对话框 QFileDialog 的使用, [here](#)
- 动态添加控件, [here](#)
- 根据控件 ID 获取控件对象

```
self.findChild(QComboBox, "name")
```

self is class

first parameter is Type

second parameter is Id String

4.3 表情识别存在的问题与发展前景

当前表情识别存在的问题

面部表情识别是一个跨学科，具有挑战性的前沿话题，但目前尚未进入实践领域。一个好的表情识别系统应该能够抵抗外加复杂背景因素的干扰，提取出有效的表情特征，并能准确高效的分析出人脸表情的所属类别。在过去的几十年中，面部识别技术从无到有，虽然面部表情识别的理论和技术取得了很大进展，但仍旧存在需要优化和改进的部分：

- 1) 由于人的多样性，面部的外观，表情和肤色可能不同，具有模式可变性，面部识别的准确性并不总是十分稳定[19]；
- 2) 再度人脸进行特征提取时很难排除人脸胡须、眼睛等因素的干扰，这些干扰因素的存在会降低实验结果的准确性；
- 3) 现实生活中对于人脸以及人脸表情的检测往往会受制于复杂背景和光照条件的影响，如在强光条件下人脸检测和表情识别的准确度都会大大降低；

- 4) 当前的研究中，各学科结合不够密切，对于人类表情的分类还停滞于 6 中基本表情；
- 5) 当前的人脸表情研究算法缺乏创新，大多数人的研究只是基于各种已有算法的叠加；
- 6) 各种算法有待优化，对于表情识别和人脸检测中，准确度高的算法往往对计算机的硬件要求高，并且运算复杂度高、运算时间长，很难应用于实际生产生活。

表情识别的发展前景

随着大数据时代的到来以及人工智能、深度学习的持续火热，人们对面部表情实时识别的需求急剧增加。未来人脸表情识别将会有更大的发展潜力，现总结如下：

- 1) 进行三维表情研究，目前的研究大都输入人脸的二位图像，手指与这种现状，很难完全反映出一个人的表情状态，二使用三维图像，可以完整、真实的传递人的表情信息，减少表情识别受光照和状态的影响，使用三维信息对人脸表情进行研究将是未来一个重要的研究课题；
- 2) 加强技术融合，目前人脸检测，表情特征提取、表情识别的方法很多，每种方法有其自身优势也相应存在部分缺陷，如何荣获人各种检测识别方法中的优势，提高表情识别的速度与准确率，会批各自缺点，也将是未来表情识别中的研究中重点；
- 3) 融入非视觉的因素，人的表情和思维具有较高的复杂度，仅仅从表情一方面很难对情绪进行准确判断，未来应结合体温、声音、环境等因素对人的心理进行总和判断；
- 4) 建立公共表情数据库，目前对人脸表情的研究离不开表情数据库的使用，而有效的、开放的表情数据库又寥寥无几，因此着力建立表情数据库也是未来表情识别发展中的一个关键问题。

4.4 改进

这次实验花了很多时间和精力，完成整个项目之后，获益匪浅。

我觉得需要改进的地方在于：

- 用 GPU 跑代码。

- 多跑几次做优化。最开始只设置了 5 次，正确率很低，后面改为 100 次了。
- 可以增加一个性别识别，下次交作业的时候可以试着完善一下。
- 界面可以再美化一下。
- 试着把 py 程序打包成.exe 文件

5. 参考文献和资料

《实感交互：人工智能下的人机交互技术》

[《Keras 中文文档》](#)

[《常用的几种卷积神经网络介绍》](#)、[《几种经典的卷积神经网络模型》](#)

参考的项目：

[集成了人脸识别系统的视频播放器 \(PyQt5\)](#)

[python 优秀项目分享-表情识别](#)

[人脸表情识别系统介绍](#)

一些博客：

[Emotient 的表情识别技术的门槛是什么，有哪些应用场景？](#)

[《人脸表情识别，智能玩转“读心术”》](#)

[【人机交互技术】人脸表情识别技术综述](#)

市场上的 API：[Microsoft Azure 人脸分析](#)、[face++情绪识别](#)

[Keras vs PyTorch：谁是「第一」深度学习框架？](#)

<https://zhuanlan.zhihu.com/p/39779767>

https://blog.csdn.net/qq_41185868/article/details/80318424

https://blog.csdn.net/qq_41185868/article/details/80323646

6. 代码附录

P. S. 一共 2 个文件，本解题报告 pdf 文件 + 项目 python 代码，代码结构如下：

```
pro
├── dataset      未上传
│   └── fer2013
├── models       训练的模型，两个 emotion 分类器
│   ├── cnn.py   网络结构
│   └── emotion_models .hdf5 模型
├── images
│   ├── test     用于测试的图片，部分上传
│   └── ui       用于美化 ui 的图片
├── qrc          qrc 资源代码
│   ├── resoure.qrc
│   └── resoure.py
├── ui           ui 界面代码
│   ├── window.ui
│   └── window.py
├── main.py      主模块，调用带 UI 界面的交互程序
├── ui_minWindow.py UI 模块
├── real_time_video_me.py 表情识别模块
├── train_emotion_classifier.py 训练模块，生成模型
├── test_confusion_matrix.py 测试模块，生成混淆矩阵
└── load_and_process.py 数据集加载模块
```