```
========================================
Bella Falbo_899793_assignsubmission_file_/
========================================
List of submitted files:
Bella Falbo.pdf
HW5.py
Untitled.ipynb
extension.txt
feedback
hamilton.jpg
robeson.jpg
========================================
File:  ../../../hw/hw5_rubric.txt
========================================
CSC 268:  Image Processing Fundamentals
Homework 5 Rubric

____ 1 pt: Explicitly identifies sources & consultations (even if none).
           Partner is clearly specified if peer programming.

____ 2 pts: integral image function

____ 2 pts: single-scale f1 value function

____ 2 pts: single-scale f2 value function

____ 4 pts: single-scale face detection function

____ 1 pts: single-scale function tested on images resized by different amounts

____ 1 pt: any scale rectangle-sum function

0.5
____ 1 pt: any scale f1 value function

0.5
____ 1 pt: any scale f2 value function

0.5
____ 1 pt: any scale face detection function

0.5
____ 1 pt: all scale face detection function

0.5
____ 1 pt: tested detection on multiple images

____ 2 pts: reflection included

     ____ 1 pt: reflects on learning goals of the assignment and whether they have been met

     ____ 1 pt: discusses choices made & reasoning behind them


       17.5
Total: ____/20
```

```
========================================

========================================
extension.txt
========================================
Bella Falbo
HW5 originally due APR11
four day extension, now due APR15
accomodations give extension day exception as discussed.
(a longer extension is taken for this assignment due to covid and I wanted to be able to ask
 questions)
========================================
HW5.py
========================================
import cv2 as cv
import numpy as np
from scipy.ndimage import label
from scipy.spatial import distance_matrix
import matplotlib.pyplot as plt
import matplotlib as mpl
from math import ceil

mpl.rc('image', cmap='gray')

def imshow(img,cmap=None):
    plt.imshow(img)
    plt.axis('off')
    if cmap:
        plt.set_cmap(cmap)
    plt.show()

robeson = cv.imread('robeson.jpg',0).astype(np.float32)/255.0
imshow(robeson)

#create integral image
def integral_image(img):
    return np.cumsum(np.cumsum(img, axis=0), axis=1)
iir = integral_image(robeson)

# return the f1 values for any image.
def f1_values(img,integral):
    r_4x12 = integral[4:,12:]+integral[:-4,:-12]-integral[4:,:-12]-integral[:-4,12:]
    f1 = r_4x12[:-4,:]-r_4x12[4:,:]
    f1pad = np.pad(f1,((4,4),(6,6)))
    plt.figure
    plt.imshow(img)
    y,x = np.nonzero(f1pad>20)#;
    plt.plot(x,y,'r.')#;
    plt.axis('off')
    plt.show()
    return (f1pad)

f1values= f1_values(robeson,iir)

def f2_values(img, integral):
    r_4x4 = integral[4:,4:]+integral[:-4,:-4]-integral[:-4,4:]-integral[4:,:-4]
    f2= 2*r_4x4[:,4:-4]-r_4x4[:,:-8]-r_4x4[:,8:]
    f2pad=np.pad (f2,((2,2),(6,6)))
    plt.figure
    plt.imshow(img)
    y,x = np.nonzero(f2pad>20)#;
    plt.plot(x,y,'r.')#;
    plt.axis('off')
```

```
    plt.show()
    return (f2pad)

f2values=f2_values(robeson,iir)

def showFaces(img,x,y):
    '''Draws approximate boxes around detected face points.'''
    plt.figure
    plt.imshow(robeson)
    ax = plt.gca()
    for i in range(len(y)):
        r = mpl.patches.Rectangle((x[i]-12,y[i]-12),24,36,edgecolor='r',fill=False)
        ax.add_patch(r)
    #plt.plot(x,y,'c.')#;
    plt.axis('off')
    plt.show()

# assumes you have already defined f1p and f2p above.
y,x = np.nonzero(np.logical_and(f1values>20,np.roll(f2values,-4,0)>16))#;
showFaces(robeson,x,y)

# TODO: write a single function that takes an image and returns the y,x coordinates of the
# faces detected using the method above.  Call it on the image scaled by different amounts:
# 1.25, 0.8, 0.64, etc. (use cv.resize).
def face_coord(img,resizeamt):
    img=cv.resize(img,None,fx=resizeamt,fy=resizeamt)
    integralimg= integral_image(img)
    f1value=f1_values(img,integralimg)
    f2value=f2_values(img,integralimg)
    y,x=np.nonzero(np.logical_and(f1value>20,np.roll(f2value,-4,0)>16))
    showFaces(img,x,y)
    return np.dstack([x,y])
```

Thresholds should be made into parameters.

```
print(face_coord(robeson, .64))
print(face_coord(robeson, .8))
print(face_coord(robeson, 1.25))
print(face_coord(robeson, 1.75))

#TODO: To complete the detection system, we therefore need to do a few things:

#Write a general rectangle-sum function that can compute the sums for rectangles of any size
def sum_any_size(img,xdim,ydim):
    integral=integral_image(img)
    block=integral[xdim:,ydim:]+integral[:-xdim,:-ydim]-integral[xdim:,:-ydim]-integral[:-xd
im,ydim:]
    return(block)

#Write functions that can compute the f1 and f2 filters, given a scale. For example, at scal
e
# 1.25 the f1 filter will use 5x15 boxes and the f2 filter will use 5x5 boxes.
def filters_scale(img, scale):
    f1xdim=int(4*scale)
    f1ydim=int(12*scale)
    f2dims=int(4*scale)
    block1=sum_any_size(img,f1xdim,f1ydim)
    f1 = block1[:-4,:]-block1[4:,:]
```

< the 4 here needs to scale also

```
    block2=sum_any_size(img,f2dims,f2dims)
    f2= 2*block2[:,4:-4]-block2[:,:-8]-block2[:,8:]
```

< as do 4 and 8 here

```
    f1xpad= ceil((img.shape[0]-f1.shape[0])/2)
    f2xpad= ceil((img.shape[0]-f2.shape[0])/2)
    f1ypad= ceil((img.shape[1]-f1.shape[1])/2)
    f2ypad= ceil((img.shape[1]-f2.shape[1])/2)
    f1pad = np.pad(f1,((f1xpad,f1xpad),(f1ypad,f1ypad)))
    f2pad= np.pad(f2,((f2xpad,f2xpad),(f2ypad,f2ypad)))
    return f1pad, f2pad
f1v,f2v= filters_scale(robeson,1.25)

#Use the filter computations in a function that takes an image plus scale as input, and retu
rns
# the y and x for all faces detected at that scale
def face_cord_scale(img, scale):
```

Thresholds should be *scale^2

```
    f1val,f2val= filters_scale(img,scale)
    y,x = np.nonzero(np.logical_and(f1val>(20*scale),np.roll(f2val,-4,0)>(16*scale)))
    return np.dstack([x,y])

#Write one more function that will call the one above in a loop, at different scales separat
ed
# by multiples of 1.25. It should return the scale and coordinates for each detection.
def face_loops(img):
```

Start at 0.8

```
    scale=1.25
    while scale <=10:
        print("coordinates for faces at scale " + str(scale))
        f1v,f2v= filters_scale(robeson, scale)
        y,x = np.nonzero(np.logical_and(f1v>(20),np.roll(f2v,-4,0)>(16)))
        showFaces(img,x,y)
        print(face_cord_scale(img,scale))
        scale=scale+1.25
```

Multiply by 1.25

```
face_loops(robeson)

#reflection
# I found this project really interesting, but i definitely see the pitfalls of doing face d
etection this way.
#By trying other iamges, the detector hardly ever picted up on faces that were not white and
 that is unsettling
# because i never really thought of these programs as being able to be racist in that sense
and that was a really
#itnersting fact to come out of this. I also noticed that there were many cases where sharp
contrast between black
# and white or dark shadows in pictures were picked as "faces" and im curious to see how to
fix that. It seems prety
# inconsistent and that's a tad bit annoying.


# I did not work with anyone. I used the Numpy documentation and scikit documentation along
with the resources you provided
========================================
```

I agree: the results here show some basic shortcomings to the algorithm. For the full Viola-Jones algorithm, this would be only the first step, and the thresholds would be deliberately set low so as to avoid ruling out any faces if at all possible.

In [ ]:
```
import cv2 as cv
import numpy as np
from scipy.ndimage import label
from scipy.spatial import distance_matrix
import matplotlib.pyplot as plt
import matplotlib as mpl
from math import ceil

mpl.rc('image', cmap='gray')

def imshow(img,cmap=None):
    plt.imshow(img)
    plt.axis('off')
    if cmap:
        plt.set_cmap(cmap)
    plt.show()

robeson = cv.imread('robeson.jpg',0).astype(np.float32)/255.0
imshow(robeson)

#create integral image
def integral_image(img):
    return np.cumsum(np.cumsum(img, axis=0), axis=1)
iir = integral_image(robeson)

# return the f1 values for any image.
def f1_values(img,integral):
    r_4x12 = integral[4:,12:]+integral[:-4,:-12]-integral[4:,:-12]-integral[:-4,12:]
    f1 = r_4x12[:-4,:]-r_4x12[4:,:]
    f1pad = np.pad(f1,((4,4),(6,6)))
    plt.figure
    plt.imshow(img)
    y,x = np.nonzero(f1pad>20)#;
    plt.plot(x,y,'r.')#;
    plt.axis('off')
    plt.show()
    return (f1pad)

f1values= f1_values(robeson,iir)

def f2_values(img, integral):
    r_4x4 = integral[4:,4:]+integral[:-4,:-4]-integral[:-4,4:]-integral[4:,:-4]
    f2= 2*r_4x4[:,4:-4]-r_4x4[:,:-8]-r_4x4[:,8:]
    f2pad=np.pad (f2,((2,2),(6,6)))
    plt.figure
    plt.imshow(img)
    y,x = np.nonzero(f2pad>20)#;
    plt.plot(x,y,'r.')#;
    plt.axis('off')
    plt.show()
    return (f2pad)

f2values=f2_values(robeson,iir)

def showFaces(img,x,y):
    '''Draws approximate boxes around detected face points.'''
    plt.figure
    plt.imshow(robeson)
```

```
    ax = plt.gca()
    for i in range(len(y)):
        r = mpl.patches.Rectangle((x[i]-12,y[i]-12),24,36,edgecolor='r',fill=False)
        ax.add_patch(r)
    #plt.plot(x,y,'c.')#;
    plt.axis('off')
    plt.show()

# assumes you have already defined f1p and f2p above.
y,x = np.nonzero(np.logical_and(f1values>20,np.roll(f2values,-4,0)>16))#;
showFaces(robeson,x,y)

# TODO: write a single function that takes an image and returns the y,x coordinates of
# faces detected using the method above.  Call it on the image scaled by different amo
# 1.25, 0.8, 0.64, etc. (use cv.resize).
def face_coord(img,resizeamt):
    img=cv.resize(img,None,fx=resizeamt,fy=resizeamt)
    integralimg= integral_image(img)
    f1value=f1_values(img,integralimg)
    f2value=f2_values(img,integralimg)
    y,x=np.nonzero(np.logical_and(f1value>20,np.roll(f2value,-4,0)>16))
    showFaces(img,x,y)
    return np.dstack([x,y])

print(face_coord(robeson, .64))
print(face_coord(robeson, .8))
print(face_coord(robeson, 1.25))
print(face_coord(robeson, 1.75))

#TODO: To complete the detection system, we therefore need to do a few things:

#Write a general rectangle-sum function that can compute the sums for rectangles of ar
def sum_any_size(img,xdim,ydim):
    integral=integral_image(img)
    block=integral[xdim:,ydim:]+integral[:-xdim,:-ydim]-integral[xdim:,:-ydim]-integra
    return(block)

#Write functions that can compute the f1 and f2 filters, given a scale. For example, a
# 1.25 the f1 filter will use 5x15 boxes and the f2 filter will use 5x5 boxes.
def filters_scale(img, scale):
    f1xdim=int(4*scale)
    f1ydim=int(12*scale)
    f2dims=int(4*scale)
    block1=sum_any_size(img,f1xdim,f1ydim)
    f1 = block1[:-4,:]-block1[4:,:]
    block2=sum_any_size(img,f2dims,f2dims)
    f2= 2*block2[:,4:-4]-block2[:,:-8]-block2[:,8:]
    f1xpad= ceil((img.shape[0]-f1.shape[0])/2)
    f2xpad= ceil((img.shape[0]-f2.shape[0])/2)
    f1ypad= ceil((img.shape[1]-f1.shape[1])/2)
    f2ypad= ceil((img.shape[1]-f2.shape[1])/2)
    f1pad = np.pad(f1,((f1xpad,f1xpad),(f1ypad,f1ypad)))
    f2pad= np.pad(f2,((f2xpad,f2xpad),(f2ypad,f2ypad)))
    return f1pad, f2pad
f1v,f2v= filters_scale(robeson,1.25)

#Use the filter computations in a function that takes an image plus scale as input, ar
# the y and x for all faces detected at that scale
def face_cord_scale(img, scale):
    f1val,f2val= filters_scale(img,scale)
```

```
    y,x = np.nonzero(np.logical_and(f1val>(20*scale),np.roll(f2val,-4,0)>(16*scale)))
    return np.dstack([x,y])

#Write one more function that will call the one above in a loop, at different scales s
# by multiples of 1.25. It should return the scale and coordinates for each detection.
def face_loops(img):
    scale=1.25
    while scale <=10:
        print("coordinates for faces at scale " + str(scale))
        f1v,f2v= filters_scale(robeson, scale)
        y,x = np.nonzero(np.logical_and(f1v>(20),np.roll(f2v,-4,0)>(16)))
        showFaces(img,x,y)
        print(face_cord_scale(img,scale))
        scale=scale+1.25
face_loops(robeson)

#reflection
# I found this project really interesting, but i definitely see the pitfalls of doing
#By trying other iamges, the detector hardly ever picted up on faces that were not whi
# because i never really thought of these programs as being able to be racist in that
#itnersting fact to come out of this. I also noticed that there were many cases where
# and white or dark shadows in pictures were picked as "faces" and im curious to see h
# inconsistent and that's a tad bit annoying.


# I did not work with anyone. I used the Numpy documentation and scikit documentation
```


S-3848-2


S-3848-2


S-3848-2


S-3848-2


S-3848-2


S-3848-2

[ ]


S-3848-2


S-3848-2


```
[[[168  71]
  [168  72]
  [111  87]
  [112  87]
  [259 103]
  [ 31 107]
  [ 31 108]
  [ 97 110]
  [ 98 110]
  [ 78 133]
  [ 79 133]
  [ 78 134]
  [ 79 134]
  [ 78 135]
  [467 135]
  [468 135]
  [ 46 175]
  [431 187]
  [432 187]
  [432 188]]]
```


S-3848-2


S-3848-2

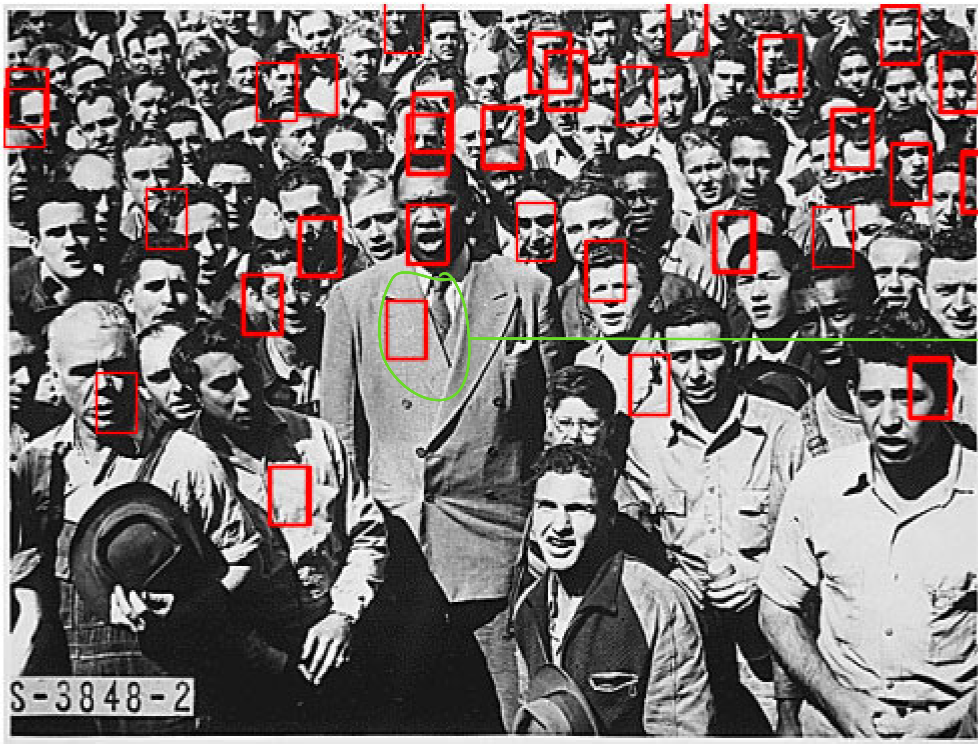

S-3848-2

```
[[[189   4]
  [190   4]
  [208   4]
  [209   4]
  [501  14]
  [500  15]
  [501  15]
  [238  20]
  [685  26]
  [417  28]
  [417  29]
  [586  30]
  [586  31]
  [276  34]
  [  9  35]
  [ 10  35]
  [275  35]
  [276  35]
  [217  52]
  [555  52]
  [217  53]
  [370  53]
  [371  53]
  [370  54]
  [371  54]
  [722  58]
  [723  58]
  [722  59]
  [723  59]
  [686  60]
  [602  62]
  [427  72]
  [ 71  88]
  [453  91]
  [453  92]
  [185  94]
  [232  94]
  [184  95]
  [185  95]
  [362  95]
  [361  96]
  [362  96]
  [361  97]
  [362  97]
  [318  98]
  [319  98]
  [318  99]
  [319  99]
  [137 101]
  [262 111]
  [263 111]
  [262 112]
  [263 112]
  [264 112]
  [628 112]
  [628 113]
  [655 124]
  [112 125]
  [112 126]
  [113 126]
```

```
 [532 135]
 [175 136]
 [174 137]
 [175 137]
 [ 76 142]
 [722 143]
 [723 143]
 [722 144]
 [723 144]
 [226 156]
 [227 156]
 [282 161]
 [404 162]
 [404 163]
 [405 163]
 [ 48 169]
 [ 49 169]
 [153 172]
 [441 182]
 [442 182]
 [679 200]
 [680 200]
 [681 200]
 [680 201]
 [681 201]
 [123 209]
 [124 209]
 [123 210]
 [124 210]
 [123 211]
 [124 211]
 [463 212]
 [731 212]
 [732 212]
 [730 213]
 [731 213]
 [732 213]
 [527 263]
 [526 264]
 [527 264]
 [177 303]
 [709 533]]]
```


S-3848-2


S-3848-2



What's going on here?

```
[[[ 244    5]
  [ 418    5]
  [ 417    6]
  [ 418    6]
  [ 548   11]
  [ 548   12]
  [ 701   21]
  [ 702   21]
  [ 643   27]
  [ 333   28]
  [ 334   28]
  [ 643   28]
  [ 333   29]
  [ 334   29]
  [ 475   30]
  [ 474   31]
  [ 475   31]
  [ 698   35]
  [ 699   35]
  [ 698   36]
  [ 699   36]
  [ 613   37]
  [ 959   37]
  [ 960   37]
  [ 959   38]
  [ 960   38]
  [ 343   39]
  [ 344   39]
  [ 959   39]
  [ 960   39]
  [ 343   40]
  [ 585   40]
  [ 584   41]
  [ 585   41]
  [ 191   42]
  [ 821   42]
  [ 191   43]
  [ 820   43]
  [ 821   43]
  [ 822   43]
  [ 820   44]
  [ 821   44]
  [ 822   44]
  [ 166   46]
  [ 386   49]
  [ 387   49]
  [  13   50]
  [  14   50]
  [  12   62]
  [ 261   65]
  [ 262   65]
  [ 261   66]
  [ 262   66]
  [ 304   73]
  [ 305   73]
  [ 778   73]
  [ 304   74]
  [ 305   74]
  [ 777   74]
  [ 778   74]
```

```
 [ 304   75]
 [ 305   75]
 [ 518   75]
 [ 519   75]
 [ 777   75]
 [ 778   75]
 [ 518   76]
 [ 519   76]
 [ 258   78]
 [ 259   78]
 [ 639   78]
 [ 640   78]
 [ 258   79]
 [ 259   79]
 [ 639   79]
 [ 640   79]
 [1012   82]
 [1012   83]
 [ 961   85]
 [ 962   85]
 [ 843   87]
 [ 842   88]
 [ 843   88]
 [ 842   89]
 [ 843   89]
 [ 554   96]
 [ 710   96]
 [ 711   96]
 [ 555   97]
 [ 597  101]
 [ 598  101]
 [ 597  102]
 [ 598  102]
 [ 597  103]
 [ 598  103]
 [  99  124]
 [ 634  128]
 [ 634  129]
 [ 635  129]
 [ 634  130]
 [ 635  130]
 [ 325  132]
 [ 259  133]
 [ 258  134]
 [ 507  135]
 [ 445  138]
 [ 446  138]
 [ 447  138]
 [ 445  139]
 [ 446  139]
 [ 447  139]
 [ 192  141]
 [ 193  141]
 [ 192  142]
 [ 193  142]
 [ 367  156]
 [ 368  156]
 [ 879  158]
 [ 880  158]
 [ 879  159]
```

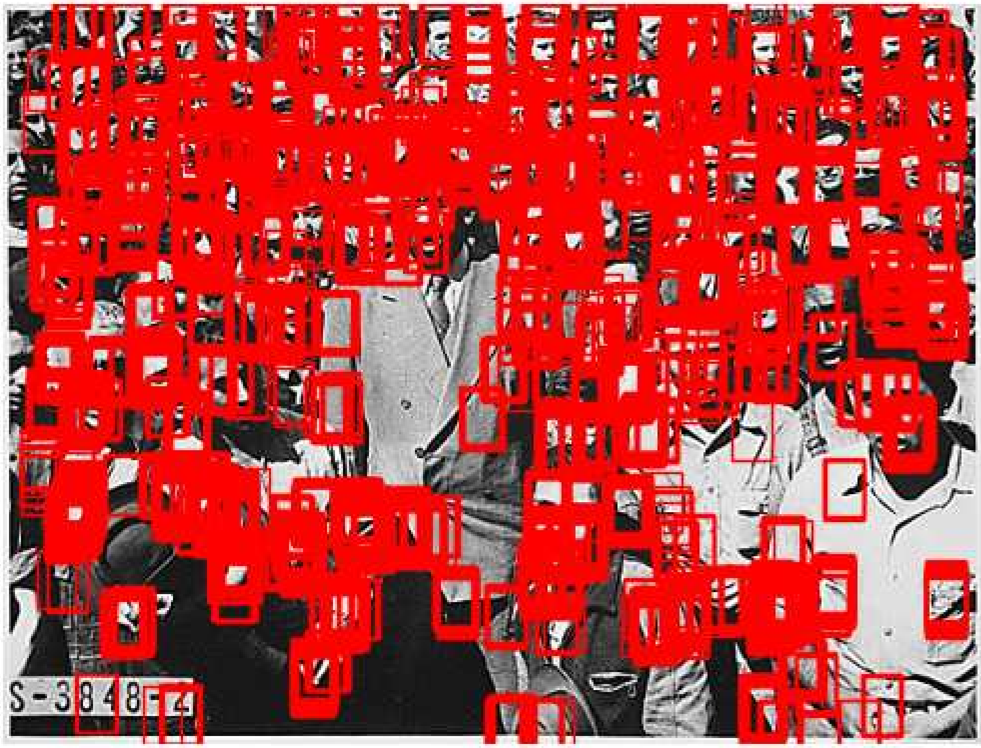
```
  [ 880  159]
  [ 879  160]
  [ 880  160]
  [ 749  172]
  [ 917  173]
  [ 918  173]
  [ 917  174]
  [ 918  174]
  [ 917  175]
  [ 918  175]
  [ 157  176]
  [ 157  177]
  [ 158  177]
  [ 245  192]
  [ 246  192]
  [ 394  226]
  [ 395  226]
  [ 566  227]
  [ 566  228]
  [ 566  229]
  [ 567  229]
  [  68  237]
  [ 619  255]
  [ 619  256]
  [ 952  281]
  [ 953  281]
  [ 173  294]
  [ 174  294]
  [ 175  294]
  [1024  298]
  [ 737  368]
  [ 737  369]
  [ 738  369]
  [ 737  370]
  [ 738  370]
  [  65  467]
  [  14  726]
  [ 191  726]]]
coordinates for faces at scale 1.25
```


S-3848-2

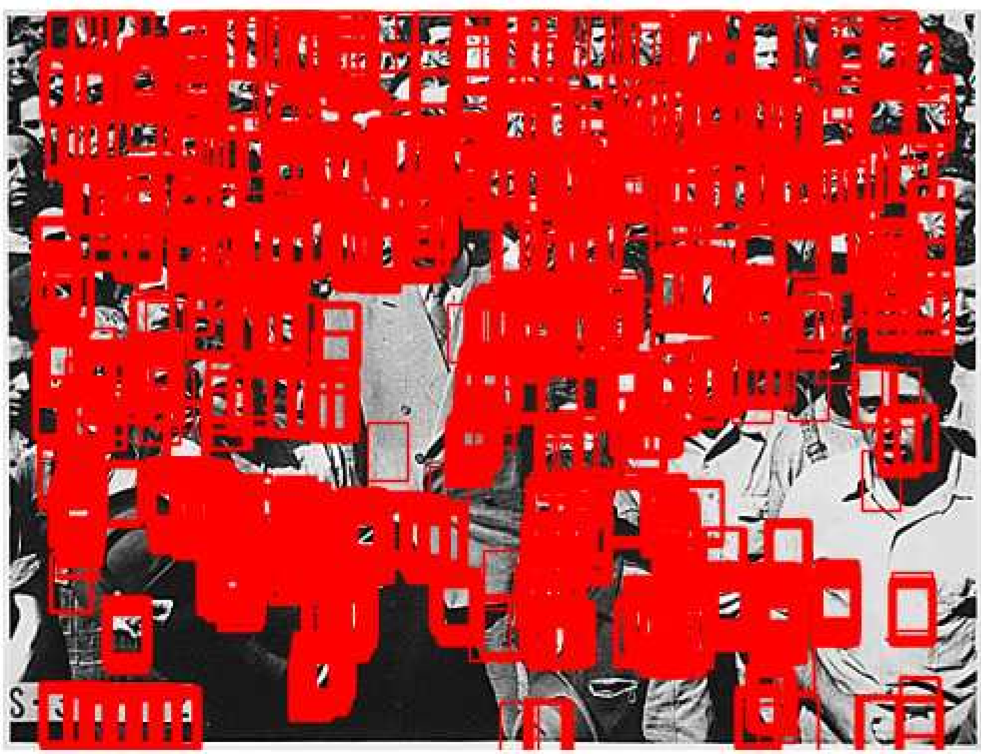
```
[[[296  43]
  [297  43]
  [296  44]
  [297  44]
  [186  75]
  [148  76]
  [186  76]
  [148  77]
  [255  78]
  [254  79]
  [254  80]
  [210  89]
  [211  89]
  [210  90]
  [211  90]
  [242  90]
  [210  91]
  [211  91]
  [ 90 101]
  [ 90 102]
  [426 108]
  [140 109]
  [140 110]
  [141 110]
  [ 60 114]
  [ 61 114]
  [324 130]
  [325 130]
  [324 131]
  [325 131]
  [355 132]
  [ 39 135]
  [ 40 135]
  [ 39 136]
  [ 40 136]
  [122 139]
  [123 139]
  [544 159]
  [545 159]
  [544 160]
  [ 99 167]
  [174 167]
  [ 99 168]
  [100 168]
  [174 168]
  [175 168]
  [ 98 169]
  [ 99 169]
  [100 169]
  [175 169]
  [585 169]
  [585 170]
  [586 170]
  [584 171]
  [585 171]
  [552 172]
  [422 211]
  [422 212]
  [540 234]
  [541 234]
```

```
  [539 235]
  [540 235]
  [541 235]
  [540 236]
  [541 236]
  [540 237]
  [340 304]
  [341 304]
  [568 426]
  [568 427]]]
coordinates for faces at scale 2.5
```


```
[]
coordinates for faces at scale 3.75
```




```
[]
coordinates for faces at scale 5.0
```



```
[]
coordinates for faces at scale 6.25
```

In [ ]:


S-3848-2