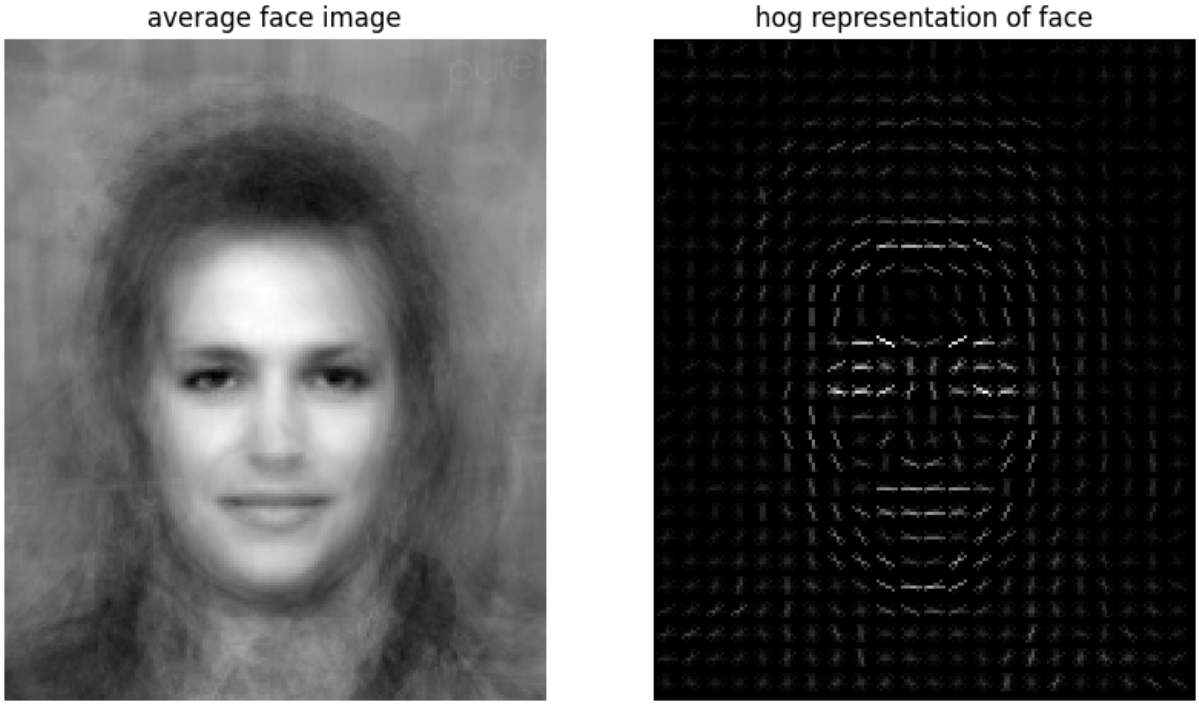
```
from __future__ import print_function
import random
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from skimage import io, color
from skimage.feature import hog
from skimage import data, color, exposure
from skimage.transform import rescale, resize, downscale_local_mean
import glob, os
import fnmatch
import time

import warnings
warnings.filterwarnings('ignore')

from detection import *
from visualization import *
from util import *

# This code is to make matplotlib figures appear inline in the
# notebook rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of p
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# Some more magic so that the notebook will reload external python m
# see http://stackoverflow.com/questions/1907993/autoreload-of-modul
%load_ext autoreload
%autoreload 2
%reload_ext autoreload
```

Part 1: Hog Representation (10 points)

In this section, we will compute the average hog representation of human faces. There are 31 aligned face images provided in the `\face` folder. They are all aligned and have the same size. We will get an average face from these images and compute a hog feature representation for the averaged face.
Use the hog function provided by skimage library, and implement a hog representation of objects. Implement **`hog_feature`** function in `detection.py`

```
image_paths = fnmatch.filter(os.listdir('./face'), '*.jpg')
list.sort(image_paths)
n = len(image_paths)
face_shape, avg_face = load_faces(image_paths, n)

(face_feature, hog_image) = hog_feature(avg_face)

print(np.sum(face_feature))
assert np.abs(np.sum(face_feature) - 499.970465079) < 1e-2

plot_part1(avg_face, hog_image)
```

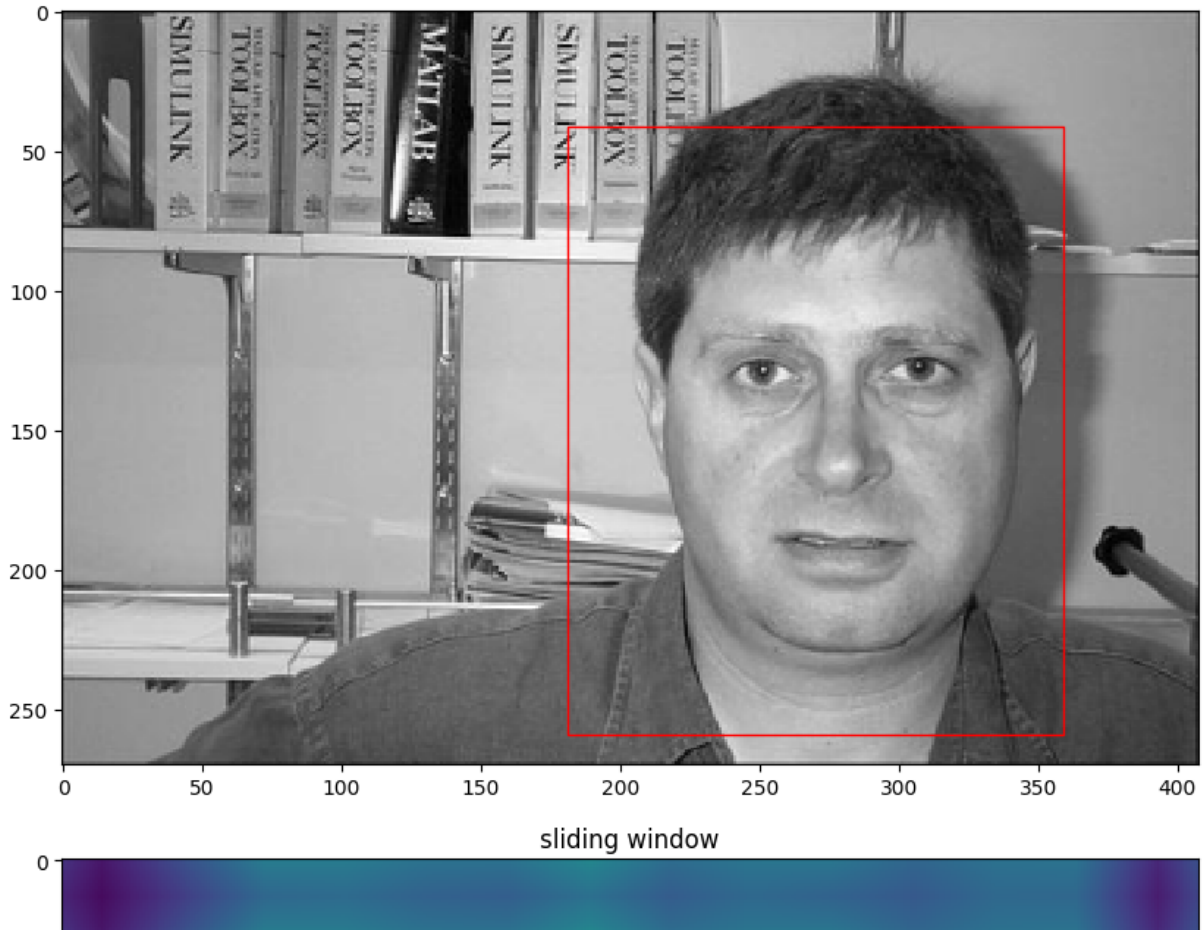
```
499.9704572768695
```

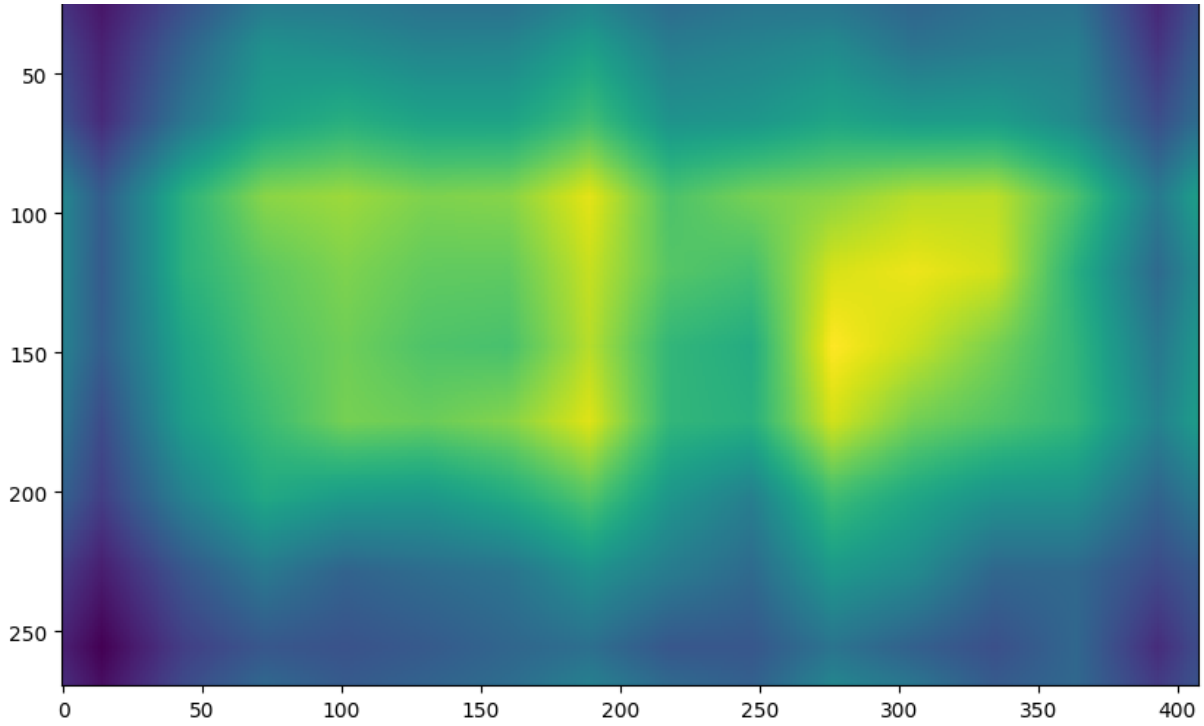


Part 2: Sliding Window (30 points)

Implement `sliding_window` function to have windows slide across an image with a specific window size. The window slides through the image and check if an object is detected with a high score at every location. These scores will generate a response map and you will be able to find the location of the window with the highest hog score.

```
image_path = 'image_0001.jpg'

image = color.rgb2gray(io.imread(image_path))
image = rescale(image, 0.8)

(hogFeature, hogImage) = hog_feature(image)

(winH, winW) = face_shape
(score, r, c, response_map) = sliding_window(image, face_feature, s
crop = image[r:r+winH, c:c+winW]

plot_part2(image, r, c, response_map, winW, winH)
```

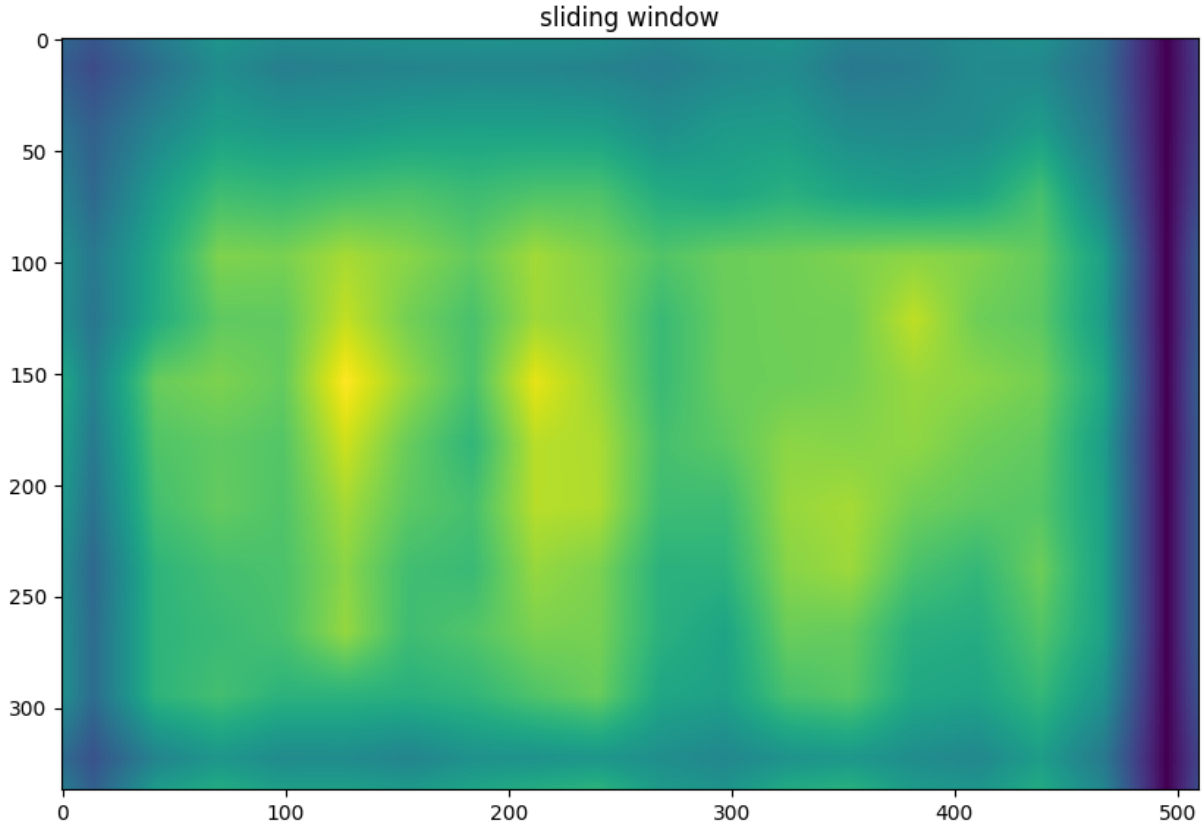




Sliding window successfully found the human face in the above example. However, in the cell below, we are only changing the scale of the image, and you can see that sliding window does not work once the scale of the image is changed.

```
image_path = 'image_0001.jpg'
image = color.rgb2gray(io.imread(image_path))
image = rescale(image, 1.0)
```

```
(winH, winW) = face_shape
(score, r, c, max_response_map) = sliding_window(image, face_featur

crop = image[r:r+winH, c:c+winW]

plot_part2(image, r, c, max_response_map, winW, winH)
```



Part 3: Image Pyramids (25 points)

In order to make sliding window work for different scales of images, you need to implement image pyramids where you resize the image to different scales and run the sliding window method on each resized image. This way you scale the objects and can detect both small and large objects.

3.1 Image Pyramid (10 points)

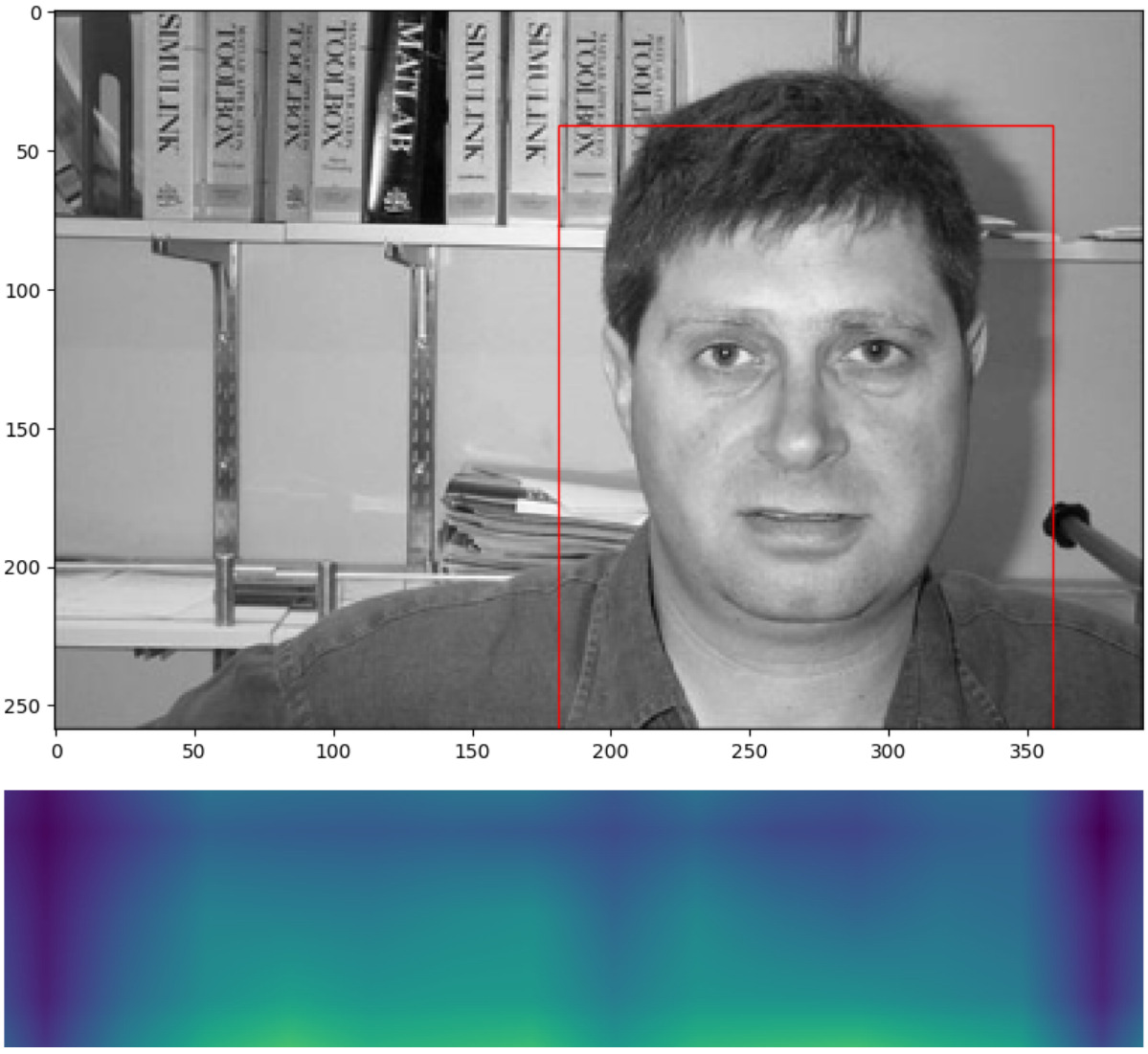
Implement `pyramid` function in `detection.py`, this will create pyramid of images at different scales. Run the following code, and you will see the shape of the original image gets smaller until it reaches a minimum size.

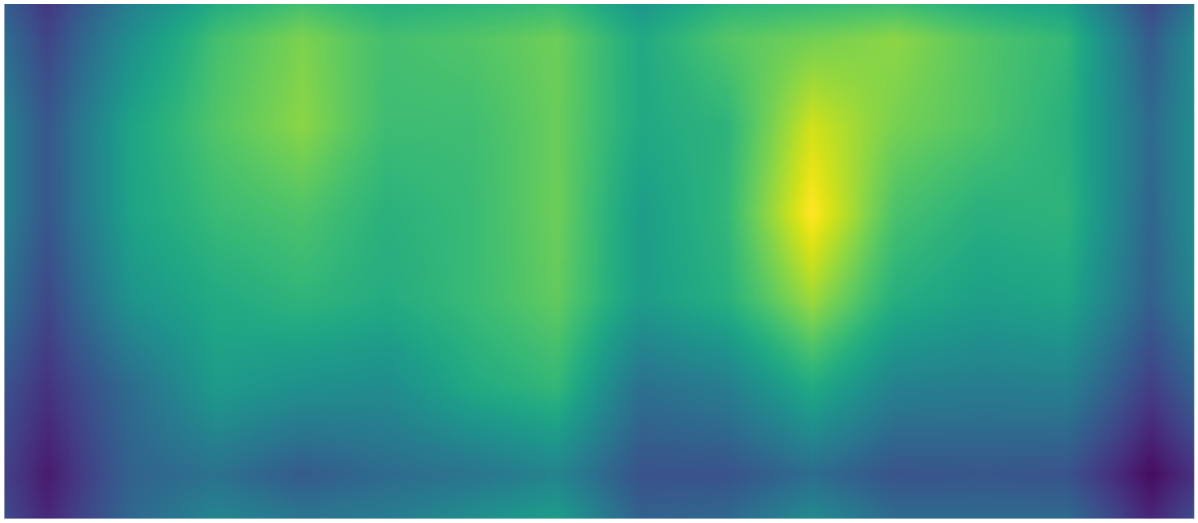
```
image_path = 'image_0001.jpg'

image = color.rgb2gray(io.imread(image_path))
image = rescale(image, 1.2)

images = pyramid(image, scale = 0.9)

plot_part3_1(images)
```

3.2 Pyramid Score (15 points)

After getting the image pyramid, we will run sliding window on all the images to find a place that gets the highest score. Implement **`pyramid_score`** function in `detection.py`. It will return the highest score and its related information in the image pyramids.

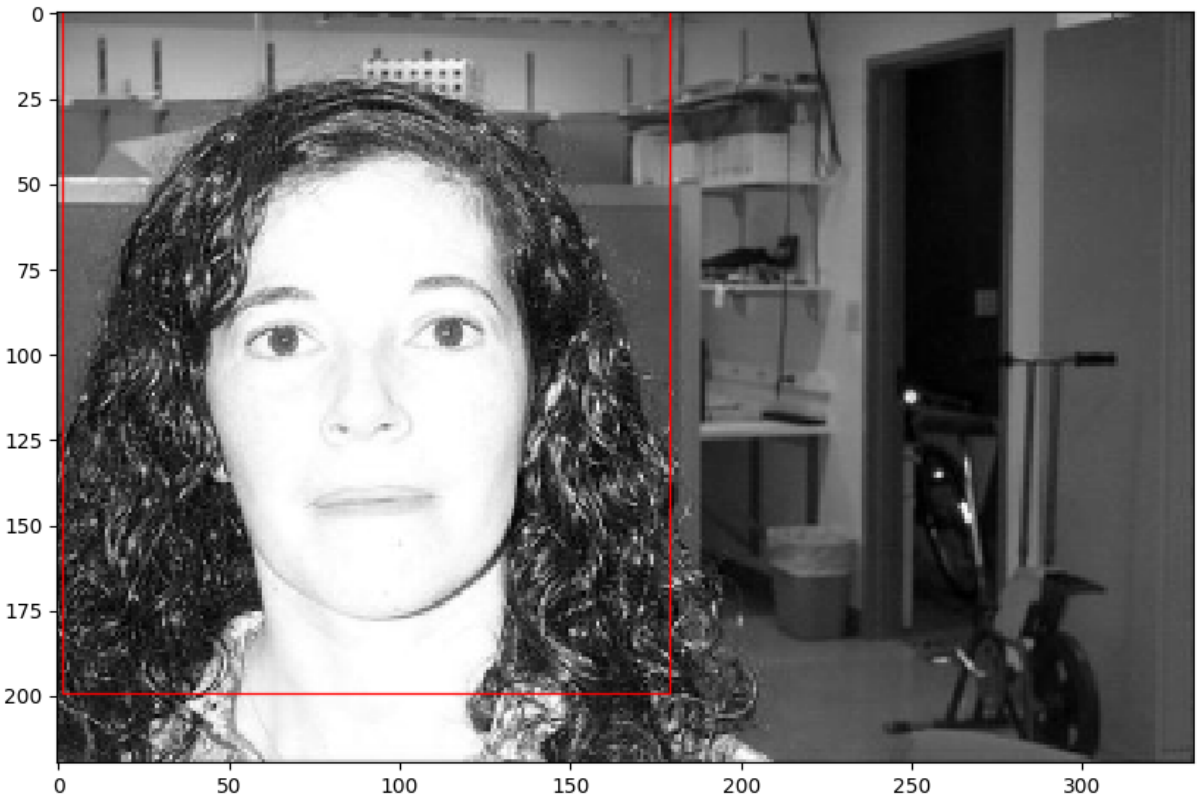
```
image_path = 'image_0001.jpg'

image = color.rgb2gray(io.imread(image_path))
image = rescale(image, 1.2)

(winH, winW) = face_shape
max_score, maxr, maxc, max_scale, max_response_map = pyramid_score
        (image, face_feature, face_shape, stepSize = 30, scale=0.8)

plot_part3_2(image, max_scale, winW, winH, maxc, maxr, max_response
```

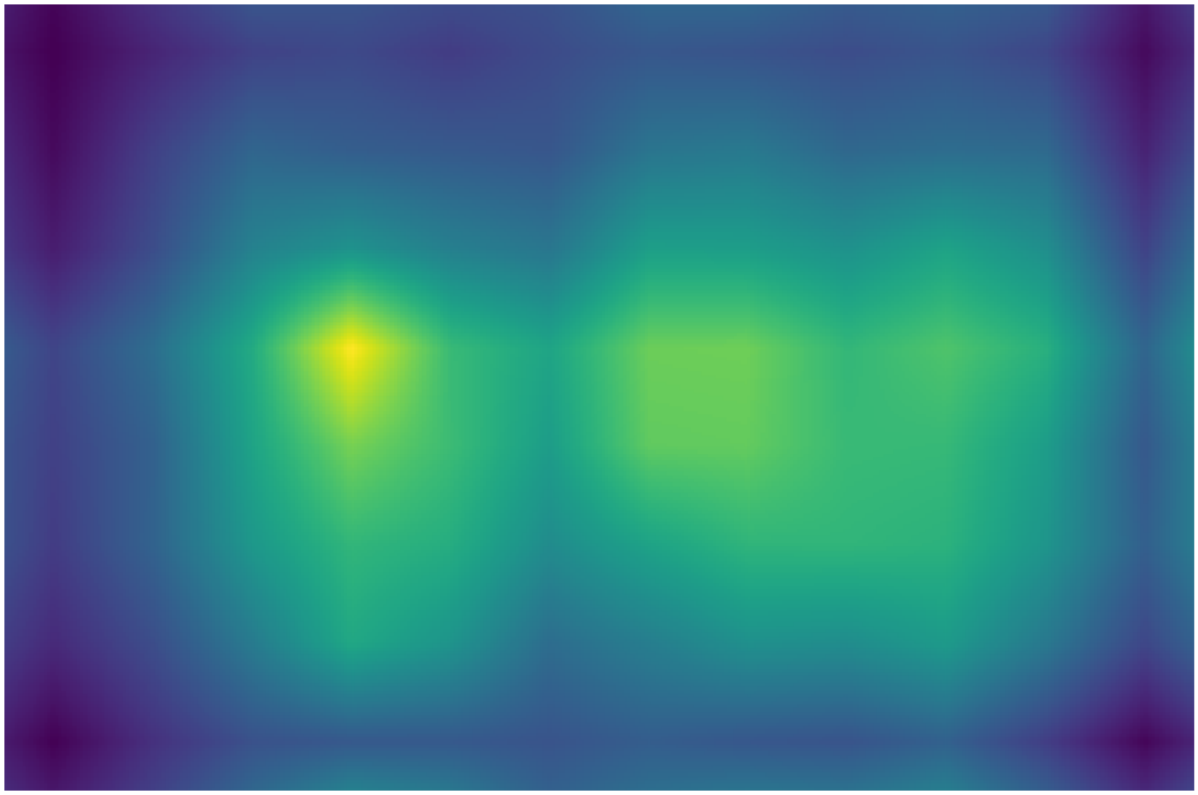




From the above example, we can see that image pyramid has fixed the problem of scaling. Then in the example below, we will try another image and implement a deformable parts model.

```
image_path = 'image_0338.jpg'
image = color.rgb2gray(io.imread(image_path))
image = rescale(image, 1.0)

(winH, winW) = face_shape

max_score, maxr, maxc, max_scale, max_response_map = pyramid_score
    (image, face_feature, face_shape, stepSize = 30, scale=0.8)

plot_part3_2(image, max_scale, winW, winH, maxc, maxr, max_response
```


0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300

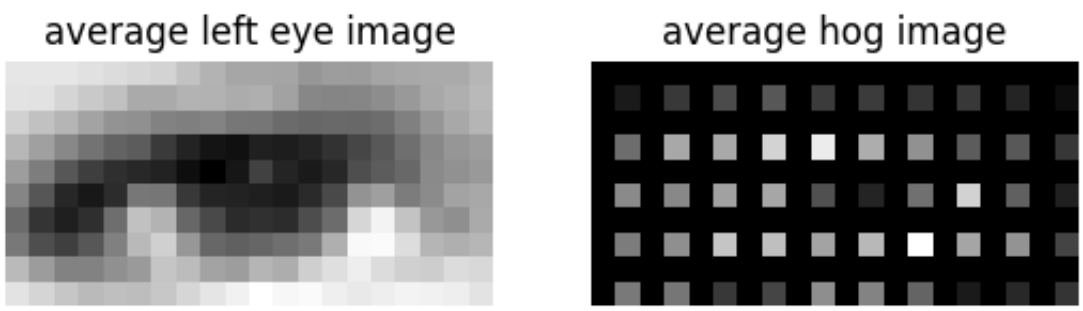


Part 4: Deformable Parts Detection

In order to solve the problem above, you will implement deformable parts model in this section, and apply it on human faces.
The first step is to get a detector for each part of the face, including left eye, right eye, nose and mouth.
For example for the left eye, we have provided the groundtruth location of left eyes for each image in the `\face` directory. This is stored in the `lefteyes` array with shape `(n,2)`, each row is the `(r,c)` location of the center of left eye. You will then find the average hog representation of the left eyes in the images.

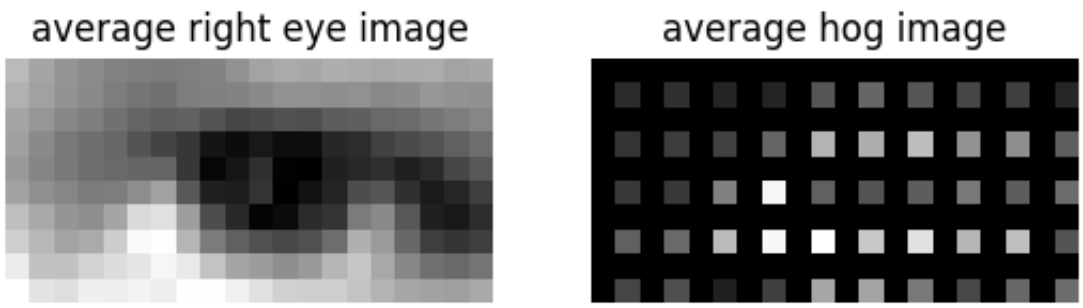
Run through the following code to get a detector for left eyes.

```
image_paths = fnmatch.filter(os.listdir('./face'), '*.jpg')

parts = read_facial_labels(image_paths)
lefteyes, righteyes, noses, mouths = parts

# Typical shape for left eye
lefteye_h = 10
lefteye_w = 20

lefteye_shape = (lefteye_h, lefteye_w)

avg_lefteye = get_detector(lefteye_h, lefteye_w, lefteyes, image_pa
(lefteye_feature, lefteye_hog) = hog_feature(avg_lefteye, pixel_per

plot_part4(avg_lefteye, lefteye_hog, 'left eye')
```



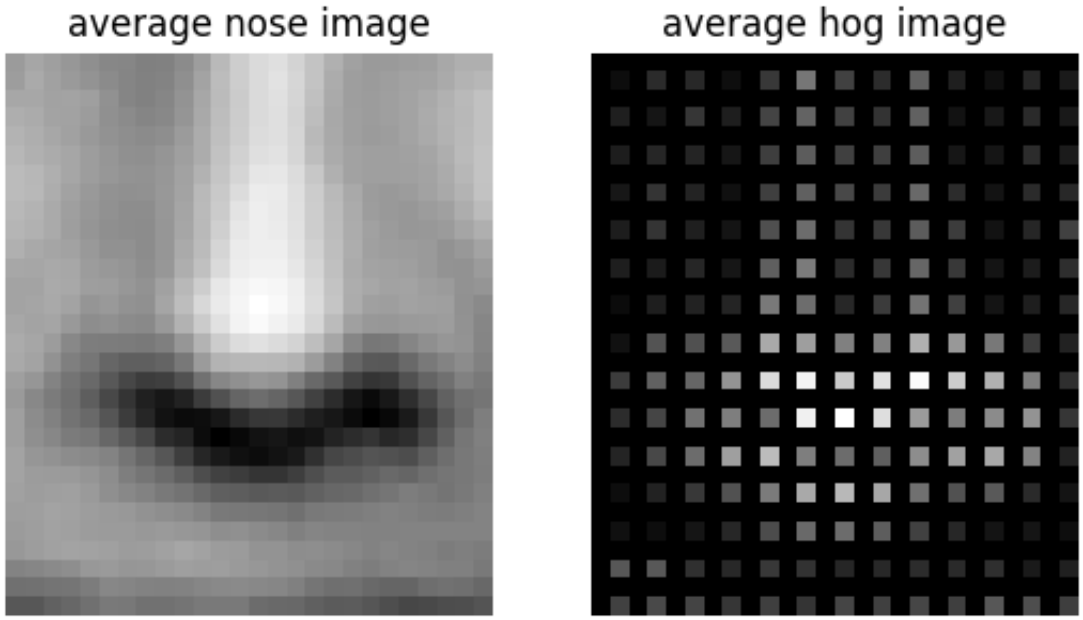
Run through the following code to get a detector for right eye.

```
righteye_h = 10
righteye_w = 20

righteye_shape = (righteye_h, righteye_w)

avg_righteye = get_detector(righteye_h, righteye_w, righteyes, imag

(righteye_feature, righteye_hog) = hog_feature(avg_righteye, pixel_

plot_part4(avg_righteye, righteye_hog, 'right eye')
```



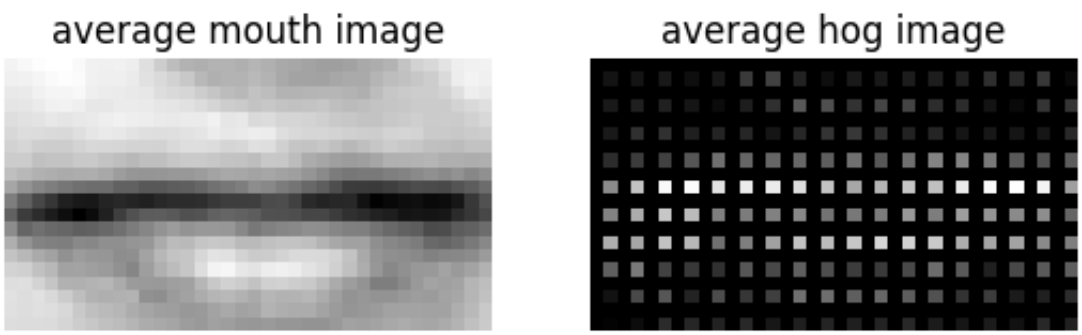
Run through the following code to get a detector for nose.

```
nose_h = 30
nose_w = 26

nose_shape = (nose_h, nose_w)

avg_nose = get_detector(nose_h, nose_w, noses, image_paths)

(nose_feature, nose_hog) = hog_feature(avg_nose, pixel_per_cell=2)

plot_part4(avg_nose, nose_hog, 'nose')
```



Run through the following code to get a detector for mouth

```
mouth_h = 20
mouth_w = 36

mouth_shape = (mouth_h, mouth_w)

avg_mouth = get_detector(mouth_h, mouth_w, mouths, image_paths)

(mouth_feature, mouth_hog) = hog_feature(avg_mouth, pixel_per_cell=

detectors_list = [lefteye_feature, righteye_feature, nose_feature,

plot_part4(avg_mouth, mouth_hog, 'mouth')
```



Part 5: Human Parts Location (15points)

5.1 Compute displacement (10 points)

Implement **`compute_displacement`** to get an average shift vector mu and standard deviation sigma for each part of the face. The vector mu is the distance from the main center, i.e the center of the face, to the center of the part.

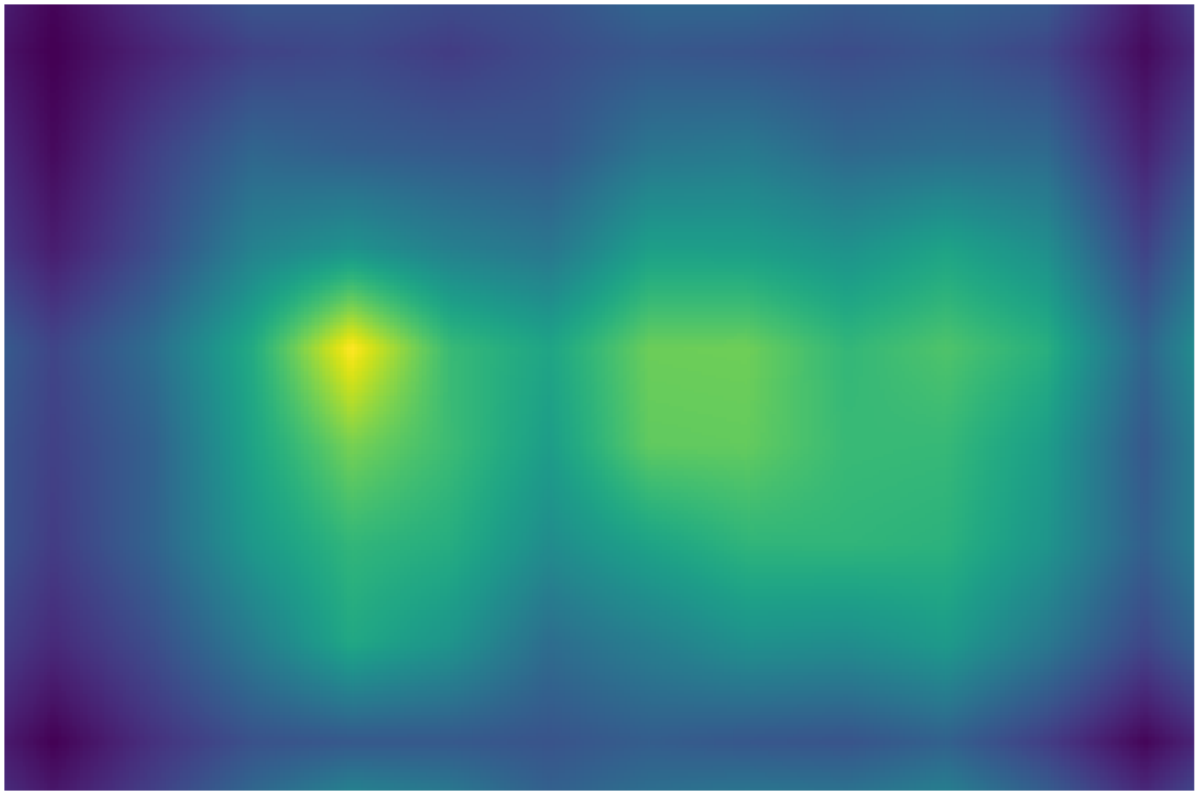
```
# test for compute_displacement
test_array = np.array([[0,1],[1,2],[2,3],[3,4]])
test_shape = (6,6)
mu, std = compute_displacement(test_array, test_shape)
assert(np.all(mu == [1,0]))
assert(np.sum(std-[ 1.11803399,  1.11803399])<1e-5)
print("Your implementation is correct!")
```

```
Your implementation is correct!
```

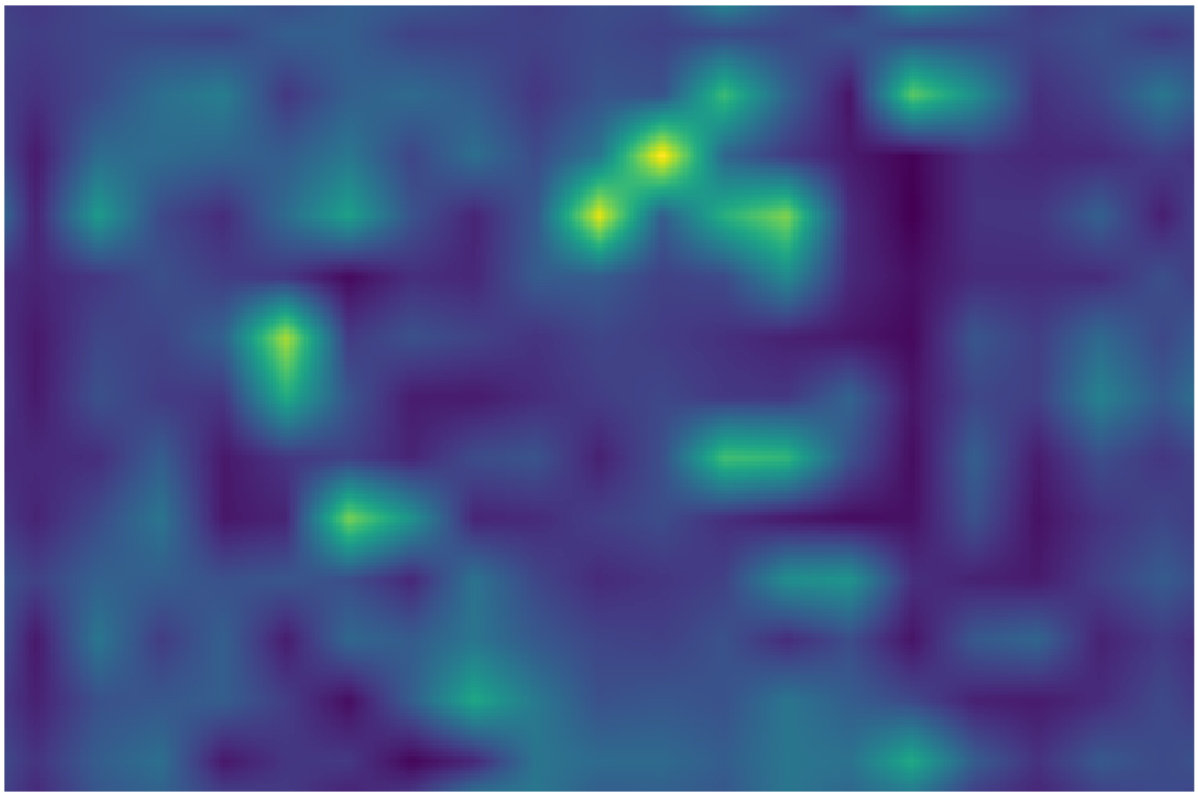
```
    lefteye_mu, lefteye_std = compute_displacement(lefteyes, face_shape
    righteye_mu, righteye_std = compute_displacement(righteyes, face_sh
    nose_mu, nose_std = compute_displacement(noses, face_shape)
    mouth_mu, mouth_std = compute_displacement(mouths, face_shape)
```

After getting the shift vectors, we can run our detector on a test image. We will first run the following code to detect each part of left eye, right eye, nose and mouth in the image. You will see a response map for each of them.

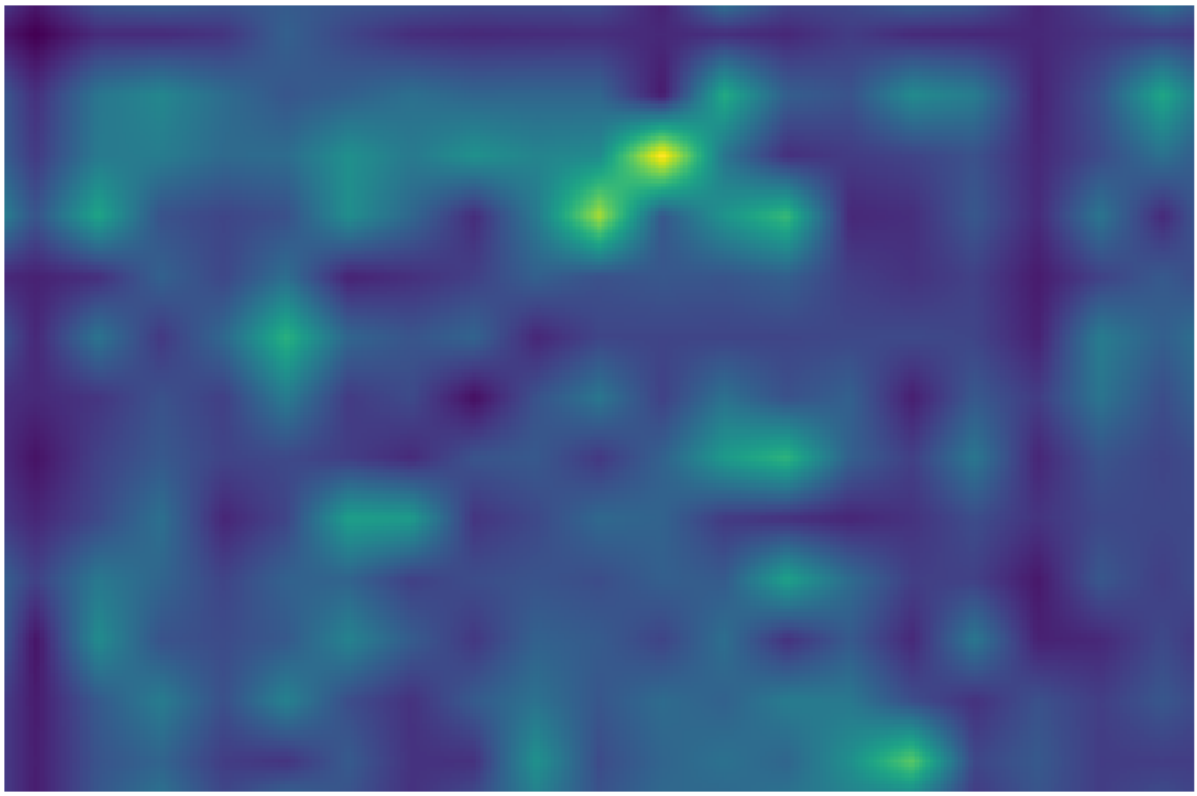
```
image_path = 'image_0338.jpg'
image = color.rgb2gray(io.imread(image_path))
image = rescale(image, 1.0)

(face_H, face_W) = face_shape
max_score, face_r, face_c, face_scale, face_response_map = pyramid_
    (image, face_feature, face_shape,stepSize = 30, scale=0.8)

plot_part5_1(face_response_map)
```



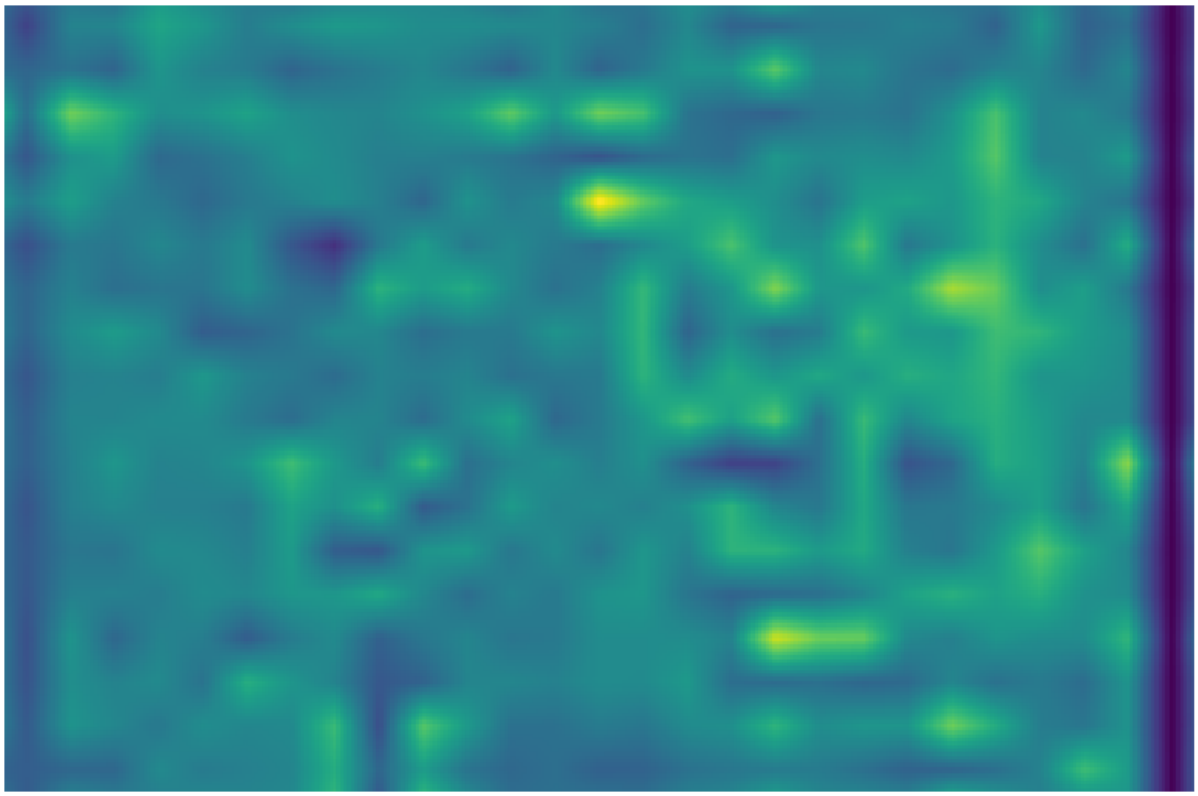
```
max_score, lefteye_r, lefteye_c, lefteye_scale, lefteye_response_ma
    pyramid_score(image, lefteye_feature,lefteye_shape, stepSize =

lefteye_response_map = resize(lefteye_response_map, face_response_m

plot_part5_1(lefteye_response_map)
```



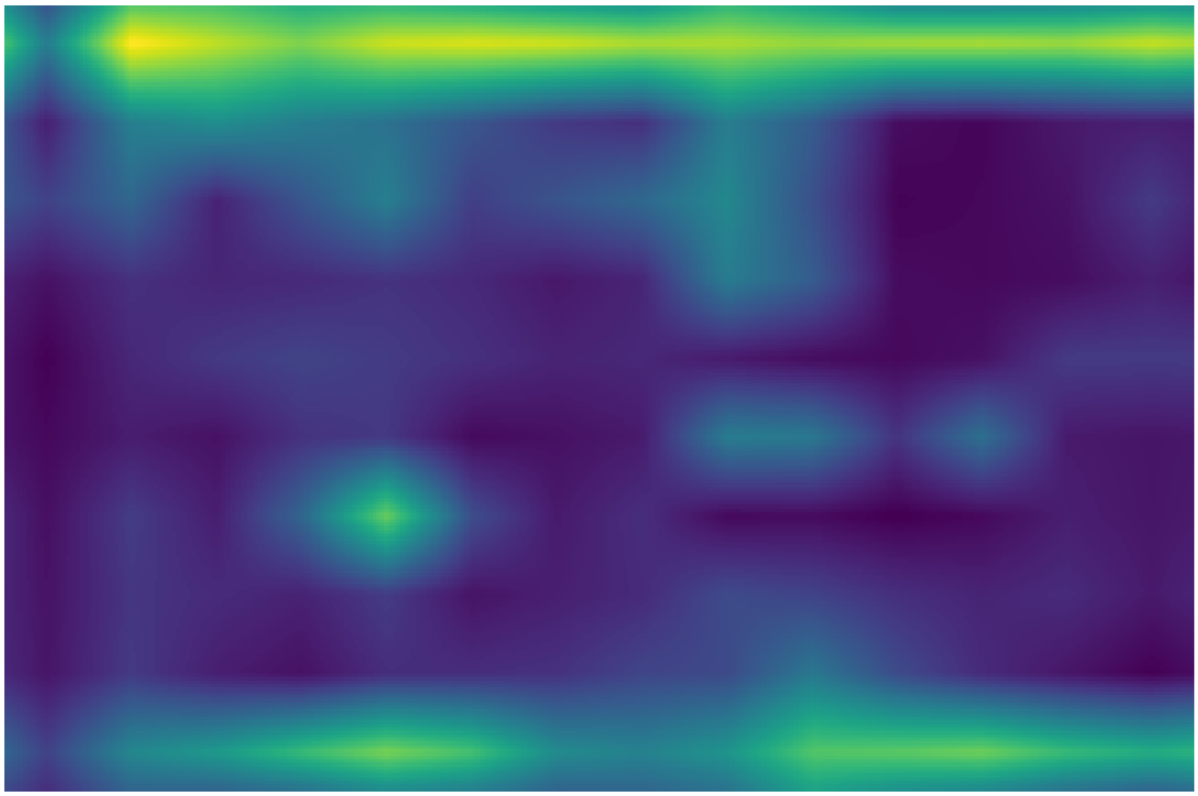
```
max_score, righteye_r, righteye_c, righteye_scale, righteye_respons
    pyramid_score (image, righteye_feature, righteye_shape, stepSiz

righteye_response_map = resize(righteye_response_map, face_response

plot_part5_1(righteye_response_map)
```



```
max_score, nose_r, nose_c, nose_scale, nose_response_map = \
    pyramid_score (image, nose_feature, nose_shape, stepSize = 20,s

nose_response_map = resize(nose_response_map, face_response_map.sha

plot_part5_1(nose_response_map)
```



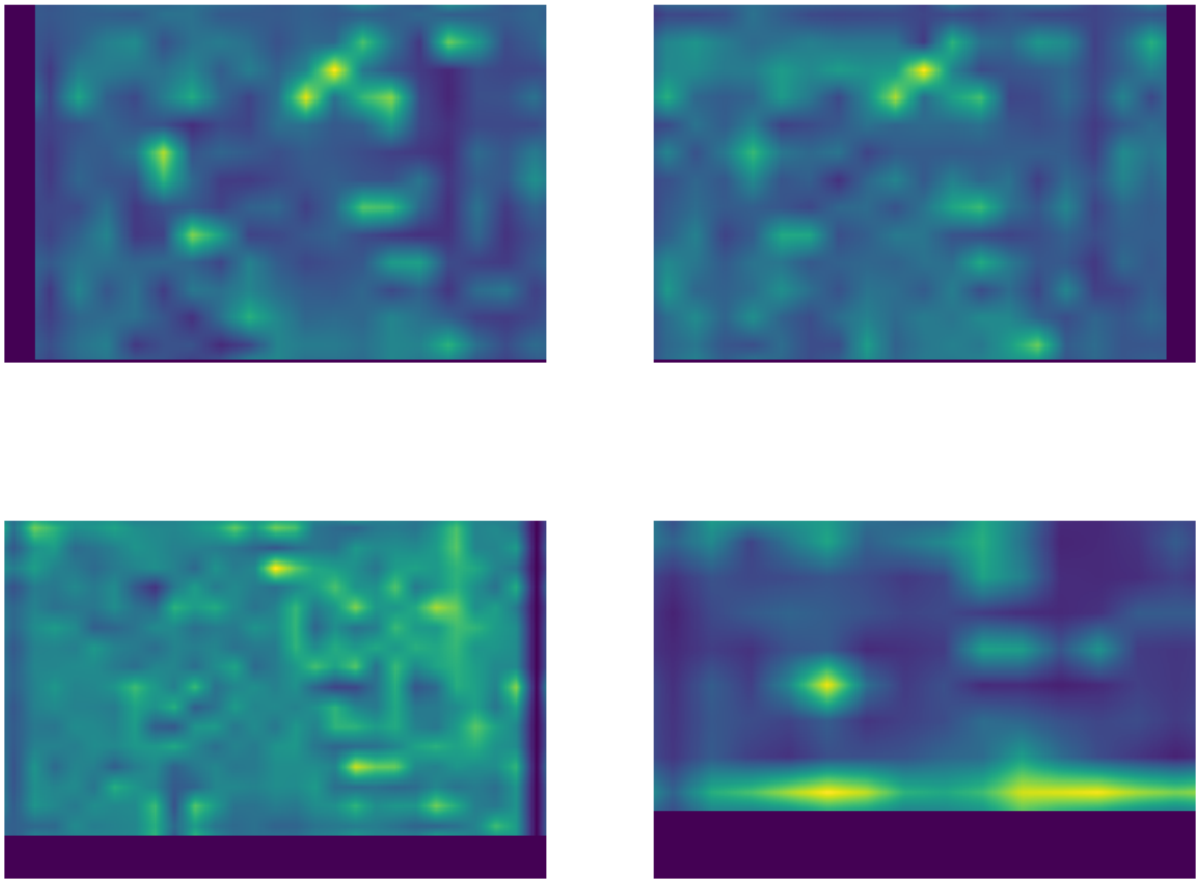
```
max_score, mouth_r, mouth_c, mouth_scale, mouth_response_map =\
    pyramid_score (image, mouth_feature, mouth_shape, stepSize = 20

mouth_response_map = resize(mouth_response_map, face_response_map.s

plot_part5_1(mouth_response_map)
```



5.2 Shift heatmap (5 points)

After getting the response maps for each part of the face, we will shift these maps so that they all have the same center as the face. We have calculated the shift vector mu in `compute_displacement`, so we are shifting based on vector mu. Implement `shift_heatmap` function in `detection.py`.

```
face_heatmap_shifted = shift_heatmap(face_response_map, [0,0])

lefteye_heatmap_shifted = shift_heatmap(lefteye_response_map, lefte
righteye_heatmap_shifted = shift_heatmap(righteye_response_map, rig
nose_heatmap_shifted = shift_heatmap(nose_response_map, nose_mu)
mouth_heatmap_shifted = shift_heatmap(mouth_response_map, mouth_mu)

plot_part5_2(lefteye_heatmap_shifted, righteye_heatmap_shifted,
                nose_heatmap_shifted, mouth_heatmap_shifted)
```



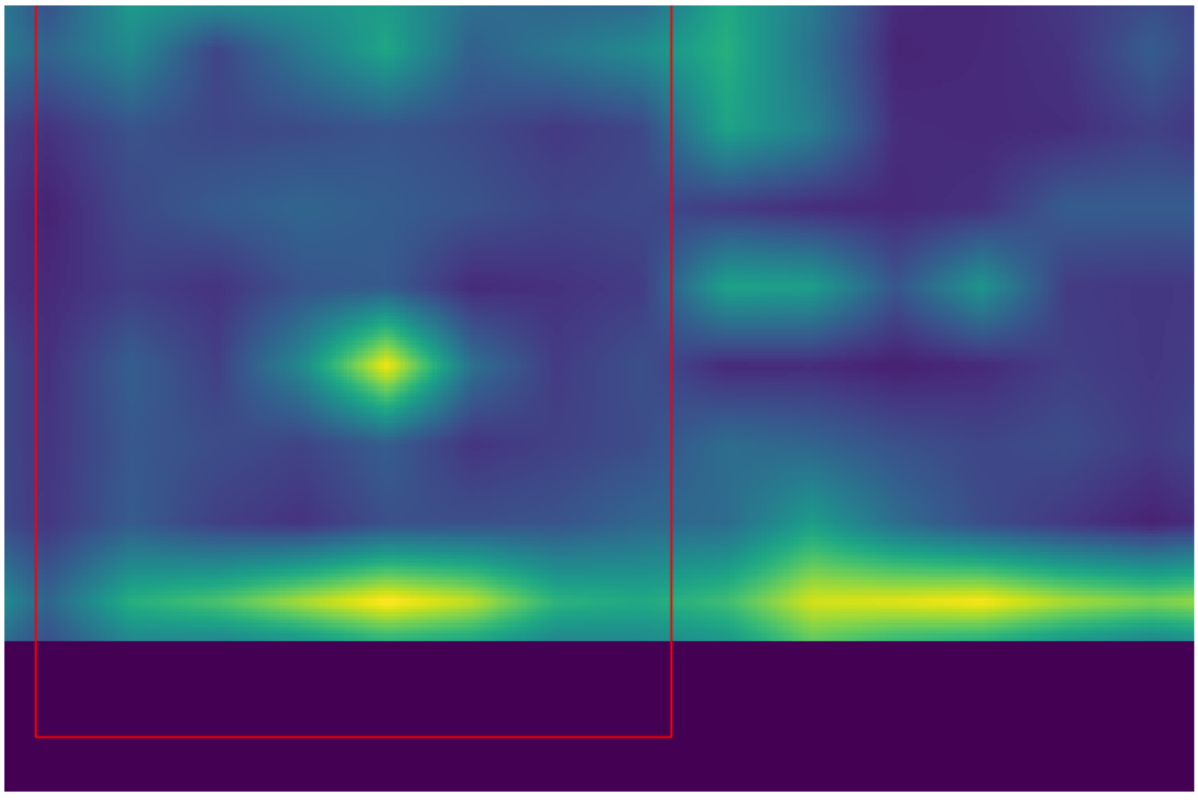
Part 6: Gaussian Filter (20 points)

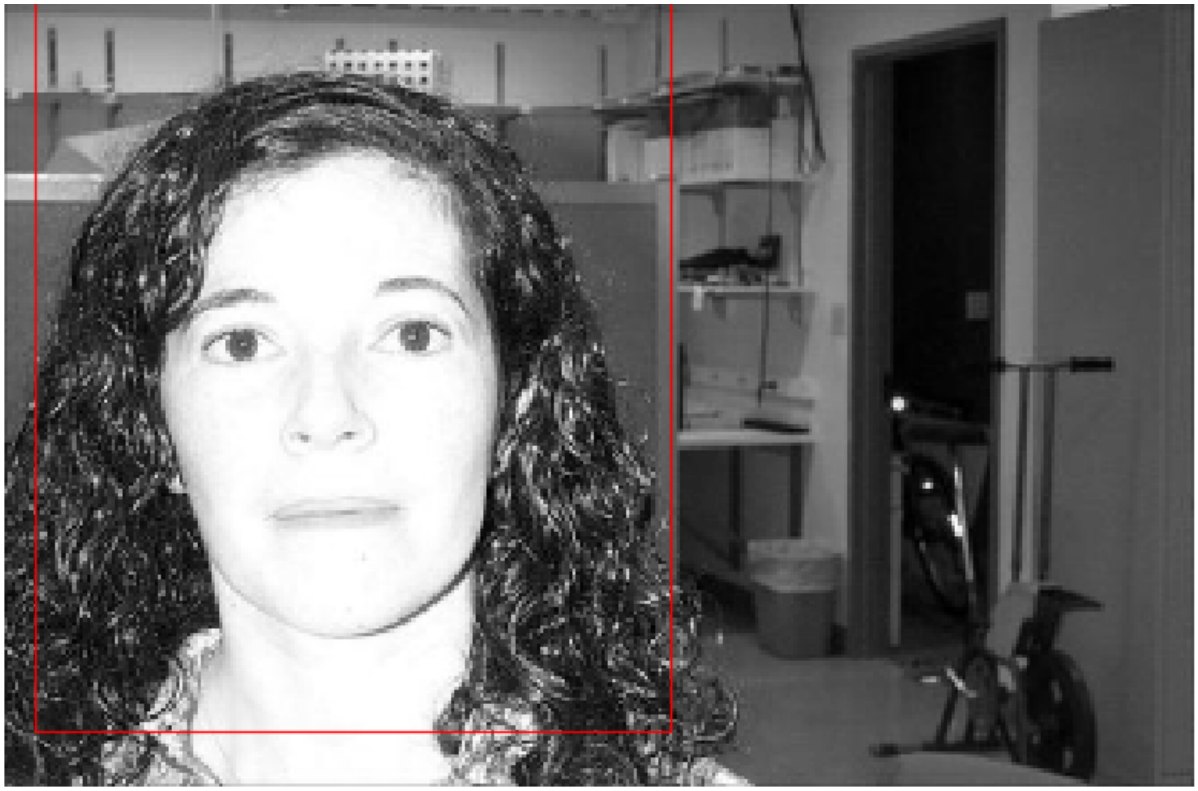
Part 6.1 Gaussian Filter

In this part, apply gaussian filter convolution to each heatmap. Blur by kernel of standard deviation sigma, and then add the heatmaps of the parts with the heatmap of the face. On the combined heatmap, find the maximum value and its location. You can use function provided by skimage to implement **`gaussian_heatmap`**.

```
heatmap_face= face_heatmap_shifted

heatmaps = [lefteye_heatmap_shifted,
            righteye_heatmap_shifted,
            nose_heatmap_shifted,
            mouth_heatmap_shifted]
sigmas = [lefteye_std, righteye_std, nose_std, mouth_std]

heatmap, i , j = gaussian_heatmap(heatmap_face, heatmaps, sigmas)
print(heatmap.shape, image.shape)
plot_part6_1(winH, winW, heatmap, image, i, j)
```

```
(220, 333) (344, 520)
```





6.2 Result Analysis (10 points)

Does your DPM work on detecting human faces? Can you think of a case where DPM may work better than the detector we had in part 3 (sliding window + image pyramid)? You can also have examples that are not faces.