Game of Thrones

Actors/Actress

How awesome of the makeup work!

Steps

1. Download the pictures of actress/actors in Game of Thrones, [1] (http://www.eonline.com/news/871858/game-of-thrones-boss-on-jon-snow-and-daenerys-targaryen-s-complicated-attraction), saved in folder, "imgs/original";
2. split each image into two, after/before makeup, and save it in folder, "imgs/final",
3. crop the part of face in each image and save it into folder in "imgs/cvdata".
4. use pre-trained features, availed by opencv official site, to recognize whether the image is face of someone or not;
5. find similar images

Source

1. Game of Thrones (http://www.eonline.com/news/871858/game-of-thrones-boss-on-jon-snow-and-daenerys-targaryen-s-complicated-attraction)
2. opencv-3.3, and opencv_contrib (http://http://opencv.org)
   - install by **pip**:

     ```
     shell> pip install opencv-python
      or
     shell> pip install opencv-python-contrib
     ```

   - install from scratch, a little computer skills required, surely patatience too, see below.
3. dlib (https://github.com/davisking/dlib), by `pip install dlib` , develop tools required, gcc, xcode for instance.
4. face_recognition (https://github.com/ageitgey/face_recognition), by `pip install face_recognition`

ö|ö Load tool for layout

Dou you have any layout/printout problem? Now you can try the new package, appmode:

```
> pip install appmode
> jupyter nbextension     enable --py --sys-prefix appmode
> jupyter serverextension enable --py --sys-prefix appmode
```

ô|ô Load necessary modules

```
diffusion
last updated: Tue Nov 21 2017 10:18:10 CST

CPython 3.6.1
IPython 6.2.1

skimage 0.13.0
opencv n

compiler   : GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)
system     : Darwin
release    : 16.7.0
machine    : x86_64
processor  : i386
CPU cores  : 8
interpreter: 64bit
Git hash   :
```

o|o Oringinal Images: **imgs/original/*.jpg**

```
imgs/original/rs_1024x759-140320150538-425.2GofT.gleeson.mh.053012.jp
g
imgs/original/rs_1024x759-140320150624-425.GofT.aiden.mh.053012.jpg
imgs/original/rs_1024x759-140320150649-425.GofT.carice.mh.053012.jpg
imgs/original/rs_1024x759-140320150707-425.GofT.alfie.mh.053012.jpg
imgs/original/rs_1024x759-140320150727-425.GofT.dinklage.mh.053012.jp
g
imgs/original/rs_1024x759-140320150746-425.GofT.christie.mh.053012.jp
g
imgs/original/rs_1024x759-140320150836-425.GofT.lena.mh.053012.jpg
imgs/original/rs_1024x759-140320150911-425.GofT.fairley.mh.053012.jpg
imgs/original/rs_1024x759-140320150941-425.GofT.mccann.mh.053012.jpg
imgs/original/rs_1024x759-140320151033-425.GofT.nik.mh.053012.jpg
imgs/original/rs_1024x759-140320151512-425.GofT.maisie.mh.053012.jpg
imgs/original/rs_1024x759-140320152103-425.GofT.emilia.mh.053012_copy
.jpg
imgs/original/rs_1024x759-140320152413-1024.kit-harrington-game-of-th
rones.ls.32014.jpg
imgs/original/rs_1024x759-140320153046-1024.natalie-dormer-game-of-th
rones.ls.32014.jpg
imgs/original/rs_1024x759-140320154115-1024.Kristian-Nairn.ls.32014.j
pg
imgs/original/rs_1024x759-140320154816-1024.Isaac-Hempstead-Wright-.l
s.32014.jpg
imgs/original/rs_1024x759-140320155416-1024.Nathalie_Emmanuel.ls.3201
4.jpg
imgs/original/rs_1024x759-140320160209-1024.sophie-turner.ls.32014.jp
g
imgs/original/rs_1024x759-150616160003-1024.Iain-Glen-game-of-thrones
.jpg
imgs/original/rs_1024x759-150616160035-1024.Liam-Cunningham-game-of-t
hrones.jpg
imgs/original/rs_1024x759-150616160051-1024.Stephen-Dillane-game-of-t
hrones.jpg
imgs/original/rs_1024x759-150616160107-1024-nell-tiger-free-game-of-t
hrones.jpg
imgs/original/rs_1024x759-150616160125-1024.Dean-Charles-Chapman-game
```

```
-of-thrones.jpg
imgs/original/rs_1024x759-150616160139-1024.Jacob-Anderson-game-of-th
rones.jpg
imgs/original/rs_1024x759-150616160206-1024.Iwan-Rheon-game-of-throne
s.jpg
imgs/original/rs_1024x759-150616160220-1024.joel-fry-game-of-thrones.
jpg
imgs/original/rs_1024x759-150616160236-1024.Kerry-Ingram-game-of-thro
nes.jpg
imgs/original/rs_1024x759-150616160252-1024.Michiel-Huisman-game-of-t
hrones.jpg
imgs/original/rs_1024x759-150616160310-1024.Indira-Varma-game-of-thro
nes.jpg
imgs/original/rs_1024x759-170531155351-1024-Kristofer-Hivju-game-of-t
hrones.jpg
imgs/original/rs_1024x759-170531155430-1024-Conleth-Hill-game-of-thro
nes.jpg
imgs/original/rs_1024x759-170531160408-1024-Ellie-Kendrick-game-of-th
rones.jpg
imgs/original/rs_1024x759-170630100059-1024-hannah-murray-game-of-thr
ones.jpg
imgs/original/rs_1024x759-170630100540-1024-John-Bradley-game-of-thro
nes.jpg
imgs/original/rs_1024x759-170630103021-1024-Richard-Dormer-game-of-th
rones.jpg
```

😕|😯 One of the actor

display image is easy in Python, for instance load by function, `matplotlib.image` , and show the image by `matplotlib.imshow` :

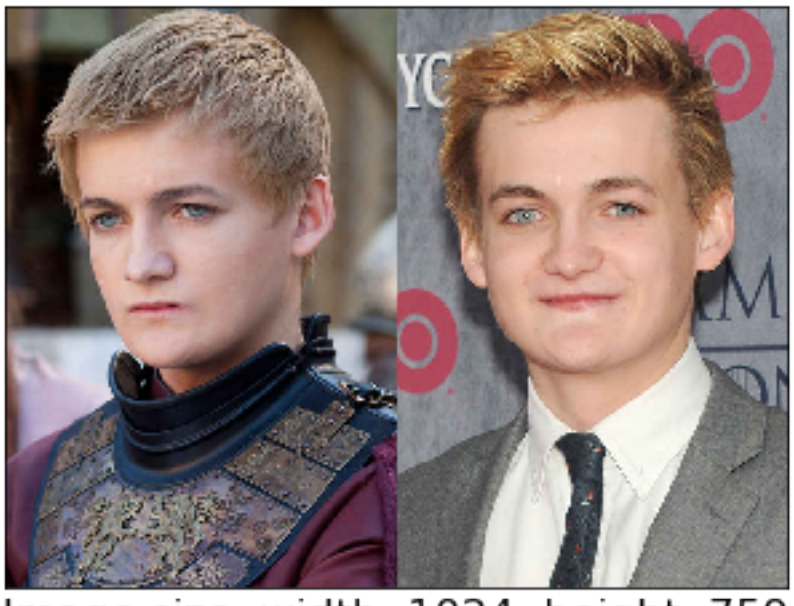


Image size, width: 1024, height: 759

```
array([[[111,  94,  68],
        [111,  94,  68],
        [111,  94,  68],
        ...,
        [139, 143, 152],
        [133, 137, 146],
        [136, 140, 149]],

       [[111,  94,  68],
        [111,  94,  68],
        [111,  94,  68],
        ...,
        [136, 140, 149],
        [130, 134, 143],
        [134, 138, 147]],

       [[111,  94,  68],
        [111,  94,  68],
        [111,  94,  68],
        ...,
        [134, 138, 147],
        [128, 132, 141],
        [132, 136, 145]],

       ...,
       [[ 47,  20,  35],
        [ 38,  14,  28],
        [ 28,   6,  19],
        ...,
        [155, 155, 155],
        [147, 147, 147],
        [129, 129, 129]],

      [[ 48,  21,  36],
       [ 36,  12,  26],
       [ 26,   4,  17],
       ...,
       [160, 160, 160],
       [149, 149, 149],
       [127, 127, 127]],

      [[ 47,  20,  35],
       [ 34,  10,  24],
       [ 23,   1,  14],
       ...,
       [160, 160, 160],
       [146, 146, 146],
       [120, 120, 120]]], dtype=uint8)
```

Generally, Image package loads image into 2D array with each element, i.e. pixel, in (r,g,b) format, each with integer value during (0,255). But opencv uses the reverse format (b,g,r)

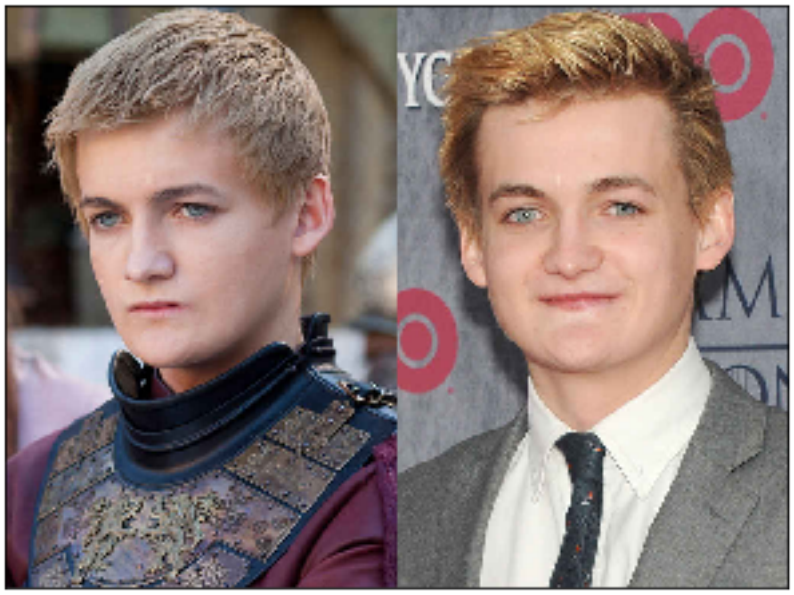


Image size, width: 1024, height: 759

Image size, width: 1024, height: 759

õ|o Split one into two equally

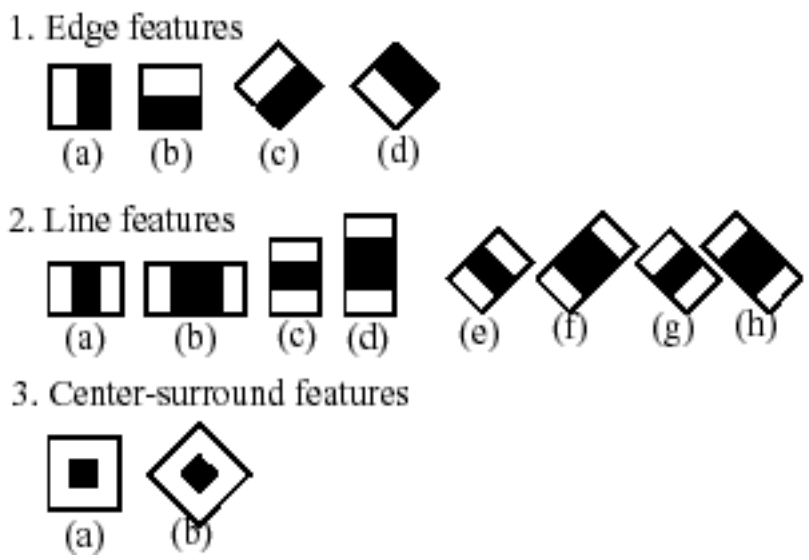
```
(0,0)--------- x ------------ (2 x grid,0)
  |            (xgrid,0)           |
  |              |                 |
  |              |                 |
  |            (xgrid,ygrid)       |
(0.ygrid)-----  x ---------- ((2 x grid,ygrid)
```

```
# by PIL/pillow

img = Image.open(files[0])
(width, height)=img.size
print("Image size, width: %s, height: %s"  %(img.size[0],img.size[1]))
```

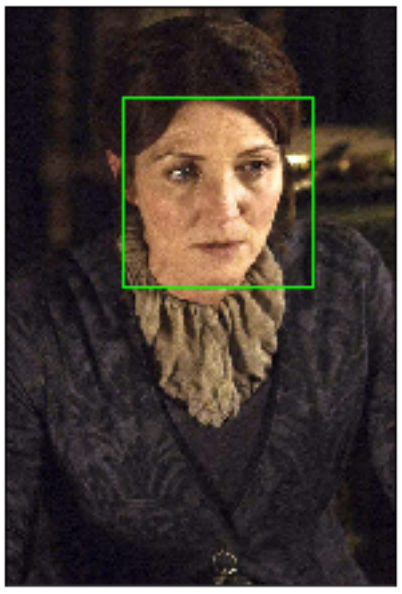
Trained data fron OpenCV

trained classifiers for detecting objects of a particular type, e.g. faces (frontal, profile) can be found at sub-folder of OpenCV (https://github.com/opencv/opencv/tree/master/data), **data**.

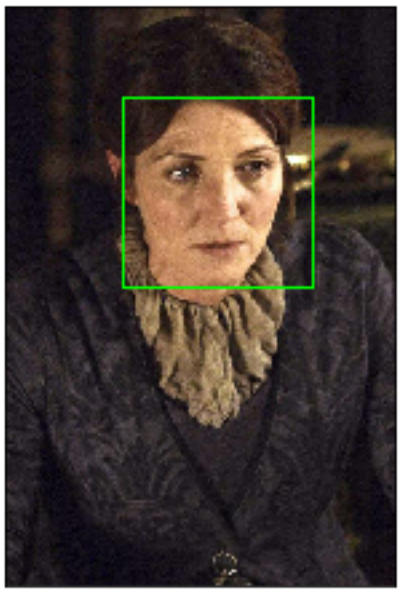


- `cv2.rectangle(img, (x, y), (x+w, y+h), (r, g, b), width)` , create a rectangle with width x height, w x h.

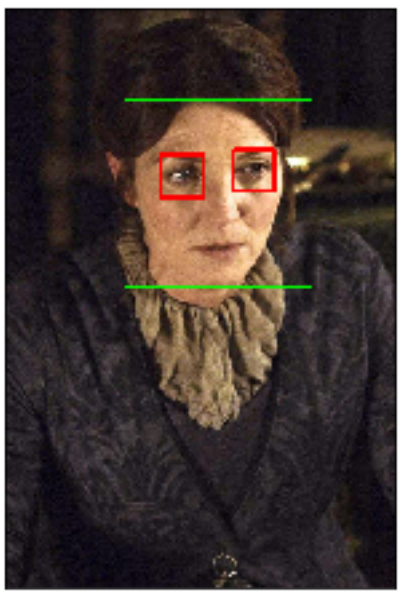
```
Found 1 faces!
Grabbed face picture saved in imgs/cvdata/, (247 px x 247 px), as m_7
_1.jpg
```



```
Found 1 faces!
Grabbed face picture saved in imgs/cvdata/, (247 px x 247 px), as m_7
_1.jpg
```



```
Found 1 faces!
Grabbed face picture saved in imgs/cvdata/, (244 px x 244 px), as m_7
_1.jpg
```



Flask

A compact Python web-development packages, which capable of Model-View-Template (MVT) framework.

1. install flask

```
pip install flask
```

Five Minutes for Basics of Flask;

1. Here the web sites host designed at box is as follows:

```
▼[ App Folder]
    ▼ [ templated]
        ▼[ imgs]
        index.html
    app.py
```

2. **app.py** is the main code of the web app:

```
# import the functions of flask
from flask import Flask, render_template

# define the falsk server
app = Flask(__name__)

# setup the home page, index.html
@app.route('/')
def index():
    return render_template('index.html')

# start the flask server while code was run
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

- we use the route decorator, @app.route('/'), to specify the URL that should trigger the execution of the index function.
- We used the `run()` function to only run the application on the server when this script is directly executed by the Python interpreter (not being imported), which we ensured using the if statement with `__ name __ == ' __ main __ `.
- The [templates] directory is the directory where Flask will look for static HTML files to render, here index.html is the defaulted one.

After completing setup, startup the Flask to provide the service:

```
> python app.py
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 ...
```

Layout Design

According to the purpose of app's function provided, the layout called by `index()` function, might be very complicated. Using wtforms, we could extend the index() function with interactivity much flexibly.

Face Recognition Web-based App

- Project Tree

```
▼[ App Folder]
    web_service_example-2.py

  ▼[ templated]
    ▼[ imgs]
    ▼  [ classifier]
         haarcascade_frontalface_default.xml
      index.html
```

- code

```
import face_recognition
from flask import Flask, jsonify, request, redirect, send_file
import cv2

# You can change this to any folder on your system
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}

app = Flask(__name__)

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/', methods=['GET', 'POST'])
def upload_image():
    # Check if a valid image file was uploaded
    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)

        file = request.files['file']

        if file.filename == '':
            return redirect(request.url)

        if file and allowed_file(file.filename):
            # The image file seems valid! Detect faces and return the resu
lt.
            return detect_faces_in_image(file)
```

```
    # If no valid image file was uploaded, show the file upload form:
    return '''
    <!doctype html>
    <title>Face Recognition Web App</title>
    <h1>Face Recognition App</h1>
    <form method="POST" enctype="multipart/form-data">
      <input type="file" name="file">
      <input type="submit" value="Upload">
    </form>
    '''
def detect_faces_in_image(file_stream):
    # Load the uploaded image file
    img = face_recognition.load_image_file(file_stream)
    #img = cv2.imread(file_stream)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    faceCascade = cv2.CascadeClassifier('classifier/haarcascade_frontalfac
e_default.xml')
    # Get face encodings for any faces in the uploaded image
    #unknown_face_encodings = face_recognition.face_encodings(img)

    faces = faceCascade.detectMultiScale(
            gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30) )
    #flags = cv2.CV_HAAR_SCALE_IMAGE

    # Draw a rectangle around the faces
    for (x, y, w, h) in faces:
        faceimage=img[y:y+h,x:x+w]
        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

    # convert color from bgr to rgb
    img1=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    cv2.imwrite("templates/imgs/test.jpg", img1)
    face_found = False

    #if len(faces) > 0:
    #    face_found = True

    # Return the result as json
    result = {
        "face_found_in_image": len(faces)
    }
    if len(faces) > 0:
       return send_file("templates/imgs/test.jpg", mimetype='image/jpeg')
    else:
       return jsonify(result)
```

```
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5001, debug=True)
```

- Now open a console under the folder with `web_service_example-2.py` , and run the web application:

  ```
  > python web_service_example-2.py
  ```

  Open browser and visit "127.0.0.1:888"; upload image and test the face recogntion example.
- Change host address if any real IP given which could provides really online service.
- This could be easily implemented as mobile app, for instance Android.

```
# import necessary packages
import face_recognition
from flask import Flask, jsonify, request, redirect, send_file
import cv2

# You can change this to any folder on your system
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}

app = Flask(__name__)

# check uploaded image whether be the valid format
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

```
# design the (web) page UI,
# if not upload yet, show the form for user to do so; if yes, and check validable
file, call
# function, dectect_face_in_image(), to work and return the result

@app.route('/', methods=['GET', 'POST'])
def upload_image():
    # Check if a valid image file was uploaded
    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)

        file = request.files['file']

        if file.filename == '':
            return redirect(request.url)

        if file and allowed_file(file.filename):
            # The image file seems valid! Detect faces and return the result.
            return detect_faces_in_image(file)

    # If no valid image file was uploaded, show the file upload form:
    return '''
    <!doctype html>
    <title>Face Recognition Web App</title>
    <h1>Face Recognition App</h1>
    <form method="POST" enctype="multipart/form-data">
      <input type="file" name="file">
      <input type="submit" value="Upload">
```

```
    </form>
    '''

#
#  Same as introduced above

def detect_faces_in_image(file_stream):
    # Load the uploaded image file
    img = face_recognition.load_image_file(file_stream)
    #img = cv2.imread(file_stream)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    faceCascade =
cv2.CascadeClassifier('classifier/haarcascade_frontalface_default.xml')
    # Get face encodings for any faces in the uploaded image
    #unknown_face_encodings = face_recognition.face_encodings(img)

    faces = faceCascade.detectMultiScale(
            gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30) )
    #flags = cv2.CV_HAAR_SCALE_IMAGE

    # Draw a rectangle around the faces
    for (x, y, w, h) in faces:
        faceimage=img[y:y+h,x:x+w]
        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

    # convert color from bgr to rgb
    img1=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    cv2.imwrite("templates/imgs/test.jpg", img1)
    face_found = False

    #if len(faces) > 0:
    #    face_found = True

    # Return the result as json
    result = {
        "face_found_in_image": len(faces)
    }

    # if any face detected, return the image with detect marker around face, if
not, return the text result
    if len(faces) > 0:
       return send_file("templates/imgs/test.jpg", mimetype='image/jpeg')
    else:
       return jsonify(result)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5001, debug=True)
```

```
70

83 faces found, 2 undetected out
Grabbed face picture, (203 px x 203 px):
```


Computing features from images

Mahotas avails submodule named mahotas.features, where feature computation functions are available.

A commonly used set of texture features is the Haralick. As with many methods in image processing, the name is due to its inventor. These features are texture-based: they distinguish between images that are smooth from those that are patterned, and between different patterns.

A commonly used set of features are the Haralick texture features. As with many methods in image processing, this method was named after its inventor. These features are texture-based: they distinguish between images that are smooth and those that are patterned and have between different patterns. With mahotas, it is very easy to compute them:

```
haralick_features = np.mean(mh.features.haralick(image),0)
```

The function mh.features.haralick returns a 4x13 array. The rst dimension refers to four possible directions in which to compute the features (up, down, left, and right). If we are not interested in the direction, we can use the mean overall directions. Based on this function, it is very easy to classify a system.

```
array(['m_0_1', 'm_0_2', 'm_10_1', 'm_10_2', 'm_11_1', 'm_11_2', 'm_1
2_1',
       'm_12_2', 'm_13_1', 'm_13_2', 'm_14_1', 'm_14_2', 'm_15_1',
       'm_15_2', 'm_16_1', 'm_16_2', 'm_17_1', 'm_17_2', 'm_18_1',
       'm_18_2', 'm_19_1', 'm_19_2', 'm_1_1', 'm_1_2', 'm_20_1', 'm_2
0_2',
       'm_21_1', 'm_21_2', 'm_22_1', 'm_22_2', 'm_23_1', 'm_23_2',
       'm_24_1', 'm_24_2', 'm_25_1', 'm_25_2', 'm_26_1', 'm_26_2',
       'm_27_1', 'm_27_2', 'm_28_1', 'm_28_2', 'm_29_1', 'm_29_2', 'm
_2_1',
       'm_2_2', 'm_30_1', 'm_30_2', 'm_31_1', 'm_31_2', 'm_32_1', 'm_
32_2',
       'm_33_1', 'm_33_2', 'm_34_1', 'm_34_2', 'm_3_1', 'm_3_2', 'm_4
_1',
       'm_4_2', 'm_5_1', 'm_5_2', 'm_6_1', 'm_6_2', 'm_7_1', 'm_7_2',
       'm_8_1', 'm_8_2', 'm_9_1', 'm_9_2'],
      dtype='<U6')

Accuracy: 0.0%

Accuracy: 0.0%
```

Training Images Data

1. link libfreetyle.dylib where it is
2. install harfbuzz-1.5.tar.bz2
3. download opencv_contrib
4. rebuild cv2

```
    opencv2/bld> cmake -DOPENCV_EXTRA_MODULES_PATH=../openc
v_contrib/modules ..
    opencv2/bld> make -j5
```

```
[0,
 0,
 2,
 3,
 3,
 5,
 6,
 7,
 8,
 9,
 10,
 11,
 12,
 13,
 14,
 15,
 15,
 15,
 18,
```

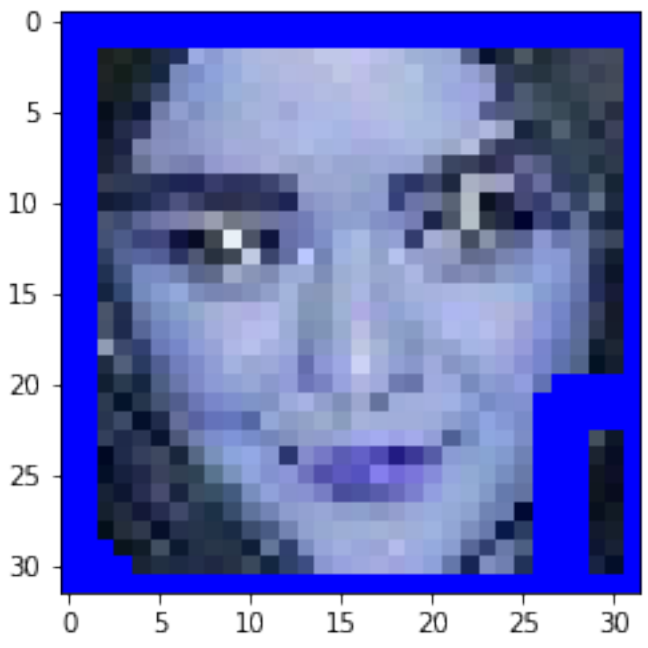

```
SSIM: 0.39700762748524776
```

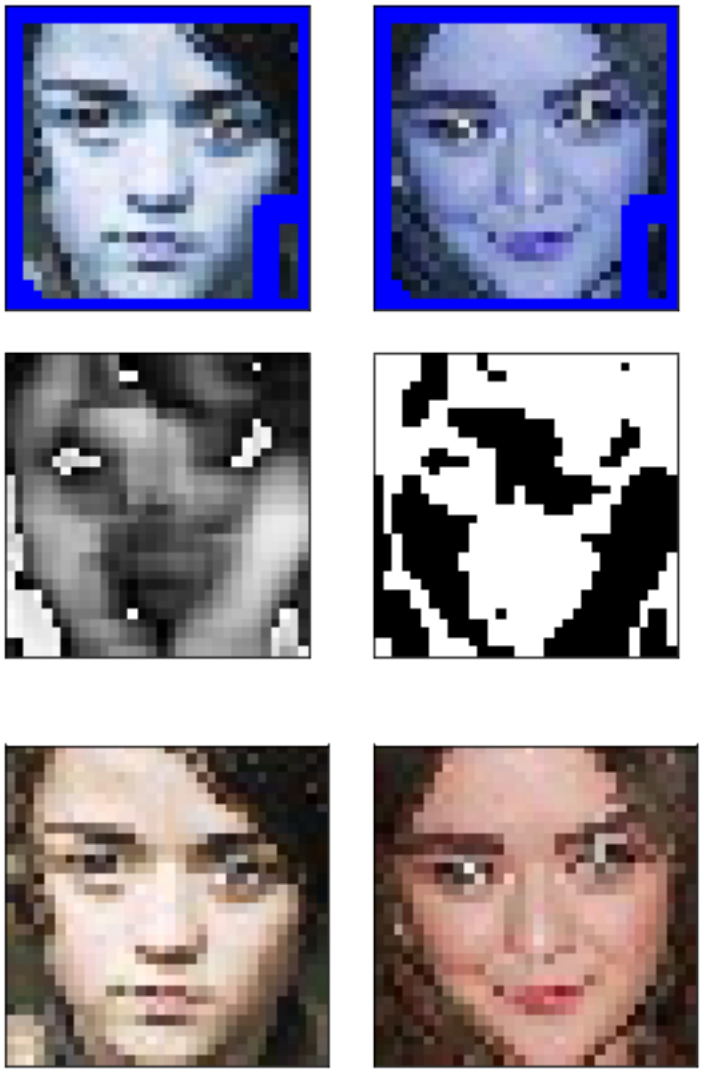
1. Using the compare_ssim function from scikit-image, we calculate a score and difference image, diff (Line 1).
2. The score represents the structural similarity index between the two input images. This value can fall into the range [-1, 1] with a value of one being a “perfect match”.
3. The diff image contains the actual image differences between the two input images that we wish to visualize. The difference image is currently represented as a floating point data type in the range [0, 1] so we first convert the array to 8-bit unsigned integers in the range [0, 255] (Line 2) before we can further process it using OpenCV.

Now, let’s find the contours so that we can place rectangles around the regions identified as “different”:

- On [Line 1] we threshold our diff image using both cv2.THRESH_BINARY_INV and cv2.THRESH_OTSU ,both of these settings are applied at the same time using the vertical bar ‘or’ symbol, | . For details on Otsu’s bimodal thresholding setting, see this OpenCV documentation.
- Subsequently we find the contours of thresh on Lines 2. The ternary operator on Line 2 simply accommodates difference between the cv2.findContours return signature in OpenCV 2.4 and OpenCV 3, respectively.

```
<matplotlib.image.AxesImage at 0x122a01fd0>
```





A image hashing library written in Python. ImageHash supports:

- average hashing (aHash)
- perception hashing (pHash)
- difference hashing (dHash)
- wavelet hashing (wHash)

```
imgs/cvdata/27-1.jpg imgs/cvdata/m_19_1.jpg
imgs/cvdata/52-1.jpg imgs/cvdata/m_2_1.jpg
imgs/cvdata/72-1.jpg imgs/cvdata/84-1.jpg
imgs/cvdata/74-1.jpg imgs/cvdata/85-1.jpg
imgs/cvdata/75-1.jpg imgs/cvdata/88-1.jpg
imgs/cvdata/76-1.jpg imgs/cvdata/87-1.jpg
imgs/cvdata/77-1.jpg imgs/cvdata/89-1.jpg imgs/cvdata/m_7_1.jpg
imgs/cvdata/78-1.jpg imgs/cvdata/90-1.jpg
imgs/cvdata/79-1.jpg imgs/cvdata/91-1.jpg
imgs/cvdata/81-1.jpg imgs/cvdata/93-1.jpg
imgs/cvdata/82-1.jpg imgs/cvdata/94-1.jpg
```

Face Recognition not detection

Link: Adam Geitgey site (https://github.com/ageitgey), face_recognition and face_recognition_models.

Facial Recognition Using Deep Learning (https://medium.com/towards-data-science/facial-recognition-using-deep-learning-a74e9059a150)

1. we will be taking: Detect/identify faces in an image (using a face detection model)—for simplicity, this tutorial will only use images with one face/person in it, not more/less
2. Predict face poses/landmarks (for the faces identified in step 1)
3. Using data from step 2 and the actual image, calculate face encodings (numbers that describe the face)
4. Compare the face encodings of known faces with those from test images to tell who is in the picture

dlib

compiler:

```
shell> mkdir build; cd build; cmake .. ; cmake --build .

Might change setting by "ccmake .", for instance, install directory.

shell> python setup.py install --no DLIB_USE_CUDA
  or
shell>   python setup.py install --yes USE_AVX_INSTRUCTIONS
```

Check the version of X11-dev, $anaconda/include/X11 and /usr/include/X11.

Pre-trained Data

1. **[dlib]**: dlib_face_recognition_resnet_model_v1.dat.bz2 (http://dlib.net/files/dlib_face_recognition_resnet_model_v1.dat.bz2)
2. **[dlib]**: shape_predictor_68_face_landmarks.dat.bz2 (http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2)

Initialize and Setup

Here we import the required library and set up the objects/parameters needed for our face recognition.

```
---------------------------------------------------------------------
------
NameError                                 Traceback (most recent call
last)
<ipython-input-48-8c05f9f76710> in <module>()
      1 # Get Face Detector from dlib
      2 # This allows us to detect faces in images
----> 3 face_detector = dlib.get_frontal_face_detector()
      4 # Get Pose Predictor from dlib
      5 # This allows us to detect landmark points in faces and under
stand the pose/angle of the face

NameError: name 'dlib' is not defined
```

Get face encodings from an image

Here we are writing the function that will take an image filename and give us the face encodings for that image.

Compare faces

Here we are writing the function that will compare a given face encoding with a list of known face encodings. It will return an array of boolean (**True/False**) values that indicate whether or not there was a match.

Find match

Here we are writing the function that will take a list of known face encodings, list of names of people (corresponding to the list of known face encodings) and a face to find a match for. It will call the function in 3a and return the name of the person with whom the given face matches.

```
# This function returns the name of the person whose image matches with the given
face (or 'Not Found')
# known_faces is a list of face encodings
# names is a list of the names of people (in the same order as the face encodings
- to match the name with an encoding)
# face is the face we are looking for
def find_match(known_faces, names, face):
    # Call compare_face_encodings to get a list of True/False values indicating
whether or not there's a match
    matches = compare_face_encodings(known_faces, face)
    # Return the name of the first match
    count = 0
    for match in matches:
        if match:
            return names[count]
        count += 1
    # Return not found if no match found
    return 'Not Found'
```

Getting face encodings

for all faces in the images folder, "imgs/svdata":

Matching each image in test folder with the known faces (one by one)

let us look at the recognition result:

Not bad for this test, right? Especially the fourth comparion, the dragon Queen.

Save/Load Training Model

1. Use the pickle operation to serialize your machine learning algorithms and save the serialized format to a file; later we can load this file to deserialize your model and use it to make new predictions.
2. Joblib is part of the SciPy ecosystem and provides utilities for pipelining Python jobs. It provides utilities for saving and loading Python objects that make use of NumPy data structures, efficiently. This can be useful for some machine learning algorithms that require a lot of parameters or store the entire dataset (like K-Nearest Neighbors).