Face Detection

In [1]:
```
import cv2
import dlib
import os
import imutils
import math
import random
import glob
import math
import sys

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import scipy.ndimage as nd
import skimage.feature as ft

from skimage import data
from scipy.spatial import distance
from imutils import face_utils
```

In [2]:
```
# settings for LBP
METHOD = 'default'
R = 2
P = 8
```

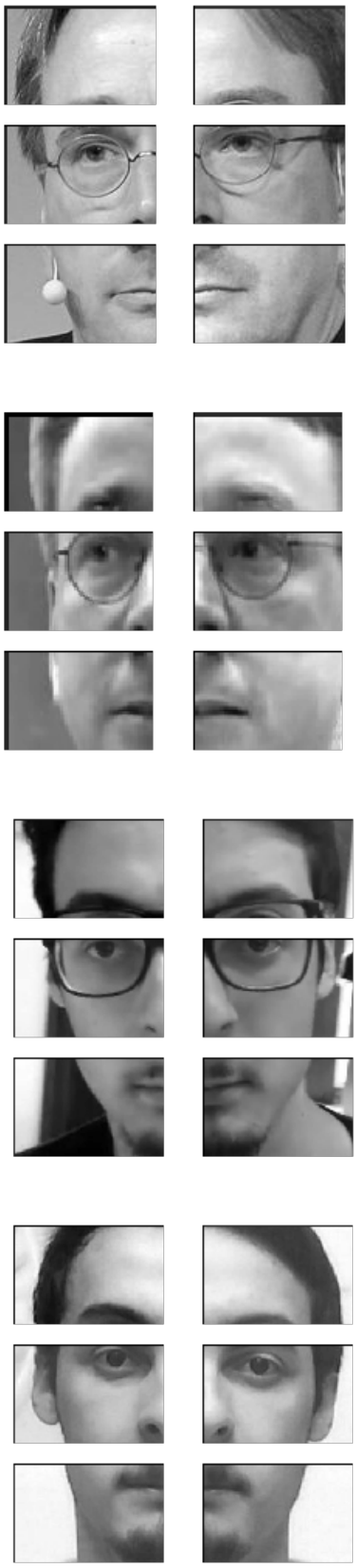
In [3]:
```
images1 = os.listdir("img/sample1")
images2 = os.listdir("img/sample2")

images1.sort()
images2.sort()
```

Esercizio 1

In [4]:
```
def face_detection(path, or_image=0):
    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

    if or_image != 0:
        img = path
    else:
        img = cv2.imread(path)

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

    crop_img = img[y:y+h, x:x+w]

    p = path.rfind("/")
    if p > -1:
        path = path[p+1:]

    cv2.imwrite("img/faces/1_"+path, crop_img)
    return crop_img
```

In [5]:
```
def get_descriptor(path, show_plot=False):
    img = cv2.imread(path)
    img = face_detection(path)
    h, w, channels = img.shape

    w1 = (w/2)-1
    h1 = (h/3)-1

    cv2.line(img, (w1, 0), (w1, h), (0, 0, 0))
    cv2.line(img, (0, h1), (w, h1), (0, 0, 0))
    cv2.line(img, (0, h1*2), (w, h1*2), (0, 0, 0))

    # convert to gray
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    #plt.imshow(img2, cmap='gray')
    #plt.show()

    d = {}
    d["tl"] = img[0:0+h1,    0:w1]      # top left
    d["tr"] = img[0:0+h1,    w1:w1*2]   # top right
    d["cl"] = img[h1:h1*2,   0:w1]      # center left
    d["cr"] = img[h1:h1*2,   w1:w1*2]   # center right
    d["bl"] = img[h1*2:h1*3, 0:w1]      # bottom left
    d["br"] = img[h1*2:h1*3, w1:w1*2]   # bottom right

    if (show_plot):
        plt.figure(figsize=[4,4])
        plt.subplot(3, 2, 1)
        plt.axis('off')
        plt.imshow(d["tl"], cmap='gray')
        plt.subplot(3, 2, 2)
        plt.axis('off')
        plt.imshow(d["tr"], cmap='gray')
        plt.subplot(3, 2, 3)
        plt.axis('off')
        plt.imshow(d["cl"], cmap='gray')
        plt.subplot(3, 2, 4)
        plt.axis('off')
        plt.imshow(d["cr"], cmap='gray')
        plt.subplot(3, 2, 5)
        plt.axis('off')
        plt.imshow(d["bl"], cmap='gray')
        plt.subplot(3, 2, 6)
        plt.axis('off')
        plt.imshow(d["br"], cmap='gray')
        plt.show()

    vett = []
    for i in d:
        tmp = ft.local_binary_pattern(d[i], P, R, METHOD)
        h, _ = np.histogram(tmp.ravel(), normed=True, bins=8, range=(0, 8))
        vett.append(h)

    #plt.close()
    v = np.hstack((vett[0], vett[1], vett[2], vett[3], vett[4], vett[5]))
    #plt.hist(v)
    #plt.show()

    return v
```

In [6]:
```
vett1 = []
vett2 = []
for i,v in enumerate(images1):
    vett1.append(get_descriptor("img/sample1/"+images1[i], True)) # True per
mostrare i plot
    vett2.append(get_descriptor("img/sample2/"+images2[i], True)) # True per
mostrare i plot
```



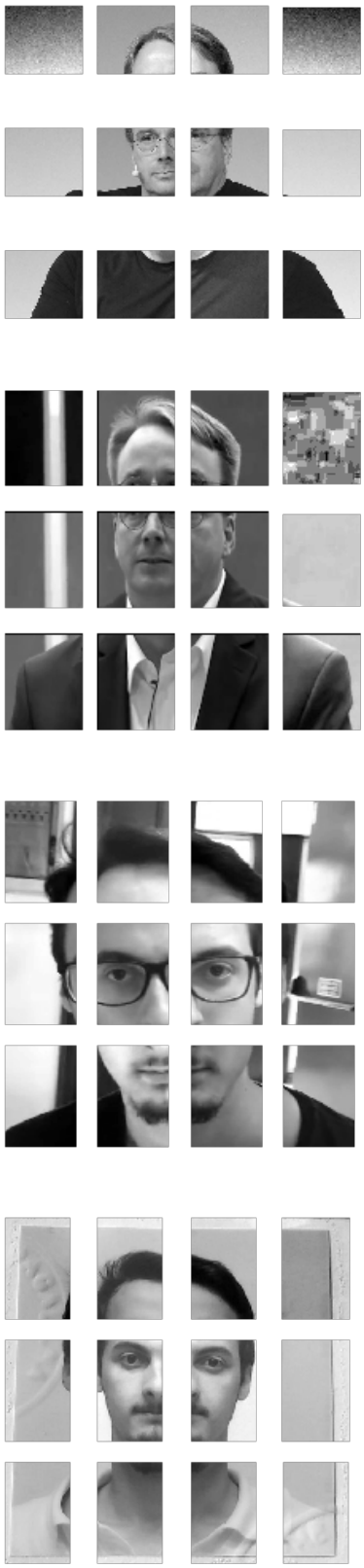
In [18]:
```
mean = 0
for i,v in enumerate(images1):
    dst = distance.euclidean(vett1[i], vett2[i])
    print images1[i][:-4] + " " + str(dst)
    mean += dst

mean /= len(images1)

print "\n\nMedia: " + str(mean) + " (soglia) "
```

```
LinusTorvalds 1.16967166475
Stefano 1.12630511159


Media: 1.14798838817 (soglia)
```

Esercizio 2

In [8]:
```
def get_descriptor2(path, show_plot=False):
    img = cv2.imread(path)
#    img = face_detection(path)
    h, w, channels = img.shape

    w1 = (w/4)-1
    h1 = (h/3)-1

    cv2.line(img, (w1, 0), (w1, h), (0, 0, 0))
    cv2.line(img, (0, h1), (w, h1), (0, 0, 0))
    cv2.line(img, (0, h1*2), (w, h1*2), (0, 0, 0))

    # convert to gray
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    #plt.imshow(img2, cmap='gray')
    #plt.show()

    d = {}
    d["tl"] = img[0:0+h1,    0:w1]         # top left
    d["tcl"] = img[0:0+h1,    w1:w1*2]     # top center left
    d["tcr"] = img[0:0+h1,    w1*2:w1*3]   # top center right
    d["tr"] = img[0:0+h1,    w1*3:w]       # top right

    d["cl"] = img[h1:h1*2,   0:w1]         # center left
    d["c2l"] = img[h1:h1*2,  w1:w1*2]      # center left (2)
    d["c2r"] = img[h1:h1*2,  w1*2:w1*3]    # center right (2)
    d["cr"] = img[h1:h1*2,   w1*3:w]       # center right

    d["bl"] = img[h1*2:h1*3, 0:w1]         # bottom left
    d["bcl"] = img[h1*2:h1*3, w1:w1*2]     # bottom center left
    d["bcr"] = img[h1*2:h1*3, w1*2:w1*3]   # bottom center left
    d["br"] = img[h1*2:h1*3, w1*3:w1*4]    # bottom right

    if (show_plot):
        plt.figure(figsize=[4,4])
        plt.subplot(3, 4, 1)
        plt.axis('off')
        plt.imshow(d["tl"], cmap='gray')
        plt.subplot(3, 4, 2)
        plt.axis('off')
        plt.imshow(d["tcl"], cmap='gray')
        plt.subplot(3, 4, 3)
        plt.axis('off')
        plt.imshow(d["tcr"], cmap='gray')
        plt.subplot(3, 4, 4)
        plt.axis('off')
        plt.imshow(d["tr"], cmap='gray')
        plt.subplot(3, 4, 5)
        plt.axis('off')
        plt.imshow(d["cl"], cmap='gray')
        plt.subplot(3, 4, 6)
        plt.axis('off')
        plt.imshow(d["c2l"], cmap='gray')
        plt.subplot(3, 4, 7)
        plt.axis('off')
        plt.imshow(d["c2r"], cmap='gray')
        plt.subplot(3, 4, 8)
        plt.axis('off')
        plt.imshow(d["cr"], cmap='gray')
        plt.subplot(3, 4, 9)
        plt.axis('off')
        plt.imshow(d["bl"], cmap='gray')
        plt.subplot(3, 4, 10)
        plt.axis('off')
        plt.imshow(d["bcl"], cmap='gray')
        plt.subplot(3, 4, 11)
        plt.axis('off')
```

In [9]:
```
vett1 = []
vett2 = []
for i,v in enumerate(images1):
    vett1.append(get_descriptor2("img/sample1/"+images1[i], True)) # True pe
r mostrare i plot
    vett2.append(get_descriptor2("img/sample2/"+images2[i], True)) # True pe
r mostrare i plot
```



In [19]:
```
mean = 0
for i,v in enumerate(images1):
    dst = distance.euclidean(vett1[i], vett2[i])
    print images1[i][:-4] + " " + str(dst)
    mean += dst

mean /= len(images1)

print "\n\nMedia: " + str(mean) + " (soglia) "
```

```
LinusTorvalds 1.16967166475
Stefano 1.12630511159


Media: 1.14798838817 (soglia)
```

Esercizio 3

In [11]:
```
def extract_face_description(img, predictor_path, model_path):
    predictor = dlib.shape_predictor(predictor_path)
    facerec = dlib.face_recognition_model_v1(model_path)

    img = imutils.resize(img, width=500)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    rect = dlib.rectangle(0,0,img.shape[0],img.shape[1])
    shape = predictor(gray, rect)

    return facerec.compute_face_descriptor(img,shape)

def compare_faces_landmark_trained_deep(face, doc, predictor_path, model_pat
h):
    face_description = extract_face_description(face, predictor_path, model_
path)
    doc_description = extract_face_description(doc, predictor_path, model_pa
th)

    p = np.asarray(face_description).flatten()
    q = np.asarray(doc_description).flatten()
    distance = np.sqrt(np.sum(np.power((p - q), 2)))

    if distance < 0.6:
        return True,distance
    else:
        return False,distance
```

In [12]:
```
for i,v in enumerate(images1):
    img1 = cv2.imread("img/sample1/" + images1[i])
    img2 = cv2.imread("img/sample2/" + images2[i])
    (if_match, tmp) = compare_faces_landmark_trained_deep(img1, img2, "shape
_predictor_68_face_landmarks.dat", "dlib_face_recognition_resnet_model_v1.da
t")

    match = "No"
    if if_match:
        match = "Si"

    print images1[i][:-4] + " " + str(tmp) + " " + match
```

```
LinusTorvalds 0.5049584830291455 Si
Stefano 0.39861340897206254 Si
```

Commenti e critiche

L'algoritmo LBP ottiene risultati se accompagnato dalla face_detection(), quindi se, come nel primo esercizio e contrariamente al secondo, lavora su un immagine dove è stata applicata la face_detection e ritagliato il relativo quadrato che individua il volto del soggetto nell'immagine.

Come soglia ho scelto la media deila distanza dei soggetti (0.52 nel primo esercizio e 0.57 nel secondo esercizio).

L'algoritmo LBP è meno accurato dell'algoritmo basato su Landmark. L'algoritmo Landmark è computazionalmente più pesante ma è più efficiente rispetto all'algoritmo LBP.