

## 人脸关键点检测实验报告

### 一、 实验要求

使用 (96, 96) 的灰度图进行 15 个人脸关键点的预测。使用自己编写的模型对训练集数据进行预测，将预测结果使用提供的函数导出到 csv 文件。

### 二、 数据分析

#### 1、数据集介绍：

训练集包含人脸图像以及相应的关键点坐标；

测试集只包含人脸图像。

2、已有的代码：读取数据集，输入标签分离，误差计算，图像和关键点可视化，结果导出代码。

根据学过的机器学习处理流程，不需要再做数据预处理，根据模型的训练结果以及现实需求，特征工程可选，必须要做的是模型的训练和保存，用训练好的模型对测试集进行预测和结果保存。

### 三、 题目分析

根据人脸图像预测关键点，属于预测问题，常用的模型有回归模型，神经网络等，可以进行集成学习。回归模型、神经网络都是以向量为输入来训练模型，但是将图片像素矩阵输入传统的 BP 神经网络，会产生很多参数，而且图像局部的关系不能很好的体现出来，所以想到用卷积神经网络：CNN 的目的是以一定的模型对事物进行特征提取，而后根据特征对该事物进行分类、识别、预测或决策等。局部感知即卷积神经网络提出每个神经元不需要感知图像中的全部像素，只对图

像的局部像素进行感知，然后在更高层（通常在卷积网络最后通过全连接层输出），将这些感受不同局部的神经元综合起来可得到全局信息。每一个卷积核在遍历整个图像的时候，卷积核的参数是固定不变的，每个卷积核都会扫描整个图像，在扫描的过程中，过滤器的参数值是固定不变的，即整个图像的所有元素都“共享”了相同的权值。也正是由于权值共享，卷积神经网络在大大减少参数量的同时，依然能够有效地提取图像的特征。

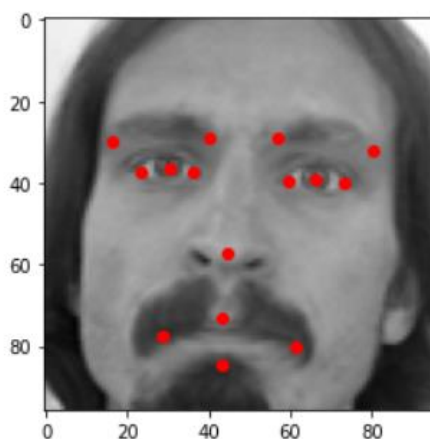
#### 四、 实验过程

1、 研究数据：使用 prepare\_data.py 读取数据集并进行标签分离  
训练集

```
[2]: X_train.shape      y_train.shape
[2]: (6001, 96, 96, 1)  (6001, 30)
```

观察数据：

```
show_result(X_train[0].reshape(96,96), y_train[0])
```



将训练数据集按 0.75:0.25 切分成训练集和验证集

```
1: from sklearn.model_selection import train_test_split # 载入数据分割函数train_test_split
   X_t,X_v,y_t,y_v=train_test_split(X_train,y_train)
   v = 0.25
```

2、 训练：TensorFlow 实现卷积神经网络

- (1) 重点：卷积核的使用：卷积核大小，步长，填充值；卷积核的特征数量。

TensorFlow 的简单学习：程序分为两个独立的部分，计算图的定义和其执行。

1) 计算图的定义：包含节点和边的网络。定义计算图要使用数据，即张量 (tensor)、对象 (常量、变量和占位符)，以及运算操作对象 (Operation Object, 简称 OP)。计算图定义神经网络的蓝图，但其中的张量还没有相关的数值。

2) 计算图的执行：使用会话对象来实现计算图的执行。会话对象封装了评估张量和操作对象的环境。这里真正实现了运算操作并将信息从网络的一层传递到另外一层。不同张量对象的值仅在会话对象中被初始化、访问和保存。在此之前张量对象只被抽象定义，在会话中才被赋予实际的意义。

- (2) **第一版**:使用课堂作业中采用 CNN 识别手写体的 TensorFlow 模型，即经典模型：LeNet 模型。

遇到的问题：

- a. 自定义数据集不能使用 `batch = mnist.train.next_batch(50)` 随机产生 50 组数据，通过网上搜索自定义了一种随机产生 50 组数据的方法：

```
def next_batch(train, target, batch_size):
    length=len(train)
    index=[i for i in range(length)]
    np.random.shuffle(index)
    cnt=length/batch_size+1
    while cnt>0:
        batch_x=[]
        batch_y=[]
        try:
            for i in range(batch_size):
                batch_x.append(train[index[i]])
                batch_y.append(target[index[i]])
                index.remove(index[i])
            except:
                index=[i for i in range(length)]
                continue

        yield (batch_x, batch_y)
a=next_batch(X_t, y_t, 50)
```

batch = next(a)等价于 batch = mnist.train.next\_batch(50)

b.在会话内部使用的函数算术运算必须使用 tf 定义的方法，不能使用 numpy 库的方法。

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

评估方法：RMSE

```
def calculate_mse(predict, label):
    mse_array = tf.reduce_mean((predict - label)**2)
    return tf.sqrt(mse_array)
```

CNN 卷积操作	输入图像 shape	激活函数	卷积核 shape	步长	输出图像 shape
第一层卷积	[batch,96,96,1]	ReLU	[3,3,1,32]	[1,2,2,1]	[batch,48,48,32]
第二层卷积	[batch,48,48,32]	ReLU	[3,3,32,64]	[1,2,2,1]	[batch,24,24,64]
MLP 全连接网络	输入 shape	激活函数	权重 shape	偏值 shape	神经元个数
第一层全连接	[batch,24*24*64]	ReLU	[24*24*64,1024]	[1024]	1024

CNN 卷积操作	输入图像 shape	激活函数	卷积核 shape	步长	输出图像 shape
第二层全连接 (keep_prob=0.5)	[batch,1024]	无	[1024,30]	[30]	30

主网络				
获取输入	占位符	类型: <code>tf.float32</code>	shape: [batch,96,96,1]	
获取标签	占位符	类型: <code>tf.float32</code>	shape: [batch,30]	
前向结构	获取卷积输出	[batch,24,24,64]		
	更改形状	[batch,24*24*64]		
	获取全连接输出	[batch,30]		
后向结构	损失	全连接输出	获取的标签	RMSE

训练结果：

```
(?, 24, 24, 64)
step 0, 测试均方误差 1008.19
step 100, 测试均方误差 39.6428
step 200, 测试均方误差 37.5549
step 300, 测试均方误差 31.4368
step 400, 测试均方误差 24.6265
step 500, 测试均方误差 22.6729
step 600, 测试均方误差 24.3214
step 700, 测试均方误差 20.5142
step 800, 测试均方误差 12.2954
step 900, 测试均方误差 17.0829
step 1000, 测试均方误差 16.7811
step 1100, 测试均方误差 19.6788
step 1200, 测试均方误差 14.2524
step 1300, 测试均方误差 20.3158
step 1400, 测试均方误差 20.2525
step 1500, 测试均方误差 14.7504
step 1600, 测试均方误差 15.7516
step 1700, 测试均方误差 11.7437
step 1800, 测试均方误差 14.2882
step 1900, 测试均方误差 12.7946
step 2000, 测试均方误差 15.3435
step 2100, 测试均方误差 7.67268
step 2200, 测试均方误差 11.9616
step 2300, 测试均方误差 18.8324
step 2400, 测试均方误差 5.93774
step 2500, 测试均方误差 7.13806
step 2600, 测试均方误差 13.2336
step 2700, 测试均方误差 7.53107
step 2800, 测试均方误差 12.3918
```

```

step 8600, 测试均方误差 9.44787
step 8700, 测试均方误差 6.9091
step 8800, 测试均方误差 12.4653
step 8900, 测试均方误差 6.67283
step 9000, 测试均方误差 5.90817
step 9100, 测试均方误差 5.66803
step 9200, 测试均方误差 9.97671
step 9300, 测试均方误差 8.04268
step 9400, 测试均方误差 12.313
step 9500, 测试均方误差 13.8313
step 9600, 测试均方误差 16.2908
step 9700, 测试均方误差 8.77708
step 9800, 测试均方误差 11.4683
step 9900, 测试均方误差 7.72119
验证集的均方误差: 8.94039

```

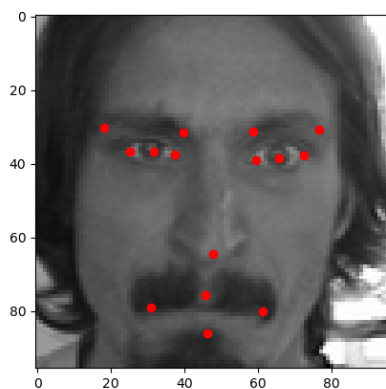
结果分析：得到的 RMSE 很大，与预期的结果不相符，考虑到 mnist 手写图片的像素是 28\*28，较简单，分类准确率很高，而人脸图像是 96\*96，所以我认为可能是卷积过程过于简单，决定调整卷积核的参数和卷积过程。

## 第二版：

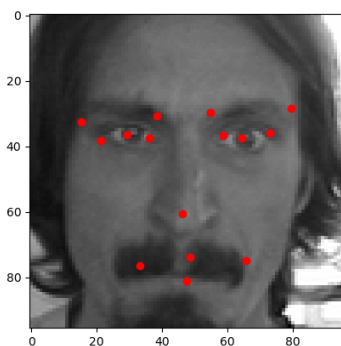
CNN 卷积操作	输入图像 shape	激活函数	卷积核 shape	步长	输出图像 shape
第一层卷积	[batch,96,96,1]	ReLU	[3,3,1,32]	[1,2,2,1]	[batch,48,48,32]
第二层卷积	[batch,48,48,32]	ReLU	[2,2,32,64]	[1,2,2,1]	[batch,24,24,64]
第三次卷积	[batch,24,24,64]	ReLU	[2,2,64,128]	[1,2,2,1]	[batch,12,12,64]
MLP 全连接网络	输入 shape	激活函数	权重 shape	偏值 shape	神经元个数
第一层全连接	[batch,12*12*128]	ReLU	[12*12*128,500]	[500]	500
第二层全连接 (keep_prob=0.5)	[batch,500]	无	[500,30]	[30]	30

训练结果：

step 7700,测试均方误差 3.63558  
step 7800,测试均方误差 2.78834  
step 7900,测试均方误差 2.60424  
step 8000,测试均方误差 3.12911  
step 8100,测试均方误差 2.90724  
step 8200,测试均方误差 3.20685  
step 8300,测试均方误差 3.08192  
step 8400,测试均方误差 4.05784  
step 8500,测试均方误差 2.68798  
step 8600,测试均方误差 2.79263  
step 8700,测试均方误差 3.23959  
step 8800,测试均方误差 3.18789  
step 8900,测试均方误差 2.78499  
step 9000,测试均方误差 3.29531  
step 9100,测试均方误差 2.70647  
step 9200,测试均方误差 2.67306  
step 9300,测试均方误差 2.90238  
step 9400,测试均方误差 3.11177  
step 9500,测试均方误差 2.82356  
step 9600,测试均方误差 2.62549  
step 9700,测试均方误差 2.75791  
step 9800,测试均方误差 2.51692  
step 9900,测试均方误差 3.24604  
验证集的均方误差: 2.85983



原图:



预测的结果:

结果分析: 和上一版相比, 误差减少了一半, 效果显著, 但是还是没有达到想象的预期结果零点几。

第三版：这次增加迭代次数：20000 次，与 10000 次比较发现，没有明显的改进，所以舍弃这种思路。

第四版：网上搜索资料：在第二版基础上增加卷积层数，每个卷积层卷积两次再进行池化。发现结果也没有明显的变化，而且预测验证集时还因为内存不够退出，所以也舍弃这种思路。

```
W_conv1 = weight_variable([3, 3, 1, 32])
b_conv1 = bias_variable([32])
# x_images = tf.reshape(x, [-1, 28, 28, 1])# -1 根据输入的实际情况判断,

h_conv1 = tf.nn.relu(conv2d(x, W_conv1) + b_conv1)
W_conv11 = weight_variable([3, 3, 32, 32])
b_conv11 = bias_variable([32])
# x_images = tf.reshape(x, [-1, 28, 28, 1])# -1 根据输入的实际情况判断,

h_conv11 = tf.nn.relu(conv2d(h_conv1, W_conv11) + b_conv11)
h_pool1 = max_pool_2x2(h_conv11)

# 第二层卷积
W_conv2 = weight_variable([2, 2, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
W_conv22 = weight_variable([2, 2, 64, 64])
b_conv22 = bias_variable([64])

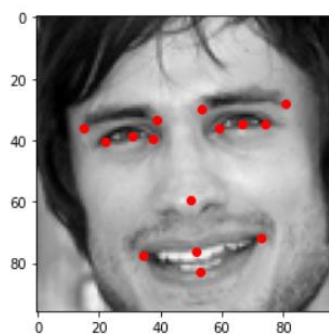
h_conv22 = tf.nn.relu(conv2d(h_conv2, W_conv22) + b_conv22)
h_pool2 = max_pool_2x2(h_conv22)
#print(h_pool2.shape)
#第三次卷积
W_conv3 = weight_variable([2, 2, 64, 128])
b_conv3 = bias_variable([128])

h_conv3 = tf.nn.relu(conv2d(h_pool2, W_conv3) + b_conv3)
W_conv33 = weight_variable([2, 2, 128, 128])
b_conv33 = bias_variable([128])

h_conv33 = tf.nn.relu(conv2d(h_conv3, W_conv33) + b_conv33)
h_pool3 = max_pool_2x2(h_conv33)
```

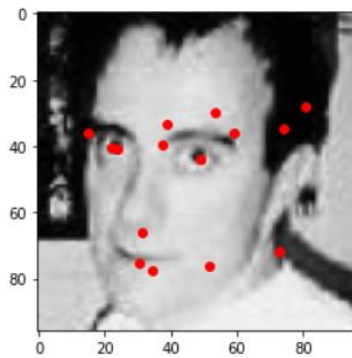
发现没有思路的时候，重新回去观察数据及特征：观察预测出的人脸加关键点的图像，并与实际作对比发现实际的关键点也并不符合人脸特征，通过观察原数据发现 2284-6001 的图像的关键点都有部分缺失的情况，

```
show_result(X_train[2283].reshape(96, 96), y_train[2283])
```





```
show_result(X_train[2284].reshape(96,96), y_train[2284])
```



第五版：只用前 2284 条正常数据进行训练，发现结果更差了，推测充足的数据是提高训练能力的重要因素。舍弃这个思路。

经同学提示发现是助教给的数据集已经处理过缺失值，采用的是填充上一条数据对应的值，所以用来训练数据并不完全正确，所以误差过大是有依据的，因为数据集已经填充好了缺失值，而不是读取时用代码填充的，所以缺失值处理部分我们不能再用其他的处理方法，猜测模型无论如何训练也达不到零点几的均方误差。所以就不再纠结如何训练出最小化误差的模型。

```
# use the previous value to fill the missing value  
train_data.fillna(method='ffill', inplace=True)
```

### 3、 保存模型

TensorFlow 提供了一个非常方便的 api, `tf.train.Saver()` 来保存和还原一个机器学习模型。程序会生成并保存四个文件:

checkpoint 文本文件，记录了模型文件的路径信息列表

mnist-10000.data-00000-of-00001 网络权重信息

mnist-10000.index .data 和.index 这两个文件是二进制文件，保存了模型中的变量参数（权重）信息

mnist-10000.meta 二进制文件，保存了模型的计算图结构信息（模型的

网络结构) protobuf

保存：想要保存需要的中间变量时，需要用 name 将变量初始化，不能省略，这是恢复模型的关键，Tensorflow 命名空间仍然存在于保存的文件中。

```
x = tf.placeholder("float", shape=[None, 9216], name="x")
y_ = tf.placeholder("float", shape=[None, 30], name='y_')

keep_prob = tf.placeholder("float", name="keep_prob")

saver = tf.train.Saver(max_to_keep=4)
ckpt_file_path = "D:/models9/mnist"
path = os.path.dirname(os.path.abspath(ckpt_file_path))
if os.path.isdir(path) is False:
    os.makedirs(path)
sess.run(tf.global_variables_initializer())

print("step %d, 测试均方误差 %g"%(i, train_accuracy))
if i%1000==0:
    tf.train.Saver().save(sess, ckpt_file_path, write_meta_graph=True)
```

加载：tf.get\_default\_graph()相当于在一张纸上画了一个计算流程图，现在我们需要在图上找到输入输出口。输入口就是我们之前定义的占位符，函数 get\_tensor\_by\_name 就是找到你之前定义的张量入口。出口同理。然后就可以开始喂数据进行预测了。

```
sess = tf.Session()

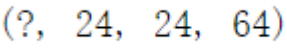
#加载模型
saver = tf.train.import_meta_graph('D:/models9/mnist.meta')
saver.restore(sess, tf.train.latest_checkpoint('D:/models9'))

graph = tf.get_default_graph()
x_ = graph.get_tensor_by_name("x:0")
keep_prob_ = graph.get_tensor_by_name("keep_prob:0")
#y = graph.get_tensor_by_name("y_:0")
# Now, access the op that you want to run.
yr = graph.get_tensor_by_name("y_conv:0")

y_conv=sess.run(yr, feed_dict={x_: X_v, keep_prob_:1.0})
```

遇到的问题：

1) 不了解保存模型的原理等，出现运行错误，比如加载模型时未初始化变量等花费了很长时间，训练保存了多个模型，比如

训练模型时出现以下 warning  没有处理，加载模型时就出错了，所以将模型的输入改为

```
# 定义输入：构建 session，在云平台上，运行快速的时候超过1000
x = tf.placeholder("float", shape=[None, 96, 96, 1])
y_ = tf.placeholder("float", shape=[None, 30])

# 定义输入：构建 session，在云平台上，运行快速的时候超过1000
x = tf.placeholder("float", shape=[None, 9216], name="x")
y_ = tf.placeholder("float", shape=[None, 30], name="y_")

def calculate_mse(predict, label):
    mse_array = tf.reduce_mean((predict - label)**2)
    return tf.sqrt(mse_array)

# 定义变量初始化
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

# 定义卷积和池化操作
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

# 第一层卷积
W_conv1 = weight_variable([3, 3, 1, 32])
b_conv1 = bias_variable([32])
x_images = tf.reshape(x, [-1, 96, 96, 1]) # -1 根据输入的实际情况判断，比如50张图片就是 50*784=>50, 28, 28, 1
```

2) 训练了十一版模型最终才有两个可以用的，多次修改代码，修改迭代次数，才发现错误其实是因为使用 jupyter 训练模型时内存中（应该是）有上一训练部分内容的保存，（猜测和 TensorFlow 的 session 有关）所以当重启 jupyter 再训练模型后，重载的模型才可以用。

3) 原代码：`y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2`，因为保存模型变量要用 name 初始化，所以只能修改为 `y_conv = tf.nn.relu(tf.matmul(h_fc1_drop, W_fc2) + b_fc2, name="y_conv")`，但是多次训练后发现误差会增加 1 左右，没有找到合适的解决方案。

```
step 8900, 测试均方误差 3.42919
step 9000, 测试均方误差 3.08801
9000
step 9100, 测试均方误差 2.79505
step 9200, 测试均方误差 2.86611
step 9300, 测试均方误差 2.49539
step 9400, 测试均方误差 2.93336
step 9500, 测试均方误差 2.502
step 9600, 测试均方误差 2.91693
step 9700, 测试均方误差 3.21037
step 9800, 测试均方误差 2.75111
step 9900, 测试均方误差 2.76774
step 10000, 测试均方误差 3.4258
10000
验证集的均方误差: 3.91522
```

## 五、实验心得与改进思路

实验心得:

做人脸关键点是为了通过人脸关键点做人脸对齐,以便优化后续的人脸识别。

掌握了 CNN 的基本原理,卷积的过程,卷积核的使用。学会了用 TensorFlow 设计 CNN,以及如何保存和加载 TensorFlow 模型。卷积神经网络 CNN 主要用来识别位移、缩放及其他形式扭曲不变性的二维图形。由于 CNN 的特征检测层通过训练数据进行学习,所以在使用 CNN 时,避免了显式的特征抽取,而隐式地从训练数据中进行学习;再者由于同一特征映射面上的神经元权值相同,所以网络可以并行学习,这也是卷积网络相对于神经元彼此相连网络的一大优势。卷积神经网络以其局部权值共享的特殊结构在语音识别和图像处理方面有着独特的优越性,其布局更接近于实际的生物神经网络,权值共享降低了网络的复杂性,特别是多维输入向量的图像可以直接输入网络这一特点避免了特征提取和分类过程中数据重建的复杂度。

TensorFlow 需要学的东西还有很多,只用充分了解背后的原理才能熟

练的使用。

改进思路：

- 1、 缺失值处理可以采用删除缺失值数据的方法、sklearn 的随机森林填充。
- 2、 可以对初始  $96 \times 96$  的图像进行翻转，镜像等处理，增加特征维度。
- 3、 超参数：CNN 模型中，不论是卷积核大小还是全连接层的神经元数，这些超参数并没有经过特别的调优，可以尝试通过调整超参数来进一步提高精确度。
- 4、 资源允许，可以使用 k 折交叉方法集成训练模型。