In [1]:

```
import pandas as pd
import numpy as np
from PIL import Image

MasterFile = pd.read_csv("/clubear/Lecture 2.1 - Linear Regression by TensorFlow/data/faces/FaceScore.csv")
```

数据准备

In [2]:

```
FileNames=MasterFile['Filename']
N=len(FileNames)
IMSIZE=128
X=np.zeros([N,IMSIZE,IMSIZE,3])
for i in range(N):
    MyFile=FileNames[i]
    Im=Image.open('/clubear/Lecture 2.1 - Linear Regression by TensorFlow/data/faces/images/'+MyFile)
    Im=Im.resize([IMSIZE,IMSIZE])
    Im=np.array(Im)/255
    X[i,]=Im
```

In [3]:

```
list_y = [] #创建一个空的list
for i in range(N):
    MyFile = FileNames[i] #提取Filenames
    if MyFile[0] == "f":
          list_y.append(0)
    elif MyFile[0] == "m":
          list_y.append(1)
#如果filename 第一个字母是f，添加0在list里；反之则添加1  （用0和1表示性别）
Y = np.asarray(list_y)
Y
```

Out[3]:

```
array([0, 0, 0, ..., 1, 1, 1])
```

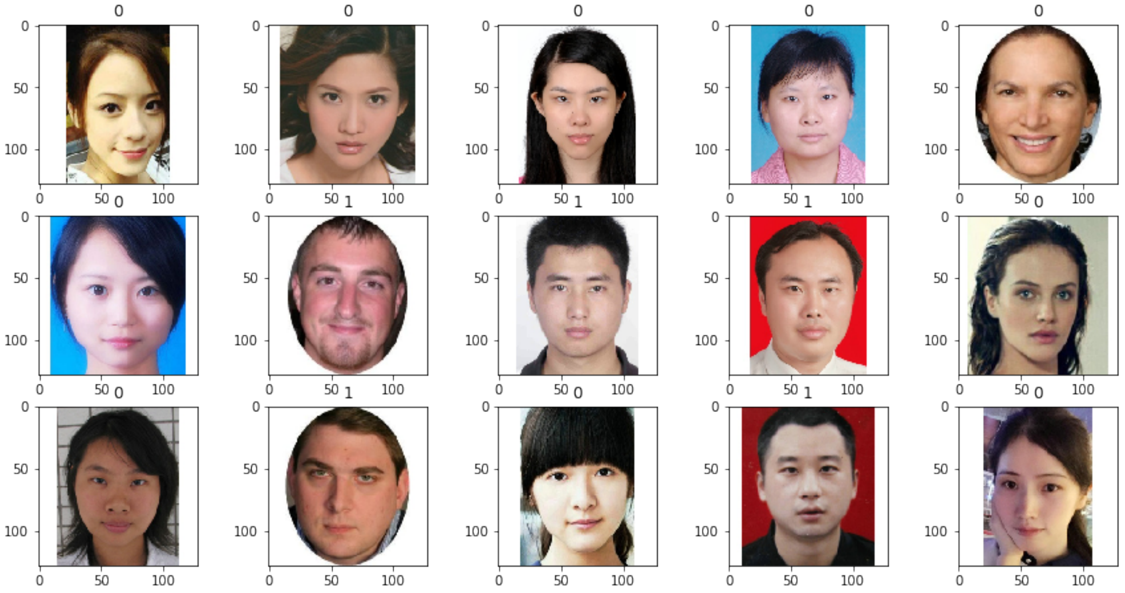
In [4]:

```
from sklearn.model_selection import train_test_split
X0,X1,Y0,Y1=train_test_split(X,Y,test_size=0.3,random_state=233)
#固定seed为233, train: test = 7:3
```

In [5]:

```
from matplotlib import pyplot as plt
plt.figure()
fig,ax=plt.subplots(3,5)
fig.set_figheight(7.5)
fig.set_figwidth(15)
ax=ax.flatten()
for i in range(15):
    ax[i].imshow(X0[i,:,:,:])
    ax[i].set_title(Y0[i])
#查看数据是否consistent ie 0和1的添加是否对应
```

```
<Figure size 432x288 with 0 Axes>
```



产生One-Hot型因变量

In [7]:

```
from keras.utils import to_categorical
YY0 = to_categorical(Y0)
YY1 = to_categorical(Y1)
YY1
```

```
Using TensorFlow backend.
```

Out[7]:

```
array([[0., 1.],
       [1., 0.],
       [0., 1.],
       ...,
       [1., 0.],
       [1., 0.],
       [0., 1.]], dtype=float32)
```

CNN模型

In [15]:

```
from keras.layers import Dense, Flatten, Input
from keras.layers import BatchNormalization, Conv2D,MaxPooling2D
from keras import Model

input_layer=Input([IMSIZE,IMSIZE,3])
x=input_layer
x=BatchNormalization()(x)            #加快训练速度，提高模型精度
x=Conv2D(10,[2,2], padding = "valid",activation='relu')(x)
#卷积层，得到新的像素矩阵
#filter = 10 (我看sample size是60000用的filter是64，这个sample没有很复杂所以尝试不是太高的filter) 用了10个卷积核
#kernel size = 4* 4, padding = “valid” 代表用的是valid 卷积
#用ReLu去掉负数值因为我们不需要

x=MaxPooling2D([16,16])(x)
#最大池化层，挑取每个小矩形16*16中的最大值

x=Flatten()(x)
x=Dense(2,activation='softmax')(x)
#softmax也是个逻辑回顾的activati哦你
output_layer=x
model=Model(input_layer,output_layer)
model.summary()
```

```
Model: "model_3"
_______________________________________________________________
__
Layer (type)                 Output Shape              Param #
===============================================================
==
input_4 (InputLayer)         (None, 128, 128, 3)       0
_______________________________________________________________
__
batch_normalization_4 (Batch (None, 128, 128, 3)       12
_______________________________________________________________
__
conv2d_3 (Conv2D)            (None, 127, 127, 10)      130
_______________________________________________________________
__
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 10)          0
_______________________________________________________________
__
flatten_3 (Flatten)          (None, 490)               0
_______________________________________________________________
__
dense_3 (Dense)              (None, 2)                 982
===============================================================
==
Total params: 1,124
Trainable params: 1,118
Non-trainable params: 6
_______________________________________________________________
__
```

参数解释

1) 第一个layer没有任何参数因为没有任何需要学习的内容

2）Normalisation layer的参数是自己学习到的，不固定

3) 卷积层有 （*22（卷积核的长宽）*3（从上一个layer留下的3个通道）+ 1（每个新通道的bias截距项））*10 （新通道） = （223 +1）*10 = 130

4）池化层没有任何学习项因为只是取了最值

5&6） 在dense层 490 * 2 + 2= 982

运用模型看精确度

In [16]:
```
from keras.optimizers import Adam
model.compile(optimizer = Adam(0.05),
             loss = "categorical_crossentropy",
             metrics = ["accuracy"])
```

In [17]:
```
model.fit(X0,YY0, validation_data = (X1,YY1),
         batch_size = 200,
          epochs = 20)
 #因为有3850个sample在train data里，batch_size不宜很多
#尝试了epochs = 10，发现accuracy还有向上升的空间，于是将epochs定为20
```

```
Train on 3850 samples, validate on 1650 samples
Epoch 1/20
3850/3850 [==============================] - 2s 501us/step - lo
ss: 1.2406 - accuracy: 0.5525 - val_loss: 0.7197 - val_accuracy
: 0.5188
Epoch 2/20
3850/3850 [==============================] - 2s 434us/step - lo
ss: 0.5627 - accuracy: 0.7119 - val_loss: 0.7289 - val_accuracy
: 0.5194
Epoch 3/20
3850/3850 [==============================] - 2s 447us/step - lo
ss: 0.4942 - accuracy: 0.7577 - val_loss: 0.7573 - val_accuracy
: 0.5055
Epoch 4/20
3850/3850 [==============================] - 2s 445us/step - lo
ss: 0.4714 - accuracy: 0.7761 - val_loss: 0.6456 - val_accuracy
```

```
: 0.6248
Epoch 5/20
3850/3850 [==============================] - 2s 498us/step - lo
ss: 0.4179 - accuracy: 0.8036 - val_loss: 0.6937 - val_accuracy
: 0.5497
Epoch 6/20
3850/3850 [==============================] - 2s 451us/step - lo
ss: 0.4095 - accuracy: 0.8104 - val_loss: 0.6317 - val_accuracy
: 0.6055
Epoch 7/20
3850/3850 [==============================] - 2s 456us/step - lo
ss: 0.3843 - accuracy: 0.8273 - val_loss: 0.5629 - val_accuracy
: 0.7491
Epoch 8/20
3850/3850 [==============================] - 2s 445us/step - lo
ss: 0.3820 - accuracy: 0.8314 - val_loss: 0.5496 - val_accuracy
: 0.7491
Epoch 9/20
3850/3850 [==============================] - 2s 475us/step - lo
ss: 0.3625 - accuracy: 0.8395 - val_loss: 0.5173 - val_accuracy
: 0.7691
Epoch 10/20
3850/3850 [==============================] - 2s 437us/step - lo
ss: 0.3540 - accuracy: 0.8395 - val_loss: 0.4808 - val_accuracy
: 0.8091
Epoch 11/20
3850/3850 [==============================] - 2s 466us/step - lo
ss: 0.3359 - accuracy: 0.8491 - val_loss: 0.4363 - val_accuracy
: 0.8073
Epoch 12/20
3850/3850 [==============================] - 2s 443us/step - lo
ss: 0.3188 - accuracy: 0.8577 - val_loss: 0.3794 - val_accuracy
: 0.8333
Epoch 13/20
3850/3850 [==============================] - 2s 467us/step - lo
ss: 0.3100 - accuracy: 0.8626 - val_loss: 0.3443 - val_accuracy
: 0.8503
Epoch 14/20
3850/3850 [==============================] - 2s 471us/step - lo
ss: 0.3170 - accuracy: 0.8569 - val_loss: 0.3679 - val_accuracy
: 0.8267
Epoch 15/20
3850/3850 [==============================] - 2s 445us/step - lo
ss: 0.3156 - accuracy: 0.8652 - val_loss: 0.3295 - val_accuracy
: 0.8636
Epoch 16/20
3850/3850 [==============================] - 2s 446us/step - lo
ss: 0.2887 - accuracy: 0.8725 - val_loss: 0.2801 - val_accuracy
: 0.8879
Epoch 17/20
3850/3850 [==============================] - 2s 451us/step - lo
ss: 0.2837 - accuracy: 0.8779 - val_loss: 0.2870 - val_accuracy
: 0.8855
Epoch 18/20
3850/3850 [==============================] - 2s 429us/step - lo
```

```
ss: 0.2926 - accuracy: 0.8722 - val_loss: 0.2595 - val_accuracy
: 0.8933
Epoch 19/20
3850/3850 [==============================] - 2s 432us/step - lo
ss: 0.2845 - accuracy: 0.8795 - val_loss: 0.2682 - val_accuracy
: 0.8897
Epoch 20/20
3850/3850 [==============================] - 2s 440us/step - lo
ss: 0.2727 - accuracy: 0.8813 - val_loss: 0.2650 - val_accuracy
: 0.8952
```

```
Out[17]: <keras.callbacks.callbacks.History at 0x7f50687d5ed0>
```

可以看到现在的accuracy是0.88，比之前直接套用logistic regression要高出0.04。

```
In [ ]:
```