学习经典模型 AlexNet

“AlexNet采用8层的神经网络，5个卷积层和3个全连接层(3个卷积层后面加了最大池化层)”。 这里和下方网络结构中有一点不一样，感觉网络结构详解中第四层应该为 MaxPooling2D（3*3), stride(2)

网络结构（详解）

> 输入层: 227×227×3的图片。
>
> 第1层：Conv2D(11×11，96), Stride(4), ReLU, Output: 55×55×96。
>
> 第2层：MaxPooling2D(3×3), Stride(2), Output: 27×27×96。
>
> 第3层：Conv2D(5×5，256），Same，Output：27×27×256。
>
> 第4层：Conv2D(3×3,256），Stride(2)，Output：13×13×256。
>
> 第5层：Conv2D(3×3,384），Same，Output：13×13×384。
>
> 第6层：Conv2D(3×3,384），Same，Output：13×13×384。
>
> 第7层：Conv2D（3×3，256），Same，Output：13×13×256。
>
> 第8层：MaxPooling2D（3×3），Stride（2），Output：6×6×256。
>
> 输出层：Flatten，Dense(4096)，Dropout(0.5)，Dense（4096），Dropout（0.5），Output。

代码实现

In [3]:
```
import tensorflow as tf
import keras
from keras.layers import Activation,Conv2D, BatchNormalization,
Dense
from keras.layers import Dropout, Flatten, Input, MaxPooling2D,
ZeroPadding2D
from keras import Model
#ZeroPadding2D "can add rows and columns of zeros at the top, bo
ttom, left and right side of an image tensor."
#但是在这里我并没有看到实际运用?

IMSIZE = 227
input_layer = Input([IMSIZE,IMSIZE,3])
x = input_layer
x = Conv2D(96,[11,11],strides = [4,4], activation = 'relu')(x)
#第一个卷积层，通道数96，卷积核大小为 11*11 ，步长为列4行4，padding 为默
认的 valid，使用的activation function是ReLu
x = MaxPooling2D([3,3], strides = [2,2])(x)
#池化层，核大小为3*3，  步长列2行2
x = Conv2D(256,[5,5],padding = "same", activation = 'relu')(x)
#第二个卷积层，通道数256，卷积核大小为 5*5 ， 步长为默认的列1行1，padding
为same，使用的activation function是ReLu
x = MaxPooling2D([3,3], strides = [2,2])(x)
#池化层，核大小为3*3，  步长列2行2
x = Conv2D(384,[3,3],padding = "same", activation = 'relu')(x)
#第三个卷积层，通道数384，卷积核大小为 3*3 ，步长为默认的列1行1，padding
为same，使用的activation function是ReLu
x = Conv2D(384,[3,3],padding = "same", activation = 'relu')(x)
#第四个卷积层，通道数384，卷积核大小为 3*3 ，步长为默认的列1行1，padding
为same，使用的activation function是ReLu
x = Conv2D(256,[3,3],padding = "same", activation = 'relu')(x)
#第五个卷积层，通道数256，卷积核大小为 3*3 ，步长为默认的列1行1，padding
为same，使用的activation function是ReLu
x = MaxPooling2D([3,3], strides = [2,2])(x)
#池化层，核大小为3*3，  步长列2行2
x = Flatten()(x)
x = Dense(4096,activation = 'relu')(x)
#第一层全联接到4096个节点，activation依旧是relu
x = Dropout(0.5)(x)
x = Dense(4096,activation = 'relu')(x)
#第二层全联接还是到4096个节点，activation依旧是relu
x = Dropout(0.5)(x)
x = Dense(2,activation = 'softmax')(x)
#第三层全联接到2个节点，因为我们需要最后的output是2，activation依旧是rel
u
output_layer=x
model=Model(input_layer,output_layer)
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
__
Layer (type)                 Output Shape              Param #
=================================================================
```

```
==
input_2 (InputLayer)         (None, 227, 227, 3)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 55, 55, 96)        34944
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 27, 27, 96)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 27, 27, 256)       614656
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 13, 13, 256)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 13, 13, 384)       885120
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 13, 13, 384)       1327488
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 13, 13, 256)       884992
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 6, 6, 256)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_4 (Dense)              (None, 4096)              37752832
_________________________________________________________________
dropout_3 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_5 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_4 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 8194
===============================================================
==
Total params: 58,289,538
Trainable params: 58,289,538
Non-trainable params: 0
_________________________________________________________________
```

模仿模型进行性别判断

数据处理

In [15]:
```
import pandas as pd
import numpy as np
from PIL import Image

MasterFile = pd.read_csv("/clubear/Lecture 2.1 - Linear Regressi
on by TensorFlow/data/faces/FaceScore.csv")
FileNames=MasterFile['Filename']
N=len(FileNames)
IMSIZE=227
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 T
ensorFlow/data/faces/images/'+MyFile)
    Im=Im.resize([IMSIZE,IMSIZE])
    Im=np.array(Im)/255
    X[i,]=Im
```

In [16]:
```
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[16]:
```
array([0, 0, 0, ..., 1, 1, 1])
```

In [17]:
```
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

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

Out[17]:
```
array([[0., 1.],
       [1., 0.],
       [0., 1.],
       ...,
       [1., 0.],
       [1., 0.],
       [0., 1.]], dtype=float32)
```

AlexNet模仿

保证使用8层，5个卷积层和3个全连接层(3个卷积层后面加了最大池化层)，参数略有调整， 保证使用ReLu作为激活函数，使用Dropout

In [33]:
```
IMSIZE = 227
input_layer = Input([IMSIZE,IMSIZE,3])
x = input_layer
x = Conv2D(96,[11,11],strides = [4,4], activation = 'relu')(x)
#第一个卷积层，通道数96，卷积核大小为 11*11 ，步长为列4行4，padding 为默认的 valid，使用的activation function是ReLu
x = MaxPooling2D([3,3], strides = [2,2])(x)
#池化层，核大小为3*3，  步长列2行2
x = Conv2D(256,[5,5],padding = "same", activation = 'relu')(x)
#第二个卷积层，通道数256，卷积核大小为 5*5 ，  步长为默认的列1行1，padding为same，使用的activation function是ReLu
x = MaxPooling2D([3,3], strides = [2,2])(x)
#池化层，核大小为3*3，  步长列2行2
x = Conv2D(384,[3,3],padding = "same", activation = 'relu')(x)
#第三个卷积层，通道数384，卷积核大小为 3*3 ，步长为默认的列1行1，padding为same，使用的activation function是ReLu
x = Conv2D(384,[3,3],padding = "same", activation = 'relu')(x)
#第四个卷积层，通道数384，卷积核大小为 3*3 ，步长为默认的列1行1，padding为same，使用的activation function是ReLu
x = Conv2D(256,[3,3],padding = "same", activation = 'relu')(x)
#第五个卷积层，通道数256，卷积核大小为 3*3 ，步长为默认的列1行1，padding为same，使用的activation function是ReLu
x = MaxPooling2D([3,3], strides = [2,2])(x)
#池化层，核大小为3*3，  步长列2行2
x = Flatten()(x)
x = Dense(4096,activation = 'relu')(x)
#第一层全联接到4096个节点，activation依旧是relu
x = Dropout(0.5)(x)
x = Dense(4096,activation = 'relu')(x)
#第二层全联接还是到4096个节点，activation依旧是relu
x = Dropout(0.5)(x)
x = Dense(2,activation = 'softmax')(x)
#第三层全联接到2个节点，因为我们需要最后的output是2，activation依旧是relu
output_layer=x
model2=Model(input_layer,output_layer)
model2.summary()
```

```
Model: "model_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_8 (InputLayer)         (None, 227, 227, 3)       0
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 55, 55, 96)        34944
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 27, 27, 96)        0
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 27, 27, 256)       614656
_________________________________________________________________
```

```
__
max_pooling2d_23 (MaxPooling (None, 13, 13, 256)       0
_________________________________________________________________
__
conv2d_38 (Conv2D)           (None, 13, 13, 384)       885120
_________________________________________________________________
__
conv2d_39 (Conv2D)           (None, 13, 13, 384)       1327488
_________________________________________________________________
__
conv2d_40 (Conv2D)           (None, 13, 13, 256)       884992
_________________________________________________________________
__
max_pooling2d_24 (MaxPooling (None, 6, 6, 256)         0
_________________________________________________________________
__
flatten_8 (Flatten)          (None, 9216)              0
_________________________________________________________________
__
dense_22 (Dense)             (None, 4096)              37752832
_________________________________________________________________
__
dropout_15 (Dropout)         (None, 4096)              0
_________________________________________________________________
__
dense_23 (Dense)             (None, 4096)              16781312
_________________________________________________________________
__
dropout_16 (Dropout)         (None, 4096)              0
_________________________________________________________________
__
dense_24 (Dense)             (None, 2)                 8194
=================================================================
==
Total params: 58,289,538
Trainable params: 58,289,538
Non-trainable params: 0
_________________________________________________________________
__
```

参数解释

第一层input layer，规定input只能是[227,227,3]的矩阵，无参数。
第二层conv2D：（11113+1） 96 = 364 96 = 34,944, 其中1111是kernel size，3是上一层遗留通道数，97是卷积核个数。
第三层池化使用[3,3]矩形，步长行列都是2，没有学习项，无参数
第四层conv2D: （5596+1） 256 = 2401 256 = 614,656, 其中55是kernel size，96是上一层遗留通道数，256是卷积核个数。
第五层池化使用[3,3]矩形，步长行列都是2，没有学习项，无参数
第六层conv2D: （33256+1） 384 = 2305 384 = 885,120, 其中33是kernel size，256是上一层遗留通道数，384是卷积核个数。
第七层conv2D: （33384+1） 384 = 3457 384 = 1,327,488, 其中33是kernel size，384是上一层遗留通道数，384是卷积核个数。
第八层conv2D: （33384+1） 256 = 3457 256 = 884,992, 其中55是kernel size，6是上一层遗留通道数，16是卷积核个数。
第九层池化使用[3,3]矩形，步长行列都是2， 没有学习项，无参数
第十层压扁，无参数
第十一层Dense： 9216 4096 + 4096 = 37748,736 + 4096 = 37,752,832
第十二层Dense： 4096 4096 + 4096 = 16,777,216 + 4096 = 16,781,312
第十三层Dense： 4096 2 + 2 = 8192 + 2 = 8,194
所以总共有58289538个参数。

Compile

In [36]:
```
from keras.optimizers import Adam
model2.compile(optimizer = Adam(0.001),
               loss = "categorical_crossentropy",
               metrics = ["accuracy"])
```

In [37]:
```
model2.fit(X0,YY0, validation_data = (X1,YY1),
         batch_size = 200,
          epochs = 20)
```

```
Train on 3850 samples, validate on 1650 samples
Epoch 1/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6932 - accuracy: 0.4995 - val_loss: 0.6931 - val_accuracy:
0.5018
Epoch 2/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6937 - accuracy: 0.4964 - val_loss: 0.6931 - val_accuracy:
0.5018
Epoch 3/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6934 - accuracy: 0.4987 - val_loss: 0.6931 - val_accuracy:
0.5018
Epoch 4/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6929 - accuracy: 0.4997 - val_loss: 0.6931 - val_accuracy:
```

```
0.5018
Epoch 5/20
3850/3850 [==============================] - 11s 3ms/step - los
s: 0.6935 - accuracy: 0.4987 - val_loss: 0.6931 - val_accuracy:
0.5018
Epoch 6/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6931 - accuracy: 0.4904 - val_loss: 0.6931 - val_accuracy:
0.4982
Epoch 7/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6932 - accuracy: 0.5010 - val_loss: 0.6932 - val_accuracy:
0.4982
Epoch 8/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6945 - accuracy: 0.5000 - val_loss: 0.6932 - val_accuracy:
0.4982
Epoch 9/20
3850/3850 [==============================] - 11s 3ms/step - los
s: 0.6932 - accuracy: 0.5003 - val_loss: 0.6932 - val_accuracy:
0.4982
Epoch 10/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6932 - accuracy: 0.4971 - val_loss: 0.6931 - val_accuracy:
0.4982
Epoch 11/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6938 - accuracy: 0.4894 - val_loss: 0.6931 - val_accuracy:
0.4982
Epoch 12/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6933 - accuracy: 0.5005 - val_loss: 0.6932 - val_accuracy:
0.4982
Epoch 13/20
3850/3850 [==============================] - 11s 3ms/step - los
s: 0.6932 - accuracy: 0.5010 - val_loss: 0.6932 - val_accuracy:
0.4982
Epoch 14/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6932 - accuracy: 0.5005 - val_loss: 0.6932 - val_accuracy:
0.4982
Epoch 15/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6931 - accuracy: 0.5010 - val_loss: 0.6932 - val_accuracy:
0.4982
Epoch 16/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6931 - accuracy: 0.5008 - val_loss: 0.6932 - val_accuracy:
0.4982
Epoch 17/20
3850/3850 [==============================] - 11s 3ms/step - los
s: 0.6932 - accuracy: 0.5005 - val_loss: 0.6932 - val_accuracy:
0.4982
Epoch 18/20
3850/3850 [==============================] - 12s 3ms/step - los
```

```
s: 0.6932 - accuracy: 0.5003 - val_loss: 0.6932 - val_accuracy:
0.4982
Epoch 19/20
3850/3850 [==============================] - 12s 3ms/step - los
s: 0.6932 - accuracy: 0.5005 - val_loss: 0.6932 - val_accuracy:
0.4982
Epoch 20/20
3850/3850 [==============================] - 11s 3ms/step - los
s: 0.6932 - accuracy: 0.5005 - val_loss: 0.6932 - val_accuracy:
0.4982
```

Out[37]:
```
<keras.callbacks.callbacks.History at 0x7fc27d37a790>
```

这个accuracy有点没有预料到...

In [ ]:
```

```