# 深度学习—人脸表情识别

摘要：

在人类的日常交流沟通中，通过人脸表情传达的信息占非常大的一部分，表情可以反映出一个人的情绪和状态。人脸表情识别是计算机视觉领域重要研究方向之一，该技术可应用于虚拟现实、人机交互以及交通安全领域方面。随着人脸识别技术的发展，研究人脸表情识别的方法也由传统的图像处理方法转变为深度学习方法，其中CNN（卷积神经网络）极大地推动了人脸表情识别的发展，TensorFlow做后端训练模型识别人类的情绪，再根据情绪选择相应的表情图片匹配。在日常工作和生活中，人们情感的表达方式主要有：语言、声音、肢体行为（如手势）、以及面部表情等。在这些行为方式中，面部表情所携带的表达人类内心情感活动的信息 为丰富，据研究表明，人类的面部表情所携带的内心活动的信息在所有的上述的形式中比例高，大约占比55%。

人类的面部表情变化可以传达出其内心的情绪变化，表情是人类内心世界的真实写照。上世纪70年代，美国著名心理学家保罗•艾克曼经过大量实验之后，将人类的基本表情定义为悲伤、害怕、厌恶、快乐、气愤和惊讶六种。同时，他们根据不同的面部表情类别建立了相应的表情图像数据库。随着研究的深入，中性表情也被研究学者加入基本面部表情中，组成了现今的人脸表情识别研究中的七种基础面部表情。



由于不同的面部表情，可以反映出在不同情景下人们的情绪变化以及心理变化，因此面部表情的识别对于研究人类行为和心理活动，具有十分重要的研究意义和实际应用价值。现如今，面部表情识别主要使用计算机对人类面部表情进行分析识别，从而分析认得情绪变化，这在人机交互、社交网络分析、远程医疗以及刑侦监测等方面都具有重要意义。

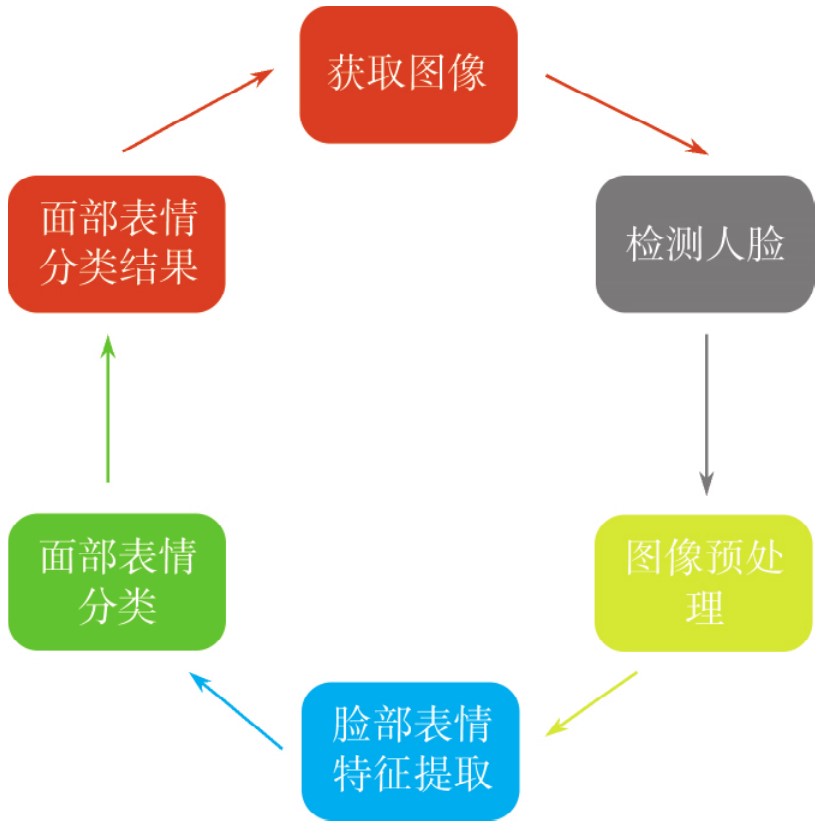
**关键词：** 人脸表情识别、深度学习、人脸识别、卷积神经网络

## 一、深度学习表情识别背景

随着科学技术的发展，传统的一些基于手工提取人脸表情图像特征的方法，因为需要研究人员具有比较丰富的经验，具有比较强的局限性，从而给人脸表情识别的研究造成了比较大的困难。随着深度学习的兴起，作为深度学习的经典代表的卷积神经网络，由于其具有自动提取人脸表情图像特征的优势，使得基于深度学习的人脸表情特征提取方法逐渐兴起，并逐步替代一些传统的人脸表情特征提取的方法。深度学习方法的主要优点在于它们可通过使用非常大的数据集进行训练学习从而获得表征这些数据的 佳功能。在深度学习中，使用卷积神经网络作为人脸表情特征提取的工具，可以更加完整的提取人脸表情特征，解决了一些传统手工方法存在的提取人脸表情特征不充足的问题。将人脸表情识别算法按照特征提取方式进行分类，其主要分为两种：一是基于传统的计算机视觉的提取算法。该类方法主要依赖于研究人员手工设计来提取人脸表情特征；二是基于深度学习的算法。该方法使用卷积神经网络，自动地提取人脸表情特征。卷积神经网络对原始图像进行简单的预处理之后，可以直接输入到网络中，使用端到端的学习方法，即不经过传统的机器学习复杂的中间建模过程，如在识别中对数据进行标注、翻转处理等，直接一次性将数据标注好，同时学习特征与进行分类，这是深度学习方法与传统方法的重要区别。相比人工的选取与设计图像特征，卷积神经网络通过自动学习的方式，获得的样本数据的深度特征信息拥有更好的抗噪声能力、投影不变性、推广与泛化能力、抽象语义表示能力。

## 二、基于深度学习的人脸表情识别研究

面部表情识别通常可以划分为四个进程。包括图像获取，面部检测，图像预处理和表情分类。其中，面部检测，脸部特征提取和面部表情分类是面部表情识别的三个关键环节面部表情识别的基本框架如下图所示。



首先是获取图像并执行面部检测，然后提取仅具有面部的图像部分。所提取的面部表情在比例和灰度上不均匀，因此有必要对面部特征区域进行分割和归一化，其中执行归一化主要是对面部光照和位置进行统一处理，将图像统一重塑为标准大小，如 48×48 像素的图片，即图像预处理。然后对脸部图像提取面部表情特征值，并进行分类。采用卷积神经网络(CNN)来完成特征提取和分类的任务，因为 CNN 是模仿人脑工作并建立卷积神经网络结构模型的著名模型，所以选择卷积神经网络作为构建模型体系结构的基础， 后不断训练，优化， 后达到较准确识别出面部表情的结果。

图像预处理采用多普勒扩展法的几何归一化分为两个主要步骤：面部校正和面部修剪。

主要目的是将图像转化为统一大小。

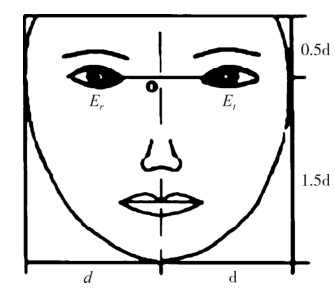
具体步骤如下:

找到特征点并对其进行标记，首先选取两眼和鼻子作为三个特征点并采用一个函数对其进行标记，

这里选择的函数是［x，y］=ginput(3)。这里重要的一点是获得特征点的坐标值，可以用鼠标进行调整。

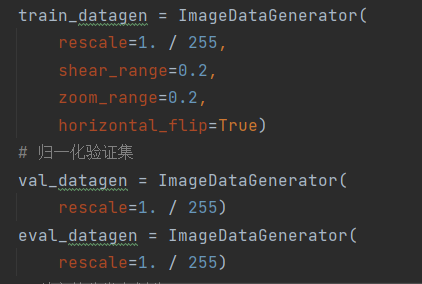
两眼的坐标值可以看作参考点，将两眼之间的距离设置为 d，找到两眼间的中点并标记为 O，然后根据参考点对图像进行旋转，这步操作是为了保证将人脸图像调到一致。

接下来以选定的 O 为基准，分别向左右两个方向各剪切距离为 d 的区域，在垂直方向剪切 0．5d 和 1．5d 的区域，这样就可以根据面部特征点和几何模型对特征区域进行确定，如下图所示：



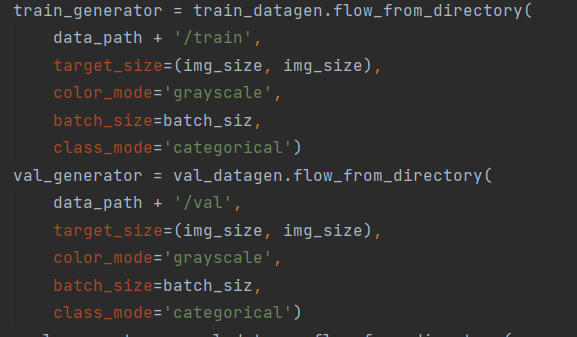
为了更好的对表情进行提取，可将表情的子区域图像裁剪成统一的 48×48 尺寸。

为了提高表情识别的性能有两种方法，一种是定义一个更深、有更多卷积操作的网络，另一种是训练更多的图片，而FER2013中供训练的图片是有限的，因此在训练时，本文使用了keras自带的ImageDataGenerator，通过实时数据增强生成张量图像数据批次，配合后面的model.fit()方法，该方法接受的第一个参数就是一个生成器，来进行训练。在处理图片时，本文对其进行了归一化处理、随机缩小、随即放大以及随机水平翻转的操作来扩充训练样本。



在处理图片的过程中，本文还会将从FER2013.csv获取的RGB图片转变为灰度图片，防止黑人和白人的肤色对模型造成影响，并且以文件分类名划分label，比如train、val、eval。

例：



## 三、基于 CNN 的人脸面部表情识别算法

我的网络架构

input--->conv1-->pool1-->conv2-->pool2--->全连接层--->输出层

input:输入层 48 × 48 × 1

conv1:卷积层 5 × 5 × 1 × 32

pool1:最大池化层 2 × 2

conv2:卷积层 3 × 3 × 32 × 64

pool2:最大池化层 2 × 2

全连接层: 包含256个神经元 输入： 12 × 12 × 64 的一维张量

输出层:包含7个神经元，对应7种表情

在这里我一开始按照书上使用的是两个卷积层、两个池化层再加上两个全连接层。但是经过训练、验证后，发现模型的精确度不是非常好，所以我改变了一下网络的架构，多加了两个卷积层，这个改变的想法取自与VGG网络架构，但是由于硬件的问题，并不是全部按照VGG的网络架构来，，改变后的网络架构如下:

input---->conv1---->conv2---->pool1---->conv3----->conv4--->pool2--->全连接层--->输出层

input:输入层 48 × 48 × 1

conv1:卷积层 3 × 3 × 32 × 64

conv2:卷积层 3 × 3 × 64 × 64

pool1:最大池化层 2 × 2

conv3:卷积层 3 × 3 × 64 × 128

conv4:卷积层 3 × 3 × 128 × 128

pool2:最大池化层 2 × 2

全连接层: 包含256个神经元 输入： 12 × 12 × 128 的一维张量

输出层:包含7个神经元，对应7种表情

经过验证，，改变后的模型准确率达到了55%,相比之前有很大的提高，但是呢，准确度并不是非常的高，如果使用更大、更多样化的训练集，干预网络参数，在进一步修改网络架构，应该可以得到更好的模型。

### 搭建模型并进行训练

import tensorflow as tf

import numpy as np

# import os, sys, inspect

from datetime import datetime

import EmotionDetectorUtils

FLAGS = tf.flags.FLAGS

tf.flags.DEFINE\_string("data\_dir", "EmotionDetector/", "Path to data files")

tf.flags.DEFINE\_string("logs\_dir", "logs/EmotionDetector\_logs/", "Path to where log files are to be saved")

tf.flags.DEFINE\_string("mode", "train", "mode: train (Default)/ test")

BATCH\_SIZE = 128

LEARNING\_RATE = 0.001 # 学习率

MAX\_ITERATIONS = 1001 # 最大迭代次数

REGULARIZATION = 1e-2 # 正则项参数大小

IMAGE\_SIZE = 48 # 图像大小

NUM\_LABELS = 7 # 输出的类别数量

VALIDATION\_PERCENT = 0.1

# 添加正则项

def add\_to\_regularization\_loss(W, b):

tf.add\_to\_collection("losses", tf.nn.l2\_loss(W))

tf.add\_to\_collection("losses", tf.nn.l2\_loss(b))

# 按照传递进来的shape形状初始化权重

def weight\_variable(shape, stddev=0.02, name=None):

initial = tf.truncated\_normal(shape, stddev=stddev)

if name is None:

return tf.Variable(initial)

else:

return tf.get\_variable(name, initializer=initial)

# 初始化偏差

def bias\_variable(shape, name=None):

initial = tf.constant(0.0, shape=shape)

if name is None:

return tf.Variable(initial)

else:

return tf.get\_variable(name, initializer=initial)

def conv2d\_basic(x, W, bias):

conv = tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding="SAME")

return tf.nn.bias\_add(conv, bias)

# 池化层操作

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1],

strides=[1, 2, 2, 1], padding="SAME")

# 模型的实现

def emotion\_cnn(dataset):

print("input dataset's shape-->", dataset.shape)

with tf.name\_scope("conv1") as scope:

tf.summary.histogram("W\_conv1", weights['wc1'])

tf.summary.histogram("b\_conv1", biases['bc1'])

conv\_1 = tf.nn.conv2d(dataset, weights['wc1'],

strides=[1, 1, 1, 1], padding="SAME")

print("conv\_1's shape--->", conv\_1.shape)

h\_conv1 = tf.nn.bias\_add(conv\_1, biases['bc1'])

h\_1 = tf.nn.relu(h\_conv1)

# h\_pool1 = max\_pool\_2x2(h\_1)

# print("h\_pool1 shape-->", h\_pool1.shape)

add\_to\_regularization\_loss(weights['wc1'], biases['bc1'])

with tf.name\_scope("conv2") as scope:

tf.summary.histogram("W\_conv2", weights['wc2'])

tf.summary.histogram("b\_conv2", biases['bc2'])

# conv\_2 = tf.nn.conv2d(h\_pool1, weights['wc2'], strides=[1, 1, 1, 1], padding="SAME")

conv\_2 = tf.nn.conv2d(h\_1, weights['wc2'], strides=[1, 1, 1, 1], padding="SAME")

print("conv\_2's shape--->", conv\_2.shape)

h\_conv2 = tf.nn.bias\_add(conv\_2, biases['bc2'])

h\_2 = tf.nn.relu(h\_conv2)

h\_pool2 = max\_pool\_2x2(h\_2)

add\_to\_regularization\_loss(weights['wc2'], biases['bc2'])

with tf.name\_scope("conv3") as scope:

tf.summary.histogram("W\_conv3", weights['wc3'])

tf.summary.histogram("b\_conv3", biases['bc3'])

conv\_3 = tf.nn.conv2d(h\_pool2, weights['wc3'], strides=[1, 1, 1, 1], padding="SAME")

print("conv\_3 shape-->", conv\_3.shape)

h\_conv3 = tf.nn.bias\_add(conv\_3, biases['bc3'])

h\_3 = tf.nn.relu(h\_conv3)

# h\_pool3 = max\_pool\_2x2(h\_3)

# print("h\_pool3 shape-->", h\_pool3.shape)

add\_to\_regularization\_loss(weights['wc3'], biases['bc3'])

with tf.name\_scope("conv4") as scope:

tf.summary.histogram("W\_conv4", weights['wc4'])

tf.summary.histogram("b\_conv4", biases['bc4'])

# conv\_4 = tf.nn.conv2d(h\_pool3, weights['wc4'], strides=[1, 1, 1, 1], padding="SAME")

conv\_4 = tf.nn.conv2d(h\_3, weights['wc4'], strides=[1, 1, 1, 1], padding="SAME")

print("conv\_4 shape-->", conv\_4.shape)

h\_conv4 = tf.nn.bias\_add(conv\_4, biases['bc4'])

h\_4 = tf.nn.relu(h\_conv4)

h\_pool4 = max\_pool\_2x2(h\_4)

print("h\_pool4 shape-->", h\_pool4.shape)

add\_to\_regularization\_loss(weights['wc4'], biases['bc4'])

with tf.name\_scope("fc\_1") as scope:

prob = 0.5

image\_size = IMAGE\_SIZE // 4

h\_flat = tf.reshape(h\_pool4, [-1, image\_size \* image\_size \* 128])

print("h\_flat shape--->", h\_flat.shape)

tf.summary.histogram("W\_fc1", weights['wf1'])

tf.summary.histogram("b\_fc1", biases['bf1'])

h\_fc1 = tf.nn.relu(tf.matmul(h\_flat, weights['wf1']) + biases['bf1'])

print("h\_fc1'shape--->", h\_fc1.shape)

h\_fc1\_dropout = tf.nn.dropout(h\_fc1, prob)

print("h\_fc1\_dropout shape-->", h\_fc1\_dropout.shape)

with tf.name\_scope("fc\_2") as scope:

tf.summary.histogram("W\_fc2", weights['wf2'])

tf.summary.histogram("b\_fc2", biases['bf2'])

pred = tf.matmul(h\_fc1\_dropout, weights['wf2']) + biases['bf2']

print("pred shape-->", pred.shape)

return pred

weights = {

'wc1': weight\_variable([3, 3, 1, 64], name="W\_conv1"),

'wc2': weight\_variable([3, 3, 64, 64], name="W\_conv2"),

'wc3': weight\_variable([3, 3, 64, 128], name="W\_conv3"),

'wc4': weight\_variable([3, 3, 128, 128], name="W\_conv4"),

'wf1': weight\_variable([(IMAGE\_SIZE // 4) \* (IMAGE\_SIZE // 4) \* 128, 256], name="W\_fc1"),

'wf2': weight\_variable([256, NUM\_LABELS], name="W\_fc2")

}

biases = {

'bc1': bias\_variable([64], name="b\_conv1"),

'bc2': bias\_variable([64], name="b\_conv2"),

'bc3': bias\_variable([128], name="b\_conv3"),

'bc4': bias\_variable([128], name="b\_conv4"),

'bf1': bias\_variable([256], name="b\_fc1"),

'bf2': bias\_variable([NUM\_LABELS], name="b\_fc2")

}

def loss(pred, label):

cross\_entropy\_loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits=pred, labels=label))

tf.summary.scalar('Entropy', cross\_entropy\_loss)

reg\_losses = tf.add\_n(tf.get\_collection("losses"))

# tf.summary.scalar('Reg\_loss', reg\_losses)

return cross\_entropy\_loss + REGULARIZATION \* reg\_losses

def train(loss, step):

return tf.train.AdamOptimizer().minimize(loss, global\_step=step)

def get\_next\_batch(images, labels, step):

offset = (step \* BATCH\_SIZE) % (images.shape[0] - BATCH\_SIZE)

batch\_images = images[offset: offset + BATCH\_SIZE]

batch\_labels = labels[offset:offset + BATCH\_SIZE]

return batch\_images, batch\_labels

# 入口函数

def main(argv=None):

# 获得数据

train\_images, train\_labels, valid\_images, valid\_labels, test\_images = EmotionDetectorUtils.read\_data(FLAGS.data\_dir)

print("Train size: %s" % train\_images.shape[0])

print('Validation size: %s' % valid\_images.shape[0])

print("Test size: %s" % test\_images.shape[0])

# 定义global\_step变量追踪当前已进行优化迭代次数,trainable=Flase意为Tensorflow不会试图优化该变量

global\_step = tf.Variable(0, trainable=False)

dropout\_prob = tf.placeholder(tf.float32)

# 为输入图像定义占位符变量 None表示该张量可以载入任意数量的图像，每个图像高和宽都为IMAGE\_SIZE像素，颜色通达数为1

input\_dataset = tf.placeholder(tf.float32, [None, IMAGE\_SIZE, IMAGE\_SIZE, 1], name="input")

# 为输入input\_data中的图像真实标签定义占位符变量

input\_labels = tf.placeholder(tf.float32, [None, NUM\_LABELS])

# 获得网络输出

pred = emotion\_cnn(input\_dataset)

# output\_pred变量为预测结果，用于网络的测试和验证

output\_pred = tf.nn.softmax(pred, name="output")

correct\_prediction = tf.equal(tf.argmax(output\_pred, 1), tf.argmax(input\_labels, 1))

# tf.cast(x, dtype, name=None) 将输入转换成dtype的类型

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

tf.summary.scalar("accuracy", accuracy)

# loss\_val变量为预测的类（pred）和输入图像的真实类（input\_labels)之间的的误差

loss\_val = loss(pred, input\_labels)

# 获得优化器对象实例

train\_op = train(loss\_val, global\_step)

# 定义summary\_op变量用于Tensorboard可视化

summary\_op = tf.summary.merge\_all()

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

summary\_writer = tf.summary.FileWriter(FLAGS.logs\_dir, sess.graph\_def)

# 定义saver变量，以存储该模型

saver = tf.train.Saver()

ckpt = tf.train.get\_checkpoint\_state(FLAGS.logs\_dir)

if ckpt and ckpt.model\_checkpoint\_path:

saver.restore(sess, ckpt.model\_checkpoint\_path)

print("Model Restored!")

# 开始训练

for step in range(MAX\_ITERATIONS):

# 获得batch\_size大小的一批训练样本

batch\_image, batch\_label = get\_next\_batch(train\_images, train\_labels, step)

# print("batch image's shape--->", batch\_image.shape)

feed\_dict = {input\_dataset: batch\_image, input\_labels: batch\_label}

# 运行优化器，将对应占位符的数据传递进去

sess.run(train\_op, feed\_dict=feed\_dict)

if step % 10 == 0:

train\_loss, summary\_str = sess.run([loss\_val, summary\_op], feed\_dict=feed\_dict)

summary\_writer.add\_summary(summary\_str, global\_step=step)

print("Training Loss: %f" % train\_loss)

# 当运行步数为100的倍数时，在验证集上验证训练出的模型，并且保存该模型

if step % 100 == 0:

valid\_loss = sess.run(loss\_val, feed\_dict={input\_dataset: valid\_images, input\_labels: valid\_labels})

print("%s Validation Loss: %f" % (datetime.now(), valid\_loss))

print("Accuracy: ", accuracy.eval(feed\_dict={input\_dataset: valid\_images, input\_labels: valid\_labels}))

saver.save(sess, FLAGS.logs\_dir + 'model.ckpt', global\_step=step)

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

tf.app.run()

还有一个辅助的py文件

from scipy import misc

import numpy as np

import matplotlib.cm as cm

import tensorflow as tf

import os, sys, inspect

from datetime import datetime

from matplotlib import pyplot as plt

import matplotlib.image as mpimg

from scipy import misc

import EmotionDetectorUtils

from EmotionDetectorUtils import testResult

import time

def rbg\_to\_gray(RGB\_JPG):

"""

将RGB彩色图像转换为灰度图像

:param RGB\_JPG: 输入的图像

:return: 灰度图像

"""

# print(type(RGB\_JPG))

return np.dot(RGB\_JPG[..., :3], [0.299, 0.587, 0.114])

l = []

image\_list = []

# 读取图像

img = mpimg.imread('myself.jpg')

img2 = mpimg.imread('author\_img.jpg')

img3 = mpimg.imread('test.jpg')

img4 = mpimg.imread('test1.jpg')

img5 = mpimg.imread('test2.jpg')

image\_list.append(img)

image\_list.append(img2)

image\_list.append(img3)

image\_list.append(img4)

image\_list.append(img5)

for img in image\_list:

# 将读入的彩色图像转换为灰度图像

gray = rbg\_to\_gray(img)

l.append(gray)

sess = tf.InteractiveSession()

# 调用之前保存的模型

new\_saver = tf.train.import\_meta\_graph('logs/EmotionDetector\_logs/model.ckpt-1000.meta')

new\_saver.restore(sess, 'logs/EmotionDetector\_logs/model.ckpt-1000')

tf.get\_default\_graph().as\_graph\_def()

x = sess.graph.get\_tensor\_by\_name("input:0")

y\_conv = sess.graph.get\_tensor\_by\_name("output:0")

tResult = testResult()

num\_evaluation = 1000

for img\_gray in l:

image\_test = np.resize(img\_gray, (1, 48, 48, 1))

# 展示图片

plt.imshow(img\_gray, cmap=plt.get\_cmap('gray'))

plt.show()

print("开始训练")

start\_time = time.time()

for i in range(0, num\_evaluation):

result = sess.run(y\_conv, feed\_dict={x: image\_test})

label = sess.run(tf.argmax(result, 1))

label = label[0]

label = int(label)

tResult.evaluate(label)

end\_time = time.time()

tResult.display\_result(num\_evaluation)

print("用时----> %s 秒" % (end\_time - start\_time))



## 四、总结

如果你想要一个效果很好的面部表情识别模型，你可以尝试去扩充数据集，或者改进模型的网络架构，更换更好的超参数等等。不过在做这些事情之前可以先根据损失函数的图像来判断模型当前是处于高偏差还是高方差的状态如果你想要一个效果很好的面部表情识别模型，你可以尝试去扩充数据集，或者改进模型的网络架构，更换更好的超参数等等。不过在做这些事情之前可以先根据损失函数的图像来判断模型当前是处于高偏差还是高方差的状态本章讲述了本次实验的环境、实验的过程以及效果，成功实现了深度学习下人脸表情的自动识别。人脸表情识别技术将在我们的日常生活中扮演越来越重要的角色，尤其是深度学习技术在不断取得进步，很多计算机视觉领域的任务都选择基于深度学习技术和卷积神经网络。因此将深度学习技术应用在解决人脸表情识别任务中是十分必要的

# 参考文献

1. 许诺.基于深度学习的人脸表情识别研究及应用[J].重庆大学,2019.6
2. 陈运军,田正卫.keras框架的人脸识别可视化参数调整[J].单片机与嵌入式系统应用,2020,20(09):4-5+10
3. 卢官明,何嘉利,闫静杰,李海波.一种用于人脸表情识别的卷积神经网络[J].南京邮电大学学报(自然科学版),2016,36(01):16-22
4. 王恒涛.基于TensorFlow、Keras与OpenCV的图像识别集成系统[J].电子测试,2020(24):53-54+124.