Import major libraries and designate the paths to image folders on desktop

In [4]:

```
import cv2
import numpy as np
import os
import glob
from tqdm import tqdm
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.externals import joblib
import keras
from keras.models import load_model
path='/Users/cheryl/Desktop/Improved'
path2='/Users/cheryl/Desktop/JessicaChastain'
```

```
/Users/cheryl/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:3
6: FutureWarning: Conversion of the second argument of issubdtype from
`float` to `np.floating` is deprecated. In future, it will be treated
as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

Function to load the images into arrays

In [2]:

```
def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        if filename.endswith(".jpg") or filename.endswith(".jpeg"):
            img = cv2.imread(os.path.join(folder, filename))
            if img is not None:
                images.append(img)
    return images
```

In [3]:

```
#Load the images
images=load_images_from_folder(path)
```

Cropping the facial region from each photo then appending them into an array.

In [4]:

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
cropped=[]
for item in images:
    map=face_cascade.detectMultiScale(item)
    for(x,y,w,h)in map:
        crop=item[y:y+h,x:x+h]
    cropped.append(crop)
```

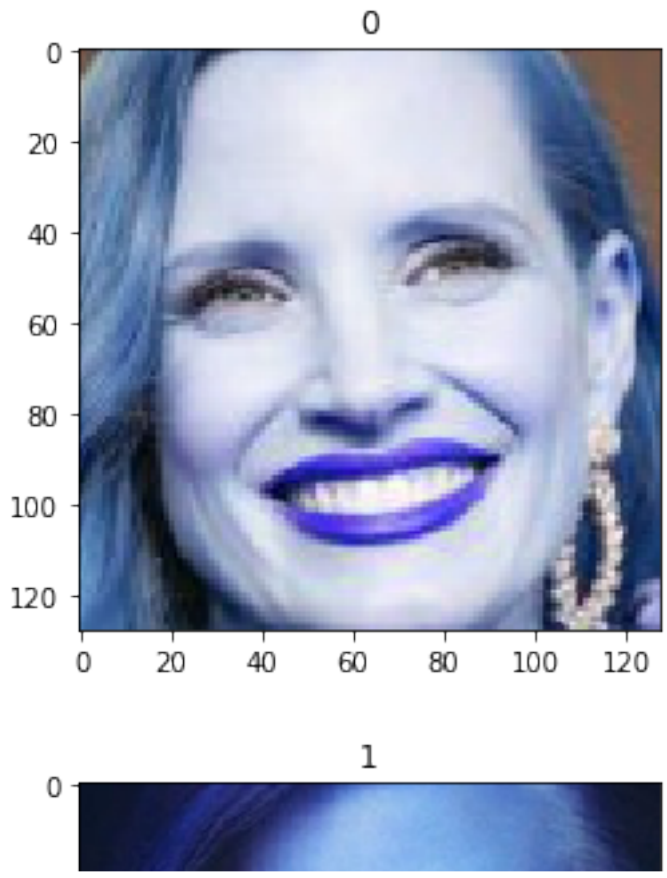
Iteraing through the cropped images then cleaning up undesirable samples from each dataset.

In [14]:

```
#indices = 3,6,9,12,14,16,25,29,32,34,37,38,41,48,51,65,67,76,87,88,91,96,97,98,102,
#112,117,122,126,132,138,140,148,149,150,157,165,173,176,179,183,191,195,204,207
#indices =81,86,91,95,101,107,109,117,119,127,134,143,146,149,153,161,165,174,177
#cropped = [i for j, i in enumerate(cropped) if j not in indices]
#indices=6,18,21,26,29,32,40,43,45,46,50,57,62,65,69,73,85,89,90,91,100,104,108,113,
#indices=155,154,152,150,130,128,120,115,104,100
#cropped = [i for j, i in enumerate(cropped) if j not in indices]
```

In [11]:

```
for i, item in enumerate(Jcropped):
    item=cv2.imread(item)
    plt.figure()
    plt.imshow(item)
    plt.title(i)
    plt.show()
```



In [17]:

```
#save the processed training data
for i,item in enumerate(cropped):
    cv2.imwrite('Bcrop%s.jpg'%i,item)
```

In [21]:

```
#save the processed training data
for i,item in enumerate(Jcropped[:159]):
    cv2.imwrite('Jcrop%s.jpg'%i,item)
```

In [4]:

```
Jimages=load_images_from_folder(path2)
```

```
Jcropped=[]
for item in Jimages:
    map=face_cascade.detectMultiScale(item)
    for(x,y,w,h)in map:
        crop=item[y:y+h,x:x+h]
    Jcropped.append(crop)
```

Saving the cropped images on local system

In [6]:

```
cropped=[]
for i in range(0,159):
    cropped.append('Bcrop%s.jpg'%i)
```

In [7]:

```
Jcropped=[]
for i in range(0,159):
    Jcropped.append('Jcrop%s.jpg'%i)
```

Creating the filter banks

In [8]:

```
theta=[0]
for i in range(1,16):
    theta.append((np.pi/16)*i)
g_kernel=[]
for i in range(0,16):
    g_kernel.append(cv2.getGaborKernel((30, 30), 3.0, theta[i], 4, 0.04, np.pi/4, kt

BryceGabor=[]


for item in cropped:
    item=cv2.imread(item)
    for i in range(0,16):
        filtered = cv2.filter2D(item, cv2.CV_8UC3, g_kernel[i])
        BryceGabor.append(filtered)

BryceFilters=[]
for gabor in BryceGabor:
    gray = cv2.cvtColor(gabor, cv2.COLOR_BGR2GRAY)
    resize=cv2.resize(gray,(64,64))
    BryceFilters.append([resize,np.array([1,0])])
JessicaGabor=[]
for item in Jcropped[:159]:
    item=cv2.imread(item)
    for i in range(0,16):
        filtered = cv2.filter2D(item, cv2.CV_8UC3, g_kernel[i])
        JessicaGabor.append(filtered)

JessicaFilters=[]
for gabor in JessicaGabor:
    gray = cv2.cvtColor(gabor, cv2.COLOR_BGR2GRAY)
    resize=cv2.resize(gray,(64,64))
    JessicaFilters.append([resize,np.array([0,1])])
```

Sample image and its Gabor filters

In [5]:

```
image=Jimages[4]
plt.figure()
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()
```

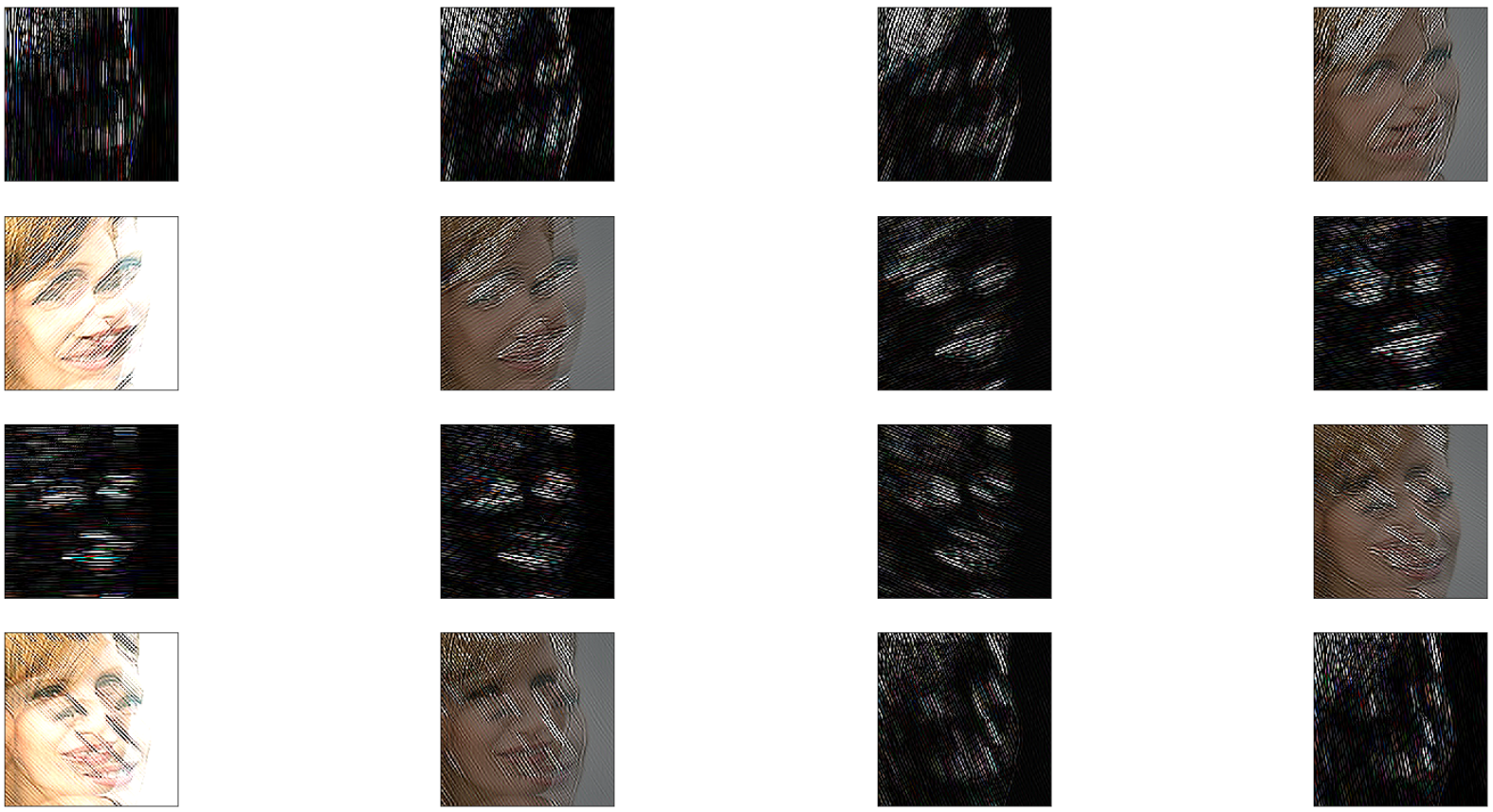

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

map=face_cascade.detectMultiScale(image)
for(x,y,w,h)in map:
    crop=Jimages[4][y:y+h,x:x+h]
plt.figure()
plt.imshow(cv2.cvtColor(crop, cv2.COLOR_BGR2RGB))
plt.show()
```

In [7]:

```
theta=[0]
for i in range(1,16):
    theta.append((np.pi/16)*i)
g_kernel=[]
for i in range(0,16):
    g_kernel.append(cv2.getGaborKernel((30, 30), 3.0, theta[i], 4, 0.04, np.pi/4, kt
list=[]
for i in range(0,16):
        filtered = cv2.filter2D(crop, cv2.CV_8UC3, g_kernel[i])
        list.append(filtered)
```

In [8]:

```
fig=plt.figure(figsize=(28,28))
for cnt,item in enumerate(list):
    y=fig.add_subplot(8,4,cnt+1)
    plt.imshow(cv2.cvtColor(item, cv2.COLOR_BGR2RGB))
    #plt.title(cnt)
    y.axes.get_xaxis().set_visible(False)
    y.axes.get_yaxis().set_visible(False)
```



Adding the training images and shuffling them

In [12]:

```
allImages=[]
allImages=BryceFilters+JessicaFilters
import random
random.shuffle(allImages)
```

Flattening the images for fitting into the neural net and extracting the labels.

In [13]:

```
trainImage=np.array([i[0] for i in allImages]).reshape(-1,64,64,1)
trainLabel=np.array([i[1] for i in allImages])
```

Building the CNN layers.

In [14]:

```
from keras.models import Sequential
from keras.layers import *
#from keras.layers import Conv3D, MaxPool3D, Flatten, Dense
model=Sequential()
model.add(Conv2D(filters=32,kernel_size=5,strides=1,padding='same',input_shape=(64,6
model.add(MaxPool2D(pool_size=5,padding='same'))
model.add(Conv2D(filters=50,kernel_size=5,strides=1,padding='same',activation='relu
model.add(MaxPool2D(pool_size=5,padding='same'))
model.add(Conv2D(filters=80,kernel_size=5,strides=1,padding='same',activation='relu
model.add(MaxPool2D(pool_size=5,padding='same'))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512,activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(2,activation='softmax'))
```

Training the neural net

In [15]:

```
adam=keras.optimizers.adam(lr=1e-3)
model.compile(optimizer=adam,loss='categorical_crossentropy',metrics=['accuracy'])
history=model.fit(x=trainImage,y=trainLabel,epochs=50,batch_size=100)
model.summary()
```

```
Epoch 1/50
5088/5088 [==============================] - 11s 2ms/step - loss: 0.0
535 - acc: 0.9860
Epoch 2/50
5088/5088 [==============================] - 10s 2ms/step - loss: 0.0
285 - acc: 0.9921
Epoch 3/50
5088/5088 [==============================] - 10s 2ms/step - loss: 0.0
201 - acc: 0.9919
Epoch 4/50
5088/5088 [==============================] - 10s 2ms/step - loss: 0.0
345 - acc: 0.9896
Epoch 5/50
5088/5088 [==============================] - 11s 2ms/step - loss: 0.0
181 - acc: 0.9945
Epoch 6/50
5088/5088 [==============================] - 10s 2ms/step - loss: 0.0
219 - acc: 0.9943
Epoch 7/50
```

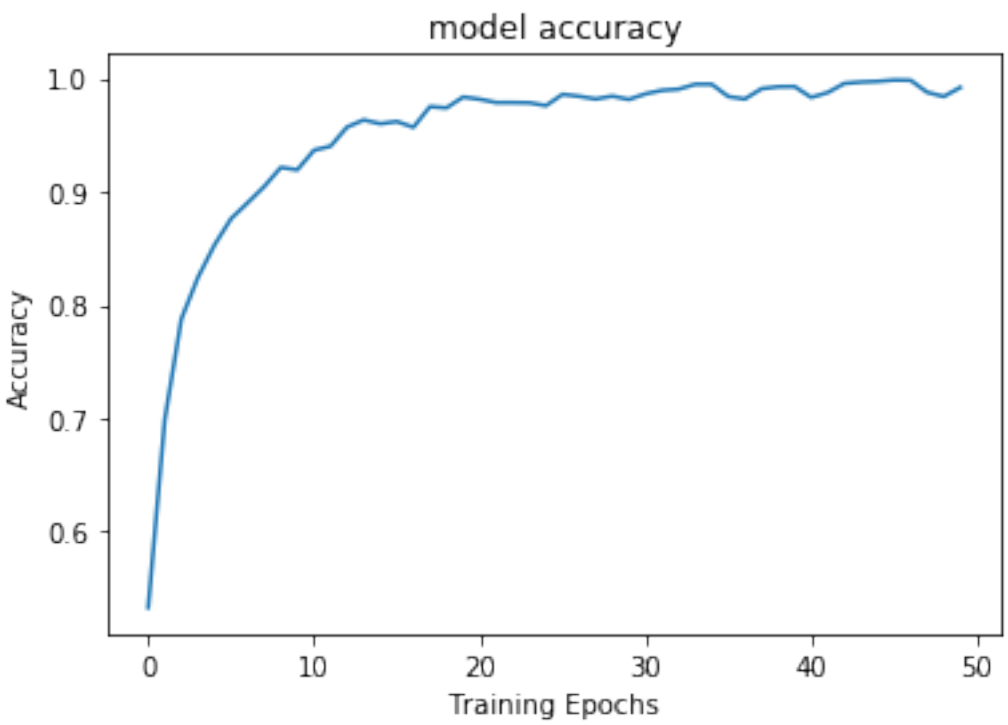
In [30]:

```
print(history.history.keys())
```

```
dict_keys(['loss', 'acc'])
```

In [31]:

```
plt.plot(history.history['acc'])
plt.title('model accuracy')
plt.ylabel("Accuracy")
plt.xlabel("Training Epochs")
plt.show()
```



Loading the testing images, appending the appropriate labels for them.

In [23]:

```
path3='/Users/cheryl/Desktop/BryceTest'
TestBryce=load_images_from_folder(path3)
path4='/Users/cheryl/Desktop/JessicaTest'
TestJessica=load_images_from_folder(path4)
BryceTest=[]
for item in TestBryce:
    BryceTest.append([item,np.array([1,0])])
JessicaTest=[]
for item in TestJessica:
    JessicaTest.append([item,np.array([0,1])])
allTest=[]
allTest=BryceTest+JessicaTest
```

Iterate through the test set, extract the facial region, flatten the array then test out the model. Printing out the testing images with the predicted labels.

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
g_kernel1=cv2.getGaborKernel((30, 30), 3.0, 0, 4, 0.04, np.pi/4,
ktype=cv2.CV_32F)
#40 test images for the 80/20 split.
fig=plt.figure(figsize=(14,14))
y_true = []
y_pred=[]
for i in range(0,40):
    y_true.append(1)
for cnt,data in enumerate(allTest[:40]):
    img=data[0]
    map=face_cascade.detectMultiScale(img)
    for(x,y,w,h)in map:
        new=img[y:y+h,x:x+h]
    y=fig.add_subplot(8,5,cnt+1)
    filtered = cv2.filter2D(new, cv2.CV_8UC3, g_kernel1)
    gray = cv2.cvtColor(filtered, cv2.COLOR_BGR2GRAY)
    resized=cv2.resize(gray,(64,64))
    data=resized.reshape(-1,64,64,1)
    model_out=model.predict([data])
    if np.argmax(model_out)==0:
        str_label='Bryce'
    else:
        str_label='Jessica'
    y_pred.append(np.argmax(model_out))
    y.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(str_label)

    y.axes.get_xaxis().set_visible(False)
    y.axes.get_yaxis().set_visible(False)
```

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
g_kernel1=cv2.getGaborKernel((30, 30), 3.0, 0, 4, 0.04, np.pi/4,
ktype=cv2.CV_32F)
fig=plt.figure(figsize=(14,14))
for cnt,data in enumerate(allTest[40:]):
    img=data[0]
    map=face_cascade.detectMultiScale(img)
    for(x,y,w,h)in map:
        new=img[y:y+h,x:x+h]
    y=fig.add_subplot(8,5,cnt+1)
    filtered = cv2.filter2D(new, cv2.CV_8UC3, g_kernel1)
    gray = cv2.cvtColor(filtered, cv2.COLOR_BGR2GRAY)
    resized=cv2.resize(gray,(64,64))
    data=resized.reshape(1,64,64,1)
    model_out=model.predict([data])
    if np.argmax(model_out)==0:
        str_label='Bryce'
    else:
        str_label='Jessica'
    y.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(str_label)
    y.axes.get_xaxis().set_visible(False)
    y.axes.get_yaxis().set_visible(False)
```

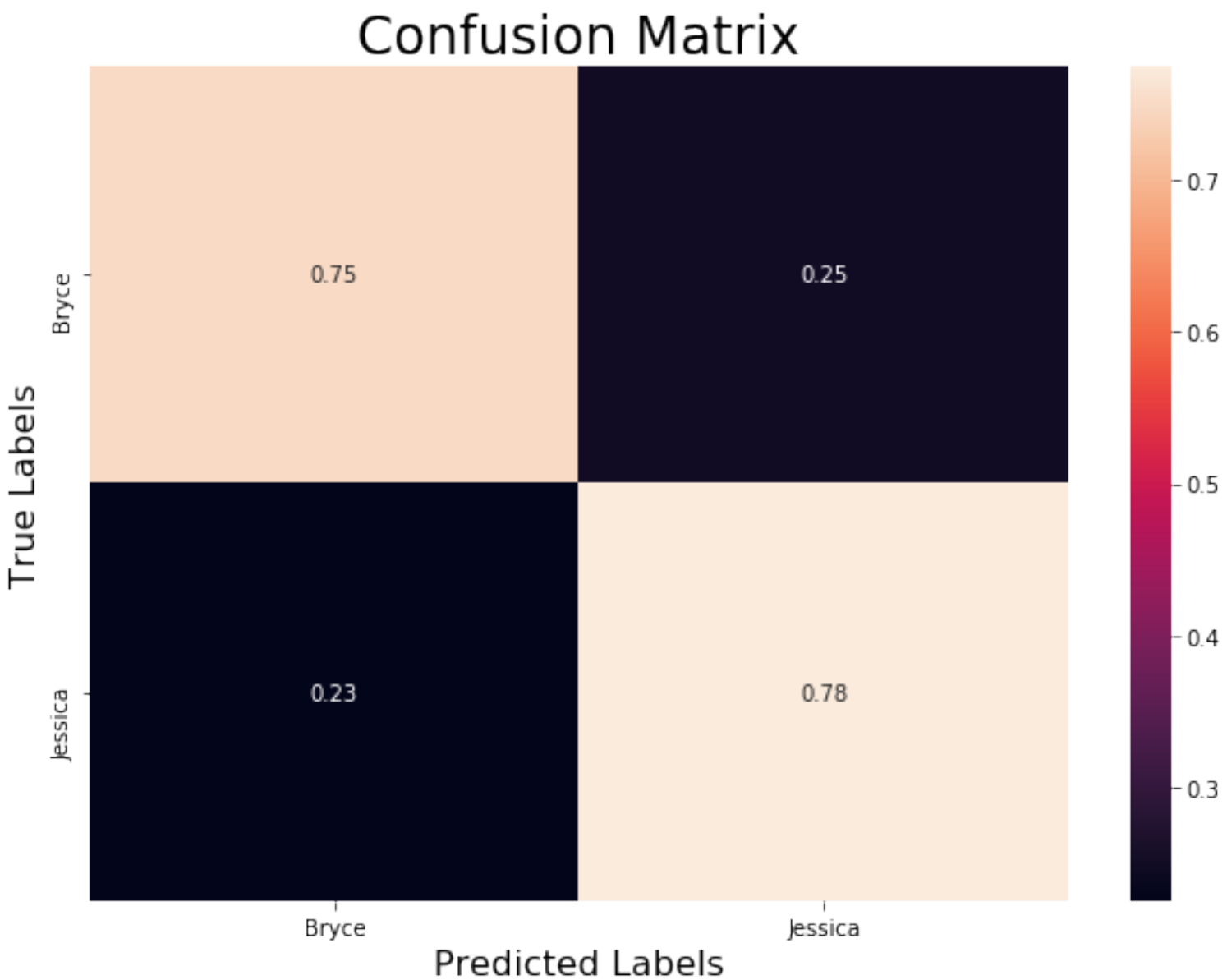
Generate confusion matrix

In [68]:

```
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
array = [[0.75,0.25],
         [0.225,0.775]]
df_cm = pd.DataFrame(array, index = ["Bryce","Jessica"],
                     columns = ["Bryce","Jessica"])
plt.figure(figsize = (10,7))
box=sn.heatmap(df_cm, annot=True)
box.set_title("Confusion Matrix",fontsize=24)
box.set_ylabel("True Labels",fontsize=16)
box.set_xlabel("Predicted Labels",fontsize=16)
```

Out[68]:

```
Text(0.5,42,'Predicted Labels')
```



Saving model to disk

In [85]:

```
joblib.dump(model,'model_joblib')
```

Out[85]:

```
['model_joblib']
```

In [11]:

```
mj=joblib.load('model_joblib')
```

In [ ]:

In [ ]:

In [80]:

In [ ]:

In [ ]:

In [39]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Import major libraries and designate the paths to image folders on desktop

In [4]:

```
/Users/cheryl/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:3
6: FutureWarning: Conversion of the second argument of issubdtype from
`float` to `np.floating` is deprecated. In future, it will be treated
as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

Function to load the images into arrays

In [2]:

In [3]:

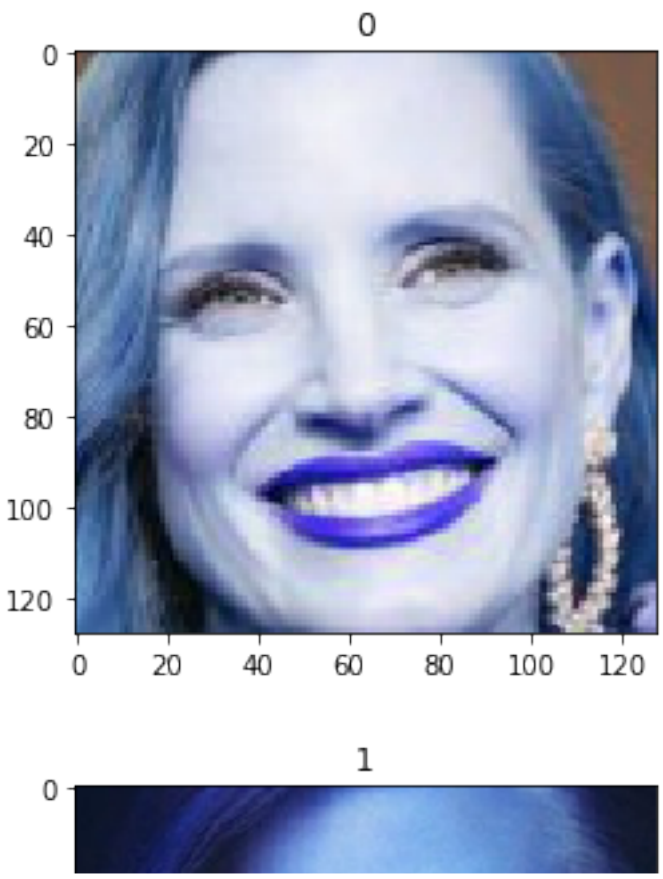
Cropping the facial region from each photo then appending them into an array.

In [4]:

Iteraing through the cropped images then cleaning up undesirable samples from each dataset.

In [14]:

In [11]:



In [17]:

In [21]:

In [4]:

Saving the cropped images on local system

In [6]:

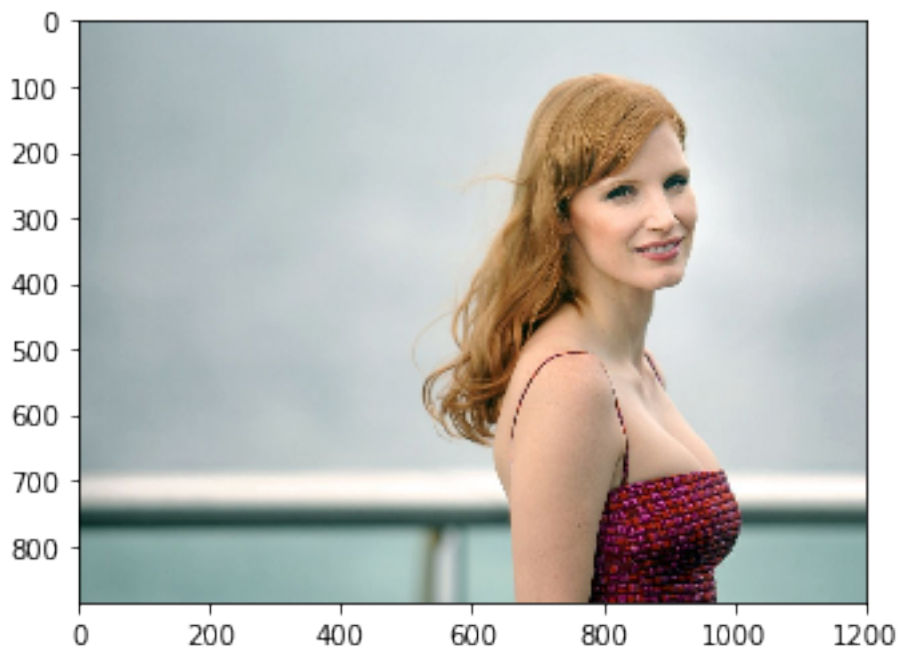
In [7]:

Creating the filter banks

In [8]:

Sample image and its Gabor filters

In [5]:



In [7]:

In [8]:

Adding the training images and shuffling them

In [12]:

Flattening the images for fitting into the neural net and extracting the labels.

In [13]:

Building the CNN layers.

In [14]:

Training the neural net

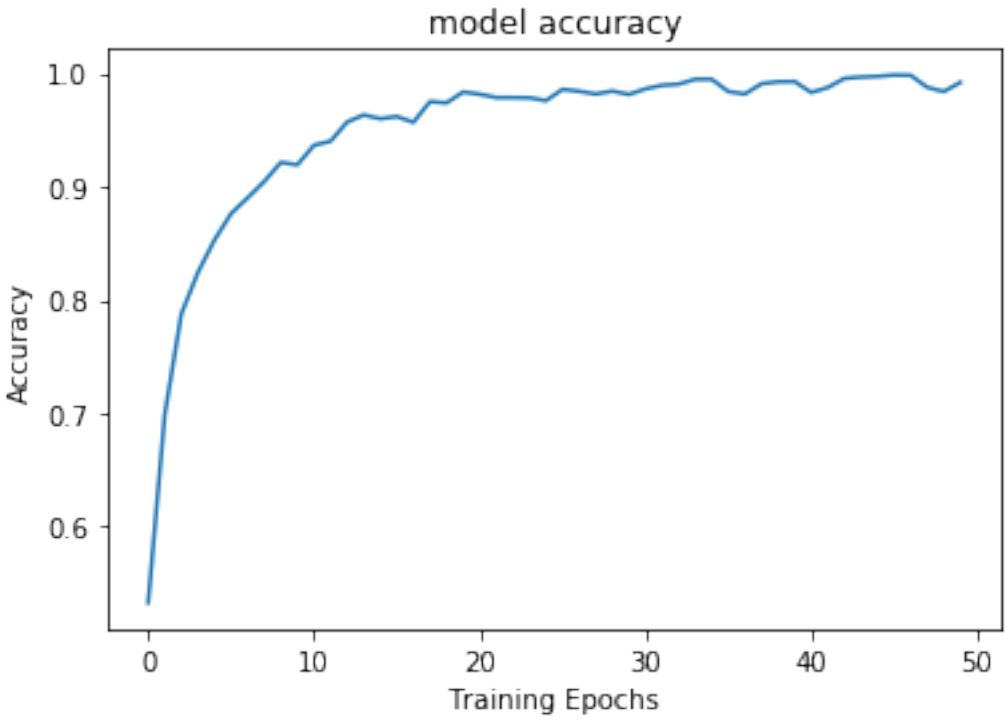
In [15]:

```
Epoch 1/50
5088/5088 [==============================] - 11s 2ms/step - loss: 0.0
535 - acc: 0.9860
Epoch 2/50
5088/5088 [==============================] - 10s 2ms/step - loss: 0.0
285 - acc: 0.9921
Epoch 3/50
5088/5088 [==============================] - 10s 2ms/step - loss: 0.0
201 - acc: 0.9919
Epoch 4/50
5088/5088 [==============================] - 10s 2ms/step - loss: 0.0
345 - acc: 0.9896
Epoch 5/50
5088/5088 [==============================] - 11s 2ms/step - loss: 0.0
181 - acc: 0.9945
Epoch 6/50
5088/5088 [==============================] - 10s 2ms/step - loss: 0.0
219 - acc: 0.9943
Epoch 7/50
```

In [30]:

```
dict_keys(['loss', 'acc'])
```

In [31]:



Loading the testing images, appending the appropriate labels for them.

In [23]:

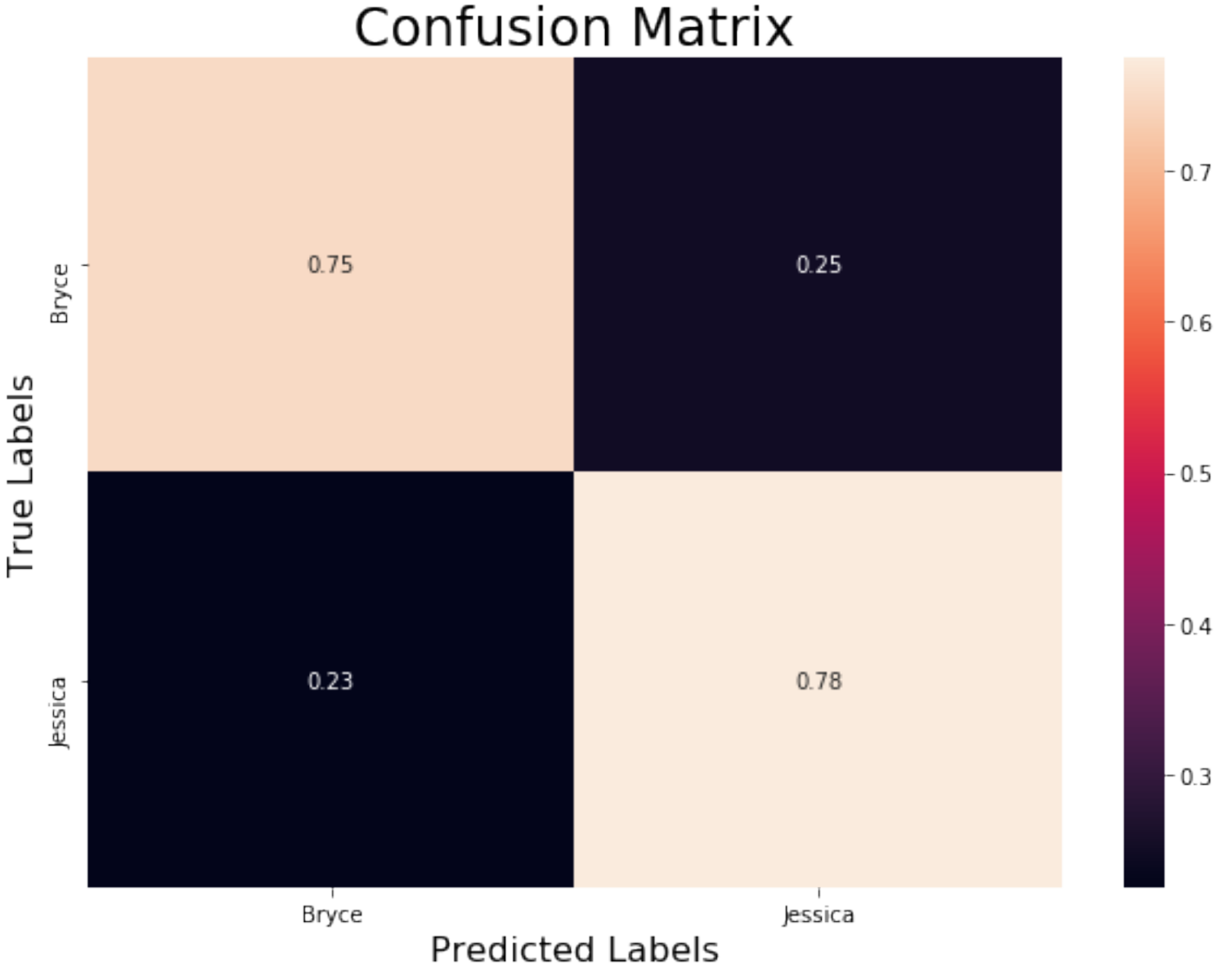
Iterate through the test set, extract the facial region, flatten the array then test out the model. Printing out the testing images with the predicted labels.

Generate confusion matrix

In [68]:

Out[68]:

```
Text(0.5,42,'Predicted Labels')
```



Saving model to disk

In [85]:

```
```

Out[85]:

```
['model_joblib']
```

In [11]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [80]:

```
```

In [ ]:

```
```

In [39]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]: