**基于卷积神经网络的高斯朴素贝叶斯分类人脸表情识别算法详解 (FaceEmotionClassifier)**

小组长：**李沅隆**

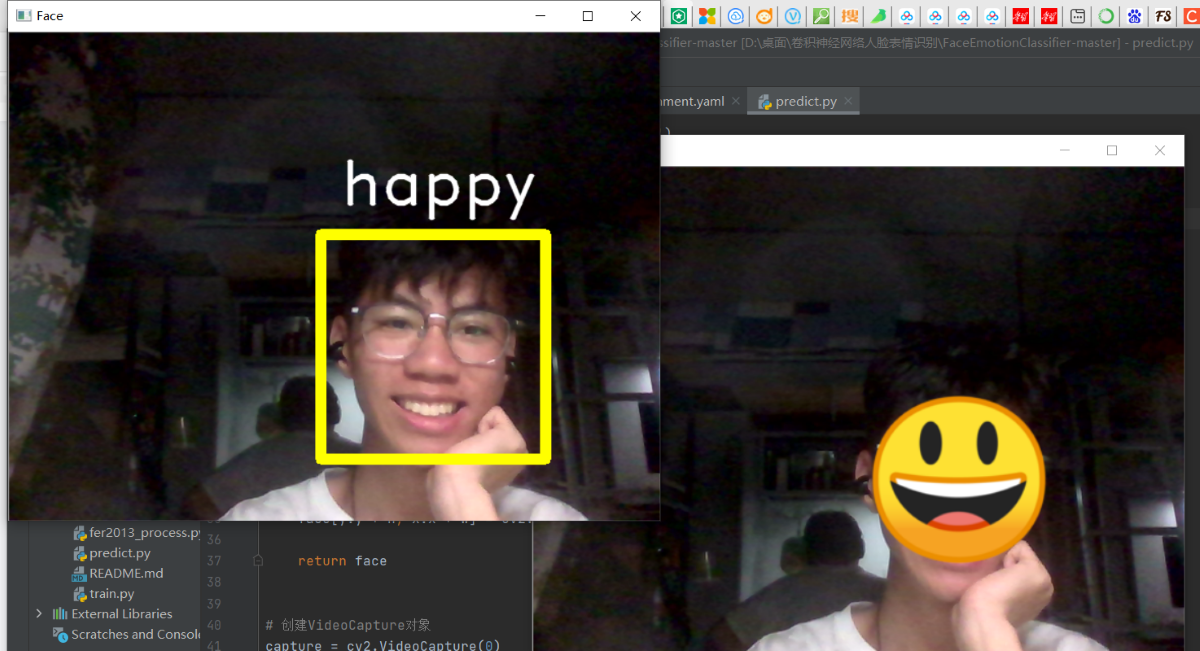
主讲人：**连亨尧**  
*Jupyter Notebook* 制作人：**连亨尧**

报告联合制作人：陈炳鑫，吴浩天，谢宇丰，李溢成，李沅隆

## 1.项目简介：

用搭载**Keras**的**tensorflow**框架通过**卷积神经网络**训练模型，使用**高斯朴素贝叶斯分类器**识别人类的情绪。根据情绪选择相应的emoji匹配  
>GitHub:https://github.com/Fosu-Hyao/GaussianBayes-FaceEmotionClassifier

### 1.1项目效果



### 1.2配置及环境：

**数据集：** Fer2013 ( kaggle挑战赛 ) ，Emoji表情集  
**神经网络框架：** Keras,Tensorflow-gpu  
**分类器：** 基于Opencv-Normal Bayes Classifier(正态贝叶斯分类)训练的贝叶斯分类器  
**python环境：** python==3.6.0 tensorflow-gpu==**1.8.0** keras-gpu==2.1.6 opencv==3.3.1

**详细环境见：** GitHub的environment.yaml文件

#加载所需的包  
  
#fer2013\_process.py  
import csv  
import os  
from PIL import Image  
import numpy as np  
  
#train.py  
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D  
from keras.models import Sequential  
from keras.preprocessing.image import ImageDataGenerator  
from keras.optimizers import SGD  
  
#perdict.py  
import cv2  
import numpy as np  
from keras.models import model\_from\_json

Using TensorFlow backend.

### 1.3总流程：

* 对训练集预处理
* 然后用处理好的fer2013的数据集训练深度卷积神经网络构建的模型识别人脸表情
* 使用训练好的模型识别人脸的表情情绪
* 根据识别结果，匹配合适的emoji遮住人脸

**流程图：**

### 

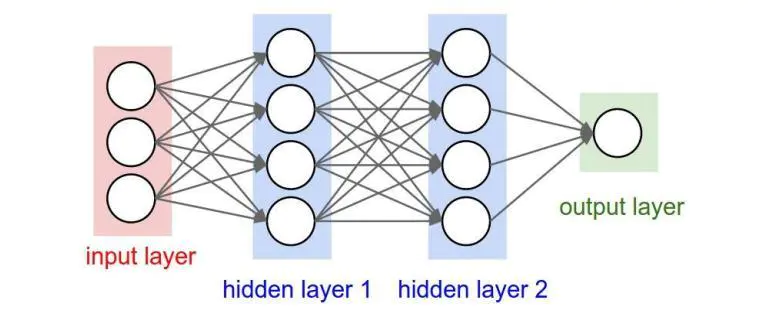
### 1.4参考文献：

* https://blog.csdn.net/qq\_42995327/article/details/114548137
* https://blog.csdn.net/zhaocj/article/details/50615049
* https://blog.csdn.net/u013841196/article/details/84262196
* https://blog.csdn.net/GUET\_DM\_LQ/article/details/103765163
* https://www.jianshu.com/p/399e5a3c7cc5
* https://www.jianshu.com/p/1ea2949c0056

### 1.5知识介绍：

#### 1.5.1 人工神经网络

通俗说就是对大脑中神经元传输信息的过程的**模拟**，抽象成一个数学模型——对应的参数乘对应的**权重的累加和**

****

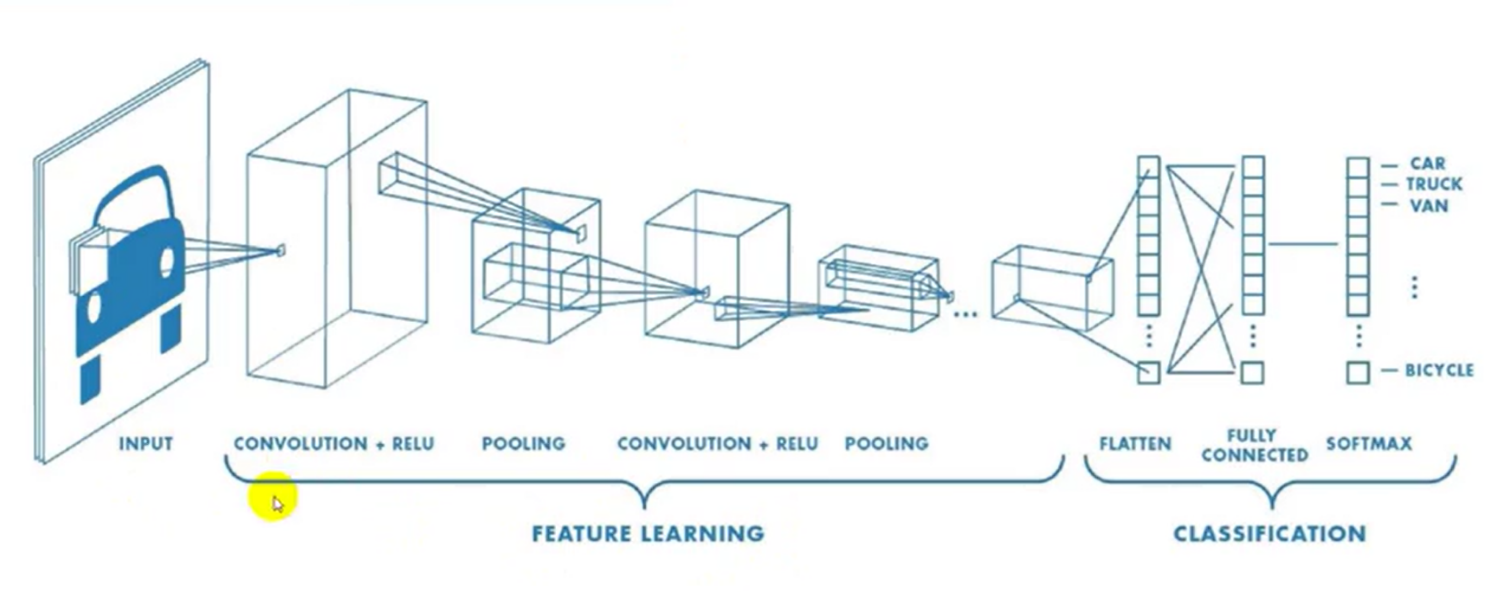
**人工神经网络缺点：**

* 局限于编程人的经验和能力
* 本质上是个分类器，无法适应复杂环境
* 全连接层非常多

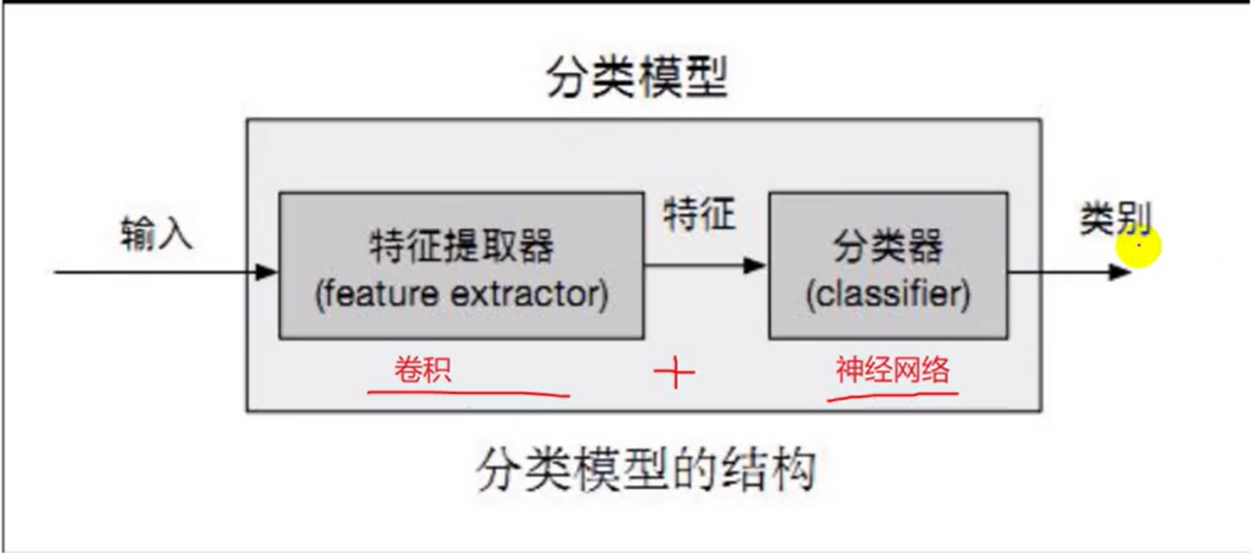
#### 

#### 1.5.2 [卷积神经网络](https://www.jianshu.com/p/1ea2949c0056)

其实卷积神经网络依旧是层级网络，只是 **层的功能和形式做了变化** ，可以说是传统神经网络的一个改进。比如下图中就多了许多传统神经网络没有的层

****

**模型实质：** 卷积+分类器

****

**卷积神经网络的层级结构：**  
+ 数据输入层/ Input layer  
+ 卷积计算层/ CONV layer  
+ ReLU激活层 / ReLU layer  
+ 池化层 / Pooling layer + 全连接层 / FC layer

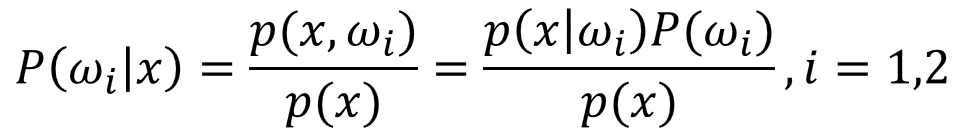
**卷积神经网络之优缺点：**  
**优点：**  
　　•共享卷积核，对高维数据处理无压力  
　　•无需手动选取特征，训练好权重，即得特征分类效果好  
**缺点：**  
　　•需要调参，需要大样本量，训练最好要GPU  
　　•物理含义不明确（也就说，我们并不知道没个卷积层到底提取到的是什么特征，而且神经网络本身就是一种难以解释的“黑箱模型”）

#### 1.5.3 朴素贝叶斯算法

**贝叶斯公式：**

贝叶斯定理就是**逆向概率**问题

“如果我们事先并不知道袋子里面黑白球的比例，而是闭着眼睛摸出一个（或好几个）球，观察这些取出来的球的颜色之后，那么我们可以就此对袋子里面的黑白球的比例作出什么样的推测”。



**朴素贝叶斯分类算法：[(链接)](https://zhuanlan.zhihu.com/p/26262151)**

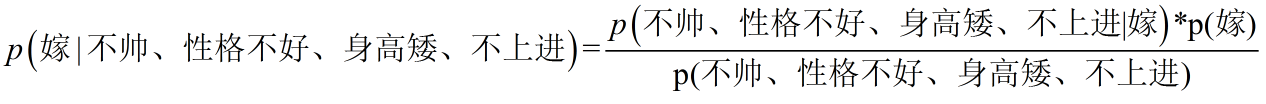
**假定数据如下：**



现在给我们的问题是，如果一对男女朋友，男生想女生求婚，男生的四个特点分别是**不帅，性格不好，身高矮，不上进** ，请你判断一下女生是嫁还是不嫁？

这是一个典型的分类问题，转为数学问题就是比较**p(嫁|(不帅、性格不好、身高矮、不上进))**与**p(不嫁|(不帅、性格不好、身高矮、不上进))**的概率，谁的概率大，我就能给出嫁或者不嫁的答案！

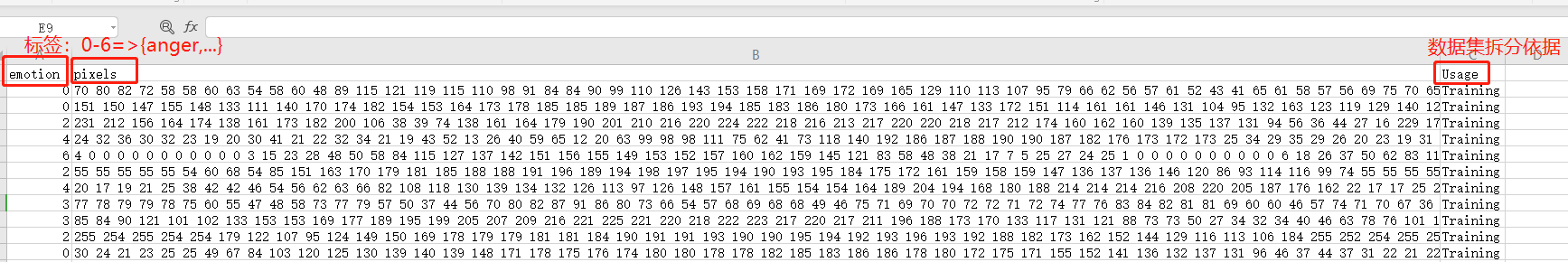
这里我们联系到朴素贝叶斯公式：



我们需要求p(嫁|(不帅、性格不好、身高矮、不上进),这是我们不知道的，但是通过朴素贝叶斯公式可以转化为好求的三个量,**p(不帅、性格不好、身高矮、不上进|嫁)、p（不帅、性格不好、身高矮、不上进)、p(嫁)** （至于为什么能求，后面会讲，那么就太好了，将待求的量转化为其它可求的值，这就相当于解决了我们的问题！）

## 2.数据集预处理

**下载得到的csv格式可以通过Excel看到格式为：**

****

**所以首先打开csv文件，根据usage把数据集分为：训练集、测试集和验证集**

data\_path = os.getcwd() + "/data/" #数据集储存位置（获取当前py文件位置+data文件夹内）  
csv\_file = data\_path + 'fer2013.csv' #原始数据集储存名称及位置  
train\_csv = data\_path + 'train.csv' #分类后训练集名称及位置  
val\_csv = data\_path + 'val.csv' #……测试集……  
test\_csv = data\_path + 'test.csv' #……测试集……  
  
#分类后图片储存位置  
train\_set = os.path.join(data\_path, 'train')   
val\_set = os.path.join(data\_path, 'val')  
test\_set = os.path.join(data\_path, 'test')  
  
#打开数据集进行分类  
with open(csv\_file) as f:  
 csv\_r = csv.reader(f)  
 header = next(csv\_r) #Python中读取文件，指向的是文件的第一行，但是第一行是标题，所以用next()指向下一行，也就是从第二行开始  
 print(header)  
 rows = [row for row in csv\_r]  
  
 #分类训练集  
 trn = [row[:-1] for row in rows if row[-1] == 'Training']  
 csv.writer(open(train\_csv, 'w+'), lineterminator='\n').writerows([header[:-1]] + trn)  
 print(len(trn))  
  
 #分类验证集  
 val = [row[:-1] for row in rows if row[-1] == 'PublicTest']  
 csv.writer(open(val\_csv, 'w+'), lineterminator='\n').writerows([header[:-1]] + val)  
 print(len(val))  
  
 #分类测试集  
 tst = [row[:-1] for row in rows if row[-1] == 'PrivateTest']  
 csv.writer(open(test\_csv, 'w+'), lineterminator='\n').writerows([header[:-1]] + tst)  
 print(len(tst))  
   
print("分类完成！")

如果直接用当前数据是一个扁平的向量，没有空间局部性。用这样的数据直接进行训练，就会失去空间结构和图像关系信息。卷积神经网络可以保留空间信息，并且更适合图像分类问题，所以要把数据转为图片方便下面采用卷积神经网络进行训练

#将分类后的数据转为图片  
  
#读取数据集地址和保存图片的地址  
for save\_path, csv\_file in [(train\_set, train\_csv), (val\_set, val\_csv), (test\_set, test\_csv)]:  
 if not os.path.exists(save\_path):  
 os.makedirs(save\_path)  
  
 num = 1  
 with open(csv\_file) as f:  
 csv\_r = csv.reader(f)  
 header = next(csv\_r)  
 for i, (label, pixel) in enumerate(csv\_r):  
 # 0 - 6 文件夹内的图片label分别为：  
 # angry ，disgust ，fear ，happy ，sad ，surprise ，neutral  
 pixel = np.asarray([float(p) for p in pixel.split()]).reshape(48, 48)  
 sub\_folder = os.path.join(save\_path, label)  
 if not os.path.exists(sub\_folder):  
 os.makedirs(sub\_folder)  
 im = Image.fromarray(pixel).convert('L')  
 image\_name = os.path.join(sub\_folder, '{:05d}.jpg'.format(i))  
 print(image\_name)  
 im.save(image\_name)

## 

## 3.卷积神经网络模型

### 3.1深度卷积神经网络模型结构：

**#卷积层：conv2D 池化层：MAaxPooling2D 激活函数：RELU**  
**#扁平层：Flaten 全连接层：Dense 损失函数：Dropout**

模型结构：

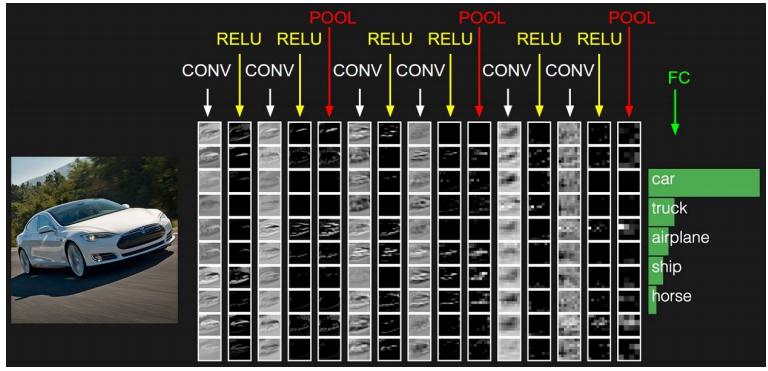
conv2D +activation-relu +conv2D + activation-relu +MaxPooling2D +

conv2D + activation-relu + MaxPooling2D +

conv2D + activation-relu + MaxPooling2D +

Flaten + Dense + activation-relu +

Dropout + Dense + activation-relu +Dropout +softmax:Dense + activation-relu3.4



代码详解：

**卷积阶段：**

使用keras添加一层二维滤波器，输出维度是**32**并且每个二维滤波器是1 \* 1的卷积层

self.model.add(Conv2D(32, (1, 1), strides=1, padding='same', input\_shape=(img\_size, img\_size, 1)))

**使用ReLU激活函数：**

self.model.add(Activation('relu'))

**池化层阶段：**

然后给网络学习32个5 \* 5的滤波器，也用ReLU激活。并且紧接着一个最大池化层方法

self.model.add(Conv2D(32, (5, 5), padding='same'))  
self.model.add(Activation('relu'))  
self.model.add(MaxPooling2D(pool\_size=(2, 2)))

之后第二层卷积阶段和第三层卷积阶段都是用ReLU激活函数，后面再次跟着最大池化层方法。第二层仍然是32个3 3大小的滤波器，第三层滤波器增加到64个5 5，**在更深的网络层增加滤波器数目是深度学习中一个普遍采用的技术**

self.model.add(Conv2D(32, (3, 3), padding='same'))  
self.model.add(Activation('relu'))  
self.model.add(MaxPooling2D(pool\_size=(2, 2)))  
  
self.model.add(Conv2D(64, (5, 5), padding='same'))  
self.model.add(Activation('relu'))  
self.model.add(MaxPooling2D(pool\_size=(2, 2)))

**深度管道的下一个阶段：**

首先用Flatten()获得一个扁平的网络

self.model.add(Flatten())

用ReLU激活一个有2048个神经元的隐藏层，用Dropout丢弃到一半的网络，再添加一个1024个神经元的隐藏层，跟着一个关闭50%神经元的dropout层

self.model.add(Activation('relu'))  
self.model.add(Dropout(0.5))  
self.model.add(Dense(1024))  
self.model.add(Activation('relu'))  
self.model.add(Dropout(0.5))

**输出层：**

添加作为输出7个类的softmax层，每个类对应一个类别

self.model.add(Dense(num\_classes))  
self.model.add(Activation('softmax'))

## 4.训练模型

### 4.1 编译模型

这里选择随机梯度下降算法作为优化器

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)  
self.model.compile(loss='categorical\_crossentropy', optimizer=sgd, metrics=['accuracy'])

### 4.2 通过数据增加改善性能

通常提高性能有两种方法，一种是定义一个更深、有更多卷积操作的网络，另一种训练更多的图片。这里用keras自带的ImageDataGenerator方法扩展数据集

# 自动扩充训练样本  
train\_datagen = ImageDataGenerator(  
 rescale=1. / 255, # 归一化处理  
 shear\_range=0.2, # 随机缩放  
 zoom\_range=0.2, # 放大  
 horizontal\_flip=True) # 随机水平翻转

### 4.3使用GPU训练模型

考虑到效率问题，keras提供了生成器针对模型的并发运行。我的理解就是**CPU处理生成图像，GPU上并行进行训练**

# 归一化验证集  
val\_datagen = ImageDataGenerator(  
 rescale=1. / 255)  
eval\_datagen = ImageDataGenerator(  
 rescale=1. / 255)  
# 以文件分类名划分label  
train\_generator = train\_datagen.flow\_from\_directory(  
 data\_path + '/train',  
 target\_size=(img\_size, img\_size),  
 color\_mode='grayscale',  
 batch\_size=batch\_siz,  
 class\_mode='categorical')  
val\_generator = val\_datagen.flow\_from\_directory(  
 data\_path + '/val',  
 target\_size=(img\_size, img\_size),  
 color\_mode='grayscale',  
 batch\_size=batch\_siz,  
 class\_mode='categorical')  
eval\_generator = eval\_datagen.flow\_from\_directory(  
 data\_path + '/test',  
 target\_size=(img\_size, img\_size),  
 color\_mode='grayscale',  
 batch\_size=batch\_siz,  
 class\_mode='categorical')  
history\_fit = self.model.fit\_generator(  
 train\_generator,  
 steps\_per\_epoch=800 / (batch\_siz / 32), # 28709  
 nb\_epoch=nb\_epoch,  
 validation\_data=val\_generator,  
 validation\_steps=2000,  
)

### 4.4保存模型结构及权重

把结构保存为JSON字串，把权重保存到HDF5文件

model\_json = self.model.to\_json()  
with open(model\_path + "/model\_json.json", "w") as json\_file:  
 json\_file.write(model\_json)  
self.model.save\_weights(model\_path + '/model\_weight.h5')  
self.model.save(model\_path + '/model.h5')

## 5.人脸表情识别模块（贝叶斯分类器）

### 5.1 加载加载权重及模型结构

# 从json中加载模型  
json\_file = open(model\_path + 'model\_json.json')  
loaded\_model\_json = json\_file.read()  
json\_file.close()  
model = model\_from\_json(loaded\_model\_json)  
  
# 加载模型权重  
model.load\_weights(model\_path + 'model\_weight.h5')

### 5.2使用基于Opencv-Normal Bayes Classifier(正态贝叶斯分类)训练的贝叶斯分类器

# 创建VideoCapture对象  
capture = cv2.VideoCapture(0)  
  
# 使用Opencv-Normal Bayes Classifier的贝叶斯人脸分类器  
cascade = cv2.CascadeClassifier(model\_path + 'haarcascade\_frontalface\_alt.xml')  
  
# 实时获得摄像头数据  
ret, frame = capture.read()  
  
# 灰度化处理  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
# 识别人脸位置  
faceLands = cascade.detectMultiScale(gray, scaleFactor=1.1,  
 minNeighbors=1, minSize=(120, 120))

#### 5.2.1创建高斯朴素贝叶斯分类器

class NaiveBayes():  
 #朴素贝叶斯分类器  
  
 #创建分类器属性  
 def \_\_init\_\_(self):  
  
 self.\_X\_train = None  
 self.\_y\_train = None  
 self.\_classes = None  
 self.\_priorlist = None  
 self.\_meanmat = None  
 self.\_varmat = None  
   
 #设置分类器模型  
 def fit(self, X\_train, y\_train):  
   
 self.\_X\_train = X\_train  
 self.\_y\_train = y\_train  
 self.\_classes = np.unique(self.\_y\_train) # 得到各个类别  
 priorlist = []  
 meanmat0 = np.array([[0, 0, 0, 0]])  
 varmat0 = np.array([[0, 0, 0, 0]])  
   
 for i, c in enumerate(self.\_classes):  
 # 计算每个种类的平均值，方差，先验概率  
 X\_Index\_c = self.\_X\_train[np.where(self.\_y\_train == c)] # 属于某个类别的样本组成的“矩阵”  
 priorlist.append(X\_Index\_c.shape[0] / self.\_X\_train.shape[0]) # 计算类别的先验概率  
 X\_index\_c\_mean = np.mean(X\_Index\_c, axis=0, keepdims=True) # 计算该类别下每个特征的均值，结果保持二维状态[[3 4 6 2 1]]  
 X\_index\_c\_var = np.var(X\_Index\_c, axis=0, keepdims=True) # 方差  
 meanmat0 = np.append(meanmat0, X\_index\_c\_mean, axis=0) # 各个类别下的特征均值矩阵罗成新的矩阵，每行代表一个类别。  
 varmat0 = np.append(varmat0, X\_index\_c\_var, axis=0)  
 self.\_priorlist = priorlist  
 self.\_meanmat = meanmat0[1:, :] #除去开始多余的第一行  
 self.\_varmat = varmat0[1:, :]  
   
 #分类器模型预测  
 def predict(self,X\_test):  
   
 eps = 1e-10 # 防止分母为0  
 classof\_X\_test = [] #用于存放测试集中各个实例的所属类别  
   
 for x\_sample in X\_test:  
 matx\_sample = np.tile(x\_sample,(len(self.\_classes),1)) #将每个实例沿列拉长，行数为样本的类别数  
 mat\_numerator = np.exp(-(matx\_sample - self.\_meanmat) \*\* 2 / (2 \* self.\_varmat + eps))  
 mat\_denominator = np.sqrt(2 \* np.pi \* self.\_varmat + eps)  
 list\_log = np.sum(np.log(mat\_numerator/mat\_denominator),axis=1)# 每个类别下的类条件概率取对数后相加  
 prior\_class\_x = list\_log + np.log(self.\_priorlist) # 加上类先验概率的对数  
 prior\_class\_x\_index = np.argmax(prior\_class\_x) # 取对数概率最大的索引  
 classof\_x = self.\_classes[prior\_class\_x\_index] # 返回一个实例对应的类别  
 classof\_X\_test.append(classof\_x)  
 return classof\_X\_test  
  
 #分类器模型准确率  
 def score(self, X\_test, y\_test):  
   
 j = 0  
 for i in range(len(self.predict(X\_test))):  
 if self.predict(X\_test)[i] == y\_test[i]:  
 j += 1  
 return ('accuracy: {:.10%}'.format(j / len(y\_test)))

### 5.3识别人脸情绪

* 根据识别出的脸部特征点，裁剪出脸部图像，然后调用模型预测情绪
* 根据识别结果，用cv的rectangle在视频流上框出脸部并且用putText打上标签
* 用EMOJI盖住人脸

if len(faceLands) > 0:  
 for faceLand in faceLands:  
 x, y, w, h = faceLand  
 images = []  
 result = np.array([0.0] \* num\_class)  
  
 # 裁剪出脸部图像  
 image = cv2.resize(gray[y:y + h, x:x + w], (img\_size, img\_size))  
 image = image / 255.0  
 image = image.reshape(1, img\_size, img\_size, 1)  
  
 # 调用模型预测情绪  
 predict\_lists = model.predict\_proba(image, batch\_size=32, verbose=1)  
 result += np.array([predict for predict\_list in predict\_lists  
 for predict in predict\_list])  
 emotion = emotion\_labels[int(np.argmax(result))]  
 print("Emotion:", emotion)  
  
# 框出脸部并且写上标签  
cv2.rectangle(frame, (x - 20, y - 20), (x + w + 20, y + h + 20),  
 (0, 255, 255), thickness=10)  
cv2.putText(frame, '%s' % emotion, (x, y - 50),  
 cv2.FONT\_HERSHEY\_DUPLEX, 2, (255, 255, 255), 2, 30)  
cv2.imshow('Face', frame)  
  
# 呈现用emoji替代后的画面  
 emoji\_show = frame.copy()  
  
# 直接把emoji图片遮盖人脸会出现emoji背景变为黑色盖上去了。所以这里要蒙版处理一下，也就是保持emoji透明背景的特性，当然，这里所有图像都要归一化处理  
  
def face2emoji(face, emotion\_index, position):  
 x, y, w, h = position  
 emotion\_image = cv2.resize(emotion\_images[emotion\_index], (w, h))  
 overlay\_img = emotion\_image[:, :, :3]/255.0  
 overlay\_bg = emotion\_image[:, :, 3:]/255.0  
 background = (1.0 - overlay\_bg)  
 face\_part = (face[y:y + h, x:x + w]/255.0) \* background  
 overlay\_part = overlay\_img \* overlay\_bg  
  
 face[y:y + h, x:x + w] = cv2.addWeighted(face\_part, 255.0, overlay\_part, 255.0, 0.0)  
  
 return face